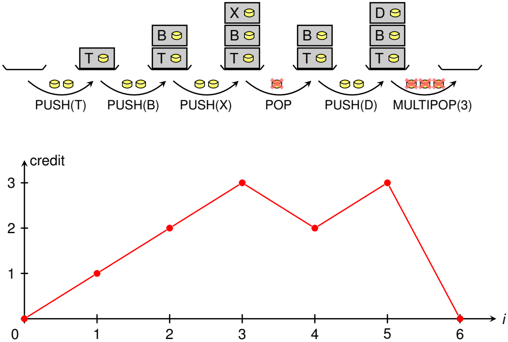


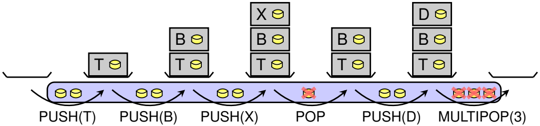
5.1: Amortized Analysis

Frank Stajano Thomas Sauerwald

Lent 2016

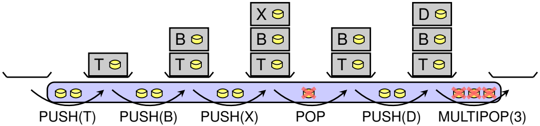
UNIVERSITY OF CAMBRIDGE

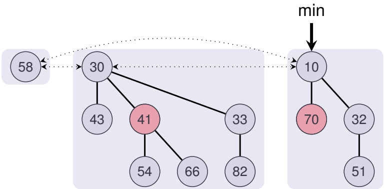
Use of Amortized Analysis

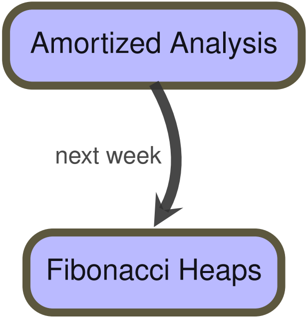


Amortized Analysis

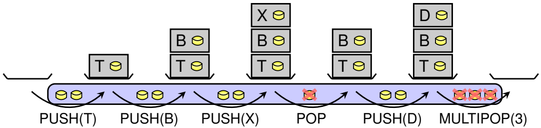
Use of Amortized Analysis

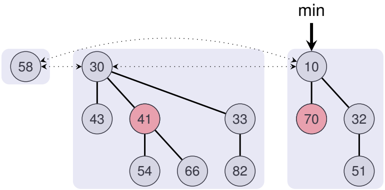


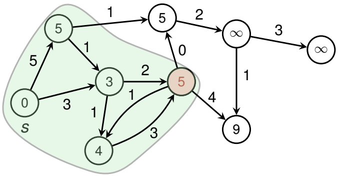


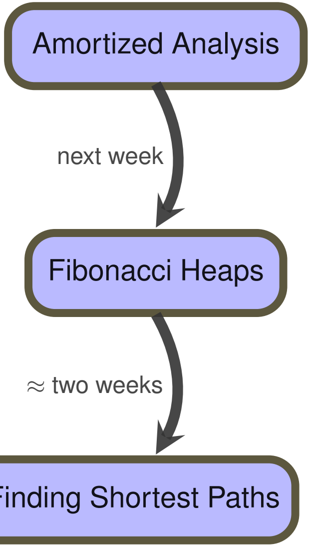


Use of Amortized Analysis









Motivating Example: Stack

Stack Operations

Motivating Example: Stack

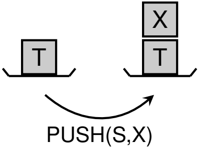
Stack Operations

- **`PUSH(S,x)`**

Motivating Example: Stack

Stack Operations

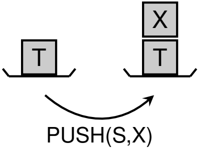
- `PUSH(S,x)`
  - pushes object x onto stack S



Motivating Example: Stack

Stack Operations

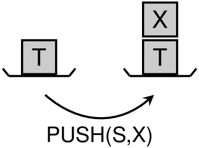
- **`PUSH(S,x)`**
  - pushes object x onto stack S
  - total cost of 1



Motivating Example: Stack

Stack Operations

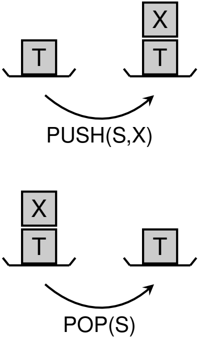
- **`PUSH(S,x)`**
  - pushes object x onto stack S
  - total cost of 1
- **`POP(S)`**



Motivating Example: Stack

Stack Operations

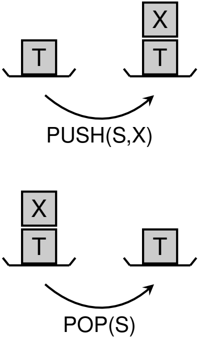
- `PUSH(S,x)`
  - pushes object x onto stack S
  - total cost of 1
- `POP(S)`
  - pops the top of (a non-empty) stack S



Motivating Example: Stack

Stack Operations

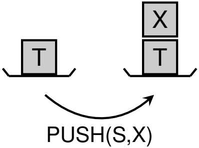
- `PUSH(S,x)`
  - pushes object x onto stack S
  - total cost of 1
- `POP(S)`
  - pops the top of (a non-empty) stack S
  - total cost of 1

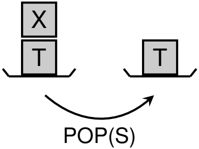


Motivating Example: Stack

Stack Operations

- **`PUSH(S,x)`**
  - pushes object x onto stack S
  - total cost of 1
- **`POP(S)`**
  - pops the top of (a non-empty) stack S
  - total cost of 1
- **`MULTIPOP(S,k)`**

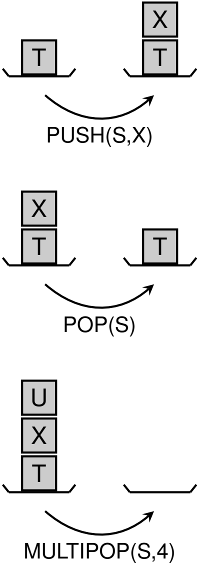




Motivating Example: Stack

Stack Operations

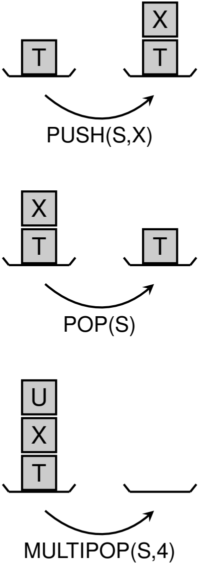
- `PUSH(S,x)`
  - pushes object x onto stack S
  - total cost of 1
- `POP(S)`
  - pops the top of (a non-empty) stack S
  - total cost of 1
- `MULTIPOP(S,k)`
  - pops the k top objects (S non-empty)



Motivating Example: Stack

Stack Operations

- **`PUSH(S,x)`**
  - pushes object x onto stack S
  - total cost of 1
- **`POP(S)`**
  - pops the top of (a non-empty) stack S
  - total cost of 1
- **`MULTIPOP(S,k)`**
  - pops the k top objects (S non-empty)
  - $\Rightarrow$ total cost of $\min\{|S|, k\}$

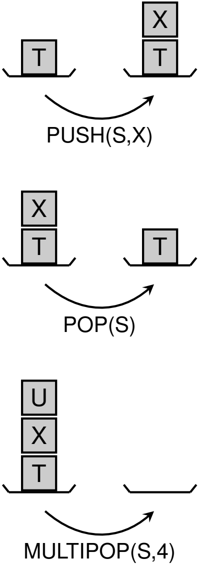


Motivating Example: Stack

Stack Operations

- **`PUSH(S,x)`**
  - pushes object x onto stack S
  - total cost of 1
- **`POP(S)`**
  - pops the top of (a non-empty) stack S
  - total cost of 1
- **`MULTIPOP(S,k)`**
  - pops the k top objects (S non-empty)
  - $\Rightarrow$ total cost of $\min\{|S|, k\}$

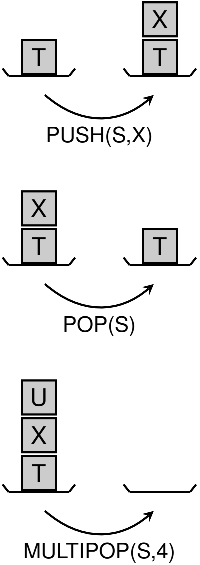
```
MULTIPOP(S,k)
while not S.empty() and k > 0
    POP(S)
    k = k - 1
```



Motivating Example: Stack

Stack Operations

- **`PUSH(S,x)`**
  - pushes object x onto stack S
  - total cost of 1
- **`POP(S)`**
  - pops the top of (a non-empty) stack S
  - total cost of 1
- **`MULTIPOP(S,k)`**
  - pops the k top objects (S non-empty)
  - $\Rightarrow$ total cost of $\min\{|S|, k\}$



What is the largest possible cost of a sequence of n stack operations (starting from an empty stack)?

Motivating Example: Stack

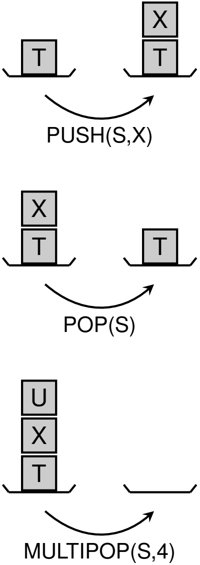
Stack Operations

- `PUSH(S,x)`
  - pushes object x onto stack S
  - total cost of 1
- `POP(S)`
  - pops the top of (a non-empty) stack S
  - total cost of 1
- `MULTIPOP(S,k)`
  - pops the k top objects (S non-empty)
  - $\Rightarrow$ total cost of $\min\{|S|, k\}$

What is the largest possible cost of a sequence of n stack operations (starting from an empty stack)?

Simple Worst-Case Bound (stack is initially empty):

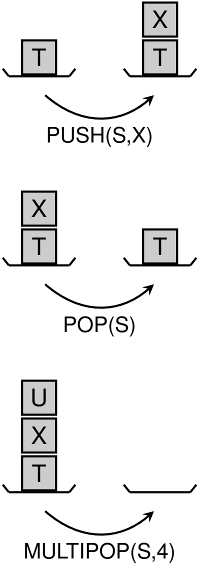
- largest cost of an operation: n
- cost is at most $n \cdot n = n^2$



Motivating Example: Stack

Stack Operations

- **`PUSH(S,x)`**
  - pushes object x onto stack S
  - total cost of 1
- **`POP(S)`**
  - pops the top of (a non-empty) stack S
  - total cost of 1
- **`MULTIPOP(S,k)`**
  - pops the k top objects (S non-empty)
  - $\Rightarrow$ total cost of $\min\{|S|, k\}$



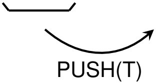
What is the largest possible cost of a sequence of n stack operations (starting from an empty stack)?

Simple Worst-Case Bound (stack is initially empty):

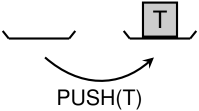
- largest cost of an operation: n
- cost is at most $n \cdot n = n^2$ **(correct, but not tight!)**

Sequence of Stack Operations

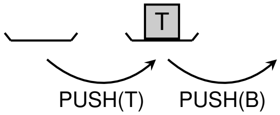
Sequence of Stack Operations



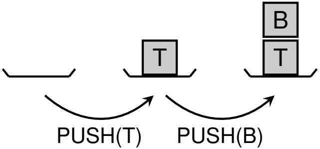
Sequence of Stack Operations



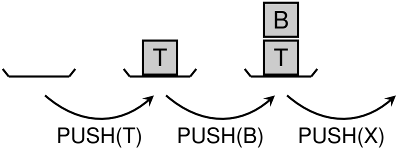
Sequence of Stack Operations



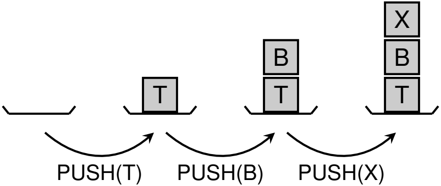
Sequence of Stack Operations



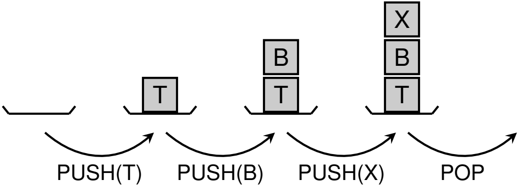
Sequence of Stack Operations



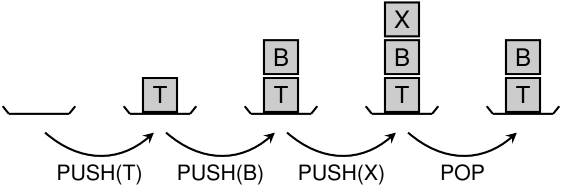
Sequence of Stack Operations



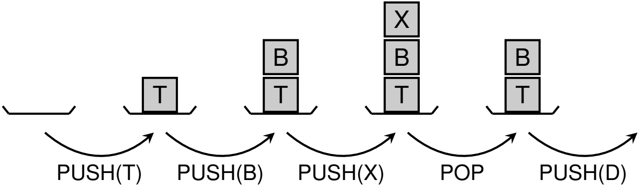
Sequence of Stack Operations



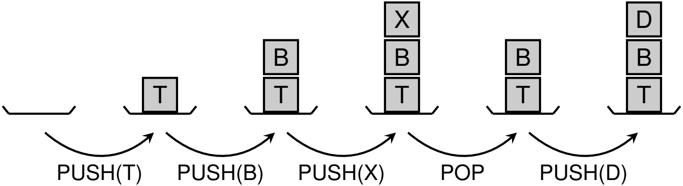
Sequence of Stack Operations



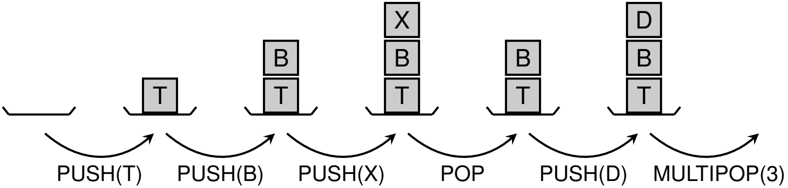
Sequence of Stack Operations



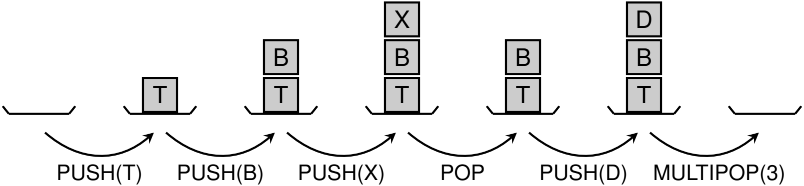
Sequence of Stack Operations



Sequence of Stack Operations



Sequence of Stack Operations



A new Analysis Tool: Amortized Analysis

Amortized Analysis

A new Analysis Tool: Amortized Analysis

Amortized Analysis

- analyse a sequence of operations

A new Analysis Tool: Amortized Analysis

Data structure operations (Heap, Stack, Queue etc.)

Amortized Analysis

- analyse a sequence of operations

A new Analysis Tool: Amortized Analysis

Amortized Analysis

- analyse a sequence of operations
- show that average cost of an operation is small

A new Analysis Tool: Amortized Analysis

Amortized Analysis

- analyse a sequence of operations
- show that average cost of an operation is small

This is **not** average case analysis!

A new Analysis Tool: Amortized Analysis

Amortized Analysis

- analyse a sequence of operations
- show that average cost of an operation is small
- concrete techniques
  - Aggregate Analysis
  - Potential Method

A new Analysis Tool: Amortized Analysis

Amortized Analysis

- analyse a sequence of operations
- show that average cost of an operation is small
- concrete techniques
  - Aggregate Analysis
  - Potential Method

Aggregate Analysis

- Determine an upper bound $T(n)$ for the total cost of any sequence of n operations
- amortized cost of each operation is the average $\frac{T(n)}{n}$

A new Analysis Tool: Amortized Analysis

Amortized Analysis

- analyse a sequence of operations
- show that average cost of an operation is small
- concrete techniques
  - Aggregate Analysis
  - Potential Method

Aggregate Analysis

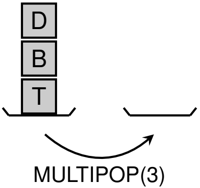
- Determine an upper bound $T(n)$ for the total cost of any sequence of n operations
- amortized cost of each operation is the average $\frac{T(n)}{n}$

Even though operations may be of different types/costs

Stack: Aggregate Analysis

Simple Worst-Case Bound:

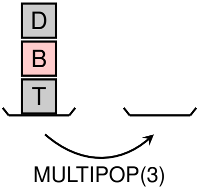
- largest cost of an operation: n
- cost is at most $n \cdot n = n^2$ **(correct, but not tight!)**



Stack: Aggregate Analysis

Simple Worst-Case Bound:

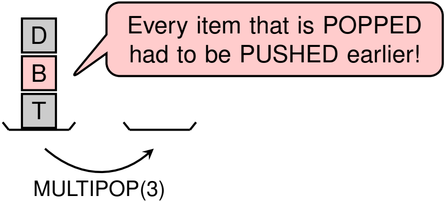
- largest cost of an operation: n
- cost is at most $n \cdot n = n^2$ **(correct, but not tight!)**



Stack: Aggregate Analysis

Simple Worst-Case Bound:

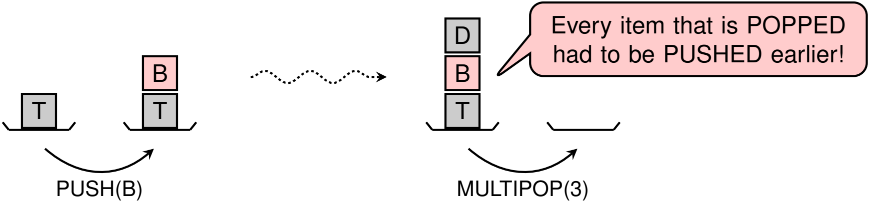
- largest cost of an operation: n
- cost is at most $n \cdot n = n^2$ **(correct, but not tight!)**



Stack: Aggregate Analysis

Simple Worst-Case Bound:

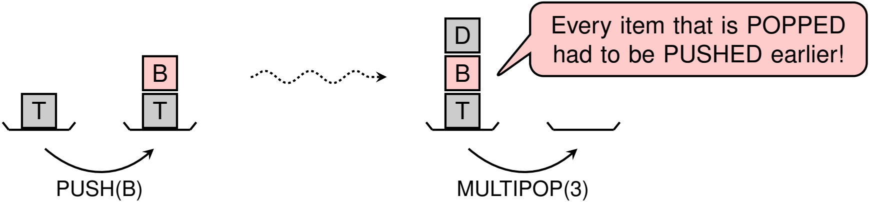
- largest cost of an operation: n
- cost is at most $n \cdot n = n^2$ **(correct, but not tight!)**



Stack: Aggregate Analysis

Simple Worst-Case Bound:

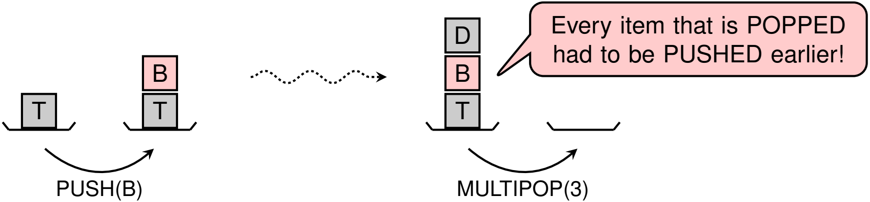
- largest cost of an operation: n
- cost is at most $n \cdot n = n^2$ **(correct, but not tight!)**



Stack: Aggregate Analysis

Simple Worst-Case Bound:

- largest cost of an operation: n
- cost is at most $n \cdot n = n^2$ **(correct, but not tight!)**

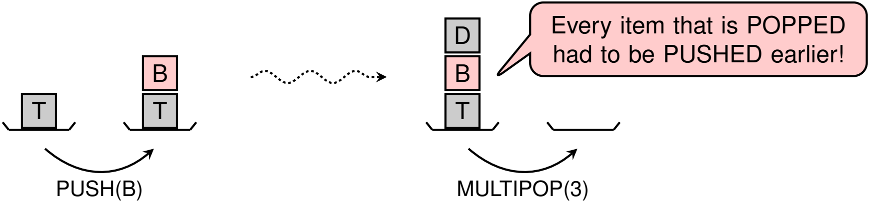


$T(n) \leq$

Stack: Aggregate Analysis

Simple Worst-Case Bound:

- largest cost of an operation: n
- cost is at most $n \cdot n = n^2$ **(correct, but not tight!)**

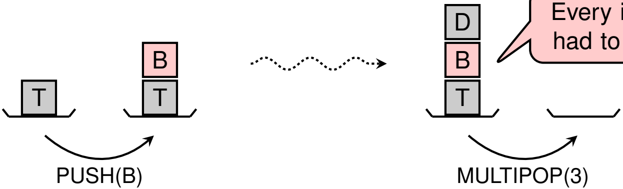


$$T(n) \leq T_{POP}(n) + T_{PUSH}(n)$$

Stack: Aggregate Analysis

Simple Worst-Case Bound:

- largest cost of an operation: n
- cost is at most $n \cdot n = n^2$ **(correct, but not tight!)**



Every item that is POPPED had to be PUSHED earlier!

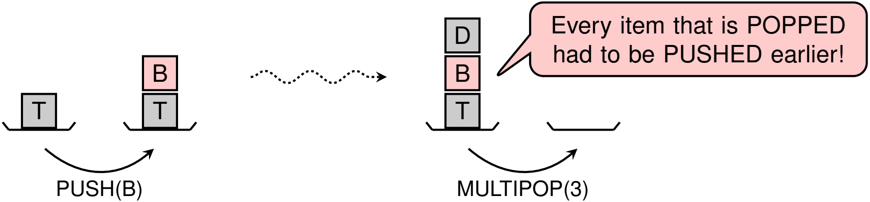
MULTIPOP(k) contributes $\min\{k, |S|\}$ to $T_{POP}(n)$

$$T(n) \leq T_{POP}(n) + T_{PUSH}(n)$$

Stack: Aggregate Analysis

Simple Worst-Case Bound:

- largest cost of an operation: n
- cost is at most $n \cdot n = n^2$ **(correct, but not tight!)**

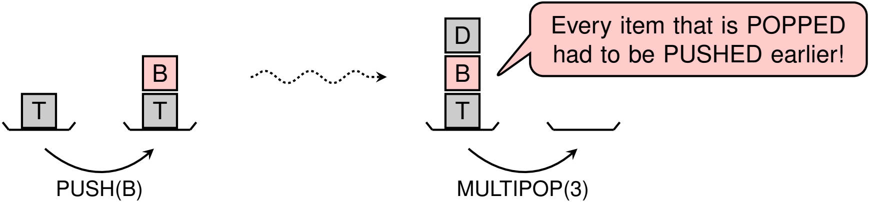


$$T(n) \leq T_{POP}(n) + T_{PUSH}(n) \leq 2 \cdot T_{PUSH}(n)$$

Stack: Aggregate Analysis

Simple Worst-Case Bound:

- largest cost of an operation: n
- cost is at most $n \cdot n = n^2$ **(correct, but not tight!)**

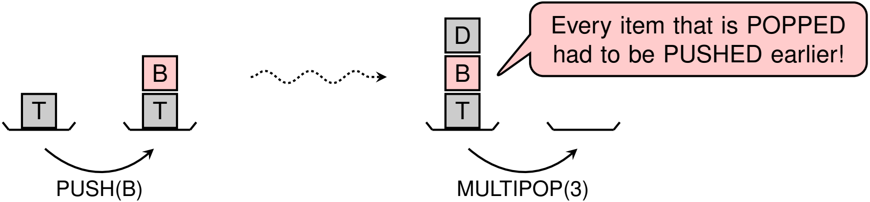


$$T(n) \leq T_{POP}(n) + T_{PUSH}(n) \leq 2 \cdot T_{PUSH}(n) \leq 2 \cdot n.$$

Stack: Aggregate Analysis

Simple Worst-Case Bound:

- largest cost of an operation: n
- cost is at most $n \cdot n = n^2$ **(correct, but not tight!)**



$$T(n) \leq T_{POP}(n) + T_{PUSH}(n) \leq 2 \cdot T_{PUSH}(n) \leq 2 \cdot n.$$

Aggregate Analysis: The amortized cost per operation is $\frac{T(n)}{n} \leq 2$

Second Technique: Potential Method

Potential Method

Second Technique: Potential Method

Potential Method

- allow different amortized costs

$\rightsquigarrow$ store **(fictitious)** credit in the data structure to cover up for expensive operations

Second Technique: Potential Method

Potential Method

- allow different amortized costs

$\rightsquigarrow$ store **(fictitious)** credit in the data structure to cover up for expensive operations

Potential of a data structure can be also thought of as

- amount of potential energy stored
- distance from an ideal state

Second Technique: Potential Method

Potential Method

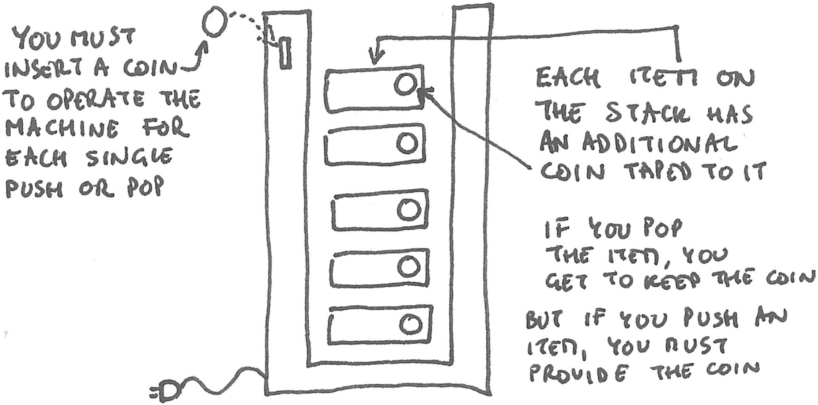
- allow different amortized costs

$\rightsquigarrow$ store **(fictitious)** credit in the data structure to cover up for expensive operations

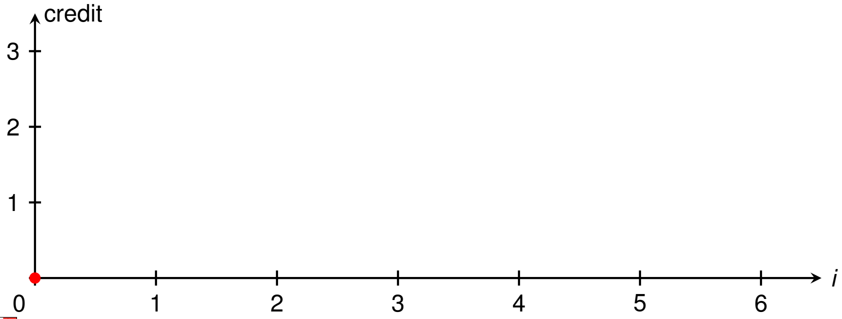
Potential of a data structure can be also thought of as

- amount of potential energy stored
- distance from an ideal state

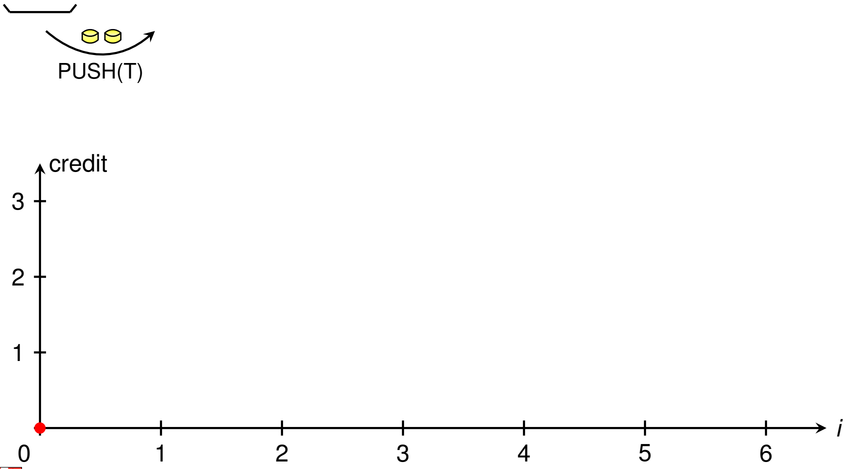
Stack as a coin-operated machine (p. 83)



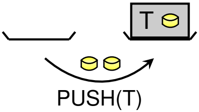
Stack and Coins

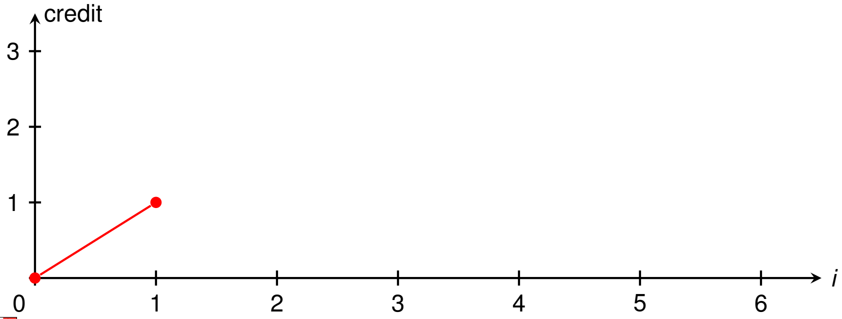


Stack and Coins

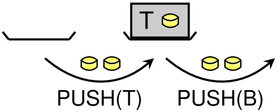


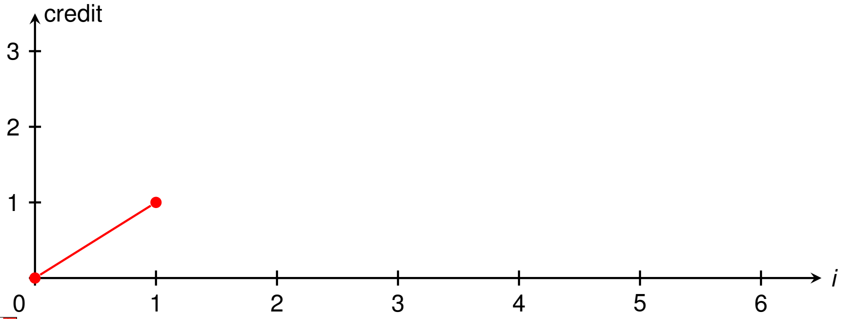
Stack and Coins



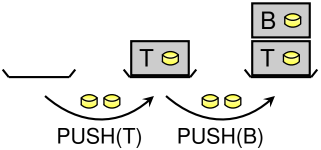


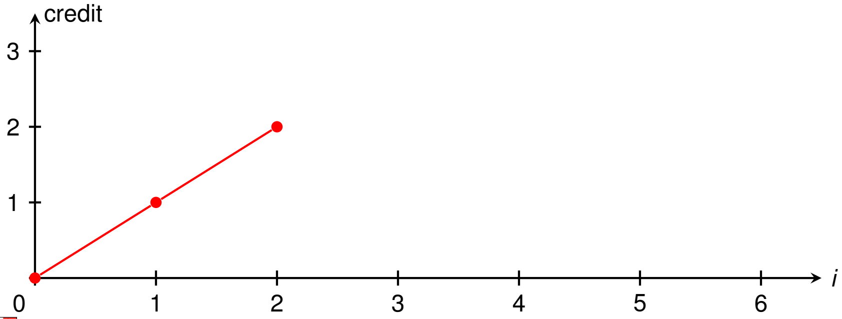
Stack and Coins



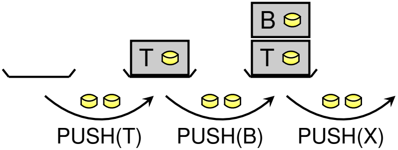


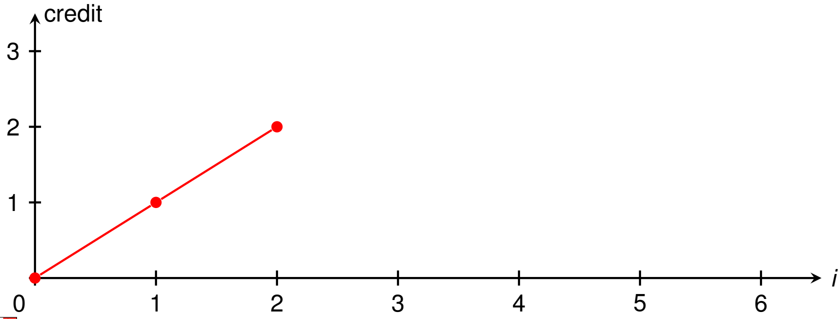
Stack and Coins



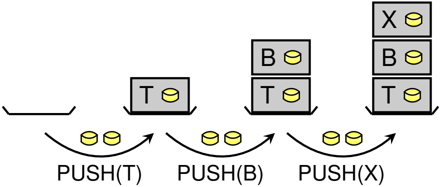


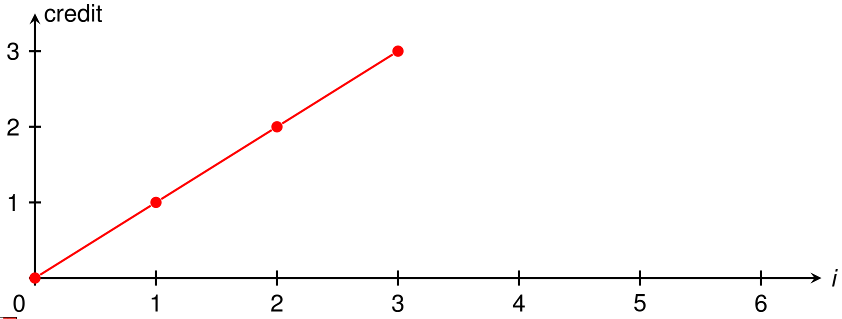
Stack and Coins



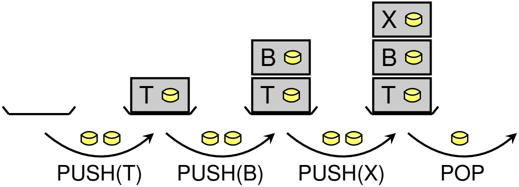


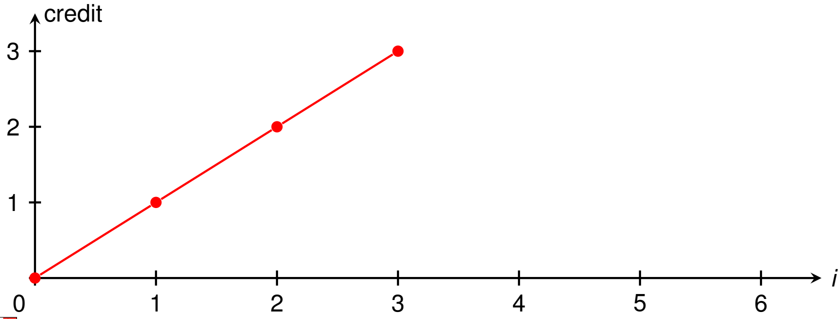
Stack and Coins



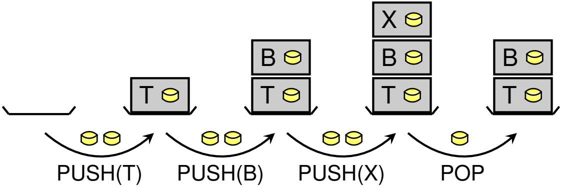


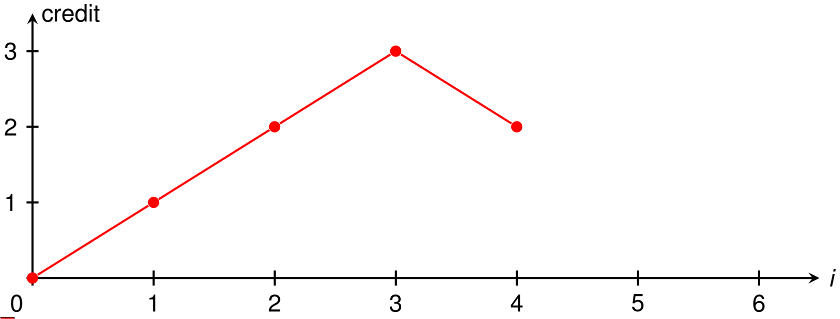
Stack and Coins



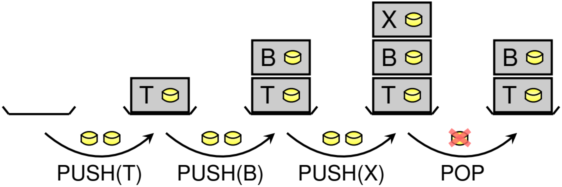


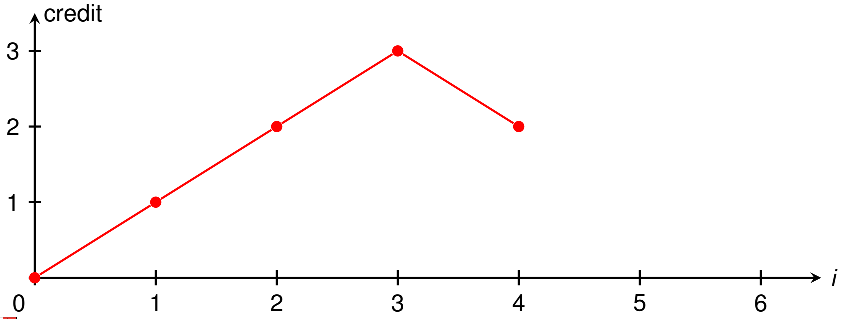
Stack and Coins



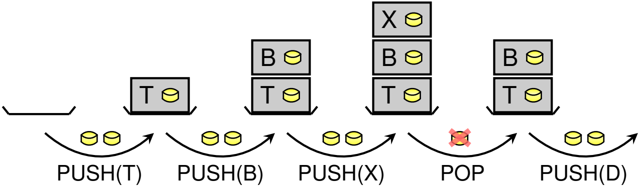


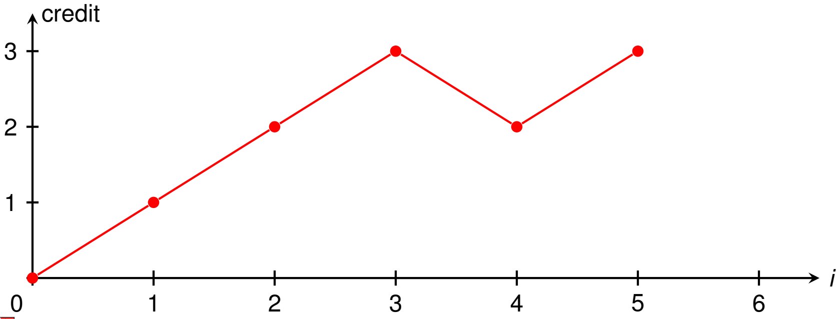
Stack and Coins



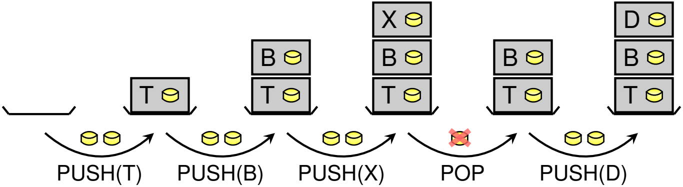


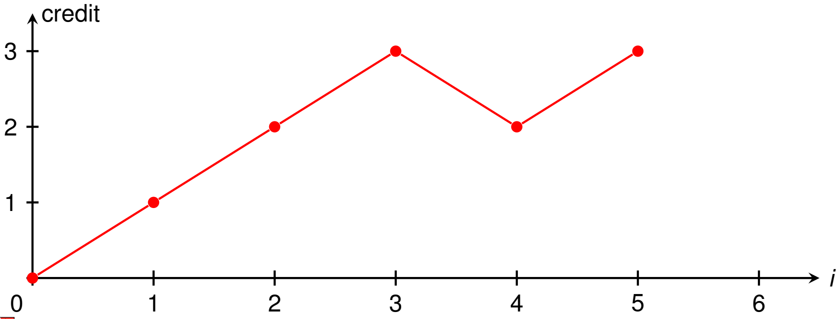
Stack and Coins



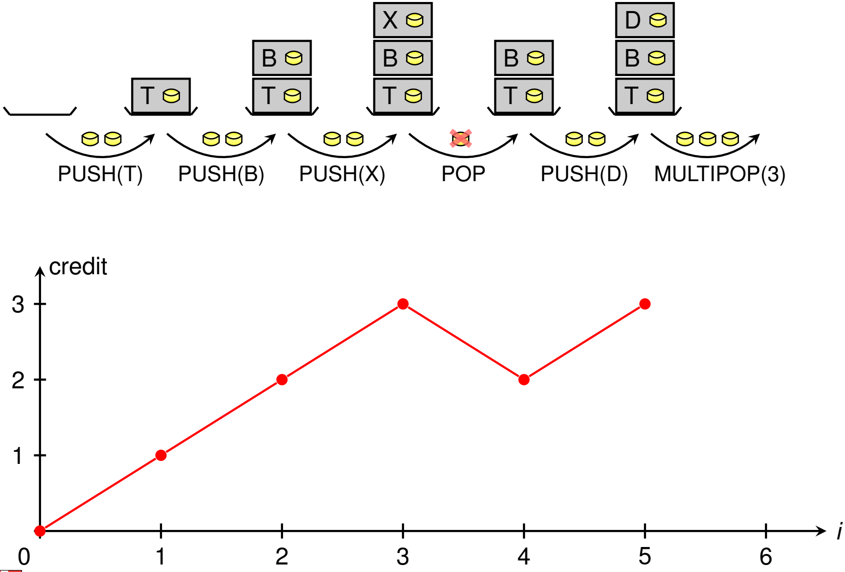


Stack and Coins

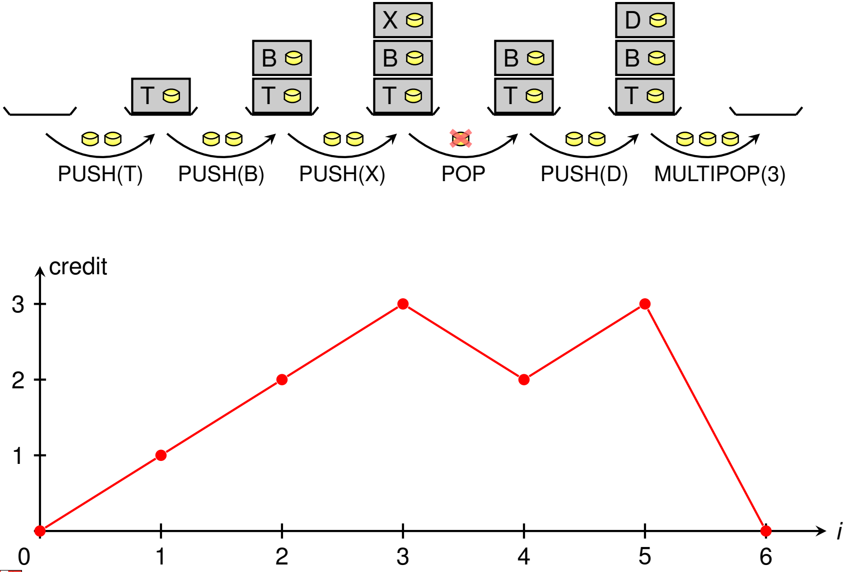




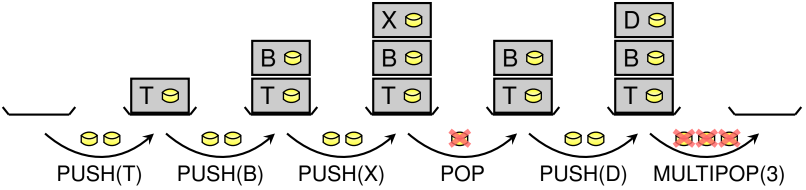
Stack and Coins

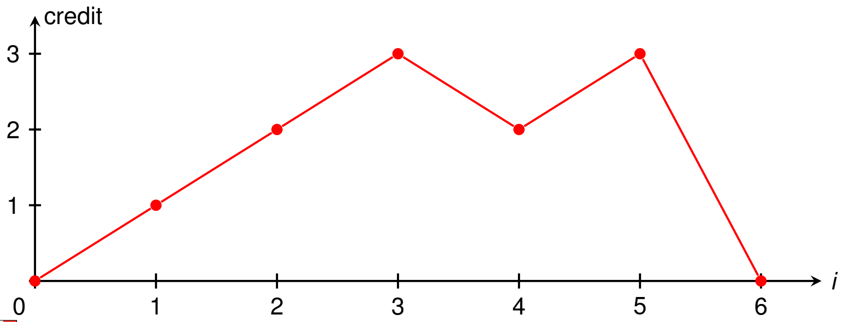


Stack and Coins

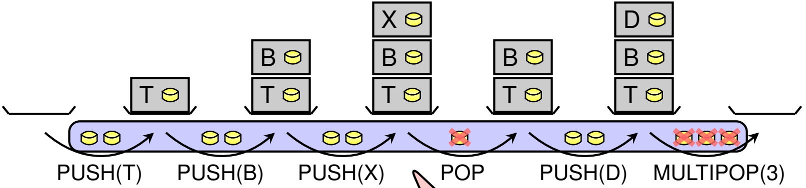


Stack and Coins

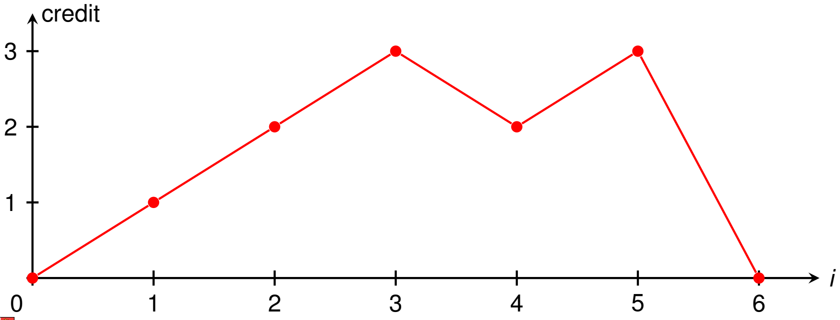




Stack and Coins



Every operation costs at most **two coins**!



Potential Method in Detail

- c_i is the actual cost of operation i

Potential Method in Detail

- c_i is the actual cost of operation i
- $\widehat{c}_i$ is the amortized cost of operation i

Potential Method in Detail

- c_i is the actual cost of operation i
- $\widehat{c}_i$ is the amortized cost of operation i

$c_i < \widehat{c}_i$, $c_i = \widehat{c}_i$ or $c_i > \widehat{c}_i$ are all possible!

Potential Method in Detail

- c_i is the actual cost of operation i
- $\widehat{c}_i$ is the amortized cost of operation i
- Φ_i is the potential stored after operation i ($\Phi_0 = 0$)

Potential Method in Detail

- c_i is the actual cost of operation i
- $\widehat{c}_i$ is the amortized cost of operation i
- Φ_i is the potential stored after operation i ($\Phi_0 = 0$)

Function that maps states of the data structure to some value

Potential Method in Detail

- c_i is the actual cost of operation i
- $\widehat{c}_i$ is the amortized cost of operation i
- Φ_i is the potential stored after operation i ($\Phi_0 = 0$)

$$\widehat{c}_i = c_i + \left(\Phi_i - \Phi_{i-1}\right)$$

Potential Method in Detail

- c_i is the actual cost of operation i
- $\widehat{c}_i$ is the amortized cost of operation i
- Φ_i is the potential stored after operation i ($\Phi_0 = 0$)

$$\widehat{c}_i = c_i + (\Phi_i - \Phi_{i-1})$$

- PUSH(): $c_i = 1$
- POP: $c_i = 1$

- PUSH(): $\Phi_i - \Phi_{i-1} = 1$
- POP: $\Phi_i - \Phi_{i-1} = -1$

Potential Method in Detail

- c_i is the actual cost of operation i
- $\widehat{c}_i$ is the amortized cost of operation i
- Φ_i is the potential stored after operation i ($\Phi_0 = 0$)

$$\widehat{c}_i = c_i + \left(\Phi_i - \Phi_{i-1}\right)$$

Potential Method in Detail

- c_i is the actual cost of operation i
- $\widehat{c}_i$ is the amortized cost of operation i
- Φ_i is the potential stored after operation i ($\Phi_0 = 0$)

$$\widehat{c}_i = c_i + \left(\Phi_i - \Phi_{i-1}\right)$$

$$\sum_{i=1}^{n} \widehat{c}_i = \sum_{i=1}^{n} \left(c_i + \Phi_i - \Phi_{i-1}\right) =$$

Potential Method in Detail

- c_i is the actual cost of operation i
- $\widehat{c}_i$ is the amortized cost of operation i
- Φ_i is the potential stored after operation i ($\Phi_0 = 0$)

$$\widehat{c}_i = c_i + \left(\Phi_i - \Phi_{i-1}\right)$$

$$\sum_{i=1}^{n} \widehat{c}_i = \sum_{i=1}^{n} (c_i + \Phi_i - \Phi_{i-1}) = \sum_{i=1}^{n} c_i + \Phi_n - \Phi_0$$

Potential Method in Detail

- c_i is the actual cost of operation i
- $\widehat{c}_i$ is the amortized cost of operation i
- Φ_i is the potential stored after operation i ($\Phi_0 = 0$)

$$\widehat{c}_i = c_i + \left(\Phi_i - \Phi_{i-1}\right)$$

$$\sum_{i=1}^{n} \widehat{c}_i = \sum_{i=1}^{n} \left(c_i + \Phi_i - \Phi_{i-1}\right) = \sum_{i=1}^{n} c_i + \Phi_n$$

Potential Method in Detail

- c_i is the actual cost of operation i
- $\widehat{c}_i$ is the amortized cost of operation i
- Φ_i is the potential stored after operation i ($\Phi_0 = 0$)

$$\widehat{c}_i = c_i + (\Phi_i - \Phi_{i-1})$$

$$\sum_{i=1}^{n} \widehat{c}_i = \sum_{i=1}^{n} (c_i + \Phi_i - \Phi_{i-1}) = \sum_{i=1}^{n} c_i + \Phi_n$$

If $\Phi_n \geq 0$ for all n, sum of amortized costs is an upper bound for the sum of actual costs!

Stack: Analysis via Potential Method

$\Phi_i =$

Stack: Analysis via Potential Method

Φ_i = # objects in the stack after ith operation (= # coins)

Stack: Analysis via Potential Method

$\Phi_i = \#$ objects in the stack after ith operation $(= \#$ coins$)$

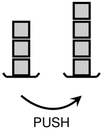
PUSH

Stack: Analysis via Potential Method

Φ_i = # objects in the stack after ith operation (= # coins)

PUSH

- actual cost: $c_i = 1$

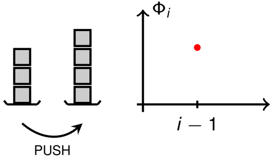


Stack: Analysis via Potential Method

Φ_i = # objects in the stack after ith operation (= # coins)

PUSH

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} =$

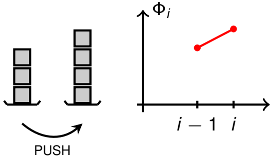


Stack: Analysis via Potential Method

$\Phi_i = \#$ objects in the stack after ith operation $(= \#$ coins$)$

PUSH

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$

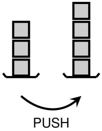


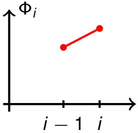
Stack: Analysis via Potential Method

Φ_i = # objects in the stack after ith operation (= # coins)

PUSH

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} =$



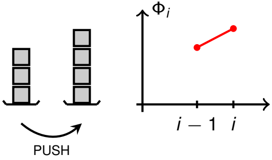


Stack: Analysis via Potential Method

$\Phi_i = \#$ objects in the stack after ith operation $(= \#$ coins$)$

PUSH

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) =$

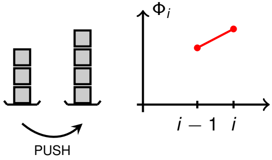


Stack: Analysis via Potential Method

Φ_i = # objects in the stack after ith operation (= # coins)

PUSH

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$

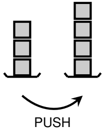


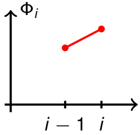
Stack: Analysis via Potential Method

Φ_i = # objects in the stack after ith operation (= # coins)

PUSH

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$





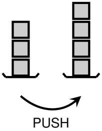
POP

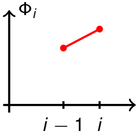
Stack: Analysis via Potential Method

$\Phi_i = \#$ objects in the stack after ith operation $(= \#$ coins$)$

PUSH

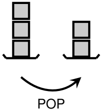
- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$





POP

- $c_i = 1$

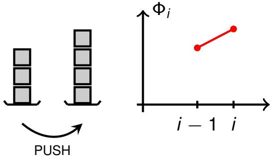


Stack: Analysis via Potential Method

$\Phi_i = \#$ objects in the stack after ith operation $\quad (= \#$ coins$)$

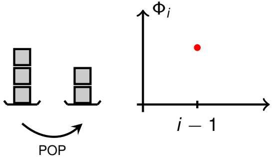
PUSH

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$



POP

- $c_i = 1$
- $\Phi_i - \Phi_{i-1} =$

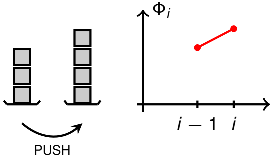


Stack: Analysis via Potential Method

Φ_i = # objects in the stack after ith operation (= # coins)

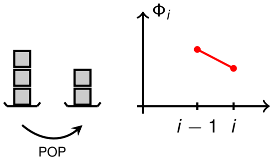
PUSH

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$



POP

- $c_i = 1$
- $\Phi_i - \Phi_{i-1} = -1$

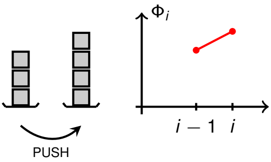


Stack: Analysis via Potential Method

Φ_i = # objects in the stack after ith operation (= # coins)

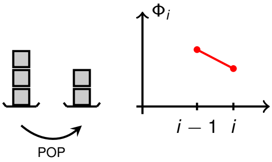
PUSH

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$



POP

- $c_i = 1$
- $\Phi_i - \Phi_{i-1} = -1$
- $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) =$

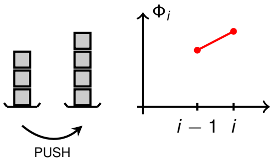


Stack: Analysis via Potential Method

Φ_i = # objects in the stack after ith operation (= # coins)

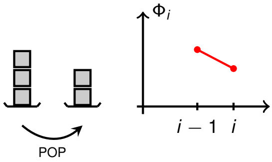
PUSH

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$



POP

- $c_i = 1$
- $\Phi_i - \Phi_{i-1} = -1$
- $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 - 1 = 0$

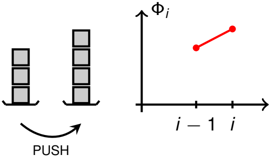


Stack: Analysis via Potential Method

$\Phi_i = \#$ objects in the stack after ith operation $(= \#$ coins$)$

PUSH

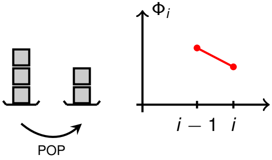
- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$



POP

- $c_i = 1$
- $\Phi_i - \Phi_{i-1} = -1$
- $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 - 1 = 0$

Stack is non-empty!

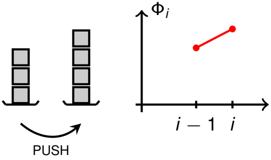


Stack: Analysis via Potential Method

Φ_i = # objects in the stack after ith operation (= # coins)

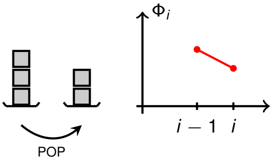
PUSH

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$



POP

- $c_i = 1$
- $\Phi_i - \Phi_{i-1} = -1$
- $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 - 1 = 0$



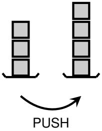
MULTIPOP(k)

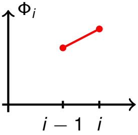
Stack: Analysis via Potential Method

$\Phi_i = \#$ objects in the stack after ith operation $(= \#$ coins$)$

PUSH

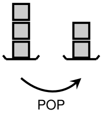
- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$

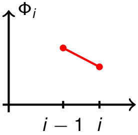




POP

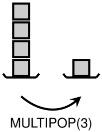
- $c_i = 1$
- $\Phi_i - \Phi_{i-1} = -1$
- $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 - 1 = 0$





MULTIPOP(k)

- $c_i = \min\{k, |S|\}$

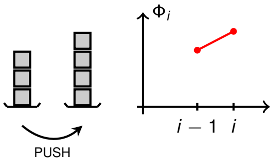


Stack: Analysis via Potential Method

Φ_i = # objects in the stack after ith operation (= # coins)

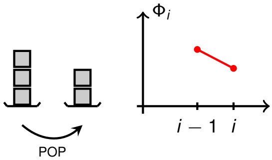
PUSH

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$



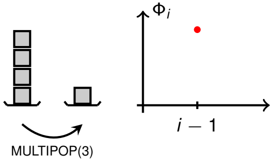
POP

- $c_i = 1$
- $\Phi_i - \Phi_{i-1} = -1$
- $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 - 1 = 0$



MULTIPOP(k)

- $c_i = \min\{k, |S|\}$
- $\Phi_i - \Phi_{i-1} =$

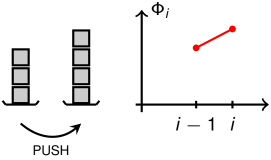


Stack: Analysis via Potential Method

$\Phi_i = \#$ objects in the stack after ith operation $(= \#$ coins$)$

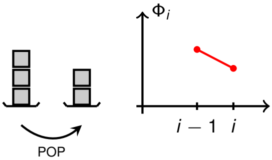
PUSH

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$



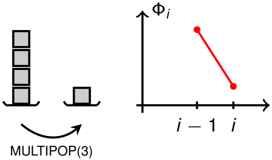
POP

- $c_i = 1$
- $\Phi_i - \Phi_{i-1} = -1$
- $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 - 1 = 0$



MULTIPOP(k)

- $c_i = \min\{k, |S|\}$
- $\Phi_i - \Phi_{i-1} = -\min\{k, |S|\}$

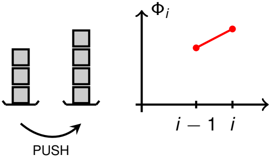


Stack: Analysis via Potential Method

$\Phi_i = \#$ objects in the stack after ith operation $(= \#$ coins$)$

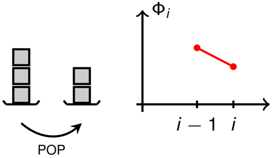
PUSH

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$



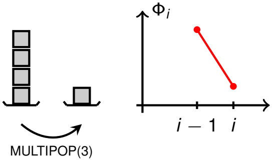
POP

- $c_i = 1$
- $\Phi_i - \Phi_{i-1} = -1$
- $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 - 1 = 0$



MULTIPOP(k)

- $c_i = \min\{k, |S|\}$
- $\Phi_i - \Phi_{i-1} = -\min\{k, |S|\}$
- $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) =$

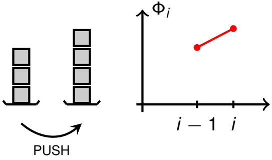


Stack: Analysis via Potential Method

$\Phi_i = \#$ objects in the stack after ith operation $\quad (= \#$ coins$)$

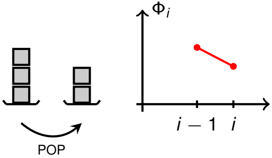
PUSH

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$



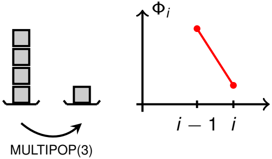
POP

- $c_i = 1$
- $\Phi_i - \Phi_{i-1} = -1$
- $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 - 1 = 0$



MULTIPOP(k)

- $c_i = \min\{k, |S|\}$
- $\Phi_i - \Phi_{i-1} = -\min\{k, |S|\}$
- $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = \min\{k, |S|\} - \min\{k, |S|\} = 0$

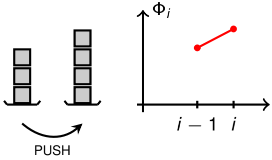


Stack: Analysis via Potential Method

Φ_i = # objects in the stack after ith operation (= # coins)

PUSH

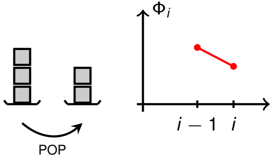
- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$



Amortized Cost $\leq 2 \Rightarrow T(n) \leq 2n$

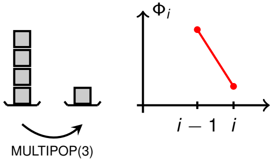
POP

- $c_i = 1$
- $\Phi_i - \Phi_{i-1} = -1$
- $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 - 1 = 0$



MULTIPOP(k)

- $c_i = \min\{k, |S|\}$
- $\Phi_i - \Phi_{i-1} = -\min\{k, |S|\}$
- $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = \min\{k, |S|\} - \min\{k, |S|\} = 0$

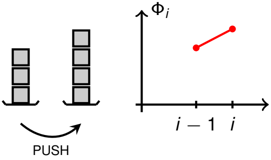


Stack: Analysis via Potential Method

$\Phi_i = \#$ objects in the stack after ith operation $(= \#$ coins$)$

PUSH

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$

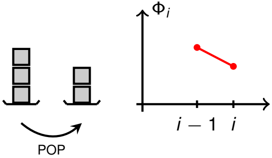


Amortized Cost $\leq 2 \Rightarrow T(n) \leq 2n$

$n/2$ PUSH, $n/2$ POP $\Rightarrow T(n) \leq n$

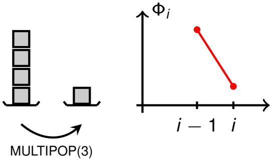
POP

- $c_i = 1$
- $\Phi_i - \Phi_{i-1} = -1$
- $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 - 1 = 0$



MULTIPOP(k)

- $c_i = \min\{k, |S|\}$
- $\Phi_i - \Phi_{i-1} = -\min\{k, |S|\}$
- $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = \min\{k, |S|\} - \min\{k, |S|\} = 0$



Second Example: Binary Counter

Binary Counter

- Array $A[k-1], A[k-2], \ldots, A[0]$ of k bits
- Use array for counting from 0 to $2^k - 1$

Second Example: Binary Counter

Binary Counter

- Array $A[k-1], A[k-2], \ldots, A[0]$ of k bits
- Use array for counting from 0 to $2^k - 1$

$A[3]$	$A[2]$	$A[1]$	$A[0]$	
1	0	1	1	11

Second Example: Binary Counter

Binary Counter

- Array $A[k-1], A[k-2], \ldots, A[0]$ of k bits
- Use array for counting from 0 to $2^k - 1$
- only operation: **`INC`**

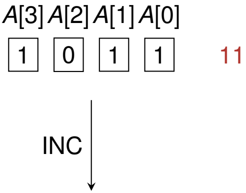
$A[3]$	$A[2]$	$A[1]$	$A[0]$	
1	0	1	1	11

INC $\downarrow$

Second Example: Binary Counter

Binary Counter

- Array $A[k-1], A[k-2], \ldots, A[0]$ of k bits
- Use array for counting from 0 to $2^k - 1$
- only operation: **`INC`**
  - increases the counter by one



Second Example: Binary Counter

Binary Counter

- Array $A[k-1], A[k-2], \ldots, A[0]$ of k bits
- Use array for counting from 0 to $2^k - 1$
- only operation: **`INC`**
  - increases the counter by one

$A[3]$	$A[2]$	$A[1]$	$A[0]$	
1	0	1	1	11

INC $\downarrow$

$A[3]$	$A[2]$	$A[1]$	$A[0]$	
1	1	0	0	12

Second Example: Binary Counter

Binary Counter

- Array $A[k-1], A[k-2], \ldots, A[0]$ of k bits
- Use array for counting from 0 to $2^k - 1$
- only operation: **`INC`**
  - increases the counter by one

```
INC(A)
i = 0
while i < k and A[i]==1
    A[i] = 0
    i = i + 1
A[i] = 1
```

$A[3]$	$A[2]$	$A[1]$	$A[0]$	
1	0	1	1	11

$\big\downarrow$ INC

$A[3]$	$A[2]$	$A[1]$	$A[0]$	
1	1	0	0	12

Second Example: Binary Counter

Binary Counter

- Array $A[k-1], A[k-2], \ldots, A[0]$ of k bits
- Use array for counting from 0 to $2^k - 1$
- only operation: **`INC`**
  - increases the counter by one
  - total cost: ??

```
INC(A)
i = 0
while i < k and A[i]==1
    A[i] = 0
    i = i + 1
A[i] = 1
```

$A[3]$	$A[2]$	$A[1]$	$A[0]$	
1	0	1	1	11

$\Big\downarrow$ INC

$A[3]$	$A[2]$	$A[1]$	$A[0]$	
1	1	0	0	12

Second Example: Binary Counter

Binary Counter

- Array $A[k-1], A[k-2], \ldots, A[0]$ of k bits
- Use array for counting from 0 to $2^k - 1$
- only operation: **`INC`**
  - increases the counter by one
  - total cost: $\leq k$

$$\begin{array}{cccccc} A[3] & A[2] & A[1] & A[0] & & \\ \boxed{1} & \boxed{0} & \boxed{1} & \boxed{1} & & 11 \end{array}$$

$$\Big\downarrow \text{INC}$$

$$\begin{array}{cccccc} A[3] & A[2] & A[1] & A[0] & & \\ \boxed{1} & \boxed{1} & \boxed{0} & \boxed{0} & & 12 \end{array}$$

Second Example: Binary Counter

Binary Counter

- Array $A[k-1], A[k-2], \ldots, A[0]$ of k bits
- Use array for counting from 0 to $2^k - 1$
- only operation: **`INC`**
  - increases the counter by one
  - total cost: $\leq k$

$A[3]$	$A[2]$	$A[1]$	$A[0]$	
1	0	1	1	11

$\downarrow$ INC

$A[3]$	$A[2]$	$A[1]$	$A[0]$	
1	1	0	0	12

What is the total cost of a sequence of n INC operations?

Second Example: Binary Counter

Binary Counter

- Array $A[k-1], A[k-2], \ldots, A[0]$ of k bits
- Use array for counting from 0 to $2^k - 1$
- only operation: **`INC`**
  - increases the counter by one
  - total cost: $\leq k$

$A[3]$	$A[2]$	$A[1]$	$A[0]$	
$\boxed{1}$	$\boxed{0}$	$\boxed{1}$	$\boxed{1}$	11

$\downarrow$ INC

$A[3]$	$A[2]$	$A[1]$	$A[0]$	
$\boxed{1}$	$\boxed{1}$	$\boxed{0}$	$\boxed{0}$	12

What is the total cost of a sequence of n INC operations?

Simple Worst-Case Bound:

- largest cost of an operation: k
- cost is at most $n \cdot k$

Second Example: Binary Counter

Binary Counter

- Array $A[k-1], A[k-2], \ldots, A[0]$ of k bits
- Use array for counting from 0 to $2^k - 1$
- only operation: **INC**
  - increases the counter by one
  - total cost: $\leq k$

$A[3]$	$A[2]$	$A[1]$	$A[0]$	
1	0	1	1	11

$\downarrow$ INC

$A[3]$	$A[2]$	$A[1]$	$A[0]$	
1	1	0	0	12

What is the total cost of a sequence of n INC operations?

Simple Worst-Case Bound:

- largest cost of an operation: k
- cost is at most $n \cdot k$ **(correct, but not tight!)**

Second Example: Binary Counter

Binary Counter

- Array $A[k-1], A[k-2], \ldots, A[0]$ of k bits
- Use array for counting from 0 to $2^k - 1$
- only operation: **`INC`**
  - increases the counter by one
  - total cost: $\cancel{\leq k}$
    number of flips (smallest index of a zero)

$A[3]$	$A[2]$	$A[1]$	$A[0]$	
1	0	1	1	11

$\Big\downarrow$ INC

$A[3]$	$A[2]$	$A[1]$	$A[0]$	
1	1	0	0	12

What is the total cost of a sequence of n INC operations?

Simple Worst-Case Bound:

- largest cost of an operation: k
- cost is at most $n \cdot k$ **(correct, but not tight!)**

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26
16	0	0	0	1	0	0	0	0	31

Incrementing a Binary Counter

Counter Value	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26
16	0	0	0	1	0	0	0	0	31

Incrementing a Binary Counter: Aggregate Analysis

Counter Value	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	3
3	0	0	1	1	4
4	0	1	0	0	7
5	0	1	0	1	8
6	0	1	1	0	10
7	0	1	1	1	11

Incrementing a Binary Counter: Aggregate Analysis

Counter Value	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	3
3	0	0	1	1	4
4	0	1	0	0	7
5	0	1	0	1	8
6	0	1	1	0	10
7	0	1	1	1	11

Incrementing a Binary Counter: Aggregate Analysis

Counter Value	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	3
3	0	0	1	1	4
4	0	1	0	0	7
5	0	1	0	1	8
6	0	1	1	0	10
7	0	1	1	1	11

Incrementing a Binary Counter: Aggregate Analysis

Counter Value	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	3
3	0	0	1	1	4
4	0	1	0	0	7
5	0	1	0	1	8
6	0	1	1	0	10
7	0	1	1	1	11

Incrementing a Binary Counter: Aggregate Analysis

Counter Value	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	3
3	0	0	1	1	4
4	0	1	0	0	7
5	0	1	0	1	8
6	0	1	1	0	10
7	0	1	1	1	11

Incrementing a Binary Counter: Aggregate Analysis

Counter Value	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	3
3	0	0	1	1	4
4	0	1	0	0	7
5	0	1	0	1	8
6	0	1	1	0	10
7	0	1	1	1	11

- Bit $A[i]$ is only flipped every 2^i increments

Incrementing a Binary Counter: Aggregate Analysis

Counter Value	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	3
3	0	0	1	1	4
4	0	1	0	0	7
5	0	1	0	1	8
6	0	1	1	0	10
7	0	1	1	1	11

- Bit $A[i]$ is only flipped every 2^i increments
- In a sequence of n increments from 0, bit $A[i]$ is flipped $\lfloor \frac{n}{2^i} \rfloor$ times

Incrementing a Binary Counter: Aggregate Analysis

Counter Value	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	3
3	0	0	1	1	4
4	0	1	0	0	7
5	0	1	0	1	8
6	0	1	1	0	10
7	0	1	1	1	11

- Bit $A[i]$ is only flipped every 2^i increments
- In a sequence of n increments from 0, bit $A[i]$ is flipped $\lfloor \frac{n}{2^i} \rfloor$ times

$$T(n) \leq$$

Incrementing a Binary Counter: Aggregate Analysis

Counter Value	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	3
3	0	0	1	1	4
4	0	1	0	0	7
5	0	1	0	1	8
6	0	1	1	0	10
7	0	1	1	1	11

- Bit $A[i]$ is only flipped every 2^i increments
- In a sequence of n increments from 0, bit $A[i]$ is flipped $\lfloor \frac{n}{2^i} \rfloor$ times

$$T(n) \leq \sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor$$

Incrementing a Binary Counter: Aggregate Analysis

Counter Value	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	3
3	0	0	1	1	4
4	0	1	0	0	7
5	0	1	0	1	8
6	0	1	1	0	10
7	0	1	1	1	11

- Bit $A[i]$ is only flipped every 2^i increments
- In a sequence of n increments from 0, bit $A[i]$ is flipped $\lfloor \frac{n}{2^i} \rfloor$ times

$$T(n) \leq \sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor \leq \sum_{i=0}^{k-1} \frac{n}{2^i}$$

Incrementing a Binary Counter: Aggregate Analysis

Counter Value	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	3
3	0	0	1	1	4
4	0	1	0	0	7
5	0	1	0	1	8
6	0	1	1	0	10
7	0	1	1	1	11

- Bit $A[i]$ is only flipped every 2^i increments
- In a sequence of n increments from 0, bit $A[i]$ is flipped $\lfloor \frac{n}{2^i} \rfloor$ times

$$T(n) \leq \sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor \leq \sum_{i=0}^{k-1} \frac{n}{2^i} = n \cdot \left(1 + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^{k-1}} \right)$$

Incrementing a Binary Counter: Aggregate Analysis

Counter Value	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	3
3	0	0	1	1	4
4	0	1	0	0	7
5	0	1	0	1	8
6	0	1	1	0	10
7	0	1	1	1	11

- Bit $A[i]$ is only flipped every 2^i increments
- In a sequence of n increments from 0, bit $A[i]$ is flipped $\lfloor \frac{n}{2^i} \rfloor$ times

$$T(n) \leq \sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor \leq \sum_{i=0}^{k-1} \frac{n}{2^i} = n \cdot \left(1 + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^{k-1}}\right) \leq 2 \cdot n.$$

Incrementing a Binary Counter: Aggregate Analysis

Counter Value	$A[3]$	$A[2]$	$A[1]$	$A[0]$	Total Cost
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	3
3	0	0	1	1	4
4	0	1	0	0	7
5	0	1	0	1	8
6	0	1	1	0	10
7	0	1	1	1	11

- Bit $A[i]$ is only flipped every 2^i increments
- In a sequence of n increments from 0, bit $A[i]$ is flipped $\lfloor \frac{n}{2^i} \rfloor$ times

Aggregate Analysis: The amortized cost per operation is $\frac{T(n)}{n} \leq 2$.

$$T(n) \leq \sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor \leq \sum_{i=0}^{k-1} \frac{n}{2^i} = n \cdot \left(1 + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^{k-1}}\right) \leq 2 \cdot n.$$

Binary Counter: Analysis via Potential Function

$$\Phi_i =$$

Binary Counter: Analysis via Potential Function

$\Phi_i = \#$ ones in the binary representation of i

Binary Counter: Analysis via Potential Function

$\Phi_0 = 0$ ✓ $\Phi_i \geq 0$ ✓

$\Phi_i = \#$ ones in the binary representation of i

Binary Counter: Analysis via Potential Function

$\Phi_0 = 0$ ✓ $\Phi_i \geq 0$ ✓

$\Phi_i = \#$ ones in the binary representation of i

Increment without Carry-Over

Binary Counter: Analysis via Potential Function

$\Phi_0 = 0$ ✓ $\Phi_i \geq 0$ ✓

$\Phi_i = $ # ones in the binary representation of i

Increment without Carry-Over

- actual cost: $c_i = 1$

1 1 0 0

Binary Counter: Analysis via Potential Function

$\Phi_0 = 0$ ✓ $\Phi_i \geq 0$ ✓

$\Phi_i = $ # ones in the binary representation of i

Increment without Carry-Over

- actual cost: $c_i = 1$

$$1\ 1\ 0\ \boxed{0} \xrightarrow{\text{INC}} 1\ 1\ 0\ 1$$

Binary Counter: Analysis via Potential Function

$\Phi_0 = 0$ ✓ $\Phi_i \geq 0$ ✓

$\Phi_i =$ # ones in the binary representation of i

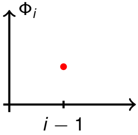
Increment without Carry-Over

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} =$

1 1 0 0

$\big\downarrow$ INC

1 1 0 1



Binary Counter: Analysis via Potential Function

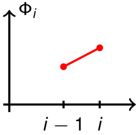
$\Phi_0 = 0 \checkmark \quad \Phi_i \geq 0 \checkmark$

$\Phi_i = \#$ ones in the binary representation of i

Increment without Carry-Over

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$

$$\begin{array}{c} 1\ 1\ 0\ \boxed{0} \\ \downarrow \text{INC} \\ 1\ 1\ 0\ 1 \end{array}$$



Binary Counter: Analysis via Potential Function

$\Phi_0 = 0$ ✓ $\Phi_i \geq 0$ ✓

$\Phi_i =$ # ones in the binary representation of i

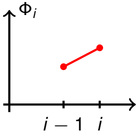
Increment without Carry-Over

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} =$

1 1 0 0

$\downarrow$ INC

1 1 0 1



Binary Counter: Analysis via Potential Function

$\Phi_0 = 0$ ✓ $\Phi_i \geq 0$ ✓

$\Phi_i = $ # ones in the binary representation of i

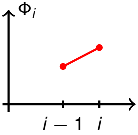
Increment without Carry-Over

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c}_i = c_i + (\Phi_i - \Phi_{i-1}) =$

1 1 0 0

$\downarrow$ INC

1 1 0 1



Binary Counter: Analysis via Potential Function

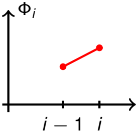
$\Phi_0 = 0$ ✓ $\Phi_i \geq 0$ ✓

$\Phi_i = \#$ ones in the binary representation of i

Increment without Carry-Over

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$

$$\begin{array}{c} 1\ 1\ 0\ \boxed{0} \\ \big\downarrow \text{INC} \\ 1\ 1\ 0\ 1 \end{array}$$



Binary Counter: Analysis via Potential Function

$\Phi_0 = 0$ ✓ $\Phi_i \geq 0$ ✓

$\Phi_i = \#$ ones in the binary representation of i

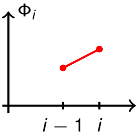
Increment without Carry-Over

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$

1 1 0 0

$\downarrow$ INC

1 1 0 1



Increment with Carry-Over

Binary Counter: Analysis via Potential Function

$\Phi_0 = 0$ ✓ $\Phi_i \geq 0$ ✓

$\Phi_i =$ # ones in the binary representation of i

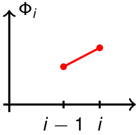
Increment without Carry-Over

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$

1 1 0 0

$\downarrow$ INC

1 1 0 1



0 1 1 1

Increment with Carry-Over

- $c_i = x + 1$, (x lowest index of a zero)

Binary Counter: Analysis via Potential Function

$\Phi_0 = 0$ ✓ $\Phi_i \geq 0$ ✓

Φ_i = # ones in the binary representation of i

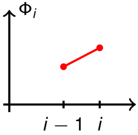
Increment without Carry-Over

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$

1 1 0 0

$\downarrow$ INC

1 1 0 1



Increment with Carry-Over

- $c_i = x + 1$, (x lowest index of a zero)

0 1 1 1

$\downarrow$ INC

1 0 0 0

Binary Counter: Analysis via Potential Function

$\Phi_0 = 0$ ✓ $\Phi_i \geq 0$ ✓

$\Phi_i =$ # ones in the binary representation of i

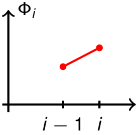
Increment without Carry-Over

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$

1 1 0 0

$\downarrow$ INC

1 1 0 1



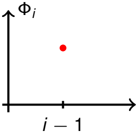
Increment with Carry-Over

- $c_i = x + 1$, (x lowest index of a zero)
- $\Phi_i - \Phi_{i-1} =$

0 1 1 1

$\downarrow$ INC

1 0 0 0



Binary Counter: Analysis via Potential Function

$\Phi_0 = 0$ ✓ $\Phi_i \geq 0$ ✓

$\Phi_i =$ # ones in the binary representation of i

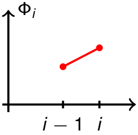
Increment without Carry-Over

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$

1 1 0 $\boxed{0}$

$\downarrow$ INC

1 1 0 1



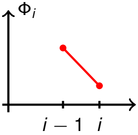
Increment with Carry-Over

- $c_i = x + 1$, (x lowest index of a zero)
- $\Phi_i - \Phi_{i-1} = -x + 1$

$\boxed{0\ 1\ 1\ 1}$

$\downarrow$ INC

1 0 0 0



Binary Counter: Analysis via Potential Function

$\Phi_0 = 0$ ✓ $\Phi_i \geq 0$ ✓

Φ_i = # ones in the binary representation of i

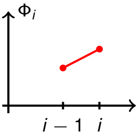
Increment without Carry-Over

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$

1 1 0 $\boxed{0}$

↓ INC

1 1 0 1



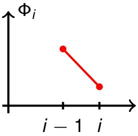
Increment with Carry-Over

- $c_i = x + 1$, (x lowest index of a zero)
- $\Phi_i - \Phi_{i-1} = -x + 1$
- $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) =$

$\boxed{0\ 1\ 1\ 1}$

↓ INC

1 0 0 0



Binary Counter: Analysis via Potential Function

$\Phi_0 = 0$ ✓ $\Phi_i \geq 0$ ✓

Φ_i = # ones in the binary representation of i

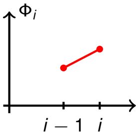
Increment without Carry-Over

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$

1 1 0 0

↓ INC

1 1 0 1



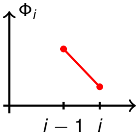
Increment with Carry-Over

- $c_i = x + 1$, (x lowest index of a zero)
- $\Phi_i - \Phi_{i-1} = -x + 1$
- $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + x - x + 1$

0 1 1 1

↓ INC

1 0 0 0



Binary Counter: Analysis via Potential Function

$\Phi_0 = 0$ ✓ $\Phi_i \geq 0$ ✓

$\Phi_i =$ # ones in the binary representation of i

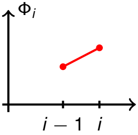
Increment without Carry-Over

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$

1 1 0 0

$\downarrow$ INC

1 1 0 1



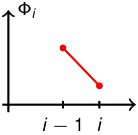
Increment with Carry-Over

- $c_i = x + 1$, (x lowest index of a zero)
- $\Phi_i - \Phi_{i-1} = -x + 1$
- $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + x - x + 1 \ = 2$

0 1 1 1

$\downarrow$ INC

1 0 0 0



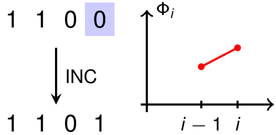
Binary Counter: Analysis via Potential Function

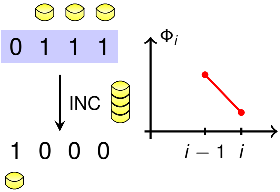
$\Phi_0 = 0$ ✓ $\Phi_i \geq 0$ ✓

Φ_i = # ones in the binary representation of i

Increment without Carry-Over

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c}_i = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$





Increment with Carry-Over

- $c_i = x + 1$, (x lowest index of a zero)
- $\Phi_i - \Phi_{i-1} = -x + 1$
- $\widehat{c}_i = c_i + (\Phi_i - \Phi_{i-1}) = 1 + x - x + 1 = 2$

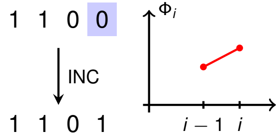
Binary Counter: Analysis via Potential Function

$\Phi_0 = 0$ ✓ $\Phi_i \geq 0$ ✓

Φ_i = # ones in the binary representation of i

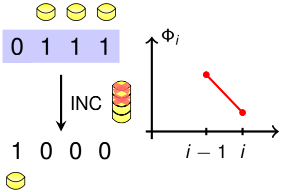
Increment without Carry-Over

- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$



Increment with Carry-Over

- $c_i = x + 1$, (x lowest index of a zero)
- $\Phi_i - \Phi_{i-1} = -x + 1$
- $\widehat{c_i} = c_i + (\Phi_i - \Phi_{i-1}) = 1 + x - x + 1 = 2$



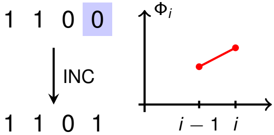
Binary Counter: Analysis via Potential Function

$\Phi_0 = 0$ ✓ $\Phi_i \geq 0$ ✓

$\Phi_i =$ # ones in the binary representation of i

Increment without Carry-Over

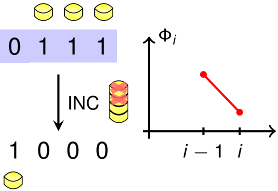
- actual cost: $c_i = 1$
- potential change: $\Phi_i - \Phi_{i-1} = 1$
- amortized cost: $\widehat{c}_i = c_i + (\Phi_i - \Phi_{i-1}) = 1 + 1 = 2$



Amortized Cost $= 2 \Rightarrow T(n) \leq 2n$

Increment with Carry-Over

- $c_i = x + 1$, (x lowest index of a zero)
- $\Phi_i - \Phi_{i-1} = -x + 1$
- $\widehat{c}_i = c_i + (\Phi_i - \Phi_{i-1}) = 1 + x - x + 1 = 2$



Summary

Amortized Analysis

- Average costs over a sequence of n operations
- overcharge cheap operations and undercharge expensive operations
- no probability/average case analysis involved!

Summary

Amortized Analysis

- Average costs over a sequence of n operations
- overcharge cheap operations and undercharge expensive operations
- no probability/average case analysis involved!

Aggregate Analysis

Summary

Amortized Analysis

- Average costs over a sequence of n operations
- overcharge cheap operations and undercharge expensive operations
- no probability/average case analysis involved!

Aggregate Analysis

- Determine an absolute upper bound $T(n)$

Summary

Amortized Analysis

- Average costs over a sequence of n operations
- overcharge cheap operations and undercharge expensive operations
- no probability/average case analysis involved!

E.g. by bounding the number of expensive operations

Aggregate Analysis

- Determine an absolute upper bound $T(n)$

Summary

Amortized Analysis

- Average costs over a sequence of n operations
- overcharge cheap operations and undercharge expensive operations
- no probability/average case analysis involved!

Aggregate Analysis

- Determine an absolute upper bound $T(n)$
- every operation has amortized cost $\frac{T(n)}{n}$

$T(n)$

Summary

Amortized Analysis

- Average costs over a sequence of n operations
- overcharge cheap operations and undercharge expensive operations
- no probability/average case analysis involved!

Aggregate Analysis

- Determine an absolute upper bound $T(n)$
- every operation has amortized cost $\frac{T(n)}{n}$

$T(n)$ 

Potential Method

Summary

Amortized Analysis

- Average costs over a sequence of n operations
- overcharge cheap operations and undercharge expensive operations
- no probability/average case analysis involved!

Aggregate Analysis

- Determine an absolute upper bound $T(n)$
- every operation has amortized cost $\frac{T(n)}{n}$

$T(n)$

Potential Method

- use savings from cheap operations to compensate for expensive ones

Summary

Amortized Analysis

- Average costs over a sequence of n operations
- overcharge cheap operations and undercharge expensive operations
- no probability/average case analysis involved!

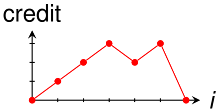
Aggregate Analysis

- Determine an absolute upper bound $T(n)$
- every operation has amortized cost $\frac{T(n)}{n}$

$T(n)$

Potential Method

- use savings from cheap operations to compensate for expensive ones



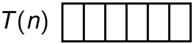
Summary

Amortized Analysis

- Average costs over a sequence of n operations
- overcharge cheap operations and undercharge expensive operations
- no probability/average case analysis involved!

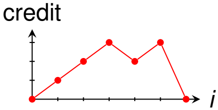
Aggregate Analysis

- Determine an absolute upper bound $T(n)$
- every operation has amortized cost $\frac{T(n)}{n}$



Potential Method

- use savings from cheap operations to compensate for expensive ones
- operations may have different amortized cost



Summary

Amortized Analysis

- Average costs over a sequence of n operations
- overcharge cheap operations and undercharge expensive operations
- no probability/average case analysis involved!

Aggregate Analysis

- Determine an absolute upper bound $T(n)$
- every operation has amortized cost $\frac{T(n)}{n}$

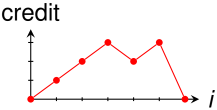
$T(n)$ 

Full power of this method will become clear later!

$T(n)$ 

Potential Method

- use savings from cheap operations to compensate for expensive ones
- operations may have different amortized cost



Next Lecture: Fibonacci Heap

Operation	Binomial heap worst-case cost
MAKE-HEAP	$\mathcal{O}(1)$
INSERT	$\mathcal{O}(\log n)$
MINIMUM	$\mathcal{O}(\log n)$
EXTRACT-MIN	$\mathcal{O}(\log n)$
UNION	$\mathcal{O}(\log n)$
DECREASE-KEY	$\mathcal{O}(\log n)$
DELETE	$\mathcal{O}(\log n)$

Next Lecture: Fibonacci Heap

Operation	Binomial heap worst-case cost	Fibonacci heap amortized cost
MAKE-HEAP	$\mathcal{O}(1)$	$\mathcal{O}(1)$
INSERT	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
MINIMUM	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
EXTRACT-MIN	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
UNION	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DECREASE-KEY	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DELETE	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

Crucial for many applications including shortest paths and minimum spanning trees!