# V. Approximation Algorithms via Exact Algorithms

Thomas Sauerwald

Easter 2018

UNIVERSITY OF CAMBRIDGE

# Outline

The Subset-Sum Problem

Parallel Machine Scheduling

# The Subset-Sum Problem

**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

**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

**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 $\sum_{i:\ x_i \in S'} x_i \leq t$.

$t = 13$ tons

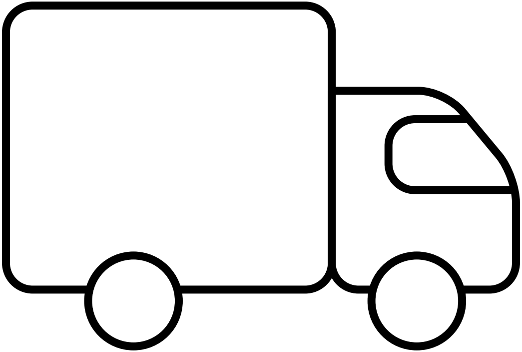
$x_1 = 10$

$x_2 = 4$

$x_3 = 5$

$x_4 = 6$

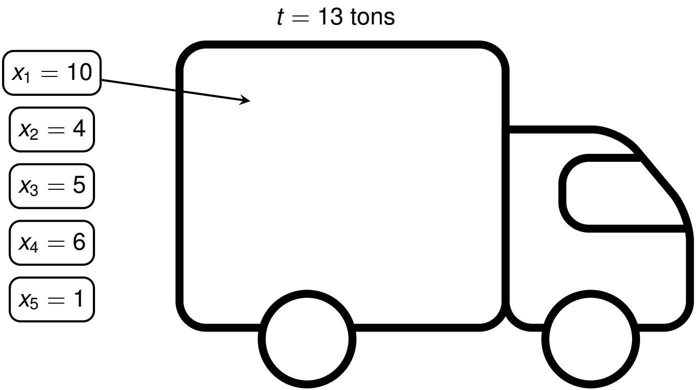
$x_5 = 1$



# The Subset-Sum Problem

**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

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 $\sum_{i:\, x_i \in S'} x_i \leq t$.

$t = 13$ tons

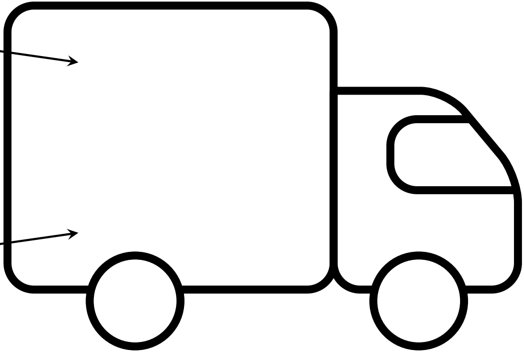
$x_1 = 10$

$x_2 = 4$

$x_3 = 5$

$x_4 = 6$

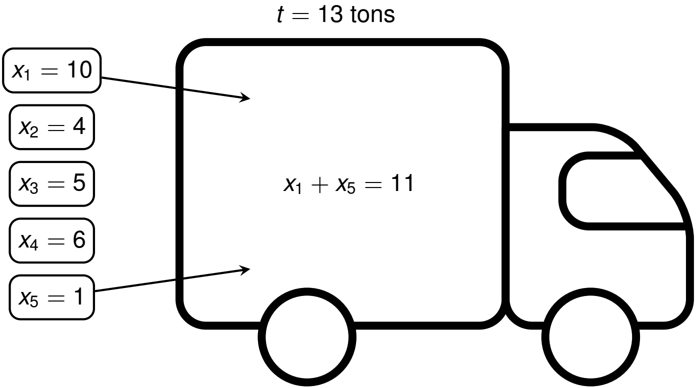
$x_5 = 1$



## The Subset-Sum Problem

The Subset-Sum Problem

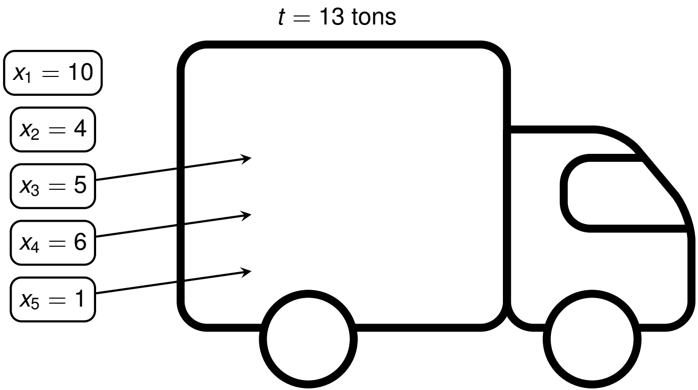
- Given: Set of positive integers $S = \{x_1, x_2, \ldots, x_n\}$ and positive integer $t$
- Goal: Find a subset $S' \subseteq S$ which maximizes $\sum_{i:\, x_i \in S'} x_i \leq t$.



# The Subset-Sum Problem

**The Subset-Sum Problem**

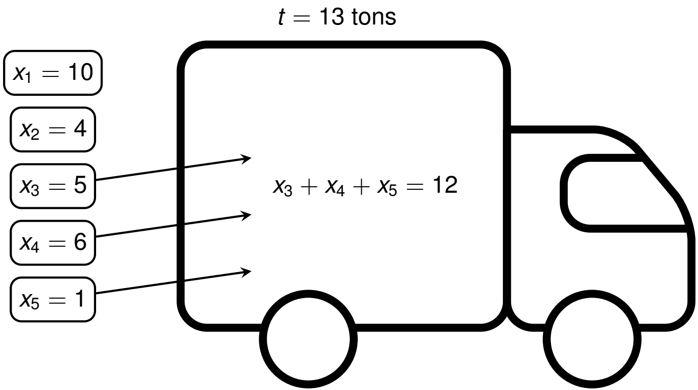
- Given: Set of positive integers $S = \{x_1, x_2, \ldots, x_n\}$ and positive integer $t$
- Goal: Find a subset $S' \subseteq S$ which maximizes $\sum_{i:\, x_i \in S'} x_i \leq t$.



# The Subset-Sum Problem

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 $\sum_{i:\, x_i \in S'} x_i \leq t$.



# An Exact (Exponential-Time) Algorithm

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

# An Exact (Exponential-Time) Algorithm

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

$\text{EXACT-SUBSET-SUM}(S, t)$

```
n = |S|
L_0 = ⟨0⟩
for i = 1 to n
    L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
    remove from L_i every element that is greater than t
return the largest element in L_n
```

# An Exact (Exponential-Time) Algorithm

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

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

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

# An Exact (Exponential-Time) Algorithm

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

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

```
n = |S|
L_0 = ⟨0⟩
for i = 1 to n
    L_i = Merge-Lists(L_{i-1}, L_{i-1} + x_i)
    remove from L_i every element that is greater than t
return the largest element in L_n
```

Returns the merged list (in sorted order and without duplicates)

$S + x := \{s + x : s \in S\}$

# An Exact (Exponential-Time) Algorithm

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

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

Returns the merged list (in sorted order and without duplicates)

$S + x := \{s + x : s \in S\}$

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

```
n = |S|
L_0 = ⟨0⟩
for i = 1 to n
    L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
    remove from L_i every element that is greater than t
return the largest element in L_n
```

# An Exact (Exponential-Time) Algorithm

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

```
EXACT-SUBSET-SUM(S, t)
n = |S|
L_0 = ⟨0⟩
for i = 1 to n
    L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
    remove from L_i every element that is greater than t
return the largest element in L_n
```

Example:

# An Exact (Exponential-Time) Algorithm

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

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

```
n = |S|
L_0 = ⟨0⟩
for i = 1 to n
    L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
    remove from L_i every element that is greater than t
return the largest element in L_n
```

Example:

- $S = \{1, 4, 5\}$, $t = 10$

# An Exact (Exponential-Time) Algorithm

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

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

$n = |S|$
$L_0 = \langle 0 \rangle$
**for** $i = 1$ **to** $n$
    $L_i =$ MERGE-LISTS$(L_{i-1}, L_{i-1} + x_i)$
    remove from $L_i$ every element that is greater than $t$
**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 Progamming: Compute bottom-up all possible sums $\leq t$

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

```
n = |S|
L_0 = ⟨0⟩
for i = 1 to n
    L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
    remove from L_i every element that is greater than t
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 Progamming:** Compute bottom-up all possible sums $\leq t$

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

$n = |S|$
$L_0 = \langle 0 \rangle$
**for** $i = 1$ **to** $n$
    $L_i =$ MERGE-LISTS$(L_{i-1}, L_{i-1} + x_i)$
    remove from $L_i$ every element that is greater than $t$
**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 Progamming: Compute bottom-up all possible sums $\leq t$

```
EXACT-SUBSET-SUM(S, t)
n = |S|
L_0 = ⟨0⟩
for i = 1 to n
    L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
    remove from L_i every element that is greater than t
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 Progamming: Compute bottom-up all possible sums $\leq t$

```
EXACT-SUBSET-SUM(S, t)
n = |S|
L_0 = ⟨0⟩
for i = 1 to n
    L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
    remove from L_i every element that is greater than t
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 Progamming: Compute bottom-up all possible sums $\leq t$

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

```
n = |S|
L_0 = ⟨0⟩
for i = 1 to n
    L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
    remove from L_i every element that is greater than t
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 Progamming: Compute bottom-up all possible sums $\leq t$

```
EXACT-SUBSET-SUM(S, t)
n = |S|
L_0 = ⟨0⟩
for i = 1 to n
    L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
    remove from L_i every element th
return the largest element in L_n
```

can be shown by induction on $n$

- **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 Progamming: Compute bottom-up all possible sums $\leq t$

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

```
n = |S|
L_0 = ⟨0⟩
for i = 1 to n
        L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
        remove from L_i every element that is greater than t
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 Progamming: Compute bottom-up all possible sums $\leq t$

```
EXACT-SUBSET-SUM(S, t)
n = |S|
L_0 = ⟨0⟩
for i = 1 to n
    L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
    remove from L_i every element that is greater than t
return the largest element in L_n
```

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

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

Example:

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

# An Exact (Exponential-Time) Algorithm

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

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

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

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

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

Better runtime if $t$ and/or $|L_i|$ are small.

Example:

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

# Towards a FPTAS

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

# 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 minimal 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 minimal 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 minimal 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 minimal sublist $L'$ so that for every $y \in L$: $\exists z \in L'$:

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

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

# 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 minimal sublist $L'$ so that for every $y \in L$: $\exists z \in L'$:

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

```
TRIM(L, δ)
let m be the length of L
L' = ⟨y_1⟩
last = y_1
for i = 2 to m
    if y_i > last · (1 + δ)        // y_i ≥ last because L is sorted
        append y_i onto the end of L'
        last = y_i
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 minimal sublist $L'$ so that for every $y \in L$: $\exists z \in L'$:

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

```
TRIM(L, δ)
let m be the length of L
L′ = ⟨y_1⟩
last = y_1
for i = 2 to m
    if y_i > last · (1 + δ)      // y_i ≥ last because L is sorted
        append y_i onto the end of L′
        last = y_i
return L′
```

TRIM works in time $\Theta(m)$, if $L$ is given in sorted order.

# Illustration of the Trim Operation

TRIM$(L, \delta)$

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

# Illustration of the Trim Operation

```
TRIM(L, δ)
let m be the length of L
L′ = ⟨y_1⟩
last = y_1
for i = 2 to m
    if y_i > last · (1 + δ)        // y_i ≥ last because L is sorted
        append y_i onto the end of L′
        last = y_i
return L′
```

$$\delta = 0.1$$

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

$$L' = \langle \rangle$$

TRIM$(L, \delta)$

let $m$ be the length of $L$<br>
$L' = \langle y_1 \rangle$<br>
$last = y_1$<br>
**for** $i = 2$ **to** $m$<br>
    **if** $y_i > last \cdot (1 + \delta)$ // $y_i \geq last$ because $L$ is sorted<br>
        append $y_i$ onto the end of $L'$<br>
        $last = y_i$<br>
**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)
let m be the length of L
L′ = ⟨y_1⟩
last = y_1
for i = 2 to m
    if y_i > last · (1 + δ)        // y_i ≥ last because L is sorted
        append y_i onto the end of L′
        last = y_i
return L′
```

$\delta = 0.1$

last

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

$L' = \langle 10 \rangle$

# Illustration of the Trim Operation

TRIM($L, \delta$)

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

$$\delta = 0.1$$

last

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

i

$$L' = \langle 10 \rangle$$

# Illustration of the Trim Operation

TRIM$(L, \delta)$

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

$$\delta = 0.1$$

last

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

i

$$L' = \langle 10 \rangle$$

# Illustration of the Trim Operation

```
TRIM(L, δ)
let m be the length of L
L′ = ⟨y_1⟩
last = y_1
for i = 2 to m
    if y_i > last · (1 + δ)      // y_i ≥ last because L is sorted
        append y_i onto the end of L′
        last = y_i
return L′
```

$\delta = 0.1$

last

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

i

$L' = \langle 10, 12 \rangle$

# Illustration of the Trim Operation

```
TRIM(L, δ)
let m be the length of L
L′ = ⟨y_1⟩
last = y_1
for i = 2 to m
    if y_i > last · (1 + δ)        // y_i ≥ last because L is sorted
        append y_i onto the end of L′
        last = y_i
return L′
```

$\delta = 0.1$

last

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

i

$L' = \langle 10, 12 \rangle$

# Illustration of the Trim Operation

TRIM$(L, \delta)$

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

$$\delta = 0.1$$

last

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

i

$$L' = \langle 10, 12 \rangle$$

# Illustration of the Trim Operation

TRIM$(L, \delta)$

```
let m be the length of L
L' = ⟨y_1⟩
last = y_1
for i = 2 to m
    if y_i > last · (1 + δ)        // y_i ≥ last because L is sorted
        append y_i onto the end of L'
        last = y_i
return L'
```

$$\delta = 0.1$$

last

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

i

$$L' = \langle 10, 12, 15 \rangle$$

## Illustration of the Trim Operation

```
TRIM(L, δ)
let m be the length of L
L′ = ⟨y_1⟩
last = y_1
for i = 2 to m
    if y_i > last · (1 + δ)        // y_i ≥ last because L is sorted
        append y_i onto the end of L′
        last = y_i
return L′
```

$\delta = 0.1$

last

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

i

$L' = \langle 10, 12, 15 \rangle$

# Illustration of the Trim Operation

```
TRIM(L, δ)
let m be the length of L
L′ = ⟨y_1⟩
last = y_1
for i = 2 to m
    if y_i > last · (1 + δ)        // y_i ≥ last because L is sorted
        append y_i onto the end of L′
        last = y_i
return L′
```

$\delta = 0.1$

last

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

i

$$L' = \langle 10, 12, 15 \rangle$$

# Illustration of the Trim Operation

```
TRIM(L, δ)
let m be the length of L
L′ = ⟨y_1⟩
last = y_1
for i = 2 to m
    if y_i > last · (1 + δ)      // y_i ≥ last because L is sorted
        append y_i onto the end of L′
        last = y_i
return L′
```

$\delta = 0.1$

last

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

i

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

# Illustration of the Trim Operation

TRIM$(L, \delta)$

```
let m be the length of L
L' = ⟨y_1⟩
last = y_1
for i = 2 to m
    if y_i > last · (1 + δ)        // y_i ≥ last because L is sorted
        append y_i onto the end of L'
        last = y_i
return L'
```

$\delta = 0.1$

last

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

i

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

# Illustration of the Trim Operation

TRIM$(L, \delta)$

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

$$\delta = 0.1$$

$$L = \langle 10, 11, 12, 15, \overset{\substack{\text{last}\\ \downarrow}}{20}, \underset{\substack{\uparrow \\ \text{i}}}{21}, 22, 23, 24, 29 \rangle$$

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

# Illustration of the Trim Operation

TRIM$(L, \delta)$

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

$\delta = 0.1$

last

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

i

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

# Illustration of the Trim Operation

TRIM$(L, \delta)$

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

$$\delta = 0.1$$

last

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

i

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

# Illustration of the Trim Operation

TRIM$(L, \delta)$

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

$\delta = 0.1$

last

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

i

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

# Illustration of the Trim Operation

TRIM$(L, \delta)$

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

$$\delta = 0.1$$

last

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

i

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

# Illustration of the Trim Operation

```
TRIM(L, δ)
let m be the length of L
L′ = ⟨y_1⟩
last = y_1
for i = 2 to m
    if y_i > last · (1 + δ)        // y_i ≥ last because L is sorted
        append y_i onto the end of L′
        last = y_i
return L′
```

$\delta = 0.1$

last

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

i

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

# Illustration of the Trim Operation

TRIM$(L, \delta)$

```
let m be the length of L
L' = ⟨y_1⟩
last = y_1
for i = 2 to m
    if y_i > last · (1 + δ)        // y_i ≥ last because L is sorted
        append y_i onto the end of L'
        last = y_i
return L'
```

$\delta = 0.1$

last

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

i

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

# Illustration of the Trim Operation

TRIM$(L, \delta)$

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

$$\delta = 0.1$$

last

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

i

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

```
TRIM(L, δ)
let m be the length of L
L′ = ⟨y_1⟩
last = y_1
for i = 2 to m
    if y_i > last · (1 + δ)        // y_i ≥ last because L is sorted
        append y_i onto the end of L′
        last = y_i
return L′
```

$\delta = 0.1$

last

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

i

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

# The FPTAS

```
APPROX-SUBSET-SUM(S, t, ε)
n = |S|
L_0 = ⟨0⟩
for i = 1 to n
    L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
    L_i = TRIM(L_i, ε/2n)
    remove from L_i every element that is greater than t
let z* be the largest value in L_n
return z*
```

# The FPTAS

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

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

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

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

# The FPTAS

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

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

Repeated application of TRIM to make sure $L_i$'s remain short.

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

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

# The FPTAS

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

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

Repeated application of TRIM to make sure $L_i$'s remain short.

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

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

- We must bound the inaccuracy introduced by repeated trimming

# The FPTAS

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

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

Repeated application of TRIM to make sure $L_i$'s remain short.

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

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

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

# The FPTAS

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

```
n = |S|
L_0 = ⟨0⟩
for i = 1 to n
    L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
    L_i = TRIM(L_i, ε/2n)
    remove from L_i every element that is greater than t
let z* be the largest value in L_n
return z*
```

Repeated application of TRIM to make sure $L_i$'s remain short.

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

```
n = |S|
L_0 = ⟨0⟩
for i = 1 to n
    L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
    remove from L_i every element that is greater than t
return the largest element in L_n
```

- 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

```
APPROX-SUBSET-SUM(S, t, ϵ)
n = |S|
L_0 = ⟨0⟩
for i = 1 to n
    L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
    L_i = TRIM(L_i, ϵ/2n)
    remove from L_i every element that is greater than t
let z* be the largest value in L_n
return z*
```

# Running through an Example

```
APPROX-SUBSET-SUM(S, t, ε)
n = |S|
L_0 = ⟨0⟩
for i = 1 to n
    L_i = MERGE-LISTS(L_{i−1}, L_{i−1} + x_i)
    L_i = TRIM(L_i, ε/2n)
    remove from L_i every element that is greater than t
let z* be the largest value in L_n
return z*
```

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

# Running through an Example

```
APPROX-SUBSET-SUM(S, t, ε)
n = |S|
L_0 = ⟨0⟩
for i = 1 to n
    L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
    L_i = TRIM(L_i, ε/2n)
    remove from L_i every element that is greater than t
let z* be the largest value in L_n
return z*
```

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

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

# Running through an Example

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

```
1  n = |S|
2  L_0 = ⟨0⟩
3  for i = 1 to n
4      L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
5      L_i = TRIM(L_i, ε/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: $S = \langle 104, 102, 201, 101 \rangle$, $t = 308$, $\epsilon = 0.4$

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

- line 2: $L_0 = \langle 0 \rangle$

# Running through an Example

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

```
1  n = |S|
2  L_0 = ⟨0⟩
3  for i = 1 to n
4      L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
5      L_i = TRIM(L_i, ε/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: $S = \langle 104, 102, 201, 101 \rangle$, $t = 308$, $\epsilon = 0.4$

$\Rightarrow$ 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

```
APPROX-SUBSET-SUM(S, t, ε)
1  n = |S|
2  L_0 = ⟨0⟩
3  for i = 1 to n
4      L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
5      L_i = TRIM(L_i, ε/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: $S = \langle 104, 102, 201, 101 \rangle$, $t = 308$, $\epsilon = 0.4$

$\Rightarrow$ 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: $L_1 = \langle 0, 104 \rangle$

# Running through an Example

```
APPROX-SUBSET-SUM(S, t, ε)
1  n = |S|
2  L_0 = ⟨0⟩
3  for i = 1 to n
4      L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
5      L_i = TRIM(L_i, ε/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: $S = \langle 104, 102, 201, 101 \rangle$, $t = 308$, $\epsilon = 0.4$

$\Rightarrow$ 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

```
APPROX-SUBSET-SUM(S, t, ε)
1  n = |S|
2  L_0 = ⟨0⟩
3  for i = 1 to n
4       L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
5       L_i = TRIM(L_i, ε/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: $S = \langle 104, 102, 201, 101 \rangle$, $t = 308$, $\epsilon = 0.4$

$\Rightarrow$ 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

```
APPROX-SUBSET-SUM(S, t, ε)
1  n = |S|
2  L_0 = ⟨0⟩
3  for i = 1 to n
4      L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
5      L_i = TRIM(L_i, ε/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: $S = \langle 104, 102, 201, 101 \rangle$, $t = 308$, $\epsilon = 0.4$

$\Rightarrow$ 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: $L_2 = \langle 0, 102, 206 \rangle$

# Running through an Example

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

```
1  n = |S|
2  L_0 = ⟨0⟩
3  for i = 1 to n
4      L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
5      L_i = TRIM(L_i, ε/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: $S = \langle 104, 102, 201, 101 \rangle$, $t = 308$, $\epsilon = 0.4$

$\Rightarrow$ 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

```
APPROX-SUBSET-SUM(S, t, ε)
1  n = |S|
2  L_0 = ⟨0⟩
3  for i = 1 to n
4      L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
5      L_i = TRIM(L_i, ε/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: $S = \langle 104, 102, 201, 101 \rangle$, $t = 308$, $\epsilon = 0.4$

$\Rightarrow$ 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

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

```
1  n = |S|
2  L_0 = ⟨0⟩
3  for i = 1 to n
4      L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
5      L_i = TRIM(L_i, ε/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: $S = \langle 104, 102, 201, 101 \rangle$, $t = 308$, $\epsilon = 0.4$

$\Rightarrow$ 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

```
APPROX-SUBSET-SUM(S, t, ε)
1  n = |S|
2  L_0 = ⟨0⟩
3  for i = 1 to n
4      L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
5      L_i = TRIM(L_i, ε/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: $L_3 = \langle 0, 102, 201, 303 \rangle$

# Running through an Example

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

```
1  n = |S|
2  L_0 = ⟨0⟩
3  for i = 1 to n
4      L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
5      L_i = TRIM(L_i, ε/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: $S = \langle 104, 102, 201, 101 \rangle$, $t = 308$, $\epsilon = 0.4$

$\Rightarrow$ 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

```
APPROX-SUBSET-SUM(S, t, ε)
1  n = |S|
2  L_0 = ⟨0⟩
3  for i = 1 to n
4      L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
5      L_i = TRIM(L_i, ε/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: $S = \langle 104, 102, 201, 101 \rangle$, $t = 308$, $\epsilon = 0.4$

$\Rightarrow$ 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

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

```
1  n = |S|
2  L_0 = ⟨0⟩
3  for i = 1 to n
4      L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
5      L_i = TRIM(L_i, ϵ/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: $S = \langle 104, 102, 201, 101 \rangle$, $t = 308$, $\epsilon = 0.4$

$\Rightarrow$ 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

```
APPROX-SUBSET-SUM(S, t, ε)
1  n = |S|
2  L_0 = ⟨0⟩
3  for i = 1 to n
4      L_i = MERGE-LISTS(L_{i−1}, L_{i−1} + x_i)
5      L_i = TRIM(L_i, ε/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: $S = \langle 104, 102, 201, 101 \rangle$, $t = 308$, $\epsilon = 0.4$

$\Rightarrow$ 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: $L_4 = \langle 0, 101, 201, 302 \rangle$

Returned solution $z^* = 302$, which is 2% within the optimum $307 = 104 + 102 + 101$

# Analysis of APPROX-SUBSET-SUM

**Theorem 35.8**

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

Proof (Approximation Ratio):

## 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 ✓

# 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

# 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.:

# 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.:

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

# 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.:

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

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.:

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

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.:

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

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.:

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

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

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.:

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

Can be shown by induction on $i$

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

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

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

Can be shown by induction on $i$

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

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

$$\frac{y^*}{z} \leq e^{\epsilon/2}$$

# 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 ✓
- 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.:

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

Can be shown by induction on $i$

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

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

$$\frac{y^*}{z} \leq e^{\epsilon/2}$$

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 ✓
- 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.:

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

Can be shown by induction on $i$

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

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

$$\frac{y^*}{z} \leq e^{\epsilon/2}$$

Taylor approximation of $e$

$$\leq 1+\epsilon/2+(\epsilon/2)^2$$

# 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 ✓
- 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.:

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

Can be shown by induction on $i$

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

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

$$\frac{y^*}{z} \leq e^{\epsilon/2}$$

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 $|L_i|$)

# 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.

# 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,

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

# 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 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,

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

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 $\lfloor \log_{1+\epsilon/(2n)} t \rfloor$ additional values. Hence,

$$\begin{aligned}\log_{1+\epsilon/(2n)} t + 2 &= \frac{\ln t}{\ln(1+\epsilon/(2n))} + 2 \\ &\leq \frac{2n(1+\epsilon/(2n))\ln t}{\epsilon} + 2 \\ &< \frac{3n \ln t}{\epsilon} + 2.\end{aligned}$$

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 $\lfloor \log_{1+\epsilon/(2n)} t \rfloor$ additional values. Hence,

$$\begin{aligned}\log_{1+\epsilon/(2n)} t + 2 &= \frac{\ln t}{\ln(1+\epsilon/(2n))} + 2 \\ &\leq \frac{2n(1+\epsilon/(2n)) \ln t}{\epsilon} + 2 \\ &< \frac{3n \ln t}{\epsilon} + 2.\end{aligned}$$

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

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

## 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,

$$\begin{aligned}
\log_{1+\epsilon/(2n)} t + 2 &= \frac{\ln t}{\ln(1+\epsilon/(2n))} + 2 \\
&\leq \frac{2n(1+\epsilon/(2n))\ln t}{\epsilon} + 2 \\
&< \frac{3n \ln t}{\epsilon} + 2.
\end{aligned}$$

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

- 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 $\sum_{i:\, x_i \in S'} x_i \leq t$.

## 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.

A more general problem than Subset-Sum

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.

A more general problem than Subset-Sum

**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.

## 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.

A more general problem than Subset-Sum

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 $\sum_{i \in S'} w_i \leq t$

Algorithm very similar to APPROX-SUBSET-SUM

Theorem

There is a FPTAS for the Knapsack problem.

# Outline

The Subset-Sum Problem

Parallel Machine Scheduling

# 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 $M_1, M_2, \ldots, M_m$

# 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 $C_{\max} = \max_{1 \leq j \leq n} C_j$, 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 $C_{\max} = \max_{1 \leq j \leq n} C_j$, where $C_k$ is the completion time of job $J_k$.

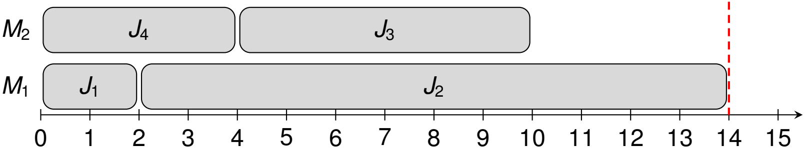
- $J_1$: $p_1 = 2$
- $J_2$: $p_2 = 12$
- $J_3$: $p_3 = 6$
- $J_4$: $p_4 = 4$

# 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 $C_{\max} = \max_{1 \le j \le n} C_j$, where $C_k$ is the completion time of job $J_k$.

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

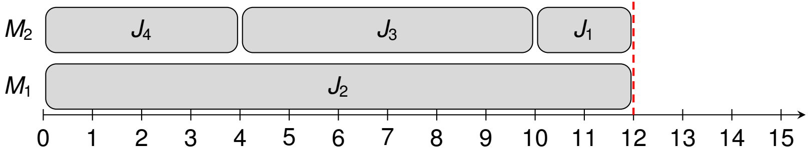


# Parallel Machine Scheduling

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 $C_{\max} = \max_{1 \leq j \leq n} C_j$, where $C_k$ is the completion time of job $J_k$.

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



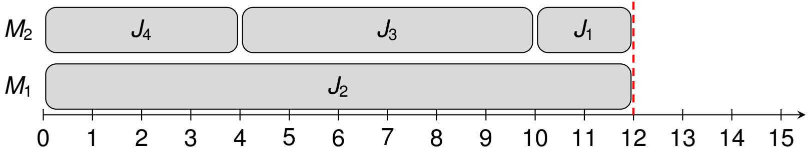
# Parallel Machine Scheduling

**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 $C_{\max} = \max_{1 \leq j \leq n} C_j$, 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 $i$.



## NP-Completeness of Parallel Machine Scheduling

**Lemma**

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

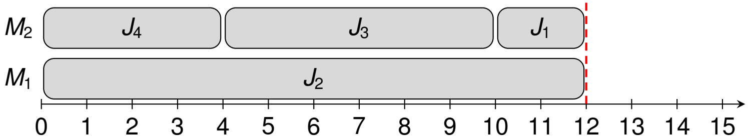
Proof Idea: Polynomial time reduction from NUMBER-PARTITIONING.

# 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.

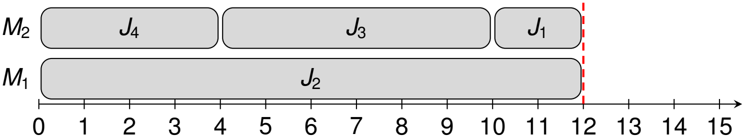


# NP-Completeness of Parallel Machine Scheduling

**Lemma**

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

Proof Idea: Polynomial time reduction from NUMBER-PARTITIONING.




LIST SCHEDULING$(J_1, J_2, \ldots, J_n, m)$

**while** there exists an unassigned job

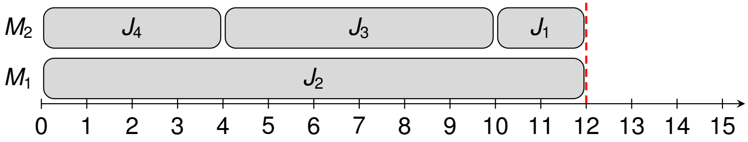
  Schedule job on the machine with the least load

# 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 [CLRS]:
Whenever a machine is idle, schedule any job that has not yet been scheduled.

LIST SCHEDULING($J_1, J_2, \ldots, J_n, m$)

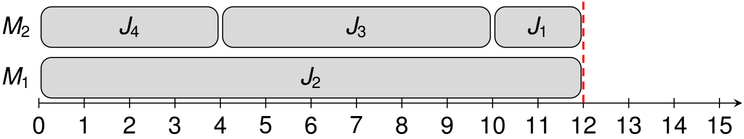
**while** there exists an unassigned job
    Schedule job on the machine with the least load

# 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 [CLRS]:
Whenever a machine is idle, schedule any job that has not yet been scheduled.

LIST SCHEDULING$(J_1, J_2, \ldots, J_n, m)$

**while** there exists an unassigned job
    Schedule job on the machine with the least load

How good is this most basic Greedy Approach?

# List Scheduling Analysis (Observations)

# 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,

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

# 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,

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

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

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

## 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,

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

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

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

Proof:

## 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,

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

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

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

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,

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

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

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

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 $\frac{1}{m} \cdot \sum_{k=1}^{n} p_k$ $\square$

# List Scheduling Analysis (Final Step)

**Ex 35-5 d. (Graham 1966)**

For the schedule returned by the greedy algorithm it holds that

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

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

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

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

Proof:

# List Scheduling Analysis (Final Step)

Ex 35-5 d. (Graham 1966)

For the schedule returned by the greedy algorithm it holds that

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

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

Proof:

- Let $J_i$ be the last job scheduled on machine $M_j$ with $C_{\max} = C_j$

## List Scheduling Analysis (Final Step)

Ex 35-5 d. (Graham 1966)

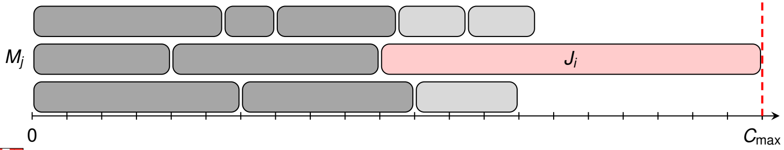
For the schedule returned by the greedy algorithm it holds that

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

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

Proof:

- Let $J_i$ be the last job scheduled on machine $M_j$ with $C_{\max} = C_j$



# List Scheduling Analysis (Final Step)

**Ex 35-5 d. (Graham 1966)**

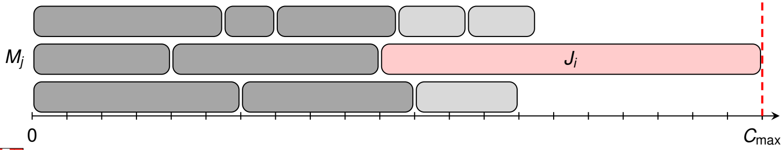
For the schedule returned by the greedy algorithm it holds that

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

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

Proof:

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



# List Scheduling Analysis (Final Step)

Ex 35-5 d. (Graham 1966)

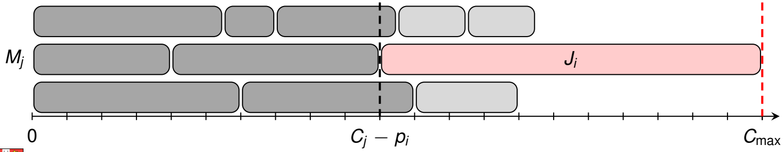
For the schedule returned by the greedy algorithm it holds that

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

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

Proof:

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



# List Scheduling Analysis (Final Step)

Ex 35-5 d. (Graham 1966)

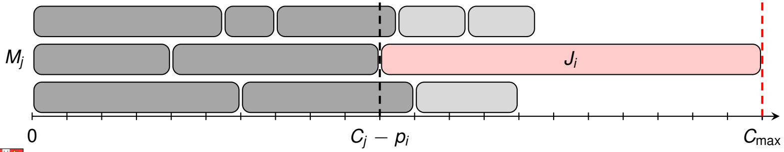
For the schedule returned by the greedy algorithm it holds that

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

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

Proof:

- Let $J_i$ be the last job scheduled on machine $M_j$ with $C_{\max} = C_j$
- 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:



## List Scheduling Analysis (Final Step)

**Ex 35-5 d. (Graham 1966)**

For the schedule returned by the greedy algorithm it holds that

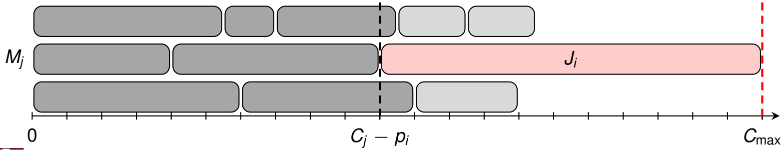
$$C_{\max} \leq \frac{1}{m} \sum_{k=1}^{n} p_k + \max_{1 \leq k \leq n} p_k.$$

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

Proof:

- Let $J_i$ be the last job scheduled on machine $M_j$ with $C_{\max} = C_j$
- 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:

$$C_j - p_i \leq \frac{1}{m} \sum_{k=1}^{m} C_k$$



# List Scheduling Analysis (Final Step)

Ex 35-5 d. (Graham 1966)

For the schedule returned by the greedy algorithm it holds that

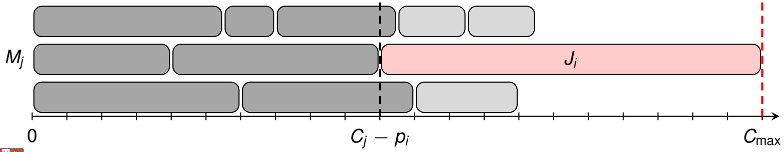
$$C_{\max} \leq \frac{1}{m}\sum_{k=1}^{n} p_k + \max_{1\leq k\leq n} p_k.$$

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

Proof:

- Let $J_i$ be the last job scheduled on machine $M_j$ with $C_{\max} = C_j$
- 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

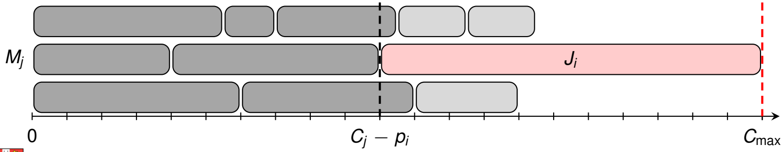
$$C_{\max} \leq \frac{1}{m} \sum_{k=1}^{n} p_k + \max_{1 \leq k \leq n} p_k.$$

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

Proof:

- Let $J_i$ be the last job scheduled on machine $M_j$ with $C_{\max} = C_j$
- 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:

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



# List Scheduling Analysis (Final Step)

**Ex 35-5 d. (Graham 1966)**

For the schedule returned by the greedy algorithm it holds that

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

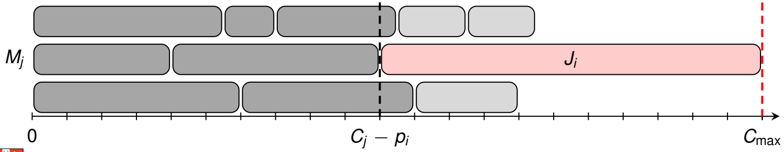
Hence list scheduling is a poly-time 2-approximation algorithm.

Proof:

- Let $J_i$ be the last job scheduled on machine $M_j$ with $C_{\max} = C_j$
- 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:

Using Ex 35-5 a. & b.

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



# List Scheduling Analysis (Final Step)

**Ex 35-5 d. (Graham 1966)**

For the schedule returned by the greedy algorithm it holds that

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

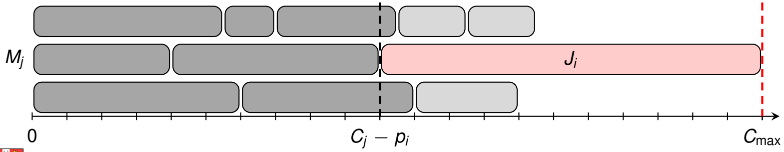
Hence list scheduling is a poly-time 2-approximation algorithm.

Proof:

- Let $J_i$ be the last job scheduled on machine $M_j$ with $C_{\max} = C_j$
- 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:

Using Ex 35-5 a. & b.

$$C_j - p_i \leq \frac{1}{m} \sum_{k=1}^{m} C_k = \frac{1}{m} \sum_{k=1}^{n} p_k \quad \Rightarrow \quad C_j \leq \frac{1}{m} \sum_{k=1}^{n} p_k + \max_{1 \leq k \leq n} p_k \leq 2 \cdot C^*_{\max}$$



## Improving Greedy

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

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: $\quad C_i = 0$
4: $\quad S_i = \emptyset$
5: **end for**
6: **for** $j = 1$ to $n$
7: $\quad i = \mathrm{argmin}_{1 \le k \le m} C_k$
8: $\quad S_i = S_i \cup \{j\}, C_i = C_i + p_j$
9: **end for**
10: **return** $S_1, \ldots, S_m$

# 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$(J_1, J_2, \ldots, J_n, m)$

Sort jobs decreasingly in their processing times
**for** $i = 1$ to $m$
  $C_i = 0$
  $S_i = \emptyset$
**end for**
**for** $j = 1$ to $n$
  $i = \text{argmin}_{1 \leq k \leq m} C_k$
  $S_i = S_i \cup \{j\}$, $C_i = C_i + p_j$
**end for**
**return** $S_1, \ldots, S_m$

Runtime:

# 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$(J_1, J_2, \ldots, J_n, m)$

Sort jobs decreasingly in their processing times
**for** $i = 1$ to $m$
    $C_i = 0$
    $S_i = \emptyset$
**end for**
**for** $j = 1$ to $n$
    $i = \text{argmin}_{1 \le k \le m} C_k$
    $S_i = S_i \cup \{j\}$, $C_i = C_i + p_j$
**end for**
**return** $S_1, \ldots, S_m$

Runtime:

- $O(n \log n)$ for sorting

# 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$(J_1, J_2, \ldots, J_n, m)$

```
Sort jobs decreasingly in their processing times
for i = 1 to m
    C_i = 0
    S_i = ∅
end for
for j = 1 to n
    i = argmin_{1≤k≤m} C_k
    S_i = S_i ∪ {j}, C_i = C_i + p_j
end for
return S_1, ..., 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).

## Analysis of Improved Greedy

**Graham 1966**

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

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 $C^*_{\max} \geq 2 \cdot p_{m+1}$.

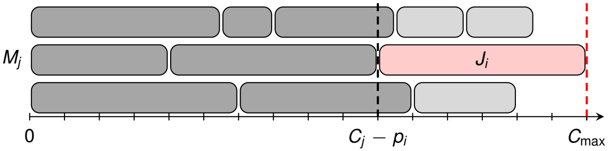
# Analysis of Improved Greedy

**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 $C^*_{\max} \geq 2 \cdot p_{m+1}$.
- As in the analysis for list scheduling



# Analysis of Improved Greedy

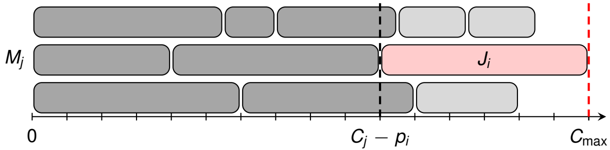
**Graham 1966**

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 $C^*_{\max} \geq 2 \cdot p_{m+1}$.
- As in the analysis for list scheduling, we have

$$C_{\max} = C_j = (C_j - p_i) + p_i$$



# Analysis of Improved Greedy

**Graham 1966**

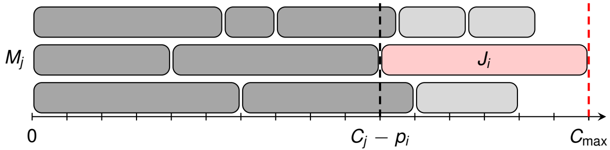
The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

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 $C^*_{\max} \geq 2 \cdot p_{m+1}$.
- As in the analysis for list scheduling, we have

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

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



# Analysis of Improved Greedy

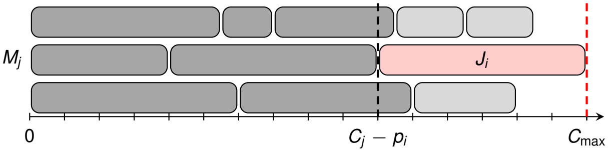
**Graham 1966**

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 $C^*_{\max} \geq 2 \cdot p_{m+1}$.
- As in the analysis for list scheduling, we have

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



# Tightness of the Bound for LPT

**Graham 1966**

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

# 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:

# 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

## 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
- $n = 2m + 1$ jobs of length $2m - 1, 2m - 2, \ldots, m$ and one job of length $m$

# Tightness of the Bound for LPT

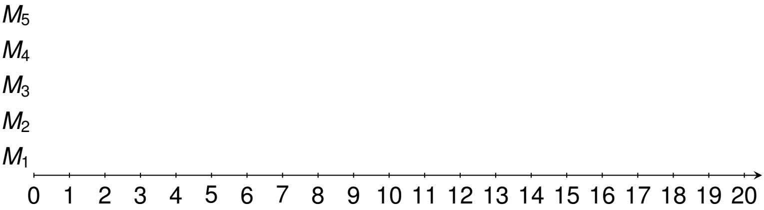
**Graham 1966**

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

Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :



## Tightness of the Bound for LPT

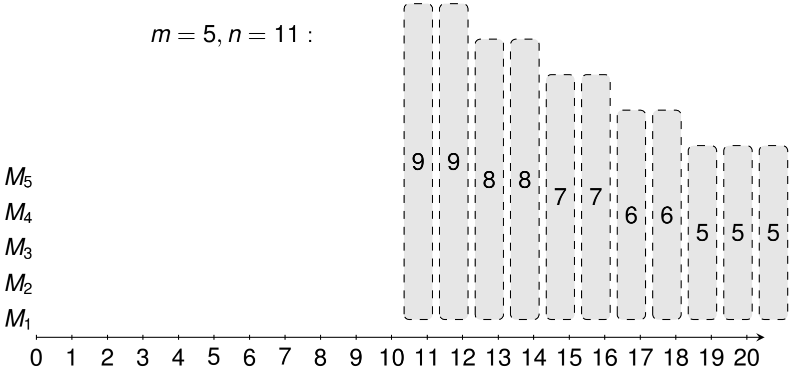
**Graham 1966**

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

Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :



## Tightness of the Bound for LPT

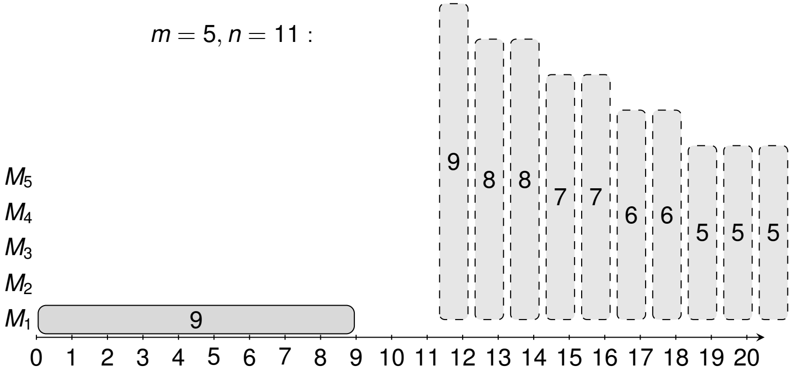
**Graham 1966**

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

Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :



# Tightness of the Bound for LPT

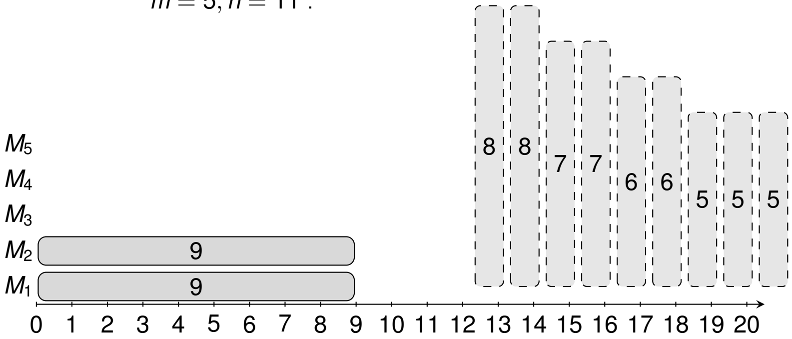
**Graham 1966**

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

Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :



# Tightness of the Bound for LPT

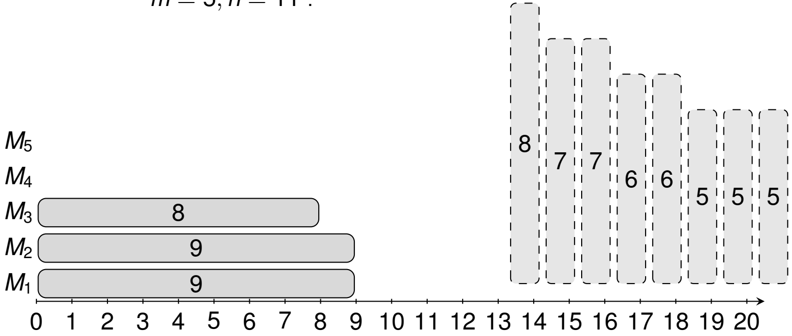
**Graham 1966**

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

Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :



# Tightness of the Bound for LPT

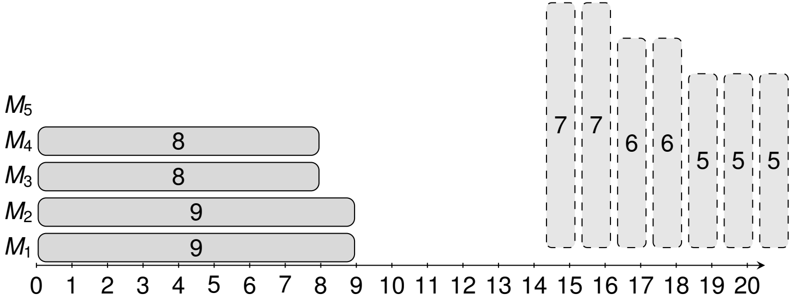
**Graham 1966**

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

Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :



## Tightness of the Bound for LPT

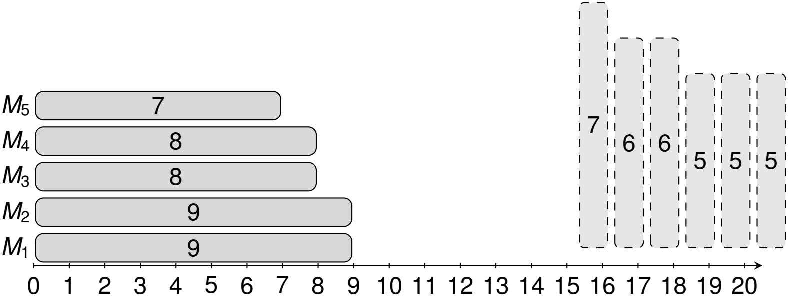
**Graham 1966**

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

Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :



## Tightness of the Bound for LPT

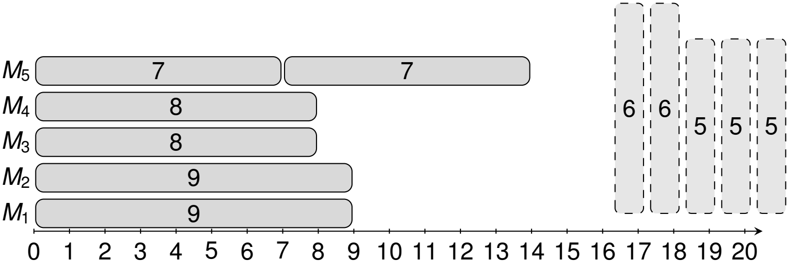
**Graham 1966**

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

Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :



# Tightness of the Bound for LPT

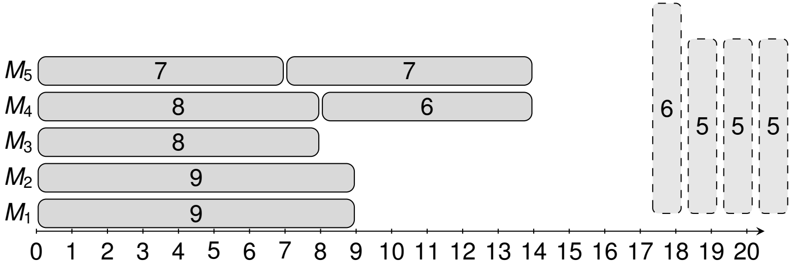
**Graham 1966**

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

Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :



## Tightness of the Bound for LPT

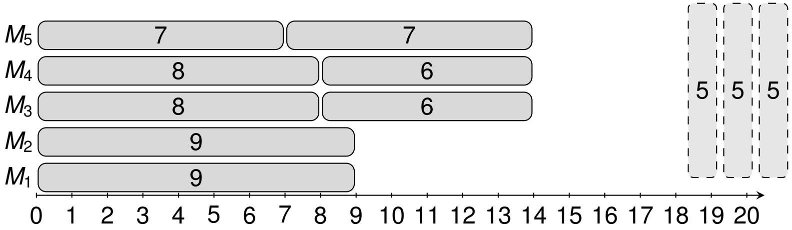
**Graham 1966**

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

Proof of an instance which shows tightness:

- $m$ machines
- $n = 2m + 1$ jobs of length $2m - 1, 2m - 2, \dots, m$ and one job of length $m$

$m = 5, n = 11$ :



# Tightness of the Bound for LPT

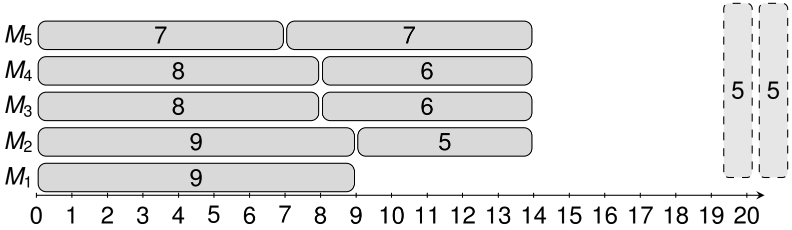
**Graham 1966**

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

Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :



# Tightness of the Bound for LPT

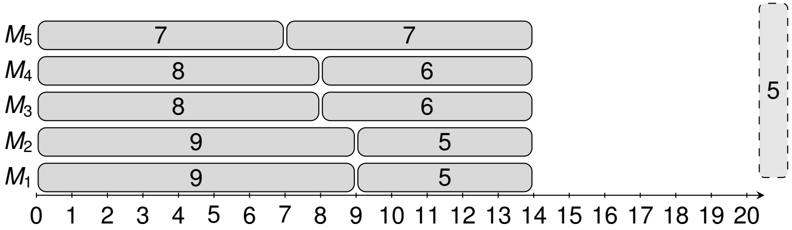
**Graham 1966**

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

Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :



# Tightness of the Bound for LPT

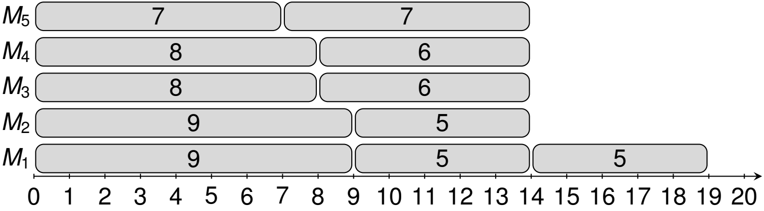
**Graham 1966**

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

Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :



# Tightness of the Bound for LPT

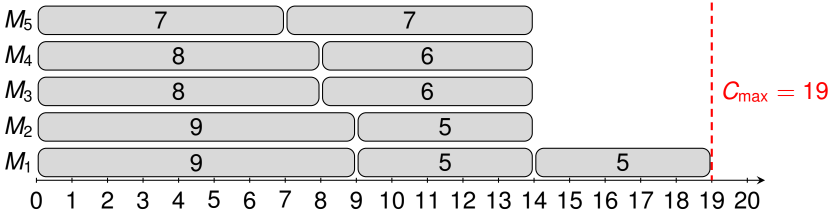
**Graham 1966**

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

Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :



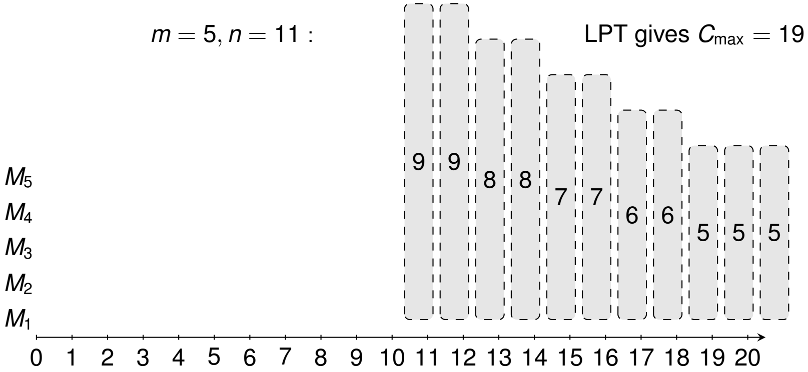
# Tightness of the Bound for LPT

**Graham 1966**

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

Proof of an instance which shows tightness:

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



# Tightness of the Bound for LPT

**Graham 1966**

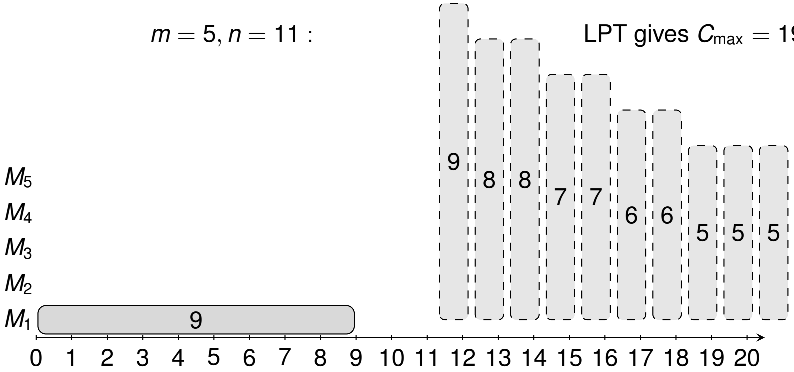
The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :

LPT gives $C_{\max} = 19$



# Tightness of the Bound for LPT

**Graham 1966**

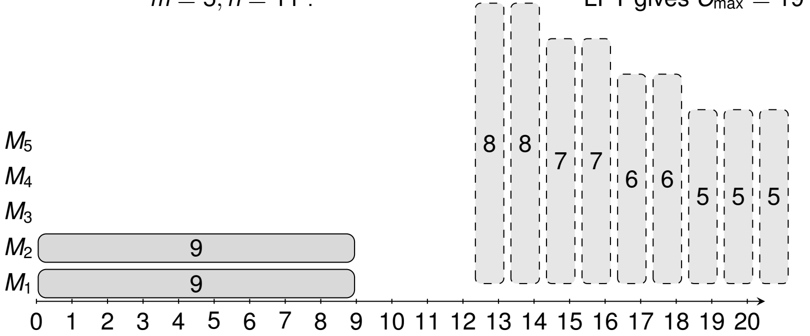
The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :

LPT gives $C_{\max} = 19$



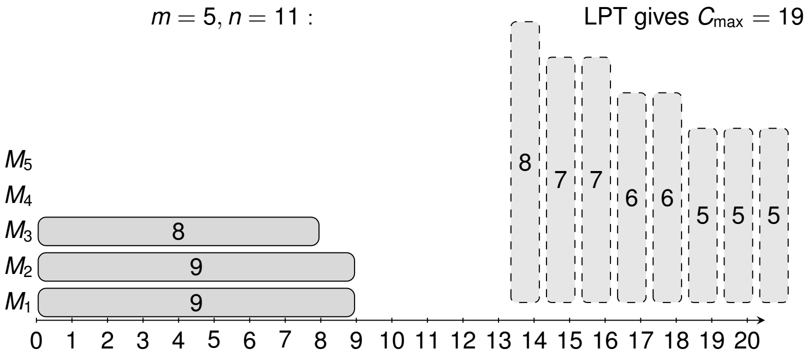
# Tightness of the Bound for LPT

**Graham 1966**

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

Proof of an instance which shows tightness:

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



# Tightness of the Bound for LPT

**Graham 1966**

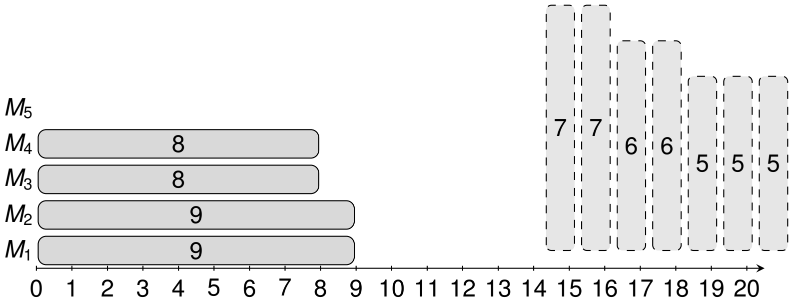
The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :

LPT gives $C_{\max} = 19$



# Tightness of the Bound for LPT

**Graham 1966**

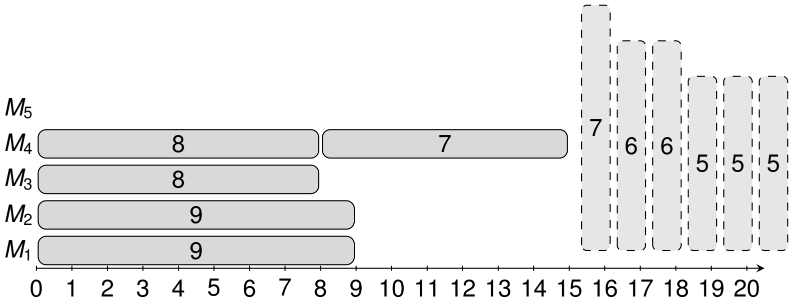
The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :

LPT gives $C_{\max} = 19$



# Tightness of the Bound for LPT

**Graham 1966**

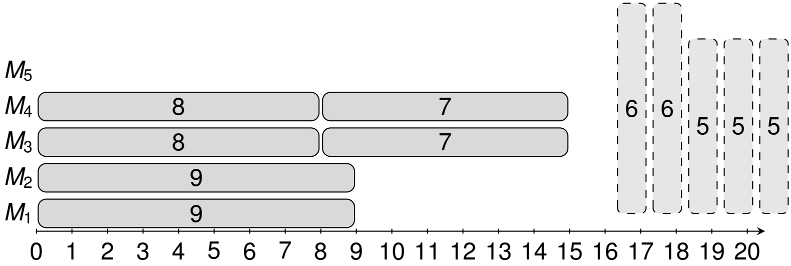
The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :

LPT gives $C_{\max} = 19$



# Tightness of the Bound for LPT

**Graham 1966**

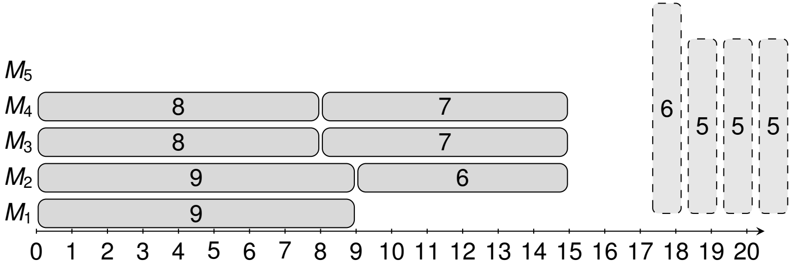
The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :

LPT gives $C_{\max} = 19$



# Tightness of the Bound for LPT

**Graham 1966**

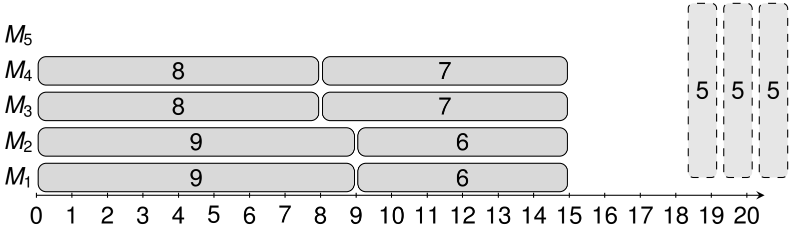
The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :

LPT gives $C_{\max} = 19$



## Tightness of the Bound for LPT

**Graham 1966**

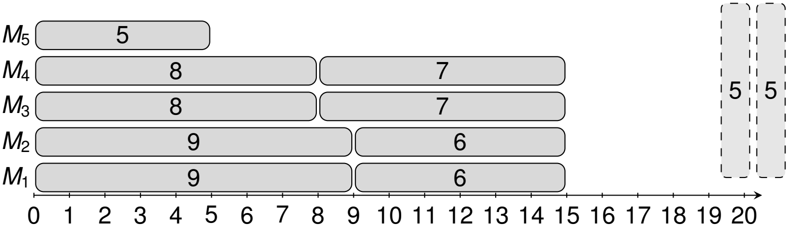
The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :

LPT gives $C_{\max} = 19$



# Tightness of the Bound for LPT

**Graham 1966**

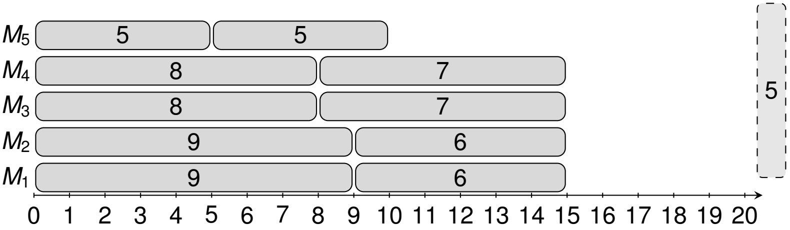
The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :

LPT gives $C_{\max} = 19$



**Graham 1966**

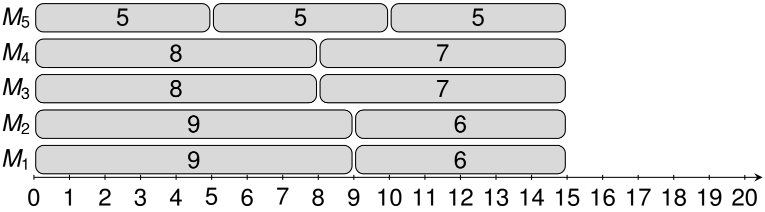
The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :

LPT gives $C_{\max} = 19$



# Tightness of the Bound for LPT

**Graham 1966**

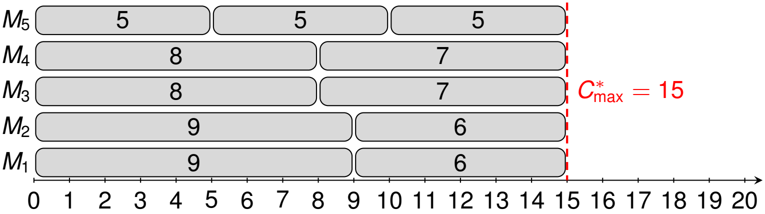
The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :

LPT gives $C_{\max} = 19$



# Tightness of the Bound for LPT

**Graham 1966**

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

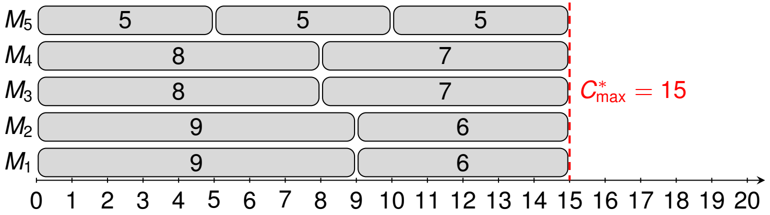
Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :

LPT gives $C_{\max} = 19$

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



# Tightness of the Bound for LPT

**Graham 1966**

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

$\frac{19}{15} = \frac{20}{15} - \frac{1}{15}$

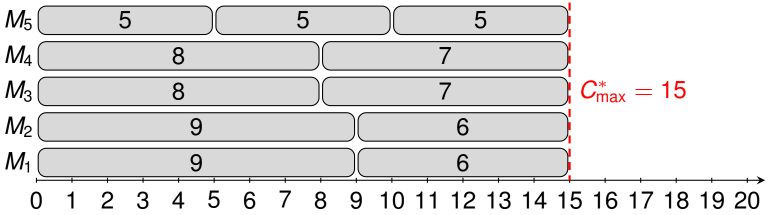
Proof of an instance which shows tightness:

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

$m = 5, n = 11$ :

LPT gives $C_{\max} = 19$

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



# A PTAS for Parallel Machine Scheduling

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.

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 $< T$

# 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 $C_{\max} \leq (1+\epsilon) \cdot \max\{T, C^*_{\max}\}$

2: Or: **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 $C_{\max} \leq (1+\epsilon) \cdot \max\{T, C^*_{\max}\}$

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 $C_{\max} \leq (1+\epsilon) \cdot \max\{T, C^*_{\max}\}$

2: Or: **Return** there is no solution with makespan $< T$

We will prove this on the next slides.

**Key Lemma**

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.

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 $< T$

We will prove this on the next slides.

**Key Lemma**

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

2: Or: **Return** there is no solution with makespan $< T$

**Key Lemma**

We will prove this on the next slides.

$\textsc{Subroutine}$ 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 $P := \sum_{k=1}^{n} p_k$.

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

Proof (using Key Lemma):

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

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

2: **Return** solution computed by $\textsc{Subroutine}(J_1, J_2, \ldots, J_n, m, T)$

# A PTAS for Parallel Machine Scheduling

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

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 $< T$

We will prove this on the next slides.

**Key Lemma**

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 $P := \sum_{k=1}^{n} p_k$.

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

Proof (using Key Lemma):

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

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

2: **Return** solution computed by 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.

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 $< T$

We will prove this on the next slides.

**Key Lemma**

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 $P := \sum_{k=1}^{n} p_k$.

polynomial in the size of the input

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

Proof (using Key Lemma):

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

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

# Implementation of Subroutine

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

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

Or: **Return** there is no solution with makespan $< T$

# Implementation of Subroutine

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: $J_{\text{small}} = \{J_i : p_i \leq \epsilon \cdot T\}$ and $J_{\text{large}} = J \setminus J_{\text{small}}$. 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 $(1+\epsilon) \cdot \max\{T, C^*_{\max}\}$.

# Implementation of Subroutine

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: $J_{\text{small}} = \{J_i : p_i \leq \epsilon \cdot T\}$ and $J_{\text{large}} = J \setminus J_{\text{small}}$. 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:

# Implementation of Subroutine

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: $J_{\text{small}} = \{J_i \colon p_i \leq \epsilon \cdot T\}$ and $J_{\text{large}} = J \setminus J_{\text{small}}$. 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

# Implementation of Subroutine

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: $J_{\text{small}} = \{J_i : p_i \leq \epsilon \cdot T\}$ and $J_{\text{large}} = J \setminus J_{\text{small}}$. 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 $(1+\epsilon) \cdot T$.

# Implementation of Subroutine

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

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: $J_{\text{small}} = \{J_i : p_i \leq \epsilon \cdot T\}$ and $J_{\text{large}} = J \setminus J_{\text{small}}$. 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 $M_j$.

# Implementation of Subroutine

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: $J_{\text{small}} = \{J_i : p_i \leq \epsilon \cdot T\}$ and $J_{\text{large}} = J \setminus J_{\text{small}}$. 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 $M_j$.

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

the "well-known" formula

## Implementation of Subroutine

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: $J_{\text{small}} = \{J_i \colon p_i \leq \epsilon \cdot T\}$ and $J_{\text{large}} = J \setminus J_{\text{small}}$. 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 $M_j$.

$$C_j - p_i \leq \frac{1}{m} \sum_{k=1}^{n} p_k \qquad \Rightarrow$$

the "well-known" formula

## Implementation of Subroutine

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: $J_{\text{small}} = \{J_i \colon p_i \leq \epsilon \cdot T\}$ and $J_{\text{large}} = J \setminus J_{\text{small}}$. 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 $M_j$.

$$C_j - p_i \leq \frac{1}{m} \sum_{k=1}^{n} p_k \qquad \Rightarrow \qquad C_j \leq p_i + \frac{1}{m} \sum_{k=1}^{n} p_k$$

the "well-known" formula

## Implementation of Subroutine

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: $J_{\text{small}} = \{J_i \colon p_i \leq \epsilon \cdot T\}$ and $J_{\text{large}} = J \setminus J_{\text{small}}$. 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 $M_j$.

$$C_j - p_i \leq \frac{1}{m} \sum_{k=1}^{n} p_k \qquad \Rightarrow \qquad C_j \leq p_i + \frac{1}{m} \sum_{k=1}^{n} p_k$$

$$\leq \epsilon \cdot T + C^*_{\max}$$

the “well-known” formula

## Implementation of Subroutine

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

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

2: Or: **Return** there is no solution with makespan $< T$

**Observation**

Divide jobs into two groups: $J_{\text{small}} = \{J_i \colon p_i \leq \epsilon \cdot T\}$ and $J_{\text{large}} = J \setminus J_{\text{small}}$. 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 $M_j$.

$$C_j - p_i \leq \frac{1}{m} \sum_{k=1}^{n} p_k \qquad \Rightarrow \qquad C_j \leq p_i + \frac{1}{m} \sum_{k=1}^{n} p_k$$

the "well-known" formula

$$\leq \epsilon \cdot T + C^*_{\max}$$

$$\leq (1+\epsilon) \cdot \max\{T, C^*_{\max}\} \qquad \square$$

# Implementation of Subroutine

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: $J_{\text{small}} = \{J_i \colon p_i \leq \epsilon \cdot T\}$ and $J_{\text{large}} = J \setminus J_{\text{small}}$. 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 $M_j$.

$$C_j - p_i \leq \frac{1}{m} \sum_{k=1}^{n} p_k \qquad \Rightarrow \qquad C_j \leq p_i + \frac{1}{m} \sum_{k=1}^{n} p_k$$

the "well-known" formula

$$\leq \epsilon \cdot T + C^*_{\max}$$

$$\leq (1+\epsilon) \cdot \max\{T, C^*_{\max}\} \qquad \square$$

# Proof of Key Lemma (non-examinable)

Use Dynamic Programming to schedule $J_{\text{large}}$ with makespan $(1+\epsilon) \cdot T$.

# 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 $1/b \leq \epsilon$.

## Proof of Key Lemma (non-examinable)

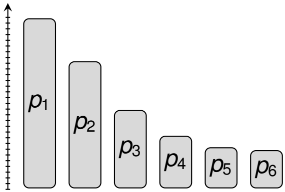
Use Dynamic Programming to schedule $J_{\text{large}}$ with makespan $(1+\epsilon)\cdot T$.

- Let $b$ be the smallest integer with $1/b \leq \epsilon$. Define processing times $p_i' = \lceil \frac{p_j b^2}{T} \rceil \cdot \frac{T}{b^2}$

# Proof of Key Lemma (non-examinable)

Use Dynamic Programming to schedule $J_{\text{large}}$ with makespan $(1+\epsilon) \cdot T$.

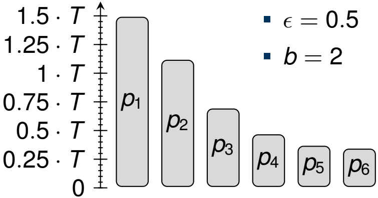
- Let $b$ be the smallest integer with $1/b \leq \epsilon$. Define processing times $p_j' = \lceil \frac{p_j b^2}{T} \rceil \cdot \frac{T}{b^2}$



# 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 $p_i' = \left\lceil \frac{p_j b^2}{T} \right\rceil \cdot \frac{T}{b^2}$

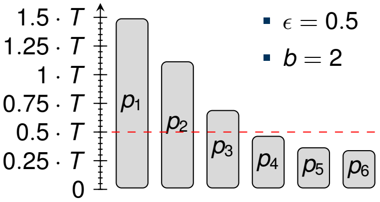



- $\epsilon = 0.5$
- $b = 2$

# 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 $p_i' = \left\lceil \frac{p_j b^2}{T} \right\rceil \cdot \frac{T}{b^2}$

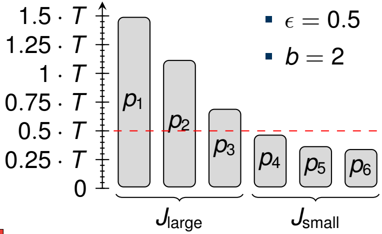



- $\epsilon = 0.5$
- $b = 2$

# 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 $p_i' = \left\lceil \frac{p_j b^2}{T} \right\rceil \cdot \frac{T}{b^2}$

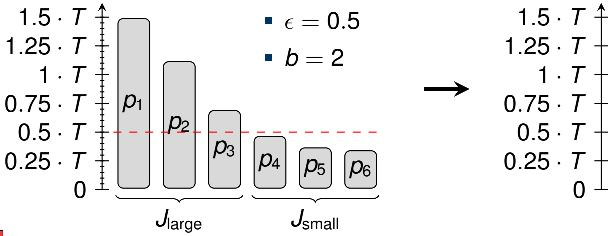



- $\epsilon = 0.5$
- $b = 2$

# Proof of Key Lemma (non-examinable)

Use Dynamic Programming to schedule $J_{\text{large}}$ with makespan $(1+\epsilon)\cdot T$.

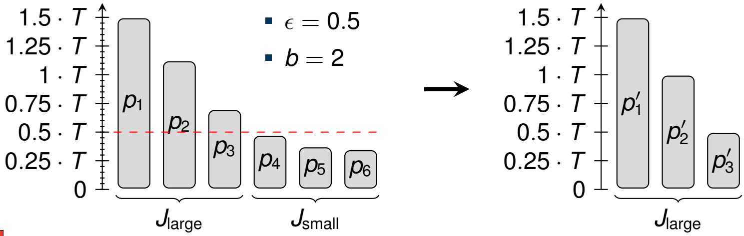
- Let $b$ be the smallest integer with $1/b \leq \epsilon$. Define processing times $p_i' = \lceil \frac{p_j b^2}{T} \rceil \cdot \frac{T}{b^2}$



# 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 $p'_i = \lceil \frac{p_j b^2}{T} \rceil \cdot \frac{T}{b^2}$



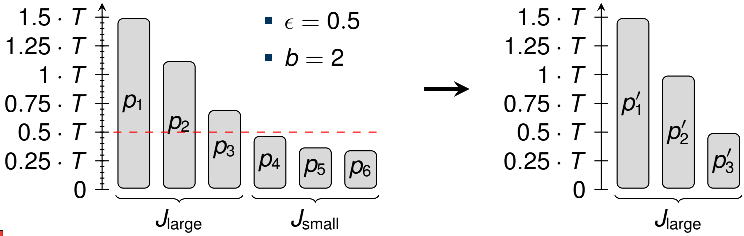
# Proof of Key Lemma (non-examinable)

**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 $p_i' = \left\lceil \frac{p_j b^2}{T} \right\rceil \cdot \frac{T}{b^2}$

$\Rightarrow$ Every $p_i' = \alpha \cdot \frac{T}{b^2}$ for $\alpha = b, b+1, \ldots, b^2$

Can assume there are no jobs with $p_j \geq T$!



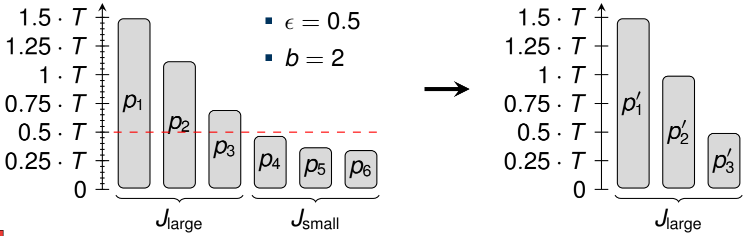
# Proof of Key Lemma (non-examinable)

**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 $p_i' = \lceil \frac{p_j b^2}{T} \rceil \cdot \frac{T}{b^2}$

$\Rightarrow$ 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 $\sum_{i=j}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T$.



# Proof of Key Lemma (non-examinable)

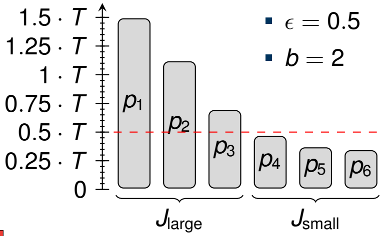
Use Dynamic Programming to schedule $J_{\text{large}}$ with makespan $(1+\epsilon)\cdot T$.

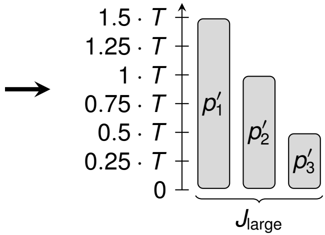
- Let $b$ be the smallest integer with $1/b \leq \epsilon$. Define processing times $p_i' = \left\lceil \frac{p_j b^2}{T} \right\rceil \cdot \frac{T}{b^2}$

$\Rightarrow$ 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 $\sum_{i=j}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T$.

Assignments to one machine with makespan $\leq T$.



# Proof of Key Lemma (non-examinable)

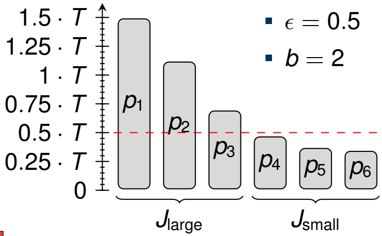
**Use Dynamic Programming to schedule $J_{\text{large}}$ with makespan $(1+\epsilon) \cdot T$.**

- Let $b$ be the smallest integer with $1/b \leq \epsilon$. Define processing times $p_i' = \lceil \frac{p_j b^2}{T} \rceil \cdot \frac{T}{b^2}$

$\Rightarrow$ 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 $\sum_{i=j}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T$.

- Let $f(n_b, n_{b+1}, \ldots, n_{b^2})$ be the minimum number of machines required to schedule all jobs with makespan $\leq T$:



# Proof of Key Lemma (non-examinable)

Use Dynamic Programming to schedule $J_{\text{large}}$ with makespan $(1+\epsilon) \cdot T$.

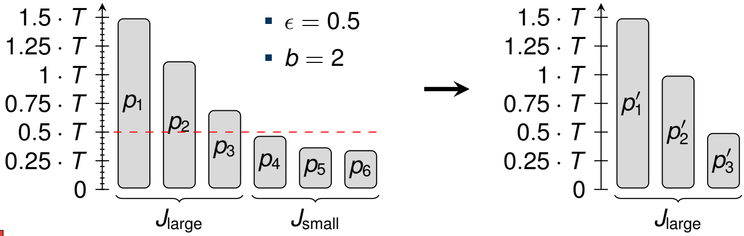
- Let $b$ be the smallest integer with $1/b \leq \epsilon$. Define processing times $p_i' = \lceil \frac{p_j b^2}{T} \rceil \cdot \frac{T}{b^2}$

$\Rightarrow$ 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 $\sum_{i=j}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T$.

- Let $f(n_b, n_{b+1}, \ldots, n_{b^2})$ be the minimum number of machines required to schedule all jobs with makespan $\leq T$:

$$f(0, 0, \ldots, 0) = 0$$



# 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 $p_i' = \left\lceil \frac{p_i b^2}{T} \right\rceil \cdot \frac{T}{b^2}$

$\Rightarrow$ Every $p_i' = \alpha \cdot \frac{T}{b^2}$ for $\alpha = b, b+1, \ldots, b^2$

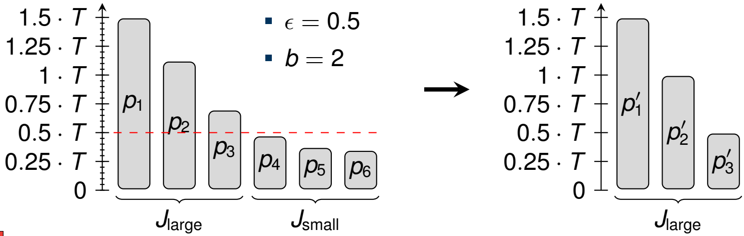
- Let $\mathcal{C}$ be all $(s_b, s_{b+1}, \ldots, s_{b^2})$ with $\sum_{i=j}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T$.

- Let $f(n_b, n_{b+1}, \ldots, n_{b^2})$ be the minimum number of machines required to schedule all jobs with makespan $\leq T$:

$$f(0,0,\ldots,0) = 0$$

$$f(n_b, n_{b+1}, \ldots, n_{b^2}) = 1 + \min_{(s_b, s_{b+1}, \ldots, s_{b^2}) \in \mathcal{C}} f(n_b - s_b, n_{b+1} - s_{b+1}, \ldots, n_{b^2} - s_{b^2}).$$

- $\epsilon = 0.5$
- $b = 2$



# 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 $p_i' = \left\lceil \frac{p_j b^2}{T} \right\rceil \cdot \frac{T}{b^2}$

$\Rightarrow$ Every $p_i' = \alpha \cdot \frac{T}{b^2}$ for $\alpha = b, b+1, \ldots, b^2$

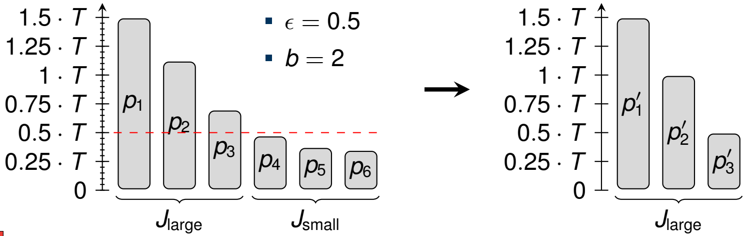
- Let $\mathcal{C}$ be all $(s_b, s_{b+1}, \ldots, s_{b^2})$ with $\sum_{i=j}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T$.

- Let $f(n_b, n_{b+1}, \ldots, n_{b^2})$ be the minimum number of machines required to schedule all jobs with makespan $\leq T$:

Assign some jobs to one machine, and then use as few machines as possible for the rest.

$$f(0,0,\ldots,0) = 0$$

$$f(n_b, n_{b+1}, \ldots, n_{b^2}) = 1 + \min_{(s_b, s_{b+1}, \ldots, s_{b^2}) \in \mathcal{C}} f(n_b - s_b, n_{b+1} - s_{b+1}, \ldots, n_{b^2} - s_{b^2}).$$



# 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 $p_i' = \left\lceil \frac{p_i b^2}{T} \right\rceil \cdot \frac{T}{b^2}$

$\Rightarrow$ 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 $\sum_{i=j}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T$.

- 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 $n^{O(b^2)}$

# 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 $p_i' = \left\lceil \frac{p_j b^2}{T} \right\rceil \cdot \frac{T}{b^2}$

$\Rightarrow$ 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 $\sum_{i=j}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T$.

- 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.

# 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 $p_i' = \left\lceil \frac{p_i b^2}{T} \right\rceil \cdot \frac{T}{b^2}$

$\Rightarrow$ 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 $\sum_{i=j}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T$.
- 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 $\leq T$,

# 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 $p_i' = \left\lceil \frac{p_i b^2}{T} \right\rceil \cdot \frac{T}{b^2}$

$\Rightarrow$ 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 $\sum_{i=j}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T$.
- 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 $\leq T$,

$$C_{\max} \leq T + b \cdot \max_{i \in J_{\text{large}}} (p_i - p_i')$$

# 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 $p_i' = \left\lceil \frac{p_j b^2}{T} \right\rceil \cdot \frac{T}{b^2}$

$\Rightarrow$ 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 $\sum_{i=j}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T$.

- 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 $\leq T$,

$$\begin{aligned} C_{\max} &\leq T + b \cdot \max_{i \in J_{\text{large}}} (p_i - p_i') \\ &\leq T + b \cdot \frac{T}{b^2} \end{aligned}$$

## 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 $p_i' = \left\lceil \frac{p_j b^2}{T} \right\rceil \cdot \frac{T}{b^2}$

$\Rightarrow$ 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 $\sum_{i=j}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T$.

- 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 $\leq T$,

$$\begin{aligned} C_{\max} &\leq T + b \cdot \max_{i \in J_{\text{large}}} (p_i - p_i') \\ &\leq T + b \cdot \frac{T}{b^2} \leq (1+\epsilon) \cdot T. \qquad \square \end{aligned}$$

# Final Remarks

**Graham 1966**

List scheduling has an approximation ratio of 2.

**Graham 1966**

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

# Final Remarks

**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 $P := \sum_{k=1}^{n} p_k$.

# Final Remarks

**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)?

## Final Remarks

**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!**

# Final Remarks

**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.