IV. Approximation Algorithms via Exact Algorithms

Thomas Sauerwald
The Subset-Sum Problem

Parallel Machine Scheduling

Bonus Material: A PTAS for Parallel Machine Scheduling (non-examinable)
The Subset-Sum Problem

- **Given:** Set of positive integers \( S = \{x_1, x_2, \ldots, x_n\} \) and positive integer \( t \)
- **Goal:** Find a subset \( S' \subseteq S \) which maximizes \( \sum_{i: x_i \in S'} x_i \leq t \).
The Subset-Sum Problem

- **Given:** Set of positive integers $S = \{x_1, x_2, \ldots, x_n\}$ and positive integer $t$
- **Goal:** Find a subset $S' \subseteq S$ which maximizes $\sum_{i: x_i \in S'} x_i \leq t$.

This problem is **NP-hard**
The Subset-Sum Problem

- **Given:** Set of positive integers \( S = \{x_1, x_2, \ldots, x_n\} \) and positive integer \( t \)
- **Goal:** Find a subset \( S' \subseteq S \) which maximizes \( \sum_{i: x_i \in S'} x_i \leq t \).

\[ x_1 = 10, \quad x_2 = 4, \quad x_3 = 5, \quad x_4 = 6, \quad x_5 = 1 \]

\( t = 13 \) tons
The Subset-Sum Problem

- **Given:** Set of positive integers $S = \{x_1, x_2, \ldots, x_n\}$ and positive integer $t$
- **Goal:** Find a subset $S' \subseteq S$ which maximizes $\sum_{i: x_i \in S'} x_i \leq t$.

IV. Approximation via Exact Algorithms

---

$t = 13$ tons

- $x_1 = 10$
- $x_2 = 4$
- $x_3 = 5$
- $x_4 = 6$
- $x_5 = 1$
The Subset-Sum Problem

- **Given:** Set of positive integers \( S = \{x_1, x_2, \ldots, x_n\} \) and positive integer \( t \)
- **Goal:** Find a subset \( S' \subseteq S \) which maximizes \( \sum_{i: x_i \in S'} x_i \leq t \).

---

IV. Approximation via Exact Algorithms

The Subset-Sum Problem

\[ t = 13 \text{ tons} \]

- \( x_1 = 10 \)
- \( x_2 = 4 \)
- \( x_3 = 5 \)
- \( x_4 = 6 \)
- \( x_5 = 1 \)
Given: Set of positive integers $S = \{x_1, x_2, \ldots, x_n\}$ and positive integer $t$
Goal: Find a subset $S' \subseteq S$ which maximizes $\sum_{i: x_i \in S'} x_i \leq t$.

$\begin{align*}
x_1 &= 10 \\
x_2 &= 4 \\
x_3 &= 5 \\
x_4 &= 6 \\
x_5 &= 1
\end{align*}$

$t = 13$ tons
The Subset-Sum Problem

Given: Set of positive integers \( S = \{x_1, x_2, \ldots, x_n\} \) and positive integer \( t \)

Goal: Find a subset \( S' \subseteq S \) which maximizes \( \sum_{i: x_i \in S'} x_i \leq t \).

IV. Approximation via Exact Algorithms
The Subset-Sum Problem

**Given:** Set of positive integers $S = \{x_1, x_2, \ldots, x_n\}$ and positive integer $t$

**Goal:** Find a subset $S' \subseteq S$ which maximizes $\sum_{i} x_{i} : x_i \in S'$ such that $x_i \leq t$.

---

IV. Approximation via Exact Algorithms

- $x_1 = 10$
- $x_2 = 4$
- $x_3 = 5$
- $x_4 = 6$
- $x_5 = 1$

$t = 13$ tons

$x_3 + x_4 + x_5 = 12$
Dynamic Programming: Compute bottom-up all possible sums $\leq t$
Dynamic Programming: Compute bottom-up all possible sums $\leq t$

**Exact-Subset-Sum** ($S, t$)

1. $n = |S|$  
2. $L_0 = \langle 0 \rangle$  
3. **for** $i = 1$ **to** $n$
   
4. $L_i = \text{MERGE-LISTS}(L_{i-1}, L_{i-1} + x_i)$
5. **remove** from $L_i$ every element that is greater than $t$
6. **return** the largest element in $L_n$
Dynamic Programming: Compute bottom-up all possible sums $\leq t$

**EXACT-SUBSET-SUM**($S, t$)

1. $n = |S|$
2. $L_0 = \langle 0 \rangle$
3. for $i = 1$ to $n$
   4. $L_i = \text{MERGE-LISTS}(L_{i-1}, L_{i-1} + x_i)$  \[
   S + x := \{s + x : s \in S\}
   \]
5. remove from $L_i$ every element that is greater than $t$
6. return the largest element in $L_n$
An Exact (Exponential-Time) Algorithm

Dynamic Programming: Compute bottom-up all possible sums \( \leq t \)

**Exact-Subset-Sum** \((S, t)\)

1. \( n = |S| \)
2. \( L_0 = \langle 0 \rangle \)
3. for \( i = 1 \) to \( n \)
4. \( L_i = \text{Merge-Lists}(L_{i-1}, L_{i-1} + x_i) \) \( S + x := \{s + x : s \in S\} \)
5. remove from \( L_i \) every element that is greater than \( t \)
6. return the largest element in \( L_n \)
An Exact (Exponential-Time) Algorithm

Dynamic Programming: Compute bottom-up all possible sums $\leq t$

**Exact-Subset-Sum**($S, t$)

1. $n = |S|$
2. $L_0 = \langle 0 \rangle$
3. **for** $i = 1$ **to** $n$
   4. $L_i = \text{Merge-Lists}(L_{i-1}, L_{i-1} + x_i)$
   5. remove from $L_i$ every element that is greater than $t$
4. **return** the largest element in $L_n$

implementable in time $O(|L_{i-1}|)$ (like Merge-Sort)

Returns the merged list (in sorted order and without duplicates)
An Exact (Exponential-Time) Algorithm

Dynamic Programming: Compute bottom-up all possible sums $\leq t$

**Exact-Subset-Sum**($S, t$)

1. $n = |S|$
2. $L_0 = \langle 0 \rangle$
3. for $i = 1$ to $n$
4. \hspace{1em} $L_i = \text{Merge-List}(L_{i-1}, L_{i-1} + x_i)$
5. \hspace{1em} remove from $L_i$ every element that is greater than $t$
6. return the largest element in $L_n$

Example:
An Exact (Exponential-Time) Algorithm

Dynamic Programming: Compute bottom-up all possible sums $\leq t$

**Exact-Subset-Sum** $(S, t)$

1. $n = |S|$
2. $L_0 = \langle 0 \rangle$
3. for $i = 1$ to $n$
4.    $L_i = \text{Merge-Lists}(L_{i-1}, L_{i-1} + x_i)$
5.    remove from $L_i$ every element that is greater than $t$
6. return the largest element in $L_n$

Example:
- $S = \{1, 4, 5\}, \ t = 10$
An Exact (Exponential-Time) Algorithm

**Dynamic Programming:** Compute bottom-up all possible sums \( \leq t \)

**Example:**

- \( S = \{1, 4, 5\}, \ t = 10 \)
- \( L_0 = \langle 0 \rangle \)

**Exact-Subset-Sum** \((S, t)\)

1. \( n = |S| \)
2. \( L_0 = \langle 0 \rangle \)
3. for \( i = 1 \) to \( n \)
   4. \( L_i = \text{Merge-Lists}(L_{i-1}, L_{i-1} + x_i) \)
   5. remove from \( L_i \) every element that is greater than \( t \)
6. return the largest element in \( L_n \)

**Example:**

- \( S = \{1, 4, 5\}, \ t = 10 \)
- \( L_0 = \langle 0 \rangle \)
An Exact (Exponential-Time) Algorithm

Dynamic Programming: Compute bottom-up all possible sums $\leq t$

**Exact-Subset-Sum** $(S, t)$

1. $n = |S|$
2. $L_0 = \langle 0 \rangle$
3. for $i = 1$ to $n$
4. \hspace{1em} $L_i = \text{Merge-Lists}(L_{i-1}, L_{i-1} + x_i)$
5. \hspace{1em} remove from $L_i$ every element that is greater than $t$
6. return the largest element in $L_n$

Example:

- $S = \{1, 4, 5\}$, $t = 10$
- $L_0 = \langle 0 \rangle$
- $L_1 = \langle 0, 1 \rangle$
An Exact (Exponential-Time) Algorithm

Dynamic Programming: Compute bottom-up all possible sums \( \leq t \)

**Exact-Subset-Sum** \((S, t)\)

1. \( n = |S| \)
2. \( L_0 = \langle 0 \rangle \)
3. for \( i = 1 \) to \( n \)
4. \( L_i = \text{Merge-Lists}(L_{i-1}, L_{i-1} + x_i) \)
5. remove from \( L_i \) every element that is greater than \( t \)
6. return the largest element in \( L_n \)

Example:

- \( S = \{1, 4, 5\}, t = 10 \)
- \( L_0 = \langle 0 \rangle \)
- \( L_1 = \langle 0, 1 \rangle \)
- \( L_2 = \langle 0, 1, 4, 5 \rangle \)
An Exact (Exponential-Time) Algorithm

Dynamic Programming: Compute bottom-up all possible sums \( \leq t \)

**Exact-Subset-Sum** \((S, t)\)

1. \( n = |S| \)
2. \( L_0 = \{0\} \)
3. for \( i = 1 \) to \( n \)
   4. \( L_i = \text{Merge-Lists}(L_{i-1}, L_{i-1} + x_i) \)
   5. remove from \( L_i \) every element that is greater than \( t \)
6. return the largest element in \( L_n \)

Example:
- \( S = \{1, 4, 5\}, t = 10 \)
- \( L_0 = \{0\} \)
- \( L_1 = \{0, 1\} \)
- \( L_2 = \{0, 1, 4, 5\} \)
- \( L_3 = \{0, 1, 4, 5, 6, 9, 10\} \)
An Exact (Exponential-Time) Algorithm

Dynamic Programming: Compute bottom-up all possible sums $\leq t$

**Exact-Subset-Sum** $(S, t)$

1. $n = |S|$
2. $L_0 = \langle 0 \rangle$
3. for $i = 1$ to $n$
4. \hspace{1em} $L_i = \text{Merge-Lists}(L_{i-1}, L_{i-1} + x_i)$
5. \hspace{1em} remove from $L_i$ every element that is greater than $t$
6. return the largest element in $L_n$

**Example:**
- $S = \{1, 4, 5\}$, $t = 10$
- $L_0 = \langle 0 \rangle$
- $L_1 = \langle 0, 1 \rangle$
- $L_2 = \langle 0, 1, 4, 5 \rangle$
- $L_3 = \langle 0, 1, 4, 5, 6, 9, 10 \rangle$
An Exact (Exponential-Time) Algorithm

Dynamic Programming: Compute bottom-up all possible sums $\leq t$

**Exact-Subset-Sum**$(S, t)$

1. $n = |S|$
2. $L_0 = \langle 0 \rangle$
3. for $i = 1$ to $n$
4. \hspace{1em} $L_i = \text{Merge-Lists}(L_{i-1}, L_{i-1} + x_i)$
5. \hspace{1em} remove from $L_i$ every element that is greater than $t$
6. return the largest element in $L_n$

- **Correctness:** $L_n$ contains all sums of $\{x_1, x_2, \ldots, x_n\}$

Example:

- $S = \{1, 4, 5\}, \ t = 10$
- $L_0 = \langle 0 \rangle$
- $L_1 = \langle 0, 1 \rangle$
- $L_2 = \langle 0, 1, 4, 5 \rangle$
- $L_3 = \langle 0, 1, 4, 5, 6, 9, 10 \rangle$
An Exact (Exponential-Time) Algorithm

**Dynamic Programming:** Compute bottom-up all possible sums \( \leq t \)

**Exact-Subset-Sum** \((S, t)\)

1. \( n = |S| \)
2. \( L_0 = \langle 0 \rangle \)
3. **for** \( i = 1 \) **to** \( n \)
   4. \( L_i = \text{Merge-Lists}\left(L_{i-1}, L_{i-1}, x_i\right) \)
   5. remove from \( L_i \) every element that is greater than \( t \)
   6. **return** the largest element in \( L_i \)

**Example:**

- \( S = \{1, 4, 5\}, \ t = 10 \)
- \( L_0 = \langle 0 \rangle \)
- \( L_1 = \langle 0, 1 \rangle \)
- \( L_2 = \langle 0, 1, 4, 5 \rangle \)
- \( L_3 = \langle 0, 1, 4, 5, 6, 9, 10 \rangle \)

- **Correctness:** \( L_n \) contains all sums of \( \{x_1, x_2, \ldots, x_n\} \)

---

**IV. Approximation via Exact Algorithms**

**The Subset-Sum Problem**

- Better runtime if \( n \) are small.
An Exact (Exponential-Time) Algorithm

### Dynamic Programming: Compute bottom-up all possible sums \( \leq t \)

**EXACT-SUBSET-SUM(\(S, t\))**

1. \( n = |S| \)
2. \( L_0 = \langle 0 \rangle \)
3. for \( i = 1 \) to \( n \)
   4. \( L_i = \text{MERGE-LISTS}(L_{i-1}, L_{i-1} + x_i) \)
   5. remove from \( L_i \) every element that is greater than \( t \)
4. return the largest element in \( L_n \)

- **Correctness:** \( L_n \) contains all sums of \( \{x_1, x_2, \ldots, x_n\} \)
- **Runtime:** \( O(2^1 + 2^2 + \cdots + 2^n) = O(2^n) \)

**Example:**

- \( S = \{1, 4, 5\}, t = 10 \)
- \( L_0 = \langle 0 \rangle \)
- \( L_1 = \langle 0, 1 \rangle \)
- \( L_2 = \langle 0, 1, 4, 5 \rangle \)
- \( L_3 = \langle 0, 1, 4, 5, 6, 9, 10 \rangle \)
An Exact (Exponential-Time) Algorithm

**Dynamic Programming:** Compute bottom-up all possible sums \( \leq t \)

**Exact-Subset-Sum** \((S, t)\)

1. \( n = |S| \)
2. \( L_0 = \langle 0 \rangle \)
3. **for** \( i = 1 \) **to** \( n \)
   4. \( L_i = \text{Merge-Lists}(L_{i-1}, L_{i-1} + x_i) \)
   5. remove from \( L_i \) every element that is greater than \( t \)
4. **return** the largest element in \( L_n \)

**Correctness:** \( L_n \) contains all sums of \( \{x_1, x_2, \ldots, x_n\} \)

**Runtime:** \( O(2^1 + 2^2 + \cdots + 2^n) = O(2^n) \)

**Example:**
- \( S = \{1, 4, 5\} \)
- \( L_0 = \langle 0 \rangle \)
- \( L_1 = \langle 0, 1 \rangle \)
- \( L_2 = \langle 0, 1, 4, 5 \rangle \)
- \( L_3 = \langle 0, 1, 4, 5, 6, 9, 10 \rangle \)
- There are \( 2^i \) subsets of \( \{x_1, x_2, \ldots, x_i\} \).
An Exact (Exponential-Time) Algorithm

**Example:**

- \( S = \{1, 4, 5\} \)
- \( L_0 = \{0\} \)
- \( L_1 = \{0, 1\} \)
- \( L_2 = \{0, 1, 4, 5\} \)
- \( L_3 = \{0, 1, 4, 5, 6, 9, 10\} \)

- **Correctness:** \( L_n \) contains all sums of \( \{x_1, x_2, \ldots, x_n\} \)
- **Runtime:** \( O(2^1 + 2^2 + \cdots + 2^n) = O(2^n) \)

There are \( 2^i \) subsets of \( \{x_1, x_2, \ldots, x_i\} \).

Better runtime if \( t \) and/or \( |L_i| \) are small.
Idea: Don’t need to maintain two values in $L$ which are close to each other.

Given a trimming parameter $0 < \delta < 1$, trimming $L$ yields a sublist $L'$ so that for every $y \in L$:

$$y_1 + \delta \leq z \leq y,$$

where $z \in L'$.

Trimming a List

$L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle$

$\delta = 0.1$

$L' = \langle 10, 12, 15, 20, 23, 29 \rangle$

1130 Chapter 35 Approximation Algorithms

that if two values in $L$ are close to each other, then since we want just an approximate solution, we do not need to maintain both of them explicitly. More precisely, we use a trimming parameter $\delta$ such that $0 < \delta < 1$. When we trim a list $L$ by $\delta$, we remove as many elements from $L$ as possible, in such a way that if $L_0$ is the result of trimming $L$, therefore element $y$ that was removed from $L$, there is an element $\tilde{y}$ still in $L_0$ that approximates $y$, that is,

$$y_1 \leq \tilde{y} \leq y + \delta.$$

We can think of such a $\tilde{y}$ as "representing" $y$ in the new list $L_0$. Each removed element $y$ is represented by a remaining element $\tilde{y}$ satisfying inequality (35.24).

For example, if $\delta = 0.1$ and $L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle$, then we can trim $L$ to obtain $L_0 = \langle 10, 12, 15, 20, 23, 29 \rangle$, where the deleted value $11$ is represented by $10$, the deleted values $21$ and $22$ are represented by $20$, and the deleted value $24$ is represented by $23$. Because every element of the trimmed version of the list is also an element of the original version of the list, trimming can dramatically decrease the number of elements kept while keeping a close (and slightly smaller) representative value in the list for each deleted element.

The following procedure trims list $L = \langle y_1; y_2; \ldots; y_m \rangle$ in time $O(m)$, given $L$ and $\delta$, assuming that $L$ is sorted into monotonically increasing order. The output of the procedure is a trimmed, sorted list.

```
TRIM(L; \delta)

let $m$ be the length of $L$

for $i$ from 2 to $m$

if $y_i > y_{i-1} + \delta$

append $y_i$ onto the end of $L_0$

return $L_0$
```

The procedure scans the elements of $L$ in monotonically increasing order. A number is appended onto the returned list $L_0$ only if it is the first element of $L$ or if it cannot be represented by the most recent number placed into $L_0$.

Given the procedure TRIM, we can construct an approximation scheme as follows. This procedure takes as input a set $S = \{x_1; x_2; \ldots; x_n\}$ of $n$ integers (in arbitrary order), a target integer $t$, and an "approximation parameter" $\epsilon$, where
Towards a FPTAS

Idea: Don’t need to maintain two values in \( L \) which are close to each other.

Trimming a List

- Given a trimming parameter \( 0 < \delta < 1 \)
Towards a FPTAS

Idea: Don’t need to maintain two values in $L$ which are close to each other.

Trimming a List

- Given a trimming parameter $0 < \delta < 1$
-Trimming $L$ yields smaller sublist $L'$ so that for every $y \in L$: $\exists z \in L'$:

\[
\frac{y}{1 + \delta} \leq z \leq y.
\]
Towards a FPTAS

Idea: Don’t need to maintain two values in $L$ which are close to each other.

Trimming a List

- Given a trimming parameter $0 < \delta < 1$
-Trimming $L$ yields smaller sublist $L'$ so that for every $y \in L$: $\exists z \in L'$:

$$\frac{y}{1 + \delta} \leq z \leq y.$$

- $L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle$
Towards a FPTAS

Idea: Don’t need to maintain two values in $L$ which are close to each other.

Trimming a List

- Given a trimming parameter $0 < \delta < 1$
-Trimming $L$ yields smaller sublist $L'$ so that for every $y \in L$: $\exists z \in L'$:

$$\frac{y}{1 + \delta} \leq z \leq y.$$ 

- $L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle$
- $\delta = 0.1$
Towards a FPTAS

Idea: Don’t need to maintain two values in \( L \) which are close to each other.

**Trimming a List**

- Given a trimming parameter \( 0 < \delta < 1 \)
- Trimming \( L \) yields smaller sublist \( L' \) so that for every \( y \in L \):
  \[
  \frac{y}{1 + \delta} \leq z \leq y.
  \]

- \( L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle \)
- \( \delta = 0.1 \)
- \( L' = \langle 10, 12, 15, 20, 23, 29 \rangle \)
Towards a FPTAS

**Idea:** Don’t need to maintain two values in \( L \) which are close to each other.

**Trimming a List**

- Given a trimming parameter \( 0 < \delta < 1 \)
-Trimming \( L \) yields smaller sublist \( L' \) so that for every \( y \in L \):
  \[
  \frac{y}{1 + \delta} \leq z \leq y.
  \]

```
TRIM(L, \delta)
1 let m be the length of L
2 L' = \{y_1\}
3 last = y_1
4 for i = 2 to m
5    if y_i > last \cdot (1 + \delta) \quad // y_i \geq last because L is sorted
6       append y_i onto the end of L'
7       last = y_i
8 return L'
```
Towards a FPTAS

Idea: Don’t need to maintain two values in $L$ which are close to each other.

Trimming a List

- Given a trimming parameter $0 < \delta < 1$
-Trimming $L$ yields smaller sublist $L'$ so that for every $y \in L$: $\exists z \in L'$:
  \[
  \frac{y}{1 + \delta} \leq z \leq y.
  \]

TRIM($L, \delta$)
1. let $m$ be the length of $L$
2. $L' = \langle y_1 \rangle$
3. $last = y_1$
4. for $i = 2$ to $m$
5. \[\text{if } y_i > last \cdot (1 + \delta) \quad // y_i \geq last \text{ because } L \text{ is sorted}\]
6. append $y_i$ onto the end of $L'$
7. $last = y_i$
8. return $L'$

TRIM works in time $\Theta(m)$, if $L$ is given in sorted order.
Illustration of the Trim Operation

**TRIM**(\(L, \delta\))

1. let \(m\) be the length of \(L\)
2. \(L' = \langle y_1 \rangle\)
3. \(last = y_1\)
4. for \(i = 2\) to \(m\)
5. if \(y_i > last \cdot (1 + \delta)\) // \(y_i \geq last\) because \(L\) is sorted
6. append \(y_i\) onto the end of \(L'\)
7. \(last = y_i\)
8. return \(L'\)
Illustration of the Trim Operation

Trim($L, \delta$)
1  let $m$ be the length of $L$
2  $L' = \langle y_1 \rangle$
3  last = $y_1$
4  for $i = 2$ to $m$
5      if $y_i > last \cdot (1 + \delta)$  // $y_i \geq last$ because $L$ is sorted
6          append $y_i$ onto the end of $L'$
7          last = $y_i$
8  return $L'$

$\delta = 0.1$

$L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle$

$L' = \langle \rangle$
Illustration of the Trim Operation

\textsc{Trim}(L, \delta)

1 let \( m \) be the length of \( L \)
2 \( L' = \langle y_1 \rangle \)
3 last = \( y_1 \)
4 \textbf{for} \( i = 2 \) \textbf{to} \( m \)
5 \textbf{if} \( y_i > \text{last} \cdot (1 + \delta) \) \quad // \( y_i \geq \text{last} \) because \( L \) is sorted
6 \quad \text{append} \ y_i \ \text{onto the end of} \ \ L'
7 \quad \text{last} = y_i
8 \textbf{return} \ L'

\[ \delta = 0.1 \]

\[ L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle \]

\[ L' = \langle 10 \rangle \]
Illustration of the Trim Operation

\textbf{TRIM}(L, \delta)

1. let \( m \) be the length of \( L \)
2. \( L' = \langle y_1 \rangle \)
3. \( \text{last} = y_1 \)
4. \textbf{for} \( i = 2 \) \textbf{to} \( m \)
5. \hspace{1em} \textbf{if} \( y_i > \text{last} \cdot (1 + \delta) \) \quad // \ y_i \geq \text{last} \text{ because } L \text{ is sorted}
6. \hspace{2em} \text{append } y_i \text{ onto the end of } L'
7. \hspace{2em} \text{last} = y_i
8. \textbf{return} \ L'

\[ \delta = 0.1 \]

\[ L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle \]

\[ L' = \langle 10 \rangle \]
Illustration of the Trim Operation

**TRIM**(L, δ)

1. Let m be the length of L
2. \( L' = \langle y_1 \rangle \)
3. \( \text{last} = y_1 \)
4. For \( i = 2 \) to \( m \)
5. \( \text{if } y_i > \text{last} \cdot (1 + \delta) \quad \text{// } y_i \geq \text{last} \text{ because } L \text{ is sorted} \)
6. Append \( y_i \) onto the end of \( L' \)
7. \( \text{last} = y_i \)
8. Return \( L' \)

\[ \delta = 0.1 \]

\( L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle \)

\( L' = \langle 10 \rangle \)
Illustration of the Trim Operation

```
TRIM(L, δ)
1    let m be the length of L
2    L' = ⟨y₁⟩
3    last = y₁
4    for i = 2 to m
5        if yₐ > last • (1 + δ)    // yₐ ≥ last because L is sorted
6            append yₐ onto the end of L'
7            last = yₐ
8    return L'
```

δ = 0.1

```
L = ⟨10, 11, 12, 15, 20, 21, 22, 23, 24, 29⟩
```

After the initialization (lines 1-3)

```
L' = ⟨10⟩
```

IV. Approximation via Exact Algorithms
The Subset-Sum Problem
Illustration of the Trim Operation

**Trim** \((L, \delta)\)

1. let \(m\) be the length of \(L\)
2. \(L' = \langle y_1 \rangle\)
3. \(last = y_1\)
4. **for** \(i = 2\) **to** \(m\)
   5. **if** \(y_i > last \cdot (1 + \delta)\)  \(\//\) \(y_i \geq last\) because \(L\) is sorted
   6. append \(y_i\) onto the end of \(L'\)
   7. \(last = y_i\)
8. **return** \(L'\)

\(\delta = 0.1\)

\[L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle\]

\[L' = \langle 10, 12 \rangle\]
Illustration of the Trim Operation

**Trim**(\(L, \delta\))

1. let \(m\) be the length of \(L\)
2. \(L' = \langle y_1 \rangle\)
3. \(last = y_1\)
4. for \(i = 2\) to \(m\)
5. \(\text{if } y_i > last \cdot (1 + \delta)\)  
   // \(y_i \geq last\) because \(L\) is sorted
6. append \(y_i\) onto the end of \(L'\)
7. \(last = y_i\)
8. return \(L'\)

\[\delta = 0.1\]

\[L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle\]

\[L' = \langle 10, 12 \rangle\]
Illustration of the Trim Operation

\( \text{TRIM}(L, \delta) \)

1. let \( m \) be the length of \( L \)
2. \( L' = \langle y_1 \rangle \)
3. last = \( y_1 \)
4. for \( i = 2 \) to \( m \)
   5. if \( y_i > \text{last} \cdot (1 + \delta) \) \( \text{// } y_i \geq \text{last} \) because \( L \) is sorted
   6. append \( y_i \) onto the end of \( L' \)
   7. last = \( y_i \)
5. return \( L' \)

\( \delta = 0.1 \)

\( L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle \)

\( L' = \langle 10, 12 \rangle \)
Illustration of the Trim Operation

Trimmed list:

```
TRIM(L, δ)
1   let m be the length of L
2   L' = ⟨y₁⟩
3   last = y₁
4   for i = 2 to m
5       if yᵢ > last · (1 + δ)  // yᵢ ≥ last because L is sorted
6           append yᵢ onto the end of L'
7           last = yᵢ
8   return L'
```

δ = 0.1

L = ⟨10, 11, 12, 15, 20, 21, 22, 23, 24, 29⟩

L' = ⟨10, 12, 15⟩

IV. Approximation via Exact Algorithms

The Subset-Sum Problem
Illustration of the Trim Operation

\[ \text{TRIM}(L, \delta) \]
\[
\begin{align*}
1 & \text{ let } m \text{ be the length of } L \\
2 & \quad L' = \langle y_1 \rangle \\
3 & \quad \text{last} = y_1 \\
4 & \text{ for } i = 2 \text{ to } m \\
5 & \quad \text{if } y_i > \text{last} \cdot (1 + \delta) \quad \text{// } y_i \geq \text{last} \text{ because } L \text{ is sorted} \\
6 & \quad \quad \text{append } y_i \text{ onto the end of } L' \\
7 & \quad \quad \text{last} = y_i \\
8 & \text{ return } L'
\end{align*}
\]

\[ \delta = 0.1 \]

\[ L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle \]

\[ L' = \langle 10, 12, 15 \rangle \]
Illustration of the Trim Operation

\text{TRIM}(L, \delta)

1. let \(m\) be the length of \(L\)
2. \(L' = \langle y_1 \rangle\)
3. \(last = y_1\)
4. \textbf{for} \(i = 2\) \textbf{to} \(m\)
5. \quad \textbf{if} \(y_i > last \cdot (1 + \delta)\) \hfill \text{\(\backslash\) \(y_i \geq last\) because \(L\) is sorted}
6. \quad \quad \text{append} \(y_i\) \text{onto the end of} \(L'\)
7. \quad \quad \text{last} = y_i
8. \quad \textbf{return} \(L'\)

\[\delta = 0.1\]

\(L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle\)

\(L' = \langle 10, 12, 15 \rangle\)
Illustration of the Trim Operation

\textbf{Trim}(L, \delta)

1. let \( m \) be the length of \( L \)
2. \( L' = \langle y_1 \rangle \)
3. \( \text{last} = y_1 \)
4. \textbf{for} \( i = 2 \) to \( m \)
5. \hspace{1em} \textbf{if} \( y_i > \text{last} \cdot (1 + \delta) \) \hspace{1em} // \( y_i \geq \text{last} \) because \( L \) is sorted
6. \hspace{2em} append \( y_i \) onto the end of \( L' \)
7. \hspace{2em} \( \text{last} = y_i \)
8. \textbf{return} \( L' \)

\[ \delta = 0.1 \]

After the initialization (lines 1-3)

The returned list \( L' \) is:

\[ L' = \langle 10, 12, 15, 20 \rangle \]

\[ L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle \]
Illustration of the Trim Operation

\[ \text{Trim}(L, \delta) \]

1. let \( m \) be the length of \( L \)
2. \( L' = \langle y_1 \rangle \)
3. \( \text{last} = y_1 \)
4. for \( i = 2 \) to \( m \)
5. \hspace{1em} if \( y_i > \text{last} \cdot (1 + \delta) \) \hspace{1em} // \( y_i \geq \text{last} \) because \( L \) is sorted
6. \hspace{2em} append \( y_i \) onto the end of \( L' \)
7. \hspace{2em} \( \text{last} = y_i \)
8. return \( L' \)

\[ \delta = 0.1 \]

\[ L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle \]

\[ L' = \langle 10, 12, 15, 20 \rangle \]
Illustration of the Trim Operation

\textbf{TRIM}(L, \delta)
\begin{algorithm}
1. let \( m \) be the length of \( L \)
2. \( L' = \langle y_1 \rangle \)
3. \( \text{last} = y_1 \)
4. \textbf{for} \( i = 2 \) \textbf{to} \( m \)
5. \quad \textbf{if} \( y_i > \text{last} \cdot (1 + \delta) \) \quad \textbf{//} \quad y_i \geq \text{last} \quad \text{because} \ L \quad \text{is sorted}
6. \quad \quad \text{append} \ y_i \quad \text{onto the end of} \ L'
7. \quad \quad \text{last} = y_i \\
8. \quad \textbf{return} \ L'
\end{algorithm}

\[ \delta = 0.1 \]

\[ L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle \]

\[ L' = \langle 10, 12, 15, 20 \rangle \]
Illustration of the Trim Operation

**Trim**($L, \delta$)
1. let $m$ be the length of $L$
2. $L' = \langle y_1 \rangle$
3. last = $y_1$
4. for $i = 2$ to $m$
   5. if $y_i > \text{last} \cdot (1 + \delta)$  // $y_i \geq \text{last}$ because $L$ is sorted
      append $y_i$ onto the end of $L'$
   6. last = $y_i$
8. return $L'$

\[ \delta = 0.1 \]

\[
L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle
\]

\[
L' = \langle 10, 12, 15, 20 \rangle
\]
Illustration of the Trim Operation

$\text{TRIM}(L, \delta)$

1. let $m$ be the length of $L$
2. $L' = \langle y_1 \rangle$
3. $\text{last} = y_1$
4. for $i = 2$ to $m$
5. \hspace{1em} if $y_i > \text{last} \cdot (1 + \delta)$ \hspace{1em} // $y_i \geq \text{last}$ because $L$ is sorted
6. \hspace{2em} append $y_i$ onto the end of $L'$
7. \hspace{2em} $\text{last} = y_i$
8. return $L'$

$\delta = 0.1$

$L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle$

$L' = \langle 10, 12, 15, 20 \rangle$
Illustration of the Trim Operation

\textbf{Trim}(L, \delta)

1. let \( m \) be the length of \( L \)
2. \( L' = \langle y_1 \rangle \)
3. \( \text{last} = y_1 \)
4. \textbf{for} \( i = 2 \) \textbf{to} \( m \)
5. \textbf{if} \( y_i > \text{last} \cdot (1 + \delta) \) \textbf{ \textit{\{}} \( y_i \geq \text{last} \) because \( L \) is sorted
6. \hspace{1em} \text{append} \( y_i \) onto the end of \( L' \)
7. \hspace{1em} \text{last} = y_i \\
8. \textbf{return} \( L' \)

\[ \delta = 0.1 \]

\( L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle \)

\( L' = \langle 10, 12, 15, 20, 23 \rangle \)
Illustration of the Trim Operation

**TRIM**\((L, \delta)\)

1. let \(m\) be the length of \(L\)
2. \(L' = \langle y_1 \rangle\)
3. \(last = y_1\)
4. **for** \(i = 2\) **to** \(m\)
5. **if** \(y_i > last \cdot (1 + \delta)\) // \(y_i \geq last\) because \(L\) is sorted
6. append \(y_i\) onto the end of \(L'\)
7. \(last = y_i\)
8. **return** \(L'\)

\(\delta = 0.1\)

\(L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle\)

\(L' = \langle 10, 12, 15, 20, 23 \rangle\)
Illustration of the Trim Operation

\textbf{TRIM}(L, \delta)

1. let \( m \) be the length of \( L \)
2. \( L' = \langle y_1 \rangle \)
3. \( \text{last} = y_1 \)
4. \textbf{for} \( i = 2 \) to \( m \)
5. \quad \textbf{if} \( y_i > \text{last} \cdot (1 + \delta) \) \hspace{1em} // \ y_i \geq \text{last} \text{ because } L \text{ is sorted}
6. \quad \text{append } y_i \text{ onto the end of } L'
7. \quad \text{last} = y_i
8. \textbf{return} \ L'

\( \delta = 0.1 \)

\( L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle \)

\( L' = \langle 10, 12, 15, 20, 23 \rangle \)
Illustration of the Trim Operation

**Trim**($L, \delta$)

1. Let $m$ be the length of $L$
2. $L' = \langle y_1 \rangle$
3. last = $y_1$
4. For $i = 2$ to $m$
   5. If $y_i > \text{last} \cdot (1 + \delta)$  
      // $y_i \geq \text{last}$ because $L$ is sorted
   6. Append $y_i$ onto the end of $L'$
   7. last = $y_i$
5. Return $L'$

\[ \delta = 0.1 \]

$L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle$

$L' = \langle 10, 12, 15, 20, 23 \rangle$
Illustration of the Trim Operation

\textbf{Trim}(L, \delta)

1. let \( m \) be the length of \( L \)
2. \( L' = \langle y_1 \rangle \)
3. \( \text{last} = y_1 \)
4. \textbf{for} \( i = 2 \) \textbf{to} \( m \)
5. \hspace{1em} \textbf{if} \( y_i > \text{last} \cdot (1 + \delta) \) \hspace{1em} \text{//} \hspace{1em} y_i \geq \text{last} \text{ because } L \text{ is sorted}
6. \hspace{2em} \text{append } y_i \text{ onto the end of } L'
7. \hspace{2em} \text{last} = y_i
8. \textbf{return} \( L' \)

\[ \delta = 0.1 \]

\[ L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle \]

\[ L' = \langle 10, 12, 15, 20, 23, 29 \rangle \]
Illustration of the Trim Operation

**TRIM**(\(L, \delta\))

1. let \(m\) be the length of \(L\)
2. \(L' = \{y_1\}\)
3. last = \(y_1\)
4. for \(i = 2\) to \(m\)
   5. if \(y_i > \text{last} \cdot (1 + \delta)\)  // \(y_i \geq \text{last}\) because \(L\) is sorted
      6. append \(y_i\) onto the end of \(L'\)
      7. last = \(y_i\)
5. return \(L'\)

\(\delta = 0.1\)

\(L = \langle10, 11, 12, 15, 20, 21, 22, 23, 24, 29\rangle\)

\(L' = \langle10, 12, 15, 20, 23, 29\rangle\)
Approximate Subset-Sum $(S, t, \epsilon)$

1. $n = |S|$
2. $L_0 = \langle 0 \rangle$
3. for $i = 1$ to $n$
4. \hspace{1em} $L_i = \text{MERGE-LISTS}(L_{i-1}, L_{i-1} + x_i)$
5. \hspace{1em} $L_i = \text{TRIM}(L_i, \epsilon/2n)$
6. \hspace{1em} remove from $L_i$ every element that is greater than $t$
7. let $z^*$ be the largest value in $L_n$
8. return $z^*$
**The FPTAS**

**APPROX-SUBSET-SUM**(S, t, ε)

1. \( n = |S| \)
2. \( L_0 = \{0\} \)
3. **for** \( i = 1 \) **to** \( n \)
4. \( L_i = \text{MERGE-LISTS}(L_{i-1}, L_{i-1} + x_i) \)
5. \( L_i = \text{TRIM}(L_i, \epsilon/2n) \)
6. **remove from** \( L_i \) **every element that is greater than** \( t \)
7. **let** \( z^* \) **be the largest value in** \( L_n \)
8. **return** \( z^* \)

**EXACT-SUBSET-SUM**(S, t)

1. \( n = |S| \)
2. \( L_0 = \{0\} \)
3. **for** \( i = 1 \) **to** \( n \)
4. \( L_i = \text{MERGE-LISTS}(L_{i-1}, L_{i-1} + x_i) \)
5. **remove from** \( L_i \) **every element that is greater than** \( t \)
6. **return** the largest element in \( L_n \)

\[ 35.5 \text{ The subset-sum problem 1129} \]
The FPTAS

\textbf{Approx-Subset-Sum}(S, t, \epsilon)

1. \( n = |S| \)
2. \( L_0 = \langle 0 \rangle \)
3. \textbf{for} \( i = 1 \) \textbf{to} \( n \)
4. \( L_i = \text{Merge-Lists}(L_{i-1}, L_{i-1} + x_i) \)
5. \( L_i = \text{Trim}(L_i, \epsilon/2n) \)
6. \text{remove from} \( L_i \) \text{every element that is greater than} \( t \)
7. \text{let} \( z^* \) \text{be the largest value in} \( L_n \)
8. \textbf{return} \( z^* \)

\textbf{Exact-Subset-Sum}(S, t)

1. \( n = |S| \)
2. \( L_0 = \langle 0 \rangle \)
3. \textbf{for} \( i = 1 \) \textbf{to} \( n \)
4. \( L_i = \text{Merge-Lists}(L_{i-1}, L_{i-1} + x_i) \)
5. \text{remove from} \( L_i \) \text{every element that is greater than} \( t \)
6. \textbf{return} \text{the largest element in} \( L_n \)

Repeated application of TRIM to make sure \( L_i \)'s remain short.
The FPTAS

**The Subset-Sum Problem**

**Approx-subset-sum** \((S, t, \epsilon)\)

1. \( n = |S| \)
2. \( L_0 = \langle 0 \rangle \)
3. **for** \( i = 1 \) **to** \( n \)
4. \( L_i = \text{merge-lists}(L_{i-1}, L_{i-1} + x_i) \)
5. \( L_i = \text{trim}(L_i, \epsilon/2n) \)
6. **remove** from \( L_i \) **every element** that is greater than \( t \)
7. **let** \( z^* \) **be the largest value** in \( L_n \)
8. **return** \( z^* \)

**Exact-subset-sum** \((S, t)\)

1. \( n = |S| \)
2. \( L_0 = \langle 0 \rangle \)
3. **for** \( i = 1 \) **to** \( n \)
4. \( L_i = \text{merge-lists}(L_{i-1}, L_{i-1} + x_i) \)
5. **remove** from \( L_i \) **every element** that is greater than \( t \)
6. **return** the largest element in \( L_n \)

Repeated application of **trim** to make sure \( L_i \)’s remain short.

- We must bound the inaccuracy introduced by repeated trimming.
We must bound the inaccuracy introduced by repeated trimming

- We must show that the algorithm is polynomial time
The FPTAS

**APPROX-SUBSET-SUM** \((S, t, \epsilon)\)

1. \(n = |S|\)
2. \(L_0 = \langle 0 \rangle\)
3. \(\text{for } i = 1 \text{ to } n\)
4. \(L_i = \text{MERGE-LISTS}(L_{i-1}, L_{i-1} + x_i)\)
5. \(L_i = \text{TRIM}(L_i, \epsilon/2n)\)
6. remove from \(L_i\) every element that is greater than \(t\)
7. let \(z^*\) be the largest value in \(L_n\)
8. return \(z^*\)

**EXACT-SUBSET-SUM** \((S, t)\)

1. \(n = |S|\)
2. \(L_0 = \langle 0 \rangle\)
3. \(\text{for } i = 1 \text{ to } n\)
4. \(L_i = \text{MERGE-LISTS}(L_{i-1}, L_{i-1} + x_i)\)
5. remove from \(L_i\) every element that is greater than \(t\)
6. return the largest element in \(L_n\)

Repeated application of \(\text{TRIM}\) to make sure \(L_i\)'s remain short.

- We must bound the inaccuracy introduced by repeated trimming
- We must show that the algorithm is polynomial time

Solution is a careful choice of \(\delta\)!
Running through an Example (CLRS3)

\textsc{Approx-Subset-Sum}(S, t, \epsilon)

1. \hspace{1em} \textbf{n} = |S|
2. \hspace{1em} L_0 = \langle 0 \rangle
3. \hspace{1em} \textbf{for } i = 1 \textbf{ to } n
4. \hspace{2em} L_i = \textsc{Merge-Lists}(L_{i-1}, L_{i-1} + x_i)
5. \hspace{2em} L_i = \textsc{Trim}(L_i, \epsilon/2n)
6. \hspace{2em} \text{remove from } L_i \text{ every element that is greater than } t
7. \hspace{1em} \textbf{let } z^* \textbf{ be the largest value in } L_n
8. \hspace{1em} \textbf{return } z^*
Running through an Example (CLRS3)

Algorithm APPROX-SUBSET-SUM($S, t, \epsilon$)
1. $n = |S|$
2. $L_0 = \langle 0 \rangle$
3. for $i = 1$ to $n$
   4. $L_i = \text{MERGE-LISTS}(L_{i-1}, L_{i-1} + x_i)$
   5. $L_i = \text{TRIM}(L_i, \epsilon/2n)$
6. remove from $L_i$ every element that is greater than $t$
7. let $z^*$ be the largest value in $L_n$
8. return $z^*$

- Input: $S = \langle 104, 102, 201, 101 \rangle$, $t = 308$, $\epsilon = 0.4$
Running through an Example (CLRS3)

APPROX-SUBSET-SUM(S, t, ε)
1  n = |S|
2  L₀ = ⟨0⟩
3  for i = 1 to n
4      Li = MERGE-LISTS(Li-1, Li-1 + xᵢ)
5  Li = TRIM(Li, ε/2n)
6      remove from Li every element that is greater than t
7  let z* be the largest value in Lₙ
8  return z*

- Input: S = ⟨104, 102, 201, 101⟩, t = 308, ε = 0.4
⇒ Trimming parameter: δ = ε/(2 · n) = ε/8 = 0.05
Running through an Example (CLRS3)

**APPROX-SUBSET-SUM(S, t, ε)**

1. \( n = |S| \)
2. \( L_0 = \langle 0 \rangle \)
3. **for** \( i = 1 \) **to** \( n \)
4. \( L_i = \text{MERGE-LISTS}(L_{i-1}, L_{i-1} + x_i) \)
5. \( L_i = \text{TRIM}(L_i, \epsilon/2n) \)
6. remove from \( L_i \) every element that is greater than \( t \)
7. let \( z^* \) be the largest value in \( L_n \)
8. **return** \( z^* \)

- **Input:** \( S = \langle 104, 102, 201, 101 \rangle \), \( t = 308 \), \( \epsilon = 0.4 \)
- **⇒ Trimming parameter:** \( \delta = \epsilon/(2 \cdot n) = \epsilon/8 = 0.05 \)
- **line 2:** \( L_0 = \langle 0 \rangle \)
Running through an Example (CLRS3)

**APPROX-SUBSET-SUM**(*S*, *t*, *ε*)

1. \( n = |S| \)
2. \( L_0 = \langle 0 \rangle \)
3. **for** \( i = 1 \) **to** \( n \)
4. \( L_i = \text{MERGE-LISTS}(L_{i-1}, L_{i-1} + x_i) \)
5. \( L_i = \text{TRIM}(L_i, \epsilon/2n) \)
6. remove from \( L_i \) every element that is greater than \( t \)
7. let \( z^* \) be the largest value in \( L_n \)
8. **return** \( z^* \)

- **Input:** \( S = \langle 104, 102, 201, 101 \rangle \), \( t = 308 \), \( \epsilon = 0.4 \)
- **Trimming parameter:** \( \delta = \epsilon/(2 \cdot n) = \epsilon/8 = 0.05 \)
  - line 2: \( L_0 = \langle 0 \rangle \)
  - line 4: \( L_1 = \langle 0, 104 \rangle \)
Running through an Example (CLRS3)

\textbf{APPROX-SUBSET-SUM}(S, t, \epsilon)

1. \hspace{1em} \texttt{n = |S|}
2. \hspace{1em} \texttt{L_0 = \langle 0 \rangle}
3. \hspace{1em} \textbf{for} \hspace{0.5em} i = 1 \hspace{0.5em} \textbf{to} \hspace{0.5em} n
4. \hspace{2.1em} \texttt{L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)}
5. \hspace{2.1em} \texttt{L_i = TRIM(L_i, \epsilon/2n)}
6. \hspace{1em} \texttt{remove from L_i every element that is greater than t}
7. \hspace{1em} \texttt{let z^* be the largest value in L_n}
8. \hspace{1em} \textbf{return} z^*

\hspace{1em} \\
\hspace{2.1em} \textbf{Input:} \hspace{1em} S = \langle 104, 102, 201, 101 \rangle, t = 308, \hspace{0.5em} \epsilon = 0.4

\hspace{2.1em} \Rightarrow \hspace{1em} \textbf{Trimming parameter:} \hspace{1em} \delta = \epsilon/(2 \cdot n) = \epsilon/8 = 0.05

\hspace{2.1em} \\
\hspace{2.1em} \hspace{1em} \textbf{line 2:} \hspace{1em} \texttt{L_0 = \langle 0 \rangle}
\hspace{2.1em} \\
\hspace{2.1em} \hspace{1em} \textbf{line 4:} \hspace{1em} \texttt{L_1 = \langle 0, 104 \rangle}
\hspace{2.1em} \\
\hspace{2.1em} \hspace{1em} \textbf{line 5:} \hspace{1em} \texttt{L_1 = \langle 0, 104 \rangle}
Running through an Example (CLRS3)

\[ \text{APPROX-SUBSET-SUM}(S, t, \epsilon) \]

1. \( n = |S| \)
2. \( L_0 = \langle 0 \rangle \)
3. \( \text{for } i = 1 \text{ to } n \)
4. \( L_i = \text{MERGE-LISTS}(L_{i-1}, L_{i-1} + x_i) \)
5. \( L_i = \text{TRIM}(L_i, \epsilon/2n) \)
6. \( \text{remove from } L_i \text{ every element that is greater than } t \)
7. \( \text{let } z^* \text{ be the largest value in } L_n \)
8. \( \text{return } z^* \)

- **Input:** \( S = \langle 104, 102, 201, 101 \rangle, t = 308, \epsilon = 0.4 \)
- **Trimming parameter:** \( \delta = \epsilon/(2 \cdot n) = \epsilon/8 = 0.05 \)
- **Line 2:** \( L_0 = \langle 0 \rangle \)
- **Line 4:** \( L_1 = \langle 0, 104 \rangle \)
- **Line 5:** \( L_1 = \langle 0, 104 \rangle \)
- **Line 6:** \( L_1 = \langle 0, 104 \rangle \)
Running through an Example (CLRS3)

Approx-Subset-Sum\((S, t, \epsilon)\)

1. \(n = |S|\)
2. \(L_0 = \langle 0 \rangle\)
3. for \(i = 1\) to \(n\) do
4. \(L_i = \text{MERGE-LISTS}(L_{i-1}, L_{i-1} + x_i)\)
5. \(L_i = \text{TRIM}(L_i, \epsilon/2n)\)
6. remove from \(L_i\) every element that is greater than \(t\)
7. let \(z^*\) be the largest value in \(L_n\)
8. return \(z^*\)

- Input: \(S = \langle 104, 102, 201, 101 \rangle\), \(t = 308\), \(\epsilon = 0.4\)

⇒ Trimming parameter: \(\delta = \epsilon/(2 \cdot n) = \epsilon/8 = 0.05\)

- line 2: \(L_0 = \langle 0 \rangle\)
- line 4: \(L_1 = \langle 0, 104 \rangle\)
- line 5: \(L_1 = \langle 0, 104 \rangle\)
- line 6: \(L_1 = \langle 0, 104 \rangle\)
- line 4: \(L_2 = \langle 0, 102, 104, 206 \rangle\)
Running through an Example (CLRS3)

**APPROX-SUBSET-SUM** \((S, t, \epsilon)\)

1. \(n = |S|\)
2. \(L_0 = \{0\}\)
3. \(\text{for } i = 1 \to n\)
4. \(L_i = \text{MERGE-LISTS}(L_{i-1}, L_{i-1} + x_i)\)
5. \(L_i = \text{TRIM}(L_i, \epsilon/2n)\)
6. \(\text{remove from } L_i \text{ every element that is greater than } t\)
7. let \(z^*\) be the largest value in \(L_n\)
8. \(\text{return } z^*\)

- **Input:** \(S = \langle 104, 102, 201, 101 \rangle, t = 308, \epsilon = 0.4\)
- **Trimming parameter:** \(\delta = \epsilon/(2 \cdot n) = \epsilon/8 = 0.05\)

- line 2: \(L_0 = \{0\}\)
- line 4: \(L_1 = \{0, 104\}\)
- line 5: \(L_1 = \{0, 104\}\)
- line 6: \(L_1 = \{0, 104\}\)
- line 4: \(L_2 = \{0, 102, 104, 206\}\)
- line 5: \(L_2 = \{0, 102, 206\}\)
Running through an Example (CLRS3)

\textbf{APPROX-SUBSET-SUM}(S, t, \epsilon)

1. \quad n = |S|
2. \quad L_0 = \langle 0 \rangle
3. \quad \text{for } i = 1 \text{ to } n
4. \quad \quad L_i = \text{MERGE-LISTS}(L_{i-1}, L_{i-1} + x_i)
5. \quad \quad L_i = \text{TRIM}(L_i, \epsilon/2n)
6. \quad \quad \text{remove from } L_i \text{ every element that is greater than } t
7. \quad \text{let } z^* \text{ be the largest value in } L_n
8. \quad \text{return } z^*

- **Input:** \( S = \langle 104, 102, 201, 101 \rangle, \ t = 308, \ \epsilon = 0.4 \)

\Rightarrow \textbf{Trimming parameter: } \delta = \epsilon/(2 \cdot n) = \epsilon/8 = 0.05

- line 2: \( L_0 = \langle 0 \rangle \)
- line 4: \( L_1 = \langle 0, 104 \rangle \)
- line 5: \( L_1 = \langle 0, 104 \rangle \)
- line 6: \( L_1 = \langle 0, 104 \rangle \)
- line 4: \( L_2 = \langle 0, 102, 104, 206 \rangle \)
- line 5: \( L_2 = \langle 0, 102, 206 \rangle \)
- line 6: \( L_2 = \langle 0, 102, 206 \rangle \)
Running through an Example (CLRS3)

\textsc{Approx-Subset-Sum}(S, t, \epsilon)
1. \quad n = |S|
2. \quad L_0 = \langle 0 \rangle
3. \quad \textbf{for} i = 1 \textbf{ to } n
4. \quad \quad L_i = \textsc{Merge-Lists}(L_{i-1}, L_{i-1} + x_i)
5. \quad \quad L_i = \textsc{Trim}(L_i, \epsilon/2n)
6. \quad \quad \text{remove from } L_i \text{ every element that is greater than } t
7. \quad \textbf{let} z^* \textbf{ be the largest value in } L_n
8. \quad \textbf{return} z^*

- Input: \( S = \langle 104, 102, 201, 101 \rangle, t = 308, \epsilon = 0.4 \)
- Trimming parameter: \( \delta = \epsilon/(2 \cdot n) = \epsilon/8 = 0.05 \)

- line 2: \( L_0 = \langle 0 \rangle \)
- line 4: \( L_1 = \langle 0, 104 \rangle \)
- line 5: \( L_1 = \langle 0, 104 \rangle \)
- line 6: \( L_1 = \langle 0, 104 \rangle \)
- line 4: \( L_2 = \langle 0, 102, 104, 206 \rangle \)
- line 5: \( L_2 = \langle 0, 102, 206 \rangle \)
- line 6: \( L_2 = \langle 0, 102, 206 \rangle \)
- line 4: \( L_3 = \langle 0, 102, 201, 206, 303, 407 \rangle \)
Running through an Example (CLRS3)

**APPROX-SUBSET-SUM(S, t, \( \epsilon \))**

1. \( n = |S| \)
2. \( L_0 = \langle 0 \rangle \)
3. for \( i = 1 \) to \( n \)
   4. \( L_i = \text{MERGE-LISTS}(L_{i-1}, L_{i-1} + x_i) \)
   5. \( L_i = \text{TRIM}(L_i, \epsilon/2n) \)
   6. remove from \( L_i \) every element that is greater than \( t \)
7. let \( z^* \) be the largest value in \( L_n \)
8. return \( z^* \)

- **Input:** \( S = \langle 104, 102, 201, 101 \rangle, t = 308, \epsilon = 0.4 \)
- **Trimming parameter:** \( \delta = \epsilon/(2 \cdot n) = \epsilon/8 = 0.05 \)

- line 2: \( L_0 = \langle 0 \rangle \)
- line 4: \( L_1 = \langle 0, 104 \rangle \)
- line 5: \( L_1 = \langle 0, 104 \rangle \)
- line 6: \( L_1 = \langle 0, 104 \rangle \)
- line 4: \( L_2 = \langle 0, 102, 104, 206 \rangle \)
- line 5: \( L_2 = \langle 0, 102, 206 \rangle \)
- line 6: \( L_2 = \langle 0, 102, 206 \rangle \)
- line 4: \( L_3 = \langle 0, 102, 201, 206, 303, 407 \rangle \)
- line 5: \( L_3 = \langle 0, 102, 201, 303, 407 \rangle \)
Running through an Example (CLRS3)

\textsc{approx-subset-sum}(S, t, \epsilon) \\
1 \ n = |S| \\
2 \ L_0 = \langle 0 \rangle \\
3 \ \textbf{for} \ i = 1 \ \textbf{to} \ n \\
4 \quad L_i = \textsc{merge-lists}(L_{i-1}, L_{i-1} + x_i) \\
5 \quad L_i = \textsc{trim}(L_i, \epsilon / 2n) \\
6 \ \quad \text{remove from } L_i \ \text{every element that is greater than } t \\
7 \ \textbf{let} \ z^* \ \textbf{be the largest value in } L_n \\
8 \ \textbf{return} \ z^* \\

- \textbf{Input:} \ S = \langle 104, 102, 201, 101 \rangle, \ t = 308, \ \epsilon = 0.4 \\
⇒ \textbf{Trimming parameter: } \delta = \epsilon / (2 \cdot n) = \epsilon / 8 = 0.05 \\
  - \textbf{line 2:} \ L_0 = \langle 0 \rangle \\
  - \textbf{line 4:} \ L_1 = \langle 0, 104 \rangle \\
  - \textbf{line 5:} \ L_1 = \langle 0, 104 \rangle \\
  - \textbf{line 6:} \ L_1 = \langle 0, 104 \rangle \\
  - \textbf{line 4:} \ L_2 = \langle 0, 102, 104, 206 \rangle \\
  - \textbf{line 5:} \ L_2 = \langle 0, 102, 206 \rangle \\
  - \textbf{line 6:} \ L_2 = \langle 0, 102, 206 \rangle \\
  - \textbf{line 4:} \ L_3 = \langle 0, 102, 201, 206, 303, 407 \rangle \\
  - \textbf{line 5:} \ L_3 = \langle 0, 102, 201, 303, 407 \rangle \\
  - \textbf{line 6:} \ L_3 = \langle 0, 102, 201, 303 \rangle
Running through an Example (CLRS3)

**APPROX-SUBSET-SUM** \((S, t, \epsilon)\)

1. \(n = |S|\)
2. \(L_0 = \langle 0 \rangle\)
3. **for** \(i = 1\) **to** \(n\)
4. \(L_i = \text{MERGE-LISTS}(L_{i-1}, L_{i-1} + x_i)\)
5. \(L_i = \text{TRIM}(L_i, \epsilon/2n)\)
6. **remove from** \(L_i\) **every element** that **is greater** than \(t\)
7. **let** \(z^*\) **be the largest** value in \(L_n\)
8. **return** \(z^*\)

- **Input:** \(S = \langle 104, 102, 201, 101 \rangle, t = 308, \epsilon = 0.4\)
- **⇒ Trimming parameter:** \(\delta = \epsilon/(2 \cdot n) = \epsilon/8 = 0.05\)

- **line 2:** \(L_0 = \langle 0 \rangle\)
- **line 4:** \(L_1 = \langle 0, 104 \rangle\)
- **line 5:** \(L_1 = \langle 0, 104 \rangle\)
- **line 6:** \(L_1 = \langle 0, 104 \rangle\)
- **line 4:** \(L_2 = \langle 0, 102, 104, 206 \rangle\)
- **line 5:** \(L_2 = \langle 0, 102, 206 \rangle\)
- **line 6:** \(L_2 = \langle 0, 102, 206 \rangle\)
- **line 4:** \(L_3 = \langle 0, 102, 201, 206, 303, 407 \rangle\)
- **line 5:** \(L_3 = \langle 0, 102, 201, 303, 407 \rangle\)
- **line 6:** \(L_3 = \langle 0, 102, 201, 303 \rangle\)
- **line 4:** \(L_4 = \langle 0, 101, 102, 201, 203, 302, 303, 404 \rangle\)
Running through an Example (CLRS3)

\textbf{APPROX-SUBSET-SUM}(S, t, \epsilon)

1. \( n = |S| \)
2. \( L_0 = \langle 0 \rangle \)
3. \textbf{for} \( i = 1 \) \textbf{to} \( n \)
4. \( L_i = \text{MERGE-LISTS}(L_{i-1}, L_{i-1} + x_i) \)
5. \( L_i = \text{TRIM}(L_i, \epsilon/2n) \)
6. \textbf{remove from} \( L_i \) \textbf{every element} \textbf{that is greater} \textbf{than} \( t \)
7. \textbf{let} \( z^* \) \textbf{be the largest value} \textbf{in} \( L_n \)
8. \textbf{return} \( z^* \)

- **Input**: \( S = \langle 104, 102, 201, 101 \rangle \), \( t = 308 \), \( \epsilon = 0.4 \)

\Rightarrow \textbf{Trimming parameter}: \( \delta = \epsilon/(2 \cdot n) = \epsilon/8 = 0.05 \)

- line 2: \( L_0 = \langle 0 \rangle \)
- line 4: \( L_1 = \langle 0, 104 \rangle \)
- line 5: \( L_1 = \langle 0, 104 \rangle \)
- line 6: \( L_1 = \langle 0, 104 \rangle \)
- line 4: \( L_2 = \langle 0, 102, 104, 206 \rangle \)
- line 5: \( L_2 = \langle 0, 102, 206 \rangle \)
- line 6: \( L_2 = \langle 0, 102, 206 \rangle \)
- line 4: \( L_3 = \langle 0, 102, 201, 206, 303, 407 \rangle \)
- line 5: \( L_3 = \langle 0, 102, 201, 303, 407 \rangle \)
- line 6: \( L_3 = \langle 0, 102, 201, 303 \rangle \)
- line 4: \( L_4 = \langle 0, 101, 102, 201, 203, 302, 303, 404 \rangle \)
- line 5: \( L_4 = \langle 0, 101, 201, 302, 404 \rangle \)
Running through an Example (CLRS3)

\[ \text{APPROX-SUBSET-SUM}(S, t, \epsilon) \]

1. \( n = |S| \)
2. \( L_0 = \langle 0 \rangle \)
3. \text{for } i = 1 \text{ to } n
4. \hspace{1em} L_i = \text{MERGE-LISTS}(L_{i-1}, L_{i-1} + x_i) \\
5. \hspace{1em} L_i = \text{TRIM}(L_i, \epsilon/2n) \\
6. \hspace{1em} \text{remove from } L_i \text{ every element that is greater than } t \\
7. \hspace{1em} \text{let } z^* \text{ be the largest value in } L_n \\
8. \hspace{1em} \text{return } z^* \\

- \textbf{Input: } S = \langle 104, 102, 201, 101 \rangle, t = 308, \epsilon = 0.4 \\

\[ \Rightarrow \text{Trimming parameter: } \delta = \epsilon/(2 \cdot n) = \epsilon/8 = 0.05 \]

- line 2: \( L_0 = \langle 0 \rangle \)
- line 4: \( L_1 = \langle 0, 104 \rangle \)
- line 5: \( L_1 = \langle 0, 104 \rangle \)
- line 6: \( L_1 = \langle 0, 104 \rangle \)
- line 4: \( L_2 = \langle 0, 102, 104, 206 \rangle \)
- line 5: \( L_2 = \langle 0, 102, 206 \rangle \)
- line 6: \( L_2 = \langle 0, 102, 206 \rangle \)
- line 4: \( L_3 = \langle 0, 102, 201, 206, 303, 407 \rangle \)
- line 5: \( L_3 = \langle 0, 102, 201, 303, 407 \rangle \)
- line 6: \( L_3 = \langle 0, 102, 201, 303 \rangle \)
- line 4: \( L_4 = \langle 0, 101, 102, 201, 203, 302, 303, 404 \rangle \)
- line 5: \( L_4 = \langle 0, 101, 201, 302, 404 \rangle \)
- line 6: \( L_4 = \langle 0, 101, 201, 302 \rangle \)
Running through an Example (CLRS3)

\textbf{APPROX-SUBSET-SUM}(S, t, \epsilon)

\begin{enumerate}
\item \( n = |S| \)
\item \( L_0 = \langle 0 \rangle \)
\item \textbf{for} \( i = 1 \) \textbf{to} \( n \)
\item \( L_i = \text{MERGE-LISTS}(L_{i-1}, L_{i-1} + x_i) \)
\item \( L_i = \text{TRIM}(L_i, \epsilon/2n) \)
\item \text{remove from} \( L_i \) \text{every element that is greater than} \( t \)
\item \text{let} \( z^* \) \text{be the largest value in} \( L_n \)
\item \textbf{return} \( z^* \)
\end{enumerate}

- \textbf{Input:} \( S = \langle 104, 102, 201, 101 \rangle \), \( t = 308 \), \( \epsilon = 0.4 \)

\implies \textbf{Trimming parameter:} \( \delta = \epsilon/(2 \cdot n) = \epsilon/8 = 0.05 \)

- line 2: \( L_0 = \langle 0 \rangle \)
- line 4: \( L_1 = \langle 0, 104 \rangle \)
- line 5: \( L_1 = \langle 0, 104 \rangle \)
- line 6: \( L_1 = \langle 0, 104 \rangle \)
- line 4: \( L_2 = \langle 0, 102, 104, 206 \rangle \)
- line 5: \( L_2 = \langle 0, 102, 206 \rangle \)
- line 6: \( L_2 = \langle 0, 102, 206 \rangle \)
- line 4: \( L_3 = \langle 0, 102, 201, 206, 303, 407 \rangle \)
- line 5: \( L_3 = \langle 0, 102, 201, 303, 407 \rangle \)
- line 6: \( L_3 = \langle 0, 102, 201, 303 \rangle \)
- line 4: \( L_4 = \langle 0, 101, 102, 201, 203, 302, 303, 404 \rangle \)
- line 5: \( L_4 = \langle 0, 101, 201, 302, 404 \rangle \)
- line 6: \( L_4 = \langle 0, 101, 201, 302 \rangle \)

\begin{center}
\textbf{Returned solution} \( z^* = 302 \), which is 2% within the optimum \( 307 = 104 + 102 + 101 \)
\end{center}
Reminder: Performance Ratios for Approximation Algorithms

Approximation Ratio

An algorithm for a problem has approximation ratio \( \rho(n) \), if for any input of size \( n \), the cost \( C \) of the returned solution and optimal cost \( C^* \) satisfy:

\[
\max \left( \frac{C}{C^*}, \frac{C^*}{C} \right) \leq \rho(n).
\]

For many problems: tradeoff between runtime and approximation ratio.

Approximation Schemes

An approximation scheme is an approximation algorithm, which given any input and \( \epsilon > 0 \), is a \((1 + \epsilon)\)-approximation algorithm.

- It is a polynomial-time approximation scheme (PTAS) if for any fixed \( \epsilon > 0 \), the runtime is polynomial in \( n \). For example, \( O(n^2/\epsilon) \).
- It is a fully polynomial-time approximation scheme (FPTAS) if the runtime is polynomial in both \( 1/\epsilon \) and \( n \). For example, \( O((1/\epsilon)^2 \cdot n^3) \).
Analysis of APPROX-SUBSET-SUM

Theorem 35.8

APPROX-SUBSET-SUM is a FPTAS for the subset-sum problem.

Proof (Approximation Ratio):

\[
\text{can be shown by induction on } i\text{.}
\]
Analysis of **APPROX-SUBSET-SUM**

**Theorem 35.8**

**APPROX-SUBSET-SUM** is a **FPTAS** for the subset-sum problem.

Proof (Approximation Ratio):
- Returned solution \( z^* \) is a valid solution \( \checkmark \)
Analysis of APPROX-SUBSET-SUM

Theorem 35.8

APPROX-SUBSET-SUM is a FPTAS for the subset-sum problem.

Proof (Approximation Ratio):

- Returned solution $z^*$ is a valid solution ✓
- Let $y^*$ denote an optimal solution

IV. Approximation via Exact Algorithms
The Subset-Sum Problem
Analysis of **APPROX-SUBSET-SUM**

**Theorem 35.8**

**APPROX-SUBSET-SUM** is a **FPTAS** for the subset-sum problem.

**Proof (Approximation Ratio):**

- Returned solution \( z^\ast \) is a valid solution ✓
- Let \( y^\ast \) denote an optimal solution
- For every possible sum \( y \leq t \) of \( x_1, \ldots, x_i \), there exists an element \( z \in L'_i \) s.t.: 

\[
y \left( 1 + \frac{\epsilon}{2n} \right)^n \leq z \leq y^\ast \left( 1 + \frac{\epsilon}{2} + \left( \frac{\epsilon}{2} \right)^2 \right)
\]
Analysis of \textsc{Approx-Subset-Sum}

\textbf{Theorem 35.8}
\textsc{Approx-Subset-Sum} is a \textbf{FPTAS} for the subset-sum problem.

\textbf{Proof (Approximation Ratio)}:
- Returned solution $z^*$ is a valid solution $\checkmark$
- Let $y^*$ denote an optimal solution
- For every possible sum $y \leq t$ of $x_1, \ldots, x_i$, there exists an element $z \in L'_i$ s.t.:

$$\frac{y}{(1 + \epsilon/(2n))^i} \leq z \leq y$$
Analysis of \textsc{Approx-Subset-Sum}

\textbf{Theorem 35.8}

\textsc{Approx-Subset-Sum} is a \textbf{FPTAS} for the subset-sum problem.

\textbf{Proof (Approximation Ratio)}:

- Returned solution $z^*$ is a valid solution $\checkmark$
- Let $y^*$ denote an optimal solution
- For every possible sum $y \leq t$ of $x_1, \ldots, x_i$, there exists an element $z \in L'_i$ s.t.:

\begin{align*}
\frac{y}{(1 + \epsilon/(2n))^i} \leq z & \leq y
\end{align*}

Can be shown by induction on $i$
Analysis of APPROX-SUBSET-SUM

Theorem 35.8

APPROX-SUBSET-SUM is a FPTAS for the subset-sum problem.

Proof (Approximation Ratio):

- Returned solution $z^*$ is a valid solution ✓
- Let $y^*$ denote an optimal solution
- For every possible sum $y \leq t$ of $x_1, \ldots, x_i$, there exists an element $z \in L'_i$ s.t.:

$$\frac{y}{(1 + \epsilon/(2n))^i} \leq z \leq y \quad y=y^*, i=n$$

Can be shown by induction on $i$
Analysis of APPROX-SUBSET-SUM

Theorem 35.8

APPROX-SUBSET-SUM is a FPTAS for the subset-sum problem.

Proof (Approximation Ratio):

- Returned solution $z^*$ is a valid solution $✓$
- Let $y^*$ denote an optimal solution
- For every possible sum $y \leq t$ of $x_1, \ldots, x_i$, there exists an element $z \in L'_i$ s.t.:

$$\frac{y}{(1 + \epsilon/(2n))^i} \leq z \leq y \quad \Rightarrow \quad \frac{y^*}{(1 + \epsilon/(2n))^n} \leq z \leq y^*$$

Can be shown by induction on $i$
Analysis of APPROX-SUBSET-SUM

Theorem 35.8

APPROX-SUBSET-SUM is a FPTAS for the subset-sum problem.

Proof (Approximation Ratio):

- Returned solution $z^*$ is a valid solution $✓$
- Let $y^*$ denote an optimal solution
- For every possible sum $y \leq t$ of $x_1, \ldots, x_i$, there exists an element $z \in L_i'$ s.t.:

$$
\frac{y}{(1 + \epsilon/(2n))^i} \leq z \leq y \quad \text{for } y = y^*, \quad i = n
$$

$$
\frac{y^*}{(1 + \epsilon/(2n))^n} \leq z \leq y^*
$$

Can be shown by induction on $i$

$$
\frac{y^*}{z} \leq \left(1 + \frac{\epsilon}{2n}\right)^n,
$$
Theorem 35.8

**APPROX-SUBSET-SUM** is a FPTAS for the subset-sum problem.

**Proof (Approximation Ratio):**

- Returned solution $z^*$ is a valid solution $✓$
- Let $y^*$ denote an optimal solution
- For every possible sum $y \leq t$ of $x_1, \ldots, x_i$, there exists an element $z \in L'_i$ s.t.:

$$\frac{y}{(1 + \epsilon/(2n))^i} \leq z \leq y \quad y=y^*, i=n \quad \frac{y^*}{(1 + \epsilon/(2n))^n} \leq z \leq y^*$$

- Can be shown by induction on $i$

and now using the fact that $\left(1 + \frac{\epsilon/2}{n}\right)^n \xrightarrow{n \to \infty} e^{\epsilon/2}$ yields
Analysis of APPROX-SUBSET-SUM

Theorem 35.8
APPROX-SUBSET-SUM is a FPTAS for the subset-sum problem.

Proof (Approximation Ratio):
- Returned solution $z^*$ is a valid solution $✓$
- Let $y^*$ denote an optimal solution
- For every possible sum $y \leq t$ of $x_1, \ldots, x_i$, there exists an element $z \in L_i'$ s.t.:

\[
\frac{y}{(1 + \epsilon/(2n))^i} \leq z \leq y \quad \Rightarrow \quad y=y^*, i=n \quad \frac{y^*}{(1 + \epsilon/(2n))^n} \leq z \leq y^*
\]

Can be shown by induction on $i$

and now using the fact that \( \left(1 + \frac{\epsilon/2}{n}\right)^n \xrightarrow{n \to \infty} e^{\epsilon/2} \) yields

\[
\frac{y^*}{z} \leq e^{\epsilon/2}
\]
Analysis of \textsc{Approx-Subset-Sum} 

\textbf{Theorem 35.8}

\textsc{Approx-Subset-Sum} is a \textsc{FPTAS} for the subset-sum problem.

Proof (Approximation Ratio):

- Returned solution $z^*$ is a valid solution $\checkmark$
- Let $y^*$ denote an optimal solution
- For every possible sum $y \leq t$ of $x_1, \ldots, x_i$, there exists an element $z \in L'_i$ s.t.:

\[
\frac{y}{(1 + \frac{\epsilon}{(2n)})^i} \leq z \leq y \quad \Rightarrow y = y^*, i = n \quad \frac{y^*}{(1 + \frac{\epsilon}{(2n)})^n} \leq z \leq y^*
\]

Can be shown by induction on $i$

and now using the fact that $\left(1 + \frac{\epsilon/2}{n}\right)^n \xrightarrow{n \to \infty} e^{\epsilon/2}$ yields

\[
\frac{y^*}{z} \leq e^{\epsilon/2} \quad \text{Taylor approximation of } e
\]
Analysis of APPROX-SUBSET-SUM

**Theorem 35.8**

APPROX-SUBSET-SUM is a FPTAS for the subset-sum problem.

**Proof (Approximation Ratio):**

- Returned solution \( z^* \) is a valid solution \( \checkmark \)
- Let \( y^* \) denote an optimal solution
- For every possible sum \( y \leq t \) of \( x_1, \ldots, x_i \), there exists an element \( z \in L' \) s.t.:

\[
\frac{y}{(1 + \epsilon/(2n))^i} \leq z \leq y \quad y=y^*, i=n
\]

\[
\frac{y^*}{(1 + \epsilon/(2n))^n} \leq z \leq y^*
\]

Can be shown by induction on \( i \)

and now using the fact that \( (1 + \frac{\epsilon}{2n})^n \xrightarrow{n \to \infty} e^{\epsilon/2} \) yields

\[
\frac{y^*}{z} \leq e^{\epsilon/2}\quad \text{Taylor approximation of } e
\]

\[
\leq 1 + \frac{\epsilon}{2} + \left(\frac{\epsilon}{2}\right)^2
\]
Analysis of **APPROX-SUBSET-SUM**

**Theorem 35.8**

**APPROX-SUBSET-SUM** is a FPTAS for the subset-sum problem.

**Proof (Approximation Ratio):**

- Returned solution $z^*$ is a valid solution $\checkmark$
- Let $y^*$ denote an optimal solution
- For every possible sum $y \leq t$ of $x_1, \ldots, x_i$, there exists an element $z \in L_i$ s.t.:

$$\frac{y}{(1 + \epsilon/(2n))^i} \leq z \leq y \quad \Rightarrow \quad \frac{y^*}{(1 + \epsilon/(2n))^n} \leq z \leq y^*$$

Can be shown by induction on $i$

and now using the fact that $\left(1 + \frac{\epsilon/2}{n}\right)^n \xrightarrow{n \rightarrow \infty} e^{\epsilon/2}$ yields

$$\frac{y^*}{z} \leq e^{\epsilon/2} \quad \text{Taylor approximation of } e$$

$$\leq 1 + \epsilon/2 + (\epsilon/2)^2 \leq 1 + \epsilon$$
Analysis of **APPROX-SUBSET-SUM**

**Theorem 35.8**

**APPROX-SUBSET-SUM** is a **FPTAS** for the subset-sum problem.

Proof (Running Time):
Analysis of APPROX-SUBSET-SUM

Theorem 35.8

APPROX-SUBSET-SUM is a FPTAS for the subset-sum problem.

Proof (Running Time):
- **Strategy:** Derive a bound on $|L_i|$ (running time is linear in $|L_i|$)

After trimming, two successive elements $z$ and $z'$ satisfy $z'/z \geq 1 + \epsilon / (2^n)$.

Hence, $\log(1 + \epsilon / (2^n))t + 2 \leq 3n \log t$.
Analysis of APPROX-SUBSET-SUM

Theorem 35.8

APPROX-SUBSET-SUM is a FPTAS for the subset-sum problem.

Proof (Running Time):

- **Strategy**: Derive a bound on $|L_i|$ (running time is linear in $|L_i|$)
- After trimming, two successive elements $z$ and $z'$ satisfy $z'/z \geq 1 + \epsilon/(2n)$
Analysis of APPROX-SUBSET-SUM

**Theorem 35.8**

APPROX-SUBSET-SUM is a FPTAS for the subset-sum problem.

Proof (Running Time):

- **Strategy**: Derive a bound on $|L_i|$ (running time is linear in $|L_i|$)
- After trimming, two successive elements $z$ and $z'$ satisfy $z'/z \geq 1 + \epsilon/(2n)$

$\Rightarrow$ Possible Values after trimming are 0, 1, and up to $\lfloor \log_{1+\epsilon/(2n)} t \rfloor$ additional values.
Theorem 35.8

**APPROX-SUBSET-SUM** is a **FPTAS** for the subset-sum problem.

Proof (Running Time):
- **Strategy**: Derive a bound on $|L_i|$ (running time is linear in $|L_i|$)
- After trimming, two successive elements $z$ and $z'$ satisfy $z'/z \geq 1 + \epsilon/(2n)$

$\Rightarrow$ Possible Values after trimming are 0, 1, and up to $\lfloor \log_{1+\epsilon/(2n)} t \rfloor$ additional values.

Hence,

$$\log_{1+\epsilon/(2n)} t + 2 =$$
**Analysis of APPROX-SUBSET-SUM**

**Theorem 35.8**

**APPROX-SUBSET-SUM** is a FPTAS for the subset-sum problem.

Proof (Running Time):
- **Strategy:** Derive a bound on $|L_i|$ (running time is linear in $|L_i|$)
- After trimming, two successive elements $z$ and $z'$ satisfy $z'/z \geq 1 + \epsilon/(2n)$

$\Rightarrow$ Possible values after trimming are 0, 1, and up to $\lfloor \log_{1+\epsilon/(2n)} t \rfloor$ additional values.

Hence,

$$\log_{1+\epsilon/(2n)} t + 2 = \frac{\ln t}{\ln(1 + \epsilon/(2n))} + 2$$
Analysis of APPROX-SUBSET-SUM

Theorem 35.8

APPROX-SUBSET-SUM is a FPTAS for the subset-sum problem.

Proof (Running Time):

- **Strategy:** Derive a bound on $|L_i|$ (running time is linear in $|L_i|$)
- After trimming, two successive elements $z$ and $z'$ satisfy $z'/z \geq 1 + \epsilon/(2n)$

$\Rightarrow$ Possible Values after trimming are 0, 1, and up to $\lfloor \log_{1+\epsilon/(2n)} t \rfloor$ additional values.

Hence,

$$\log_{1+\epsilon/(2n)} t + 2 = \frac{\ln t}{\ln(1 + \epsilon/(2n))} + 2$$

For $x > -1$, $\ln(1 + x) \geq \frac{x}{1+x}$
Analysis of \textsc{Approx-Subset-Sum}

**Theorem 35.8**

\textsc{Approx-Subset-Sum} is a FPTAS for the subset-sum problem.

**Proof (Running Time):**

- **Strategy:** Derive a bound on $|L_i|$ (running time is linear in $|L_i|$)
- After trimming, two successive elements $z$ and $z'$ satisfy $z'/z \geq 1 + \epsilon/(2n)$

$\Rightarrow$ Possible Values after trimming are 0, 1, and up to $\lfloor \log_{1+\epsilon/(2n)} t \rfloor$ additional values. Hence,

$$\log_{1+\epsilon/(2n)} t + 2 = \frac{\ln t}{\ln(1 + \epsilon/(2n))} + 2 \leq \frac{2n(1 + \epsilon/(2n)) \ln t}{\epsilon} + 2$$

For $x > -1$, $\ln(1 + x) \geq \frac{x}{1 + x}$
Analysis of APPROX-SUBSET-SUM

Theorem 35.8

APPROX-SUBSET-SUM is a FPTAS for the subset-sum problem.

Proof (Running Time):
- **Strategy:** Derive a bound on $|L_i|$ (running time is linear in $|L_i|$)
- After trimming, two successive elements $z$ and $z'$ satisfy $z'/z \geq 1 + \epsilon/(2n)$

$\Rightarrow$ Possible Values after trimming are 0, 1, and up to $\lceil \log_{1+\epsilon/(2n)} t \rceil$ additional values. Hence,

$$\log_{1+\epsilon/(2n)} t + 2 = \frac{\ln t}{\ln(1 + \epsilon/(2n))} + 2 \leq \frac{2n(1 + \epsilon/(2n)) \ln t}{\epsilon} + 2$$

**For $x > -1$, $\ln(1 + x) \geq \frac{x}{1+x}$**

IV. Approximation via Exact Algorithms

The Subset-Sum Problem
**Analysis of APPROX-SUBSET-SUM**

**Theorem 35.8**

**APPROX-SUBSET-SUM** is a **FPTAS** for the subset-sum problem.

**Proof (Running Time):**

- **Strategy:** Derive a bound on $|L_i|$ (running time is linear in $|L_i|$)
- After trimming, two successive elements $z$ and $z'$ satisfy $z'/z \geq 1 + \epsilon/(2n)$

$\Rightarrow$ Possible Values after trimming are 0, 1, and up to $\lfloor \log_{1+\epsilon/(2n)} t \rfloor$ additional values.

Hence,

$$\log_{1+\epsilon/(2n)} t + 2 = \frac{\ln t}{\ln(1 + \epsilon/(2n))} + 2$$

$$\leq \frac{2n(1 + \epsilon/(2n)) \ln t}{\epsilon} + 2$$

For $x > -1$, $\ln(1 + x) \geq \frac{x}{1+x}$

$$< \frac{3n \ln t}{\epsilon} + 2.$$ 

- This bound on $|L_i|$ is polynomial in the size of the input and in $1/\epsilon$. 

---

IV. Approximation via Exact Algorithms

The Subset-Sum Problem

10
Analysis of APPROX-SUBSET-SUM

**Theorem 35.8**

APPROX-SUBSET-SUM is a FPTAS for the subset-sum problem.

Proof (Running Time):

- **Strategy**: Derive a bound on $|L_i|$ (running time is linear in $|L_i|$)
- After trimming, two successive elements $z$ and $z'$ satisfy $z'/z \geq 1 + \epsilon/(2n)$

$\Rightarrow$ Possible Values after trimming are 0, 1, and up to $\lfloor \log_{1+\epsilon/(2n)} t \rfloor$ additional values. Hence,

$$\log_{1+\epsilon/(2n)} t + 2 = \ln t + 2 \leq \frac{\ln(1 + \epsilon/(2n))}{\ln(1 + \epsilon/(2n))} + 2 \leq \frac{2n(1 + \epsilon/(2n)) \ln t}{\epsilon} + 2$$

For $x > -1$, $\ln(1 + x) \geq \frac{x}{1 + x}$

$$\ln t < \frac{3n \ln t}{\epsilon} + 2.$$  

- This bound on $|L_i|$ is polynomial in the size of the input and in $1/\epsilon$.

Need $\log(t)$ bits to represent $t$ and $n$ bits to represent $S$.
Concluding Remarks

The Subset-Sum Problem

- **Given:** Set of positive integers $S = \{x_1, x_2, \ldots, x_n\}$ and positive integer $t$
- **Goal:** Find a subset $S' \subseteq S$ which maximizes $\sum_{i: x_i \in S'} x_i \leq t$. 

The Subset-Sum Problem

A **PPROX**-SUBSET-SUM is a FPTAS for the subset-sum problem.

**Theorem 35.8**

Given: Items $i = 1, 2, \ldots, n$ with weights $w_i$ and values $v_i$, and integer $t$
- **Goal:** Find a subset $S' \subseteq S$ which
  1. maximizes $\sum_{i \in S'} v_i$
  2. satisfies $\sum_{i \in S'} w_i \leq t$

The Knapsack Problem

A more general problem than Subset-Sum

There is a FPTAS for the Knapsack problem.

**Theorem**

Algorithm very similar to A **PPROX**-SUBSET-SUM

IV. Approximation via Exact Algorithms
Concluding Remarks

The Subset-Sum Problem

- **Given:** Set of positive integers \( S = \{x_1, x_2, \ldots, x_n\} \) and positive integer \( t \)
- **Goal:** Find a subset \( S' \subseteq S \) which maximizes \( \sum_{i: x_i \in S'} x_i \leq t \).

**Theorem 35.8**

**APPROX-SUBSET-SUM** is a FPTAS for the subset-sum problem.
Concluding Remarks

The Subset-Sum Problem

- **Given:** Set of positive integers \( S = \{x_1, x_2, \ldots, x_n\} \) and positive integer \( t \)
- **Goal:** Find a subset \( S' \subseteq S \) which maximizes \( \sum_{i: x_i \in S'} x_i \leq t \).

**Theorem 35.8**

**APPROX-SUBSET-SUM** is a **FPTAS** for the subset-sum problem.

The Knapsack Problem

- **Given:** Items \( i = 1, 2, \ldots, n \) with weights \( w_i \) and values \( v_i \), and integer \( t \)
Concluding Remarks

The Subset-Sum Problem

- **Given**: Set of positive integers \( S = \{x_1, x_2, \ldots, x_n\} \) and positive integer \( t \)
- **Goal**: Find a subset \( S' \subseteq S \) which maximizes \( \sum_{i: x_i \in S'} x_i \leq t \).

Theorem 35.8

**APPROX-SUBSET-SUM** is a **FPTAS** for the subset-sum problem.

The Knapsack Problem

- **Given**: Items \( i = 1, 2, \ldots, n \) with weights \( w_i \) and values \( v_i \), and integer \( t \)
- **Goal**: Find a subset \( S' \subseteq S \) which
Concluding Remarks

The Subset-Sum Problem

- **Given**: Set of positive integers \( S = \{x_1, x_2, \ldots, x_n\} \) and positive integer \( t \)
- **Goal**: Find a subset \( S' \subseteq S \) which maximizes \( \sum_{i: x_i \in S'} x_i \leq t \).

Theorem 35.8

APPROX-SUBSET-SUM is a FPTAS for the subset-sum problem.

The Knapsack Problem

- **Given**: Items \( i = 1, 2, \ldots, n \) with weights \( w_i \) and values \( v_i \), and integer \( t \)
- **Goal**: Find a subset \( S' \subseteq S \) which
  1. maximizes \( \sum_{i \in S'} v_i \)
  2. satisfies \( \sum_{i \in S'} w_i \leq t \)
Concluding Remarks

The Subset-Sum Problem

- **Given:** Set of positive integers \( S = \{x_1, x_2, \ldots, x_n\} \) and positive integer \( t \)
- **Goal:** Find a subset \( S' \subseteq S \) which maximizes \( \sum_{i: x_i \in S'} x_i \leq t \).

Theorem 35.8

**APPROX-SUBSET-SUM** is a **FPTAS** for the subset-sum problem.

The Knapsack Problem

- **Given:** Items \( i = 1, 2, \ldots, n \) with weights \( w_i \) and values \( v_i \), and integer \( t \)
- **Goal:** Find a subset \( S' \subseteq S \) which
  1. maximizes \( \sum_{i \in S'} v_i \)
  2. satisfies \( \sum_{i \in S'} w_i \leq t \)

A more general problem than Subset-Sum
Concluding Remarks

The Subset-Sum Problem

- **Given:** Set of positive integers \( S = \{x_1, x_2, \ldots, x_n\} \) and positive integer \( t \)
- **Goal:** Find a subset \( S' \subseteq S \) which maximizes \( \sum_{i: x_i \in S'} x_i \leq t \).

Theorem 35.8

**APPROX-SUBSET-SUM** is a FPTAS for the subset-sum problem.

The Knapsack Problem

- **Given:** Items \( i = 1, 2, \ldots, n \) with weights \( w_i \) and values \( v_i \), and integer \( t \)
- **Goal:** Find a subset \( S' \subseteq S \) which
  1. maximizes \( \sum_{i \in S'} v_i \)
  2. satisfies \( \sum_{i \in S'} w_i \leq t \)

**Theorem**

There is a FPTAS for the Knapsack problem.

IV. Approximation via Exact Algorithms

The Subet-Sum Problem
Concluding Remarks

The Subset-Sum Problem
- **Given**: Set of positive integers $S = \{x_1, x_2, \ldots, x_n\}$ and positive integer $t$
- **Goal**: Find a subset $S' \subseteq S$ which maximizes $\sum_{i: x_i \in S'} x_i \leq t$.

**Theorem 35.8**

**APPROX-SUBSET-SUM** is a FPTAS for the subset-sum problem.

The Knapsack Problem
- **Given**: Items $i = 1, 2, \ldots, n$ with weights $w_i$ and values $v_i$, and integer $t$
- **Goal**: Find a subset $S' \subseteq S$ which
  1. maximizes $\sum_{i \in S'} v_i$
  2. satisfies $\sum_{i \in S'} w_i \leq t$

**Theorem**

There is a FPTAS for the Knapsack problem.
Outline

The Subset-Sum Problem

Parallel Machine Scheduling

Bonus Material: A PTAS for Parallel Machine Scheduling (non-examinable)
Parallel Machine Scheduling

Machine Scheduling Problem

- Given: $n$ jobs $J_1, J_2, \ldots, J_n$ with processing times $p_1, p_2, \ldots, p_n$, and $m$ identical machines $M_1, M_2, \ldots, M_m$
Given: $n$ jobs $J_1, J_2, \ldots, J_n$ with processing times $p_1, p_2, \ldots, p_n$, and $m$ identical machines $M_1, M_2, \ldots, M_m$

Goal: Schedule the jobs on the machines minimizing the makespan $C_{\text{max}} = \max_{1 \leq j \leq n} C_j$, where $C_k$ is the completion time of job $J_k$. 
Parallel Machine Scheduling

Machine Scheduling Problem

- **Given:** \( n \) jobs \( J_1, J_2, \ldots, J_n \) with processing times \( p_1, p_2, \ldots, p_n \), and \( m \) identical machines \( M_1, M_2, \ldots, M_m \)

- **Goal:** Schedule the jobs on the machines minimizing the makespan \( C_{\text{max}} = \max_{1 \leq j \leq n} C_j \), where \( C_k \) is the completion time of job \( J_k \).

- \( J_1: p_1 = 2 \)
- \( J_2: p_2 = 12 \)
- \( J_3: p_3 = 6 \)
- \( J_4: p_4 = 4 \)
Parallel Machine Scheduling

Machine Scheduling Problem

- Given: \( n \) jobs \( J_1, J_2, \ldots, J_n \) with processing times \( p_1, p_2, \ldots, p_n \), and \( m \) identical machines \( M_1, M_2, \ldots, M_m \)

- Goal: Schedule the jobs on the machines minimizing the makespan \( C_{\text{max}} = \max_{1 \leq j \leq n} C_j \), where \( C_k \) is the completion time of job \( J_k \).

For the analysis, it will be convenient to denote by \( C_i \) the completion time of a machine \( i \).

- \( J_1: p_1 = 2 \)
- \( J_2: p_2 = 12 \)
- \( J_3: p_3 = 6 \)
- \( J_4: p_4 = 4 \)
Parallel Machine Scheduling

Machine Scheduling Problem

- Given: $n$ jobs $J_1, J_2, \ldots, J_n$ with processing times $p_1, p_2, \ldots, p_n$, and $m$ identical machines $M_1, M_2, \ldots, M_m$
- Goal: Schedule the jobs on the machines minimizing the makespan $C_{\text{max}} = \max_{1 \leq j \leq n} C_j$, where $C_k$ is the completion time of job $J_k$.

- $J_1: p_1 = 2$
- $J_2: p_2 = 12$
- $J_3: p_3 = 6$
- $J_4: p_4 = 4$

For the analysis, it will be convenient to denote by $C_i$ the completion time of a machine $i$. 

IV. Approximation via Exact Algorithms
Parallel Machine Scheduling

Machine Scheduling Problem

- **Given:** \( n \) jobs \( J_1, J_2, \ldots, J_n \) with processing times \( p_1, p_2, \ldots, p_n \), and \( m \) identical machines \( M_1, M_2, \ldots, M_m \)
- **Goal:** Schedule the jobs on the machines minimizing the makespan \( C_{\text{max}} = \max_{1 \leq j \leq n} C_j \), where \( C_k \) is the completion time of job \( J_k \).

- \( J_1: p_1 = 2 \)
- \( J_2: p_2 = 12 \)
- \( J_3: p_3 = 6 \)
- \( J_4: p_4 = 4 \)

For the analysis, it will be convenient to denote by \( C_i \) the completion time of a machine \( i \).

IV. Approximation via Exact Algorithms

Parallel Machine Scheduling
NP-Completeness of Parallel Machine Scheduling

Lemma

Parallel Machine Scheduling is NP-complete even if there are only two machines.

Proof Idea: Polynomial time reduction from NUMBER-PARTITIONING.
NP-Completeness of Parallel Machine Scheduling

Lemma

Parallel Machine Scheduling is NP-complete even if there are only two machines.

Proof Idea: Polynomial time reduction from NUMBER-PARTITIONING.

![Diagram showing two machines with tasks scheduled over time]

IV. Approximation via Exact Algorithms
Lemma

Parallel Machine Scheduling is NP-complete even if there are only two machines.

Proof Idea: Polynomial time reduction from NUMBER-PARTITIONING.

**List Scheduling** $(J_1, J_2, \ldots, J_n, m)$

1. **while** there exists an unassigned job
2. Schedule job on the machine with the least load
NP-Completeness of Parallel Machine Scheduling

Lemma

Parallel Machine Scheduling is NP-complete even if there are only two machines.

Proof Idea: Polynomial time reduction from NUMBER-PARTITIONING.

Equivalent to the following Online Algorithm [CLRS3]: Whenever a machine is idle, schedule the next job on that machine.

\textbf{LIST SCHEDULING}\((J_1, J_2, \ldots, J_n, m)\)
\begin{enumerate}
  \item \textbf{while} there exists an unassigned job
  \item Schedule job on the machine with the least load
\end{enumerate}
NP-Completeness of Parallel Machine Scheduling

Lemma

Parallel Machine Scheduling is NP-complete even if there are only two machines.

Proof Idea: Polynomial time reduction from NUMBER-PARTITIONING.

Equivalent to the following Online Algorithm [CLRS3]: Whenever a machine is idle, schedule the next job on that machine.

LIST SCHEDULING(\(J_1, J_2, \ldots, J_n, m\))
1: while there exists an unassigned job
2: Schedule job on the machine with the least load

How good is this most basic Greedy Approach?
List Scheduling Analysis (Observations)

a. The optimal makespan is at least as large as the greatest processing time, that is,
   \[ C^*_{\text{max}} \geq \max_{1 \leq k \leq n} p_k. \]

b. The optimal makespan is at least as large as the average machine load, that is,
   \[ C^*_{\text{max}} \geq \frac{1}{m} \sum_{k=1}^{n} p_k. \]

Proof:

The total processing times of all \( n \) jobs equals
   \[ \sum_{k=1}^{n} p_k \]

\[ \Rightarrow \quad \text{One machine must have a load of at least} \]
   \[ \frac{1}{m} \cdot \sum_{k=1}^{n} p_k. \]
List Scheduling Analysis (Observations)

Ex 35-5 a.&b.

a. The optimal makespan is at least as large as the greatest processing time, that is,

\[ C_{\text{max}}^* \geq \max_{1 \leq k \leq n} p_k. \]
Ex 35-5 a.&b.

a. The optimal makespan is at least as large as the greatest processing time, that is,

\[ C^*_{\text{max}} \geq \max_{1 \leq k \leq n} p_k. \]

b. The optimal makespan is at least as large as the average machine load, that is,

\[ C^*_{\text{max}} \geq \frac{1}{m} \sum_{k=1}^{n} p_k. \]
List Scheduling Analysis (Observations)

Ex 35-5 a.&b.

a. The optimal makespan is at least as large as the greatest processing time, that is,

\[ C^*_\text{max} \geq \max_{1 \leq k \leq n} p_k. \]

b. The optimal makespan is at least as large as the average machine load, that is,

\[ C^*_\text{max} \geq \frac{1}{m} \sum_{k=1}^{n} p_k. \]

Proof:
Ex 35-5 a.&b.

a. The optimal makespan is at least as large as the greatest processing time, that is,

$$C^*_\text{max} \geq \max_{1 \leq k \leq n} p_k.$$ 

b. The optimal makespan is at least as large as the average machine load, that is,

$$C^*_\text{max} \geq \frac{1}{m} \sum_{k=1}^{n} p_k.$$ 

Proof:

b. The total processing times of all $n$ jobs equals $\sum_{k=1}^{n} p_k$
List Scheduling Analysis (Observations)

Ex 35-5 a.&b.

a. The optimal makespan is at least as large as the greatest processing time, that is,

\[ C_{\text{max}}^* \geq \max_{1 \leq k \leq n} p_k. \]

b. The optimal makespan is at least as large as the average machine load, that is,

\[ C_{\text{max}}^* \geq \frac{1}{m} \sum_{k=1}^{n} p_k. \]

Proof:

b. The total processing times of all \( n \) jobs equals \( \sum_{k=1}^{n} p_k \)

\[ \Rightarrow \] One machine must have a load of at least \( \frac{1}{m} \cdot \sum_{k=1}^{n} p_k \) \[ \square \]
For the schedule returned by the greedy algorithm it holds that

\[ C_{\text{max}} \leq \frac{1}{m} \sum_{k=1}^{n} p_k + \max_{1 \leq k \leq n} p_k. \]

Hence list scheduling is a poly-time 2-approximation algorithm.
List Scheduling Analysis (Final Step)

Ex 35-5 d. (Graham 1966)

For the schedule returned by the greedy algorithm it holds that

\[ C_{\text{max}} \leq \frac{1}{m} \sum_{k=1}^{n} p_k + \max_{1 \leq k \leq n} p_k. \]

Hence list scheduling is a \text{poly-time 2-approximation algorithm.}

Proof:
List Scheduling Analysis (Final Step)

Ex 35-5 d. (Graham 1966)

For the schedule returned by the greedy algorithm it holds that

\[ C_{\text{max}} \leq \frac{1}{m} \sum_{k=1}^{n} p_k + \max_{1 \leq k \leq n} p_k. \]

Hence list scheduling is a poly-time 2-approximation algorithm.

Proof:

- Let \( J_i \) be the last job scheduled on machine \( M_j \) with \( C_{\text{max}} = C_j \)
List Scheduling Analysis (Final Step)

Ex 35-5 d. (Graham 1966)

For the schedule returned by the greedy algorithm it holds that

\[ C_{\text{max}} \leq \frac{1}{m} \sum_{k=1}^{n} p_k + \max_{1 \leq k \leq n} p_k. \]

Hence list scheduling is a poly-time 2-approximation algorithm.

Proof:
- Let \( J_i \) be the last job scheduled on machine \( M_j \) with \( C_{\text{max}} = C_j \).
List Scheduling Analysis (Final Step)

Ex 35-5 d. (Graham 1966)

For the schedule returned by the greedy algorithm it holds that

\[ C_{\text{max}} \leq \frac{1}{m} \sum_{k=1}^{n} p_k + \max_{1 \leq k \leq n} p_k. \]

Hence list scheduling is a poly-time 2-approximation algorithm.

Proof:
- Let \( J_i \) be the last job scheduled on machine \( M_j \) with \( C_{\text{max}} = C_j \)
- When \( J_i \) was scheduled to machine \( M_j \), \( C_j - p_i \leq C_k \) for all \( 1 \leq k \leq m \)

\[ 0 \quad C_j - p_i \quad C_{\text{max}} \]

IV. Approximation via Exact Algorithms
Parallel Machine Scheduling
List Scheduling Analysis (Final Step)

Ex 35-5 d. (Graham 1966)

For the schedule returned by the greedy algorithm it holds that

\[ C_{\text{max}} \leq \frac{1}{m} \sum_{k=1}^{n} p_k + \max_{1 \leq k \leq n} p_k. \]

Hence list scheduling is a poly-time 2-approximation algorithm.

Proof:

- Let \( J_i \) be the last job scheduled on machine \( M_j \) with \( C_{\text{max}} = C_j \)
- When \( J_i \) was scheduled to machine \( M_j \), \( C_j - p_i \leq C_k \) for all \( 1 \leq k \leq m \)

![Diagram showing the last job \( J_i \) scheduled on machine \( M_j \) and the relationship between \( C_j \) and \( p_i \).]
List Scheduling Analysis (Final Step)

Ex 35-5 d. (Graham 1966)

For the schedule returned by the greedy algorithm it holds that

\[ C_{\text{max}} \leq \frac{1}{m} \sum_{k=1}^{n} p_k + \max_{1 \leq k \leq n} p_k. \]

Hence list scheduling is a poly-time 2-approximation algorithm.

Proof:

- Let \( J_i \) be the last job scheduled on machine \( M_j \) with \( C_{\text{max}} = C_j \)
- When \( J_i \) was scheduled to machine \( M_j \), \( C_j - p_i \leq C_k \) for all \( 1 \leq k \leq m \)
- Averaging over \( k \) yields:

![Diagram showing the schedule with machine \( M_j \) and job \( J_i \) and their respective start and end times.]
List Scheduling Analysis (Final Step)

Ex 35-5 d. (Graham 1966)

For the schedule returned by the greedy algorithm it holds that

\[ C_{\text{max}} \leq \frac{1}{m} \sum_{k=1}^{n} p_k + \max_{1 \leq k \leq n} p_k. \]

Hence list scheduling is a poly-time 2-approximation algorithm.

Proof:
- Let \( J_i \) be the last job scheduled on machine \( M_j \) with \( C_{\text{max}} = C_j \)
- When \( J_i \) was scheduled to machine \( M_j \), \( C_j - p_i \leq C_k \) for all \( 1 \leq k \leq m \)
- Averaging over \( k \) yields:

\[ C_j - p_i \leq \frac{1}{m} \sum_{k=1}^{m} C_k \]
For the schedule returned by the greedy algorithm it holds that

\[ C_{\text{max}} \leq \frac{1}{m} \sum_{k=1}^{n} p_k + \max_{1 \leq k \leq n} p_k. \]

Hence list scheduling is a poly-time 2-approximation algorithm.

**Proof:**
- Let \( J_i \) be the last job scheduled on machine \( M_j \) with \( C_{\text{max}} = C_j \)
- When \( J_i \) was scheduled to machine \( M_j \), \( C_j - p_i \leq C_k \) for all \( 1 \leq k \leq m \)
- Averaging over \( k \) yields:

\[ C_j - p_i \leq \frac{1}{m} \sum_{k=1}^{m} C_k = \frac{1}{m} \sum_{k=1}^{n} p_k \]
List Scheduling Analysis (Final Step)

Ex 35-5 d. (Graham 1966)

For the schedule returned by the greedy algorithm it holds that

\[ C_{\text{max}} \leq \frac{1}{m} \sum_{k=1}^{n} p_k + \max_{1 \leq k \leq n} p_k. \]

Hence list scheduling is a poly-time 2-approximation algorithm.

Proof:

- Let \( J_i \) be the last job scheduled on machine \( M_j \) with \( C_{\text{max}} = C_j \)
- When \( J_i \) was scheduled to machine \( M_j \), \( C_j - p_i \leq C_k \) for all \( 1 \leq k \leq m \)
- Averaging over \( k \) yields:

\[ C_j - p_i \leq \frac{1}{m} \sum_{k=1}^{m} C_k = \frac{1}{m} \sum_{k=1}^{n} p_k \Rightarrow C_j \leq \frac{1}{m} \sum_{k=1}^{n} p_k + \max_{1 \leq k \leq n} p_k \]
List Scheduling Analysis (Final Step)

Ex 35-5 d. (Graham 1966)

For the schedule returned by the greedy algorithm it holds that

\[ C_{\text{max}} \leq \frac{1}{m} \sum_{k=1}^{n} p_k + \max_{1 \leq k \leq n} p_k. \]

Hence list scheduling is a poly-time 2-approximation algorithm.

Proof:

- Let \( J_i \) be the last job scheduled on machine \( M_j \) with \( C_{\text{max}} = C_j \)
- When \( J_i \) was scheduled to machine \( M_j \), \( C_j - p_i \leq C_k \) for all \( 1 \leq k \leq m \)
- Averaging over \( k \) yields:

\[
C_j - p_i \leq \frac{1}{m} \sum_{k=1}^{m} C_k = \frac{1}{m} \sum_{k=1}^{n} p_k \quad \Rightarrow \quad C_j \leq \frac{1}{m} \sum_{k=1}^{n} p_k + \max_{1 \leq k \leq n} p_k
\]

Using Ex 35-5 a. & b.
List Scheduling Analysis (Final Step)

Ex 35-5 d. (Graham 1966)

For the schedule returned by the greedy algorithm it holds that

\[
C_{\text{max}} \leq \frac{1}{m} \sum_{k=1}^{n} p_k + \max_{1 \leq k \leq n} p_k.
\]

Hence list scheduling is a poly-time 2-approximation algorithm.

Proof:

- Let \( J_i \) be the last job scheduled on machine \( M_j \) with \( C_{\text{max}} = C_j \)
- When \( J_i \) was scheduled to machine \( M_j \), \( C_j - p_i \leq C_k \) for all \( 1 \leq k \leq m \)
- Averaging over \( k \) yields:

\[
C_j - p_i \leq \frac{1}{m} \sum_{k=1}^{m} C_k = \frac{1}{m} \sum_{k=1}^{n} p_k \implies C_j \leq \frac{1}{m} \sum_{k=1}^{n} p_k + \max_{1 \leq k \leq n} p_k \leq 2 \cdot C^*_{\text{max}}
\]

Using Ex 35-5 a. & b.

IV. Approximation via Exact Algorithms

Parallel Machine Scheduling
Analysis can be shown to be almost tight. Is there a better algorithm?
Improving Greedy

The problem of the List-Scheduling Approach were the large jobs. Analysis can be shown to be almost tight. Is there a better algorithm?
Improving Greedy Analysis can be shown to be almost tight. Is there a better algorithm?

The problem of the List-Scheduling Approach were the large jobs

Analysis can be shown to be almost tight. Is there a better algorithm?

**Least Processing Time**($J_1, J_2, \ldots, J_n, m$)

1. Sort jobs decreasingly in their processing times
2. for $i = 1$ to $m$
3. $C_i = 0$
4. $S_i = \emptyset$
5. end for
6. for $j = 1$ to $n$
7. $i = \text{argmin}_{1 \leq k \leq m} C_k$
8. $S_i = S_i \cup \{j\}$, $C_i = C_i + p_j$
9. end for
10. return $S_1, \ldots, S_m$

**Runtime:**
- $O(n \log n)$ for sorting
- $O(n \log m)$ for extracting (and re-inserting) the minimum (use priority queue).
Improving Greedy

The problem of the List-Scheduling Approach were the large jobs

Analysis can be shown to be almost tight. Is there a better algorithm?

**Least Processing Time** ($J_1, J_2, \ldots, J_n, m$)

1: Sort jobs decreasingly in their processing times
2: for $i = 1$ to $m$
3:     $C_i = 0$
4:     $S_i = \emptyset$
5: end for
6: for $j = 1$ to $n$
7:     $i = \text{argmin}_{1 \leq k \leq m} C_k$
8:     $S_i = S_i \cup \{j\}$, $C_i = C_i + p_j$
9: end for
10: return $S_1, \ldots, S_m$

Runtime:

Runtime:

**IV. Approximation via Exact Algorithms**

Parallel Machine Scheduling
Improving Greedy Analysis can be shown to be almost tight. Is there a better algorithm?

The problem of the List-Scheduling Approach were the large jobs.

Least Processing Time \((J_1, J_2, \ldots, J_n, m)\)

1: Sort jobs decreasingly in their processing times
2: for \(i = 1\) to \(m\)
3: \(C_i = 0\)
4: \(S_i = \emptyset\)
5: end for
6: for \(j = 1\) to \(n\)
7: \(i = \arg\min_{1 \leq k \leq m} C_k\)
8: \(S_i = S_i \cup \{j\}, C_i = C_i + p_j\)
9: end for
10: return \(S_1, \ldots, S_m\)

Runtime:
- \(O(n \log n)\) for sorting
Improving Greedy Analysis can be shown to be almost tight. Is there a better algorithm?

The problem of the List-Scheduling Approach were the large jobs

Least Processing Time \((J_1, J_2, \ldots, J_n, m)\)

1: Sort jobs decreasingly in their processing times
2: for \(i = 1\) to \(m\)
3: \(C_i = 0\)
4: \(S_i = \emptyset\)
5: end for
6: for \(j = 1\) to \(n\)
7: \(i = \arg\min_{1 \leq k \leq m} C_k\)
8: \(S_i = S_i \cup \{j\}, C_i = C_i + p_j\)
9: end for
10: return \(S_1, \ldots, S_m\)

Runtime:
- \(O(n \log n)\) for sorting
- \(O(n \log m)\) for extracting (and re-inserting) the minimum (use priority queue).
Analysis of Improved Greedy

Graham 1966

The LPT algorithm has an approximation ratio of \(4/3 - 1/(3m)\).

This can be shown to be tight (see next slide).
The LPT algorithm has an approximation ratio of $\frac{4}{3} - \frac{1}{3m}$.

Proof (of approximation ratio 3/2).
Analysis of Improved Greedy

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof (of approximation ratio 3/2).
- **Observation 1**: If there are at most $m$ jobs, then the solution is optimal.
Analysis of Improved Greedy

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof (of approximation ratio 3/2).

- **Observation 1**: If there are at most $m$ jobs, then the solution is optimal.
- **Observation 2**: If there are more than $m$ jobs, then $C^*_\text{max} \geq 2 \cdot p_{m+1}$. 

IV. Approximation via Exact Algorithms
Analysis of Improved Greedy

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof (of approximation ratio $3/2$).
- Observation 1: If there are at most $m$ jobs, then the solution is optimal.
- Observation 2: If there are more than $m$ jobs, then $C^*_{\text{max}} \geq 2 \cdot p_{m+1}$.
- As in the analysis for list scheduling
Analysis of Improved Greedy

Graham 1966

The LPT algorithm has an approximation ratio of \( 4/3 - 1/(3m) \).

Proof (of approximation ratio 3/2).

- **Observation 1:** If there are at most \( m \) jobs, then the solution is optimal.
- **Observation 2:** If there are more than \( m \) jobs, then \( C^*_\text{max} \geq 2 \cdot p_{m+1} \).
- As in the analysis for list scheduling, we have

\[
C_{\text{max}} = C_j = (C_j - p_i) + p_i
\]
Analysis of Improved Greedy

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$. 

Proof (of approximation ratio $3/2$).

- Observation 1: If there are at most $m$ jobs, then the solution is optimal.
- Observation 2: If there are more than $m$ jobs, then $C^*_\text{max} \geq 2 \cdot p_{m+1}$.
- As in the analysis for list scheduling, we have

$$C^\text{max} = C_j = (C_j - p_i) + p_i \leq C^*_\text{max} + \frac{1}{2} C^*_\text{max}$$

This is for the case $i \geq m+1$ (otherwise, an even stronger inequality holds)
Analysis of Improved Greedy

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof (of approximation ratio 3/2).
- **Observation 1**: If there are at most $m$ jobs, then the solution is optimal.
- **Observation 2**: If there are more than $m$ jobs, then $C^*_\text{max} \geq 2 \cdot p_{m+1}$.
- As in the analysis for list scheduling, we have

$$C_{\text{max}} = C_j = (C_j - p_i) + p_i \leq C^*_\text{max} + \frac{1}{2} C^*_\text{max} = \frac{3}{2} C_{\text{max}}.$$

IV. Approximation via Exact Algorithms

Parallel Machine Scheduling
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$. 

Proof of an instance which shows tightness:
machines and 

\[ n = 2^m + 1 \]

jobs:
two of length $2^m - 1$, $2^m - 2$, ..., $m$ and one extra job of length $m$.

LPT gives 

\[ C_{\text{max}} = 19 \]

Optimum is 

\[ C^*_{\text{max}} = 15 \]
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $\frac{4}{3} - \frac{1}{3m}$.

Proof of an instance which shows tightness:
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $\frac{4}{3} - \frac{1}{3m}$.

Proof of an instance which shows tightness:
- $m$ machines and $n = 2m + 1$ jobs:
Tightness of the Bound for LPT

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:
- $m$ machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length $m$
Tightness of the Bound for LPT

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:

- $m$ machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length $m$

$$m = 5, n = 11 :$$

IV. Approximation via Exact Algorithms
Parallel Machine Scheduling
Tightness of the Bound for LPT

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:
- $m$ machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length $m$

$m = 5, n = 11$:
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:
- $m$ machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length $m$

$m = 5, n = 11$:
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:
- $m$ machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length $m$

$m = 5, n = 11$:

```
\begin{align*}
M_5 & \quad M_4 & \quad M_3 & \quad M_2 & \quad M_1 \\
\phantom{M_5} & \quad \phantom{M_4} & \quad \phantom{M_3} & \quad 9 & \quad 9 \\
\phantom{M_5} & \quad \phantom{M_4} & \quad \phantom{M_3} & \quad 8 & \quad 8 \\
\phantom{M_5} & \quad \phantom{M_4} & \quad \phantom{M_3} & \quad 7 & \quad 7 \\
\phantom{M_5} & \quad \phantom{M_4} & \quad \phantom{M_3} & \quad 6 & \quad 6 \\
\phantom{M_5} & \quad \phantom{M_4} & \quad \phantom{M_3} & \quad 5 & \quad 5 & \quad 5 \\
\end{align*}
```

$LPT$ gives $C_{\text{max}} = 19$

Optimum is $C^*_{\text{max}} = 15$
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:

- $m$ machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length $m$

$m = 5, n = 11$:

```
M_5
M_4
M_3 8
M_2 9
M_1 9
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

$C^*_{\text{max}} = 15$  
$LPT$ gives $C_{\text{max}} = 19$  
Optimum is $C^*_{\text{max}} = 15$
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:

- $m$ machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length $m$

\[ m = 5, \quad n = 11 : \]

IV. Approximation via Exact Algorithms
Parallel Machine Scheduling
Tightness of the Bound for LPT

The LPT algorithm has an approximation ratio of \(\frac{4}{3} - \frac{1}{(3m)}\).

Proof of an instance which shows tightness:
- \(m\) machines and \(n = 2m + 1\) jobs:
- two of length \(2m - 1, 2m - 2, \ldots, m\) and one extra job of length \(m\)

\[
m = 5, \ n = 11:
\]

\[
\begin{array}{c}
M_5 \quad \quad \quad \quad 7 \\
M_4 \quad \quad \quad \quad 8 \\
M_3 \quad \quad \quad \quad 8 \\
M_2 \quad \quad \quad \quad 9 \\
M_1 \quad \quad \quad \quad 9
\end{array}
\]
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of \( \frac{4}{3} - \frac{1}{(3m)} \).

Proof of an instance which shows tightness:

- \( m \) machines and \( n = 2m + 1 \) jobs:
- two of length \( 2m - 1, 2m - 2, \ldots, m \) and one extra job of length \( m \)

\[
m = 5, \quad n = 11:
\]

\[
\begin{array}{c}
M_5 & 7 & 7 \\
M_4 & 8 \\
M_3 & 8 \\
M_2 & 9 \\
M_1 & 9 \\
\end{array}
\]
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of \( 4/3 - 1/(3m) \).

Proof of an instance which shows tightness:

- \( m \) machines and \( n = 2m + 1 \) jobs:
- two of length \( 2m - 1, 2m - 2, \ldots, m \) and one extra job of length \( m \)

\[
m = 5, \ n = 11:
\]

\[
M_5 \quad 7 \quad M_3 \quad 8 \quad M_1 \quad 9
\]
\[
M_4 \quad 8 \quad M_2 \quad 9
\]
\[
C^\ast_{\text{max}} = 15 \quad C_{\text{max}} = 19
\]

LPT gives \( C_{\text{max}} = 19 \)

Optimum is \( C^\ast_{\text{max}} = 15 \)
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $\frac{4}{3} - \frac{1}{(3m)}$.

Proof of an instance which shows tightness:

- $m$ machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length $m$

$m = 5, n = 11$:

![Diagram showing an example instance for LPT](image)

LPT gives $C_{\text{max}} = 19$.

Optimum is $C^*_{\text{max}} = 15$. 

IV. Approximation via Exact Algorithms

Parallel Machine Scheduling
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $\frac{4}{3} - \frac{1}{3m}$.

Proof of an instance which shows tightness:
- $m$ machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length $m$

$m = 5, n = 11$:
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $\frac{4}{3} - \frac{1}{3m}$.

Proof of an instance which shows tightness:

- $m$ machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length $m$

$m = 5, n = 11:$

\[ C^*_{\text{max}} = 15 \]
\[ C_{\text{max}} = 19 \]
Tightness of the Bound for LPT

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:
- $m$ machines and $n = 2m + 1$ jobs:
- two of length $2m - 1$, $2m - 2$, \ldots, $m$ and one extra job of length $m$

$m = 5$, $n = 11$:

\[
\begin{array}{ccccccccccccccc}
M_5 & 7 & & & & & & & & & & & & 7 \\
M_4 & 8 & 6 & & & & & & & & & & & \\
M_3 & 8 & 6 & & & & & & & & & & & \\
M_2 & 9 & 5 & & & & & & & & & & & \\
M_1 & 9 & 5 & & & & & & & & & & & \\
\end{array}
\]
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:

- $m$ machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length $m$

$$m = 5, n = 11 :$$

\[
\begin{align*}
M_5 & \quad 7 \quad \quad 7 \\
M_4 & \quad 8 \quad \quad 6 \\
M_3 & \quad 8 \quad \quad 6 \\
M_2 & \quad 9 \quad \quad 5 \\
M_1 & \quad 9 \quad \quad 5 \quad \quad 5
\end{align*}
\]

$C_{\text{max}} = 19$
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:
- $m$ machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length $m$

$m = 5, n = 11$:

LPT gives $C_{\text{max}} = 19$
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:
- $m$ machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length $m$

$m = 5, n = 11$:

LPT gives $C_{\text{max}} = 19$
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:
- $m$ machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length $m$

$m = 5, n = 11$:

LPT gives $C_{\text{max}} = 19$
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:
- $m$ machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length $m$

$m = 5, n = 11$:
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:
- $m$ machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length $m$

$m = 5, n = 11$:

LPT gives $C_{\text{max}} = 19$
The LPT algorithm has an approximation ratio of \( \frac{4}{3} - \frac{1}{3m} \).

Proof of an instance which shows tightness:
- \( m \) machines and \( n = 2m + 1 \) jobs:
- two of length \( 2m - 1, 2m - 2, \ldots, m \) and one extra job of length \( m \)

For \( m = 5, n = 11 \):

LPT gives \( C_{\text{max}} = 19 \)
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:
- $m$ machines and $n = 2m + 1$ jobs:
  - two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length $m$

$$m = 5, n = 11: \quad \text{LPT gives } C_{\text{max}} = 19$$
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:
- $m$ machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length $m$

$m = 5, n = 11$:

LPT gives $C_{\text{max}} = 19$
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:
- $m$ machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots , m$ and one extra job of length $m$

$m = 5, n = 11$:

LPT gives $C_{\text{max}} = 19$
**Tightness of the Bound for LPT**

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:
- $m$ machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length $m$

$m = 5, n = 11$:

LPT gives $C_{\text{max}} = 19$
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:
- $m$ machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length $m$

$$m = 5, \ n = 11 :$$

LPT gives $C_{\text{max}} = 19$

\[
\begin{array}{c}
M_5 & 5 & 5 \\
M_4 & 8 & 7 \\
M_3 & 8 & 7 \\
M_2 & 9 & 6 \\
M_1 & 9 & 6 \\
\end{array}
\]
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:
- $m$ machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length $m$

$$m = 5, \ n = 11 : \quad \text{LPT gives} \ C_{\text{max}} = 19$$
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:

- $m$ machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length $m$

$m = 5, n = 11$:

LPT gives $C_{\text{max}} = 19$

$C^*_{\text{max}} = 15$
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:
- $m$ machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length $m$

$$m = 5, \ n = 11 :$$

LPT gives $C_{\text{max}} = 19$
Optimum is $C^*_{\text{max}} = 15$
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:
- $m$ machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length $m$

$m = 5, n = 11$:

LPT gives $C_{\text{max}} = 19$
Optimum is $C^*_{\text{max}} = 15$
Conclusion

List scheduling has an approximation ratio of 2.

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.
Conclusion

Graham 1966

List scheduling has an approximation ratio of 2.

Graham 1966

The LPT algorithm has an approximation ratio of \(4/3 - 1/(3m)\).

Theorem (Hochbaum, Shmoys’87)

There exists a PTAS for Parallel Machine Scheduling which runs in time \(O(n^{O(1/\epsilon^2)} \cdot \log P)\), where \(P := \sum_{k=1}^{n} p_k\).
Conclusion

Graham 1966

List scheduling has an approximation ratio of 2.

Graham 1966

The LPT algorithm has an approximation ratio of \(4/3 - 1/(3m)\).

Theorem (Hochbaum, Shmoys’87)

There exists a PTAS for Parallel Machine Scheduling which runs in time \(O(n^{O(1/\epsilon^2)} \cdot \log P)\), where \(P := \sum_{k=1}^{n} p_k\).

Can we find a FPTAS (for polynomially bounded processing times)?
Conclusion

Graham 1966

List scheduling has an approximation ratio of 2.

Graham 1966

The LPT algorithm has an approximation ratio of \( \frac{4}{3} - \frac{1}{3m} \).

Theorem (Hochbaum, Shmoys’87)

There exists a PTAS for Parallel Machine Scheduling which runs in time \( O(n^{O(1/\epsilon^2)} \cdot \log P) \), where \( P := \sum_{k=1}^{n} p_k \).

Can we find a FPTAS (for polynomially bounded processing times)?

No!
Conclusion

Graham 1966

List scheduling has an approximation ratio of 2.

Graham 1966

The LPT algorithm has an approximation ratio of \(4/3 - 1/(3m)\).

Theorem (Hochbaum, Shmoys’87)

There exists a PTAS for Parallel Machine Scheduling which runs in time \(O(n^{O(1/\epsilon^2)} \cdot \log P)\), where \(P := \sum_{k=1}^{n} p_k\).

Can we find a FPTAS (for polynomially bounded processing times)?

No!

Because for sufficiently small approximation ratio \(1 + \epsilon\), the computed solution has to be optimal, and Parallel Machine Scheduling is strongly NP-hard.
Exercise (easy): Run the LPT algorithm on three machines and jobs having processing times \{3, 4, 4, 3, 5, 3, 5\}. Which allocation do you get?

1. [3, 3, 5], [4, 5], [4, 3]
2. [5, 3], [5, 4], [4, 3, 3]
3. [3, 3, 3], [5, 4], [5, 4]
Outline

The Subset-Sum Problem

Parallel Machine Scheduling

Bonus Material: A PTAS for Parallel Machine Scheduling (non-examinable)
Basic Idea: For \((1 + \epsilon)\)-approximation, don’t have to work with exact \(p_k\)’s.
A PTAS for Parallel Machine Scheduling

Basic Idea: For \((1 + \epsilon)\)-approximation, don’t have to work with exact \(p_k\)’s.

\textbf{SUBROUTINE}(\(J_1, J_2, \ldots, J_n, m, T\))

1: Either: \textbf{Return} a solution with \(C_{\text{max}} \leq (1 + \epsilon) \cdot \max\{T, C^*_{\text{max}}\}\)

2: Or: \textbf{Return} there is no solution with makespan < \(T\)
A PTAS for Parallel Machine Scheduling

Basic Idea: For \((1 + \epsilon)\)-approximation, don’t have to work with exact \(p_k\)’s.

SUBROUTINE\((J_1, J_2, \ldots, J_n, m, T)\)
1. Either: \textbf{Return} a solution with \(C_{\text{max}} \leq (1 + \epsilon) \cdot \max\{T, C^*_{\text{max}}\}\)
2. Or: \textbf{Return} there is no solution with makespan \(< T\)

Key Lemma

SUBROUTINE can be implemented in time \(n^{O(1/\epsilon^2)}\).
A PTAS for Parallel Machine Scheduling

Basic Idea: For \((1 + \epsilon)\)-approximation, don’t have to work with exact \(p_k\)’s.

**Subroutine**\((J_1, J_2, \ldots, J_n, m, T)\)

1. Either: **Return** a solution with \(C_{\text{max}} \leq (1 + \epsilon) \cdot \max\{T, C^*_{\text{max}}\}\)
2. Or: **Return** there is no solution with makespan < \(T\)

**Key Lemma**

We will prove this on the next slides.

**Subroutine** can be implemented in time \(n^{O(1/\epsilon^2)}\).
A PTAS for Parallel Machine Scheduling

Basic Idea: For \((1 + \epsilon)\)-approximation, don’t have to work with exact \(p_k\)’s.

**SUBROUTINE** \((J_1, J_2, \ldots, J_n, m, T)\)

1. Either: **Return** a solution with \(C_{\text{max}} \leq (1 + \epsilon) \cdot \max\{T, C^*_{\text{max}}\}\)
2. Or: **Return** there is no solution with makespan \(< T\)

**Key Lemma**

We will prove this on the next slides.

**SUBROUTINE** can be implemented in time \(n^{O(1/\epsilon^2)}\).

**Theorem (Hochbaum, Shmoys’87)**

There exists a **PTAS** for Parallel Machine Scheduling which runs in time \(O(n^{O(1/\epsilon^2)} \cdot \log P)\), where \(P := \sum_{k=1}^{n} p_k\).
A PTAS for Parallel Machine Scheduling

Basic Idea: For \((1 + \epsilon)\)-approximation, don’t have to work with exact \(p_k\)’s.

\[
\text{SUBROUTINE}(J_1, J_2, \ldots, J_n, m, T)
\]

1: Either: \textbf{Return} a solution with \(C_{\text{max}} \leq (1 + \epsilon) \cdot \max\{T, C^*_{\text{max}}\}\)

2: Or: \textbf{Return} there is no solution with makespan < \(T\)

**Key Lemma**

We will prove this on the next slides.

\text{SUBROUTINE} can be implemented in time \(n^{O(1/\epsilon^2)}\).

**Theorem (Hochbaum, Shmoys’87)**

There exists a \textbf{PTAS} for Parallel Machine Scheduling which runs in time \(O(n^{O(1/\epsilon^2)} \cdot \log P)\), where \(P := \sum_{k=1}^{n} p_k\).

Proof (using Key Lemma):

\[
\text{PTAS}(J_1, J_2, \ldots, J_n, m)
\]

1: Do binary search to find smallest \(T\) s.t. \(C_{\text{max}} \leq (1 + \epsilon) \cdot \max\{T, C^*_{\text{max}}\}\).

2: \textbf{Return} solution computed by \text{SUBROUTINE}(J_1, J_2, \ldots, J_n, m, T)
A PTAS for Parallel Machine Scheduling

Basic Idea: For \((1 + \epsilon)\)-approximation, don’t have to work with exact \(p_k\)'s.

\textsc{Subroutine}(\(J_1, J_2, \ldots, J_n, m, T\))

1. Either: \textbf{Return} a solution with \(C_{\text{max}} \leq (1 + \epsilon) \cdot \max\{T, C^*_{\text{max}}\}\)
2. Or: \textbf{Return} there is no solution with makespan < \(T\)

Key Lemma

\textsc{Subroutine} can be implemented in time \(n^{O(1/\epsilon^2)}\).

Theorem (Hochbaum, Shmoys’87)

There exists a PTAS for Parallel Machine Scheduling which runs in time \(O(n^{O(1/\epsilon^2)} \cdot \log P)\), where \(P := \sum_{k=1}^{n} p_k\).

Proof (using Key Lemma):

PTAS(\(J_1, J_2, \ldots, J_n, m\))

1. Do binary search to find smallest \(T\) s.t. \(C_{\text{max}} \leq (1 + \epsilon) \cdot \max\{T, C^*_{\text{max}}\}\).
2. \textbf{Return} solution computed by \textsc{Subroutine}(\(J_1, J_2, \ldots, J_n, m, T\))

Since \(0 \leq C^*_{\text{max}} \leq P\) and \(C^*_{\text{max}}\) is integral, binary search terminates after \(O(\log P)\) steps.
A PTAS for Parallel Machine Scheduling

Basic Idea: For \((1 + \epsilon)\)-approximation, don’t have to work with exact \(p_k\)’s.

**SUBROUTINE**\((J_1, J_2, \ldots, J_n, m, T)\)

1. Either: **Return** a solution with \(C_{\text{max}} \leq (1 + \epsilon) \cdot \max\{T, C^*_\text{max}\}\)
2. Or: **Return** there is no solution with makespan \(< T\)

**Key Lemma**

We will prove this on the next slides.

**SUBROUTINE** can be implemented in time \(n^{O(1/\epsilon^2)}\).

**Theorem (Hochbaum, Shmoys’87)**

There exists a PTAS for Parallel Machine Scheduling which runs in time \(O(n^{O(1/\epsilon^2)} \cdot \log P)\), where \(P := \sum_{k=1}^{n} p_k\).

**Proof (using Key Lemma):**

PTAS\((J_1, J_2, \ldots, J_n, m)\)

1. Do binary search to find smallest \(T\) s.t. \(C_{\text{max}} \leq (1 + \epsilon) \cdot \max\{T, C^*_\text{max}\}\).
2. **Return** solution computed by **SUBROUTINE**\((J_1, J_2, \ldots, J_n, m, T)\)

Since \(0 \leq C^*_\text{max} \leq P\) and \(C^*_\text{max}\) is integral, binary search terminates after \(O(\log P)\) steps.
A PTAS for Parallel Machine Scheduling

Basic Idea: For \((1 + \epsilon)\)-approximation, don’t have to work with exact \(p_k\)’s.

**SUBROUTINE**\((J_1, J_2, \ldots, J_n, m, T)\)

1. Either: **Return** a solution with \(C_{\text{max}} \leq (1 + \epsilon) \cdot \max\{T, C^*_{\text{max}}\}\)
2. Or: **Return** there is no solution with makespan \(< T\)

**Key Lemma**
We will prove this on the next slides.

**SUBROUTINE** can be implemented in time \(n^{O(1/\epsilon^2)}\).

**Theorem (Hochbaum, Shmoys’87)**
There exists a **PTAS** for Parallel Machine Scheduling which runs in time \(O(n^{O(1/\epsilon^2)} \cdot \log P)\), where \(P := \sum_{k=1}^{n} p_k\).

**Proof (using Key Lemma):**

**PTAS**\((J_1, J_2, \ldots, J_n, m)\)

1. Do binary search to find smallest \(T\) s.t. \(C_{\text{max}} \leq (1 + \epsilon) \cdot \max\{T, C^*_{\text{max}}\}\).
2. **Return** solution computed by **SUBROUTINE**\((J_1, J_2, \ldots, J_n, m, T)\)

Since \(0 \leq C^*_{\text{max}} \leq P\) and \(C^*_{\text{max}}\) is integral, binary search terminates after \(O(\log P)\) steps.

IV. Approximation via Exact Algorithms Bonus Material: A PTAS for Parallel Machine Scheduling (non-examinable)
Implementation of Subroutine

\text{SUBROUTINE}(J_1, J_2, \ldots, J_n, m, T)

1. Either: \textbf{Return} a solution with $C_{\text{max}} \leq (1 + \epsilon) \cdot \max\{T, C^*_{\text{max}}\}$

2. Or: \textbf{Return} there is no solution with makespan $< T$
Implementation of Subroutine

\textbf{SUBROUTINE}(J_1, J_2, \ldots, J_n, m, T)

1: Either: \textbf{Return} a solution with \( C_{\text{max}} \leq (1 + \epsilon) \cdot \max\{T, C^{*}_{\text{max}}\} \)

2: Or: \textbf{Return} there is no solution with makespan < \( T \)

---

Observation

Divide jobs into two groups: \( J_{\text{small}} = \{i : p_i \leq \epsilon \cdot T\} \) and \( J_{\text{large}} = [n] \setminus J_{\text{small}} \).

Given a solution for \( J_{\text{large}} \) only with makespan \((1 + \epsilon) \cdot T\), then greedily placing \( J_{\text{small}} \) yields a solution with makespan \((1 + \epsilon) \cdot \max\{T, C^{*}_{\text{max}}\} \).
Implementation of Subroutine

**SUBROUTINE**\((J_1, J_2, \ldots, J_n, m, T)\)

1: Either: **Return** a solution with \(C_{\text{max}} \leq (1 + \epsilon) \cdot \max\{T, C^*_{\text{max}}\}\)

2: Or: **Return** there is no solution with makespan \(< T\)

---

**Observation**

**Divide** jobs into two groups: \(J_{\text{small}} = \{i : p_i \leq \epsilon \cdot T\}\) and \(J_{\text{large}} = [n] \setminus J_{\text{small}}\).

Given a solution for \(J_{\text{large}}\) only with makespan \((1 + \epsilon) \cdot T\), then **greedily placing** \(J_{\text{small}}\) **yields** a solution with makespan \((1 + \epsilon) \cdot \max\{T, C^*_{\text{max}}\}\).

Proof:
Implementation of Subroutine

```plaintext
SUBROUTINE(J₁, J₂, . . . , Jₙ, m, T)
1: Either: Return a solution with \( C_{\text{max}} \leq (1 + \epsilon) \cdot \max\{T, C^*_{\text{max}}\} \)
2: Or: Return there is no solution with makespan < T
```

Observation

Divide jobs into two groups: \( J_{\text{small}} = \{i : p_i \leq \epsilon \cdot T\} \) and \( J_{\text{large}} = [n] \setminus J_{\text{small}} \).
Given a solution for \( J_{\text{large}} \) only with makespan \( (1 + \epsilon) \cdot T \), then greedily placing \( J_{\text{small}} \) yields a solution with makespan \( (1 + \epsilon) \cdot \max\{T, C^*_{\text{max}}\} \).

Proof:
- Let \( M_j \) be the machine with the largest load
Implementation of Subroutine

**SUBROUTINE**\((J_1, J_2, \ldots, J_n, m, T)\)

1. Either: **Return** a solution with \(C_{\text{max}} \leq (1 + \epsilon) \cdot \max\{T, C_{\text{max}}^*\}\)
2. Or: **Return** there is no solution with makespan < \(T\)

---

Observation

**Divide** jobs into two groups: \(J_{\text{small}} = \{i : p_i \leq \epsilon \cdot T\}\) and \(J_{\text{large}} = [n] \setminus J_{\text{small}}\).

Given a solution for \(J_{\text{large}}\) only with makespan \((1 + \epsilon) \cdot T\), then greedily placing \(J_{\text{small}}\) yields a solution with makespan \((1 + \epsilon) \cdot \max\{T, C_{\text{max}}^*\}\).

---

**Proof:**

- Let \(M_j\) be the machine with largest load
- If there are no jobs from \(J_{\text{small}}\), then makespan is at most \((1 + \epsilon) \cdot T\).
**Implementation of Subroutine**

\[\text{SUBROUTINE}(J_1, J_2, \ldots, J_n, m, T)\]

1: Either: **Return** a solution with \(C_{\text{max}} \leq (1 + \epsilon) \cdot \max\{T, C^*_\text{max}\}\)

2: Or: **Return** there is no solution with makespan \(< T\)

---

**Observation**

*Divide* jobs into two groups: \(J_{\text{small}} = \{i : p_i \leq \epsilon \cdot T\}\) and \(J_{\text{large}} = [n] \setminus J_{\text{small}}\).

Given a solution for \(J_{\text{large}}\) only with makespan \((1 + \epsilon) \cdot T\), then *greedily* placing \(J_{\text{small}}\) yields a solution with makespan \((1 + \epsilon) \cdot \max\{T, C^*_\text{max}\}\).

**Proof:**

- Let \(M_j\) be the machine with largest load
- If there are no jobs from \(J_{\text{small}}\), then makespan is at most \((1 + \epsilon) \cdot T\).
- Otherwise, let \(i \in J_{\text{small}}\) be the last job added to \(M_j\).
Implementation of Subroutine

\textbf{SUBROUTINE}(J_1, J_2, \ldots, J_n, m, T)

1: Either: \textbf{Return} a solution with $C_{\text{max}} \leq (1 + \epsilon) \cdot \max\{ T, C^*_{\text{max}} \}$

2: Or: \textbf{Return} there is no solution with makespan $< T$

---

Observation

Divide jobs into two groups: $J_{\text{small}} = \{ i : p_i \leq \epsilon \cdot T \}$ and $J_{\text{large}} = [n] \setminus J_{\text{small}}$. Given a solution for $J_{\text{large}}$ only with makespan $(1 + \epsilon) \cdot T$, then greedily placing $J_{\text{small}}$ yields a solution with makespan $(1 + \epsilon) \cdot \max\{ T, C^*_{\text{max}} \}$.

---

Proof:

- Let $M_j$ be the machine with largest load
- If there are no jobs from $J_{\text{small}}$, then makespan is at most $(1 + \epsilon) \cdot T$.
- Otherwise, let $i \in J_{\text{small}}$ be the last job added to $M_j$.

\[
C_j - p_i \leq \frac{1}{m} \sum_{k=1}^{n} p_k
\]

the “well-known” formula
SUBROUTINE\( (J_1, J_2, \ldots, J_n, m, T) \)

1. Either: **Return** a solution with \( C_{\max} \leq (1 + \epsilon) \cdot \max\{T, C^*_{\max}\} \)
2. Or: **Return** there is no solution with makespan \( < T \)

---

**Observation**

Divide jobs into two groups: \( J_{\text{small}} = \{i : p_i \leq \epsilon \cdot T\} \) and \( J_{\text{large}} = [n] \setminus J_{\text{small}} \).

Given a solution for \( J_{\text{large}} \) only with makespan \( (1 + \epsilon) \cdot T \), then greedily placing \( J_{\text{small}} \) yields a solution with makespan \( (1 + \epsilon) \cdot \max\{T, C^*_{\max}\} \).

**Proof:**

- Let \( M_j \) be the machine with largest load
- If there are no jobs from \( J_{\text{small}} \), then makespan is at most \( (1 + \epsilon) \cdot T \).
- Otherwise, let \( i \in J_{\text{small}} \) be the last job added to \( M_j \).

\[
C_j - p_i \leq \frac{1}{m} \sum_{k=1}^{n} p_k \quad \Rightarrow
\]

the “well-known” formula
Implementation of Subroutine

\begin{align*}
\text{SUBROUTINE}(J_1, J_2, \ldots, J_n, m, T) \\
1: \text{Either: } \textbf{Return} \text{ a solution with } C_{\text{max}} \leq (1 + \epsilon) \cdot \max\{T, C^*_{\text{max}}\} \\
2: \text{Or: } \textbf{Return} \text{ there is no solution with makespan } < T
\end{align*}

Observation

Divide jobs into two groups: \( J_{\text{small}} = \{ i : p_i \leq \epsilon \cdot T \} \) and \( J_{\text{large}} = [n] \setminus J_{\text{small}} \). Given a solution for \( J_{\text{large}} \) only with makespan \((1 + \epsilon) \cdot T\), then greedily placing \( J_{\text{small}} \) yields a solution with makespan \((1 + \epsilon) \cdot \max\{T, C^*_{\text{max}}\}\).

Proof:

- Let \( M_j \) be the machine with largest load
- If there are no jobs from \( J_{\text{small}} \), then makespan is at most \((1 + \epsilon) \cdot T\).
- Otherwise, let \( i \in J_{\text{small}} \) be the last job added to \( M_j \).

\[
C_j - p_i \leq \frac{1}{m} \sum_{k=1}^{n} p_k \quad \Rightarrow \quad C_j \leq p_i + \frac{1}{m} \sum_{k=1}^{n} p_k
\]

the “well-known” formula
Implementation of Subroutine

\text{SUBROUTINE}(J_1, J_2, \ldots, J_n, m, T)\]

1: Either: \textbf{Return} a solution with $C_{\text{max}} \leq (1 + \epsilon) \cdot \max\{T, C_{\text{max}}^*\}$
2: Or: \textbf{Return} there is no solution with makespan $< T$

---

Observation

Divide jobs into two groups: $J_{\text{small}} = \{i : p_i \leq \epsilon \cdot T\}$ and $J_{\text{large}} = [n] \setminus J_{\text{small}}$.
Given a solution for $J_{\text{large}}$ only with makespan $(1 + \epsilon) \cdot T$, then greedily placing $J_{\text{small}}$ yields a solution with makespan $(1 + \epsilon) \cdot \max\{T, C_{\text{max}}^*\}$.

---

Proof:

- Let $M_j$ be the machine with largest load
- If there are no jobs from $J_{\text{small}}$, then makespan is at most $(1 + \epsilon) \cdot T$.
- Otherwise, let $i \in J_{\text{small}}$ be the last job added to $M_j$.

\[
C_j - p_i \leq \frac{1}{m} \sum_{k=1}^{n} p_k \quad \Rightarrow \quad C_j \leq p_i + \frac{1}{m} \sum_{k=1}^{n} p_k \leq \epsilon \cdot T + C_{\text{max}}^* \]

the “well-known” formula
Implementation of Subroutine

**SUBROUTINE**($J_1, J_2, \ldots, J_n, m, T$)

1: Either: **Return** a solution with $C_{\text{max}} \leq (1 + \epsilon) \cdot \max\{T, C^*_{\text{max}}\}$
2: Or: **Return** there is no solution with makespan $< T$

---

**Observation**

**Divide** jobs into two groups: $J_{\text{small}} = \{i : p_i \leq \epsilon \cdot T\}$ and $J_{\text{large}} = [n] \setminus J_{\text{small}}$.

Given a solution for $J_{\text{large}}$ only with makespan $(1 + \epsilon) \cdot T$, then **greedily placing** $J_{\text{small}}$ yields a solution with makespan $(1 + \epsilon) \cdot \max\{T, C^*_{\text{max}}\}$.

---

**Proof:**

- Let $M_j$ be the machine with largest load
- If there are no jobs from $J_{\text{small}}$, then makespan is at most $(1 + \epsilon) \cdot T$.
- Otherwise, let $i \in J_{\text{small}}$ be the last job added to $M_j$.

\[
C_j - p_i \leq \frac{1}{m} \sum_{k=1}^{n} p_k \
\Rightarrow 
C_j \leq p_i + \frac{1}{m} \sum_{k=1}^{n} p_k \
\leq \epsilon \cdot T + C^*_{\text{max}} \\
\leq (1 + \epsilon) \cdot \max\{T, C^*_{\text{max}}\}
\]


IV. Approximation via Exact Algorithms Bonus Material: A PTAS for Parallel Machine Scheduling (non-examinable) 24


**Implementation of Subroutine**

\[ \text{SUBROUTINE}(J_1, J_2, \ldots, J_n, m, T) \]

1. Either: **Return** a solution with \( C_{\text{max}} \leq (1 + \epsilon) \cdot \max\{T, C^*_\text{max}\} \)
2. Or: **Return** there is no solution with makespan \(< T \)

---

**Observation**

Divide jobs into two groups: \( J_{\text{small}} = \{i: p_i \leq \epsilon \cdot T\} \) and \( J_{\text{large}} = [n] \setminus J_{\text{small}} \).

Given a solution for \( J_{\text{large}} \) only with makespan \((1 + \epsilon) \cdot T\), then greedily placing \( J_{\text{small}} \) yields a solution with makespan \((1 + \epsilon) \cdot \max\{T, C^*_\text{max}\}\).

---

**Proof:**

- Let \( M_j \) be the machine with largest load
- If there are no jobs from \( J_{\text{small}} \), then makespan is at most \((1 + \epsilon) \cdot T\).
- Otherwise, let \( i \in J_{\text{small}} \) be the last job added to \( M_j \).

\[
C_j - p_i \leq \frac{1}{m} \sum_{k=1}^{n} p_k \quad \Rightarrow \quad C_j \leq p_i + \frac{1}{m} \sum_{k=1}^{n} p_k
\]

the “well-known” formula

\[
\leq \epsilon \cdot T + C^*_\text{max}
\]

\[
\leq (1 + \epsilon) \cdot \max\{T, C^*_\text{max}\} \quad \square
\]
Proof of Key Lemma (non-examinable)

Use **Dynamic Programming** to schedule $J_{\text{large}}$ with makespan $(1 + \epsilon) \cdot T$. 

IV. Approximation via Exact Algorithms  Bonus Material: A PTAS for Parallel Machine Scheduling (non-examinable)
Proof of Key Lemma (non-examinable)

Use Dynamic Programming to schedule $J_{\text{large}}$ with makespan $(1 + \epsilon) \cdot T$.

- Let $b$ be the smallest integer with $1/b \leq \epsilon$. 

IV. Approximation via Exact Algorithms
Bonus Material: A PTAS for Parallel Machine Scheduling (non-examinable)

25
Proof of Key Lemma (non-examinable)

Use Dynamic Programming to schedule $J_{\text{large}}$ with makespan $(1 + \epsilon) \cdot T$.

- Let $b$ be the smallest integer with $1/b \leq \epsilon$. Define processing times $p'_i = \left\lfloor \frac{p_j b^2}{T} \right\rfloor \cdot \frac{T}{b^2}$.
Proof of Key Lemma (non-examinable)

Use **Dynamic Programming** to schedule $J_{\text{large}}$ with makespan $(1 + \epsilon) \cdot T$.

- Let $b$ be the smallest integer with $1/b \leq \epsilon$. Define processing times $p'_i = \left\lfloor \frac{p_i b^2}{T} \right\rfloor \cdot \frac{T}{b^2}$.
Proof of Key Lemma (non-examinable)

Use Dynamic Programming to schedule $J_{\text{large}}$ with makespan $(1 + \epsilon) \cdot T$.

- Let $b$ be the smallest integer with $1/b \leq \epsilon$. Define processing times $p'_i = \left\lfloor \frac{p_i b^2}{T} \right\rfloor \cdot \frac{T}{b^2}$.
Use Dynamic Programming to schedule \( J_{\text{large}} \) with makespan \((1 + \epsilon) \cdot T\).

- Let \( b \) be the smallest integer with \( 1/b \leq \epsilon \). Define processing times \( p'_i = \left\lfloor \frac{p_i b^2}{T} \right\rfloor \cdot \frac{T}{b^2} \)

### Example

- \( \epsilon = 0.5 \)
- \( b = 2 \)
Proof of Key Lemma (non-examinable)

**Use Dynamic Programming to schedule** $J_{\text{large}}$ **with makespan** $(1 + \epsilon) \cdot T$.

- Let $b$ be the smallest integer with $1/b \leq \epsilon$. Define processing times $p'_i = \left\lceil \frac{p_j b^2}{T} \right\rceil \cdot \frac{T}{b^2}$

![Diagram of scheduled jobs]

- $\epsilon = 0.5$
- $b = 2$

IV. Approximation via Exact Algorithms Bonus Material: A PTAS for Parallel Machine Scheduling (non-examinable)
Use Dynamic Programming to schedule \( J_{\text{large}} \) with makespan \((1 + \epsilon) \cdot T\).

- Let \( b \) be the smallest integer with \( 1/b \leq \epsilon \). Define processing times \( p'_i = \left\lfloor \frac{p_i b^2}{T} \right\rfloor \cdot \frac{T}{b^2} \).
Proof of Key Lemma (non-examinable)

Use **Dynamic Programming** to schedule \( J_{\text{large}} \) with makespan \((1 + \epsilon) \cdot T\).

- Let \( b \) be the smallest integer with \( 1/b \leq \epsilon \). Define processing times \( p'_i = \left\lfloor \frac{p_j b^2}{T} \right\rfloor \cdot \frac{T}{b^2} \)

\[ \begin{align*}
J_{\text{large}} & \quad J_{\text{small}} \\
1.5 \cdot T & \quad \epsilon = 0.5 \\
1.25 \cdot T & \quad b = 2 \\
1.0 \cdot T & \\
0.75 \cdot T & \\
0.5 \cdot T & \\
0.25 \cdot T & \\
0 & \\
\end{align*} \]

\[ \begin{align*}
J'_{\text{large}} & \\
1.5 \cdot T & \\
1.25 \cdot T & \\
1.0 \cdot T & \\
0.75 \cdot T & \\
0.5 \cdot T & \\
0.25 \cdot T & \\
0 & \\
\end{align*} \]
Proof of Key Lemma (non-examinable)

Use Dynamic Programming to schedule $J_{\text{large}}$ with makespan $(1 + \epsilon) \cdot T$.

- Let $b$ be the smallest integer with $1/b \leq \epsilon$. Define processing times $p'_i = \left\lfloor \frac{p_j b^2}{T} \right\rfloor \cdot \frac{T}{b^2}$
- $\Rightarrow$ Every $p'_i = \alpha \cdot \frac{T}{b^2}$ for $\alpha = b, b + 1, \ldots, b^2$

Can assume there are no jobs with $p_j \geq T$.

- $\epsilon = 0.5$
- $b = 2$

IV. Approximation via Exact Algorithms Bonus Material: A PTAS for Parallel Machine Scheduling (non-examinable)
Proof of Key Lemma (non-examinable)

Use **Dynamic Programming** to schedule $J_{\text{large}}$ with makespan $(1 + \epsilon) \cdot T$.

- Let $b$ be the smallest integer with $1/b \leq \epsilon$. Define processing times $p'_i = \left\lfloor \frac{p_i b^2}{T} \right\rfloor \cdot \frac{T}{b^2}$

  $\Rightarrow$ Every $p'_i = \alpha \cdot \frac{T}{b^2}$ for $\alpha = b, b + 1, \ldots, b^2$

- Let $C$ be all $(s_b, s_{b+1}, \ldots, s_{b^2})$ with $\sum_{j=b}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T.$

### IV. Approximation via Exact Algorithms

**Bonus Material:** A PTAS for Parallel Machine Scheduling (non-examinable)
Proof of Key Lemma (non-examinable)

Use **Dynamic Programming** to schedule \(J_{\text{large}}\) with makespan \((1 + \epsilon) \cdot T\).

- Let \(b\) be the smallest integer with \(1/b \leq \epsilon\). Define processing times \(p'_i = \left\lfloor \frac{p_j b^2}{T} \right\rfloor \cdot \frac{T}{b^2}\)

\(\Rightarrow\) Every \(p'_i = \alpha \cdot \frac{T}{b^2}\) for \(\alpha = b, b + 1, \ldots, b^2\)

- Let \(C\) be all \((s_b, s_{b+1}, \ldots, s_{b^2})\) with \(\sum_{j=b}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T\)

Assignments to one machine with makespan \(\leq T\).

IV. Approximation via Exact Algorithms Bonus Material: A PTAS for Parallel Machine Scheduling (non-examinable)
Proof of Key Lemma (non-examinable)

Use Dynamic Programming to schedule \( J_{\text{large}} \) with makespan \((1 + \epsilon) \cdot T\).

- Let \( b \) be the smallest integer with \( \frac{1}{b} \leq \epsilon \). Define processing times \( p_i' = \left\lfloor \frac{p_j b^2}{T} \right\rfloor \cdot \frac{T}{b^2} \)

\( \Rightarrow \) Every \( p_i' = \alpha \cdot \frac{T}{b^2} \) for \( \alpha = b, b + 1, \ldots, b^2 \)

- Let \( C \) be all \((s_b, s_{b+1}, \ldots, s_{b^2})\) with \( \sum_{j=b}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T \).

- Let \( f(n_b, n_{b+1}, \ldots, n_{b^2}) \) be the minimum number of machines required to schedule all jobs with makespan \( \leq T \):

\[ f(n_b, n_{b+1}, \ldots, n_{b^2}) = \begin{cases} 0 & \text{if } f(n_b, n_{b+1}, \ldots, n_{b^2}) \leq m \\
1 + \min_{(s_b, s_{b+1}, \ldots, s_{b^2}) \in C} f(n_b - s_b, n_{b+1} - s_{b+1}, \ldots, n_{b^2} - s_{b^2}) & \text{otherwise} \end{cases} \]

Number of table entries is at most \( n b^2 \), hence filling all entries takes \( n O(b^2) \).

If \( f(n_b, n_{b+1}, \ldots, n_{b^2}) \leq m \) (for the jobs with \( p_i' \)), then return yes, otherwise no.

As every machine is assigned at most \( b \) jobs (\( p_i' \geq \frac{T}{b} \)) and the makespan is \( \leq T \),

\[ C_{\text{max}} \leq T + b \cdot \max_{i \in J_{\text{large}}} (p_i - p_i') \leq (1 + \epsilon) \cdot T \]

Can assume there are no jobs with \( p_j \geq T ! \) 

Assignments to one machine with makespan \( \leq T \).

Assign some jobs to one machine, and then use as few machines as possible for the rest.

IV. Approximation via Exact Algorithms Bonus Material: A PTAS for Parallel Machine Scheduling (non-examinable)
Proof of Key Lemma (non-examinable)

Use Dynamic Programming to schedule \( J_{\text{large}} \) with makespan \((1 + \epsilon) \cdot T\).

- Let \( b \) be the smallest integer with \( 1/b \leq \epsilon \). Define processing times \( p'_i = \left\lfloor \frac{p_i b^2}{T} \right\rfloor \cdot \frac{T}{b^2} \).

\[
\Rightarrow \quad \text{Every } p'_i = \alpha \cdot \frac{T}{b^2} \text{ for } \alpha = b, b + 1, \ldots, b^2
\]

- Let \( C \) be all \((s_b, s_{b+1}, \ldots, s_{b^2})\) with \( \sum_{j=b}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T \).

- Let \( f(n_b, n_{b+1}, \ldots, n_{b^2}) \) be the minimum number of machines required to schedule all jobs with makespan \( \leq T \):

\[
f(0, 0, \ldots, 0) = 0
\]

IV. Approximation via Exact Algorithms Bonus Material: A PTAS for Parallel Machine Scheduling (non-examinable)
Use Dynamic Programming to schedule $J_{\text{large}}$ with makespan $\left(1 + \epsilon\right) \cdot T$.

- Let $b$ be the smallest integer with $1/b \leq \epsilon$. Define processing times $p'_i = \left\lfloor \frac{p_j b^2}{T} \right\rfloor \cdot \frac{T}{b^2}$

  $\Rightarrow$ Every $p'_i = \alpha \cdot \frac{T}{b^2}$ for $\alpha = b, b + 1, \ldots, b^2$

- Let $C$ be all $(s_b, s_{b+1}, \ldots, s_{b^2})$ with $\sum_{j=b}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T$.

- Let $f(n_b, n_{b+1}, \ldots, n_{b^2})$ be the minimum number of machines required to schedule all jobs with makespan $\leq T$:

  \[
  f(0, 0, \ldots, 0) = 0 \\
  f(n_b, n_{b+1}, \ldots, n_{b^2}) = 1 + \min_{(s_b, s_{b+1}, \ldots, s_{b^2}) \in C} f(n_b - s_b, n_{b+1} - s_{b+1}, \ldots, n_{b^2} - s_{b^2}).
  \]

IV. Approximation via Exact Algorithms

Bonus Material: A PTAS for Parallel Machine Scheduling (non-examinable)
Proof of Key Lemma (non-examinable)

Use Dynamic Programming to schedule $J_{\text{large}}$ with makespan $(1 + \epsilon) \cdot T$.

- Let $b$ be the smallest integer with $1/b \leq \epsilon$. Define processing times $p'_i = \left\lfloor \frac{p_j b^2}{T} \right\rfloor \cdot \frac{T}{b^2}$.

  ⇒ Every $p'_i = \alpha \cdot \frac{T}{b^2}$ for $\alpha = b, b + 1, \ldots, b^2$.

- Let $C$ be all $(s_b, s_{b+1}, \ldots, s_{b^2})$ with $\sum_{j=b}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T$.

- Let $f(n_b, n_{b+1}, \ldots, n_{b^2})$ be the minimum number of machines required to schedule all jobs with makespan $\leq T$:

  
  - $f(0, 0, \ldots, 0) = 0$
  
  - $f(n_b, n_{b+1}, \ldots, n_{b^2}) = 1 + \min_{(s_b, s_{b+1}, \ldots, s_{b^2}) \in C} f(n_b - s_b, n_{b+1} - s_{b+1}, \ldots, n_{b^2} - s_{b^2})$.

Assign some jobs to one machine, and then use as few machines as possible for the rest.

IV. Approximation via Exact Algorithms Bonus Material: A PTAS for Parallel Machine Scheduling (non-examinable)
Proof of Key Lemma (non-examinable)

**Use Dynamic Programming** to schedule $J_{\text{large}}$ with makespan $(1 + \epsilon) \cdot T$.

- Let $b$ be the smallest integer with $1/b \leq \epsilon$. Define processing times $p'_i = \left\lfloor \frac{p_j b^2}{T} \right\rfloor \cdot \frac{T}{b^2}$

  \[ \Rightarrow \]  
  Every $p'_i = \alpha \cdot \frac{T}{b^2}$ for $\alpha = b, b + 1, \ldots, b^2$

- Let $\mathcal{C}$ be all $(s_b, s_{b+1}, \ldots, s_{b^2})$ with $\sum_{j=b}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T$.

- Let $f(n_b, n_{b+1}, \ldots, n_{b^2})$ be the minimum number of machines required to schedule all jobs with makespan $\leq T$:

  \[
  f(0, 0, \ldots, 0) = 0 \\
  f(n_b, n_{b+1}, \ldots, n_{b^2}) = 1 + \min_{(s_b, s_{b+1}, \ldots, s_{b^2}) \in \mathcal{C}} f(n_b - s_b, n_{b+1} - s_{b+1}, \ldots, n_{b^2} - s_{b^2}).
  \]

- Number of table entries is at most $n^{b^2}$, hence filling all entries takes $n^{O(b^2)}$
Use Dynamic Programming to schedule \( J_{\text{large}} \) with makespan \((1 + \epsilon) \cdot T\).

- Let \( b \) be the smallest integer with \( 1/b \leq \epsilon \). Define processing times \( p'_i = \left\lfloor \frac{p_j b^2}{T} \right\rfloor \cdot \frac{T}{b^2} \).

\[ \Rightarrow \] Every \( p'_i = \alpha \cdot \frac{T}{b^2} \) for \( \alpha = b, b + 1, \ldots, b^2 \).

- Let \( C \) be all \((s_b, s_{b+1}, \ldots, s_{b^2})\) with \( \sum_{j=b}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T \).

- Let \( f(n_b, n_{b+1}, \ldots, n_{b^2}) \) be the minimum number of machines required to schedule all jobs with makespan \( \leq T \):

\[ f(0, 0, \ldots, 0) = 0 \]

\[ f(n_b, n_{b+1}, \ldots, n_{b^2}) = 1 + \min_{(s_b, s_{b+1}, \ldots, s_{b^2}) \in C} f(n_b - s_b, n_{b+1} - s_{b+1}, \ldots, n_{b^2} - s_{b^2}). \]

- Number of table entries is at most \( n^{b^2} \), hence filling all entries takes \( n^{O(b^2)} \).
- If \( f(n_b, n_{b+1}, \ldots, n_{b^2}) \leq m \) (for the jobs with \( p' \)), then return yes, otherwise no.
Proof of Key Lemma (non-examinable)

Use **Dynamic Programming** to schedule \(J_{\text{large}}\) with makespan \((1 + \epsilon) \cdot T\).

- Let \(b\) be the smallest integer with \(1/b \leq \epsilon\). Define processing times \(p'_i = \left\lfloor \frac{p_i b^2}{T} \right\rfloor \cdot \frac{T}{b^2}\).

  \[
  \Rightarrow \quad \text{Every } p'_i = \alpha \cdot \frac{T}{b^2} \text{ for } \alpha = b, b + 1, \ldots, b^2
  \]

- Let \(C\) be all \((s_b, s_{b+1}, \ldots, s_{b^2})\) with \(\sum_{j=b}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T\).

- Let \(f(n_b, n_{b+1}, \ldots, n_{b^2})\) be the minimum number of machines required to schedule all jobs with makespan \(\leq T\):

  \[
  f(0, 0, \ldots, 0) = 0
  \]

  \[
  f(n_b, n_{b+1}, \ldots, n_{b^2}) = 1 + \min_{(s_b, s_{b+1}, \ldots, s_{b^2}) \in C} f(n_b - s_b, n_{b+1} - s_{b+1}, \ldots, n_{b^2} - s_{b^2}).
  \]

- **Number of table entries is at most** \(n^{b^2}\), hence filling all entries takes \(n^{O(b^2)}\).
- If \(f(n_b, n_{b+1}, \ldots, n_{b^2}) \leq m\) (for the jobs with \(p'\)), then return yes, otherwise no.
- As every machine is assigned at most \(b\) jobs (\(p'_i \geq \frac{T}{b}\)) and the makespan is \(\leq T\),
Proof of Key Lemma (non-examinable)

Use Dynamic Programming to schedule $J_{\text{large}}$ with makespan $(1 + \epsilon) \cdot T$.

- Let $b$ be the smallest integer with $1/b \leq \epsilon$. Define processing times $p'_i = \left\lfloor \frac{p_i b^2}{T} \right\rfloor \cdot \frac{T}{b^2}$
  - Every $p'_i = \alpha \cdot \frac{T}{b^2}$ for $\alpha = b, b + 1, \ldots, b^2$
- Let $\mathcal{C}$ be all $(s_b, s_{b+1}, \ldots, s_{b^2})$ with $\sum_{j=b}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T$.
- Let $f(n_b, n_{b+1}, \ldots, n_{b^2})$ be the minimum number of machines required to schedule all jobs with makespan $\leq T$:
  \[
f(0, 0, \ldots, 0) = 0
  \]
  \[
f(n_b, n_{b+1}, \ldots, n_{b^2}) = 1 + \min_{(s_b, s_{b+1}, \ldots, s_{b^2}) \in \mathcal{C}} f(n_b - s_b, n_{b+1} - s_{b+1}, \ldots, n_{b^2} - s_{b^2}).
  \]
- Number of table entries is at most $n^{b^2}$, hence filling all entries takes $n^{O(b^2)}$
- If $f(n_b, n_{b+1}, \ldots, n_{b^2}) \leq m$ (for the jobs with $p'$), then return yes, otherwise no.
- As every machine is assigned at most $b$ jobs ($p'_i \geq \frac{T}{b}$) and the makespan is $\leq T$,
  \[
  C_{\text{max}} \leq T + b \cdot \max_{i \in J_{\text{large}}} (p_i - p'_i)
  \]
Proof of Key Lemma (non-examinable)

**Use Dynamic Programming to schedule \(J_{\text{large}}\) with makespan \((1 + \epsilon) \cdot T\).**

- Let \(b\) be the smallest integer with \(1/b \leq \epsilon\). Define processing times \(p'_i = \left\lfloor \frac{p_i b^2}{T} \right\rfloor \cdot \frac{T}{b^2}\)

\[\Rightarrow \text{Every } p'_i = \alpha \cdot \frac{T}{b^2} \text{ for } \alpha = b, b + 1, \ldots, b^2\]

- Let \(C\) be all \((s_b, s_{b+1}, \ldots, s_{b^2})\) with \(\sum_{j=b}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T\).

- Let \(f(n_b, n_{b+1}, \ldots, n_{b^2})\) be the minimum number of machines required to schedule all jobs with makespan \(\leq T\):

\[
f(0, 0, \ldots, 0) = 0 \]
\[
f(n_b, n_{b+1}, \ldots, n_{b^2}) = 1 + \min_{(s_b, s_{b+1}, \ldots, s_{b^2}) \in C} f(n_b - s_b, n_{b+1} - s_{b+1}, \ldots, n_{b^2} - s_{b^2}).
\]

- **Number of table entries is at most** \(n^{b^2}\), hence filling all entries takes \(n^{O(b^2)}\)

- **If** \(f(n_b, n_{b+1}, \ldots, n_{b^2}) \leq m\) (for the jobs with \(p'\)), **then return yes**, otherwise **no**.

- **As every machine is assigned at most** \(b\) jobs \((p'_i \geq \frac{T}{b})\) and the makespan is \(\leq T\),

\[
C_{\text{max}} \leq T + b \cdot \max_{i \in J_{\text{large}}} (p_i - p'_i)
\]
\[
\leq T + b \cdot \frac{T}{b^2}
\]
Proof of Key Lemma (non-examinable)

**Use Dynamic Programming** to schedule $J_{\text{large}}$ with makespan $(1 + \epsilon) \cdot T$.

- Let $b$ be the smallest integer with $1/b \leq \epsilon$. Define processing times $p'_i = \left\lfloor \frac{p_i b^2}{T} \right\rfloor \cdot \frac{T}{b^2}$.

$\Rightarrow$ Every $p'_i = \alpha \cdot \frac{T}{b^2}$ for $\alpha = b, b+1, \ldots, b^2$.

- Let $C$ be all $(s_b, s_{b+1}, \ldots, s_{b^2})$ with $\sum_{j=b}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T$.

- Let $f(n_b, n_{b+1}, \ldots, n_{b^2})$ be the minimum number of machines required to schedule all jobs with makespan $\leq T$:

\[
f(0, 0, \ldots, 0) = 0
\]

\[
f(n_b, n_{b+1}, \ldots, n_{b^2}) = 1 + \min_{(s_b, s_{b+1}, \ldots, s_{b^2}) \in C} f(n_b - s_b, n_{b+1} - s_{b+1}, \ldots, n_{b^2} - s_{b^2}).
\]

- Number of table entries is at most $n^{b^2}$, hence filling all entries takes $n^{O(b^2)}$.

- If $f(n_b, n_{b+1}, \ldots, n_{b^2}) \leq m$ (for the jobs with $p'$), then return yes, otherwise no.

- As every machine is assigned at most $b$ jobs ($p'_i \geq \frac{T}{b}$) and the makespan is $\leq T$,

\[
C_{\text{max}} \leq T + b \cdot \max_{i \in J_{\text{large}}} (p_i - p'_i)
\]

\[
\leq T + b \cdot \frac{T}{b^2} \leq (1 + \epsilon) \cdot T.
\]