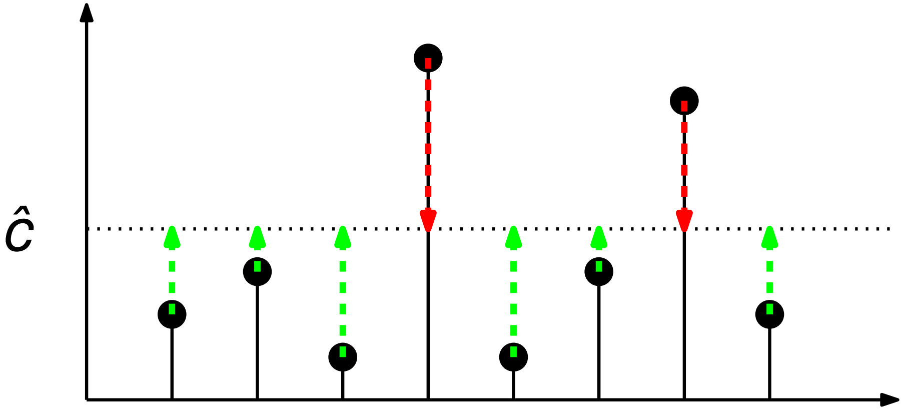
INFORMATION & COMPUTER SCIENCES
UNIVERSITY *of* HAWAIʻI *at* MĀNOA

AlgoPARC

ICS 621: Analysis of Algorithms

Prof. Nodari Sitchinava



Amortized Analysis

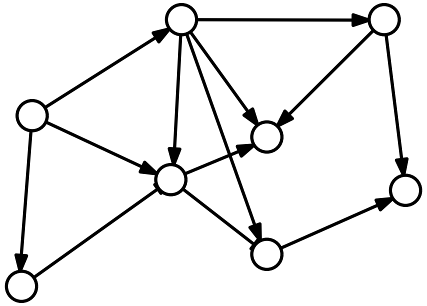
Last time: Simple analysis

```
procedure TRIPLELOOP($A, n$)
    for $k = 1$ to 10 do
        for $i = 1$ to $n$ do
            for $j = n$ downto $i$ do
                $A[i] = A[j] + 1$
```

```
function FOO($n$)
    if  then$n < 2$
        return 1
    else
        for  do$i = 1$ to $n$
            $x = x + 1$
        for  do$i = 1$ to 2
            $x = x + \text{FOO}(n/4)$
        return $x$
```

Today: More complex analysis

```
procedure ProcessGraph($G$)
    for each vertex $u \in G.V$ do
        $u.color$ = WHITE
        $u.\pi$ = NIL
    $time = 0$
    for each vertex $u \in G.V$ do
        if $u.color$ == WHITE then
            Visit($G, u$)
```

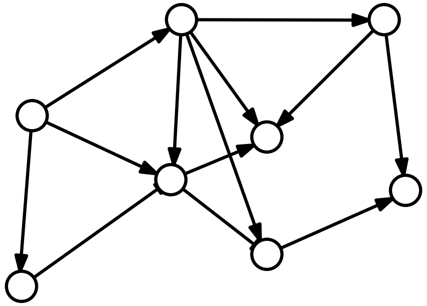


```
procedure Visit($G, u$)
    $time = time + 1$
    $u.d = time$
    $u.color$ = GRAY
    for each $v \in G.Adj[u]$ do
        if $v.color$ == WHITE then
            $v.\pi = u$
            Visit($G, v$)
    $u.color$ = BLACK
    $time = time + 1$
    $u.f = time$
```

Today: More complex analysis

```
procedure ProcessGraph($G$)
    for each vertex $u \in G.V$ do
        $u.color$ = WHITE
        $u.\pi$ = NIL
    $time = 0$
    for each vertex $u \in G.V$ do
        if $u.color$ == WHITE then
            Visit($G, u$)
```

```
procedure Visit($G, u$)
    $time = time + 1$
    $u.d = time$
    $u.color$ = GRAY
    for each $v \in G.Adj[u]$ do
        if $v.color$ == WHITE then
            $v.\pi = u$
            Visit($G, v$)
    $u.color$ = BLACK
    $time = time + 1$
    $u.f = time$
```

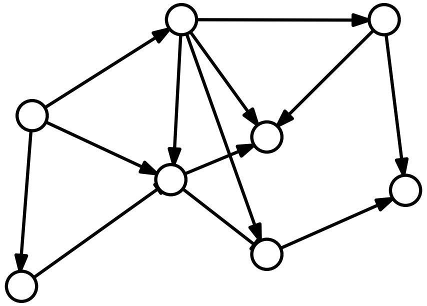


$$T(n) \leq (n-1) \cdot T(n-1) + O(1)$$

Today: More complex analysis

```
procedure ProcessGraph($G$)
    for each vertex $u \in G.V$ do
        $u.color$ = WHITE
        $u.\pi$ = NIL
    $time = 0$
    for each vertex $u \in G.V$ do
        if $u.color$ == WHITE then
            Visit($G, u$)
```

```
procedure Visit($G, u$)
    $time = time + 1$
    $u.d = time$
    $u.color$ = GRAY
    for each $v \in G.Adj[u]$ do
        if $v.color$ == WHITE then
            $v.\pi = u$
            Visit($G, v$)
    $u.color$ = BLACK
    $time = time + 1$
    $u.f = time$
```



$$T(n) \leq (n-1) \cdot T(n-1) + O(1) \leq \Theta(n^2)$$

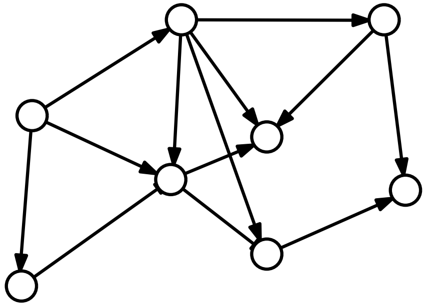
More precise analysis

of vertices

$$\sum_{i=1}^{n}(1 + d_i) = n + \sum_{i=1}^{n} d_i = n + m$$

out-degree of
the *i*-th vertex

# of edges



Amortized analysis

If *i*-th operation takes time (”costs”) c_i time steps, then total cost of an algorithm with *n* operations:

$$T(n) = \sum_{i=1}^{n} c_i$$

Amortized analysis

If i-th operation takes time (”costs”) c_i time steps, then total cost of an algorithm with n operations:

$$T(n) = \sum_{i=1}^{n} c_i$$

Amortized cost is the “average” cost of each operation:

$$\hat{c} = \frac{T(n)}{n}$$

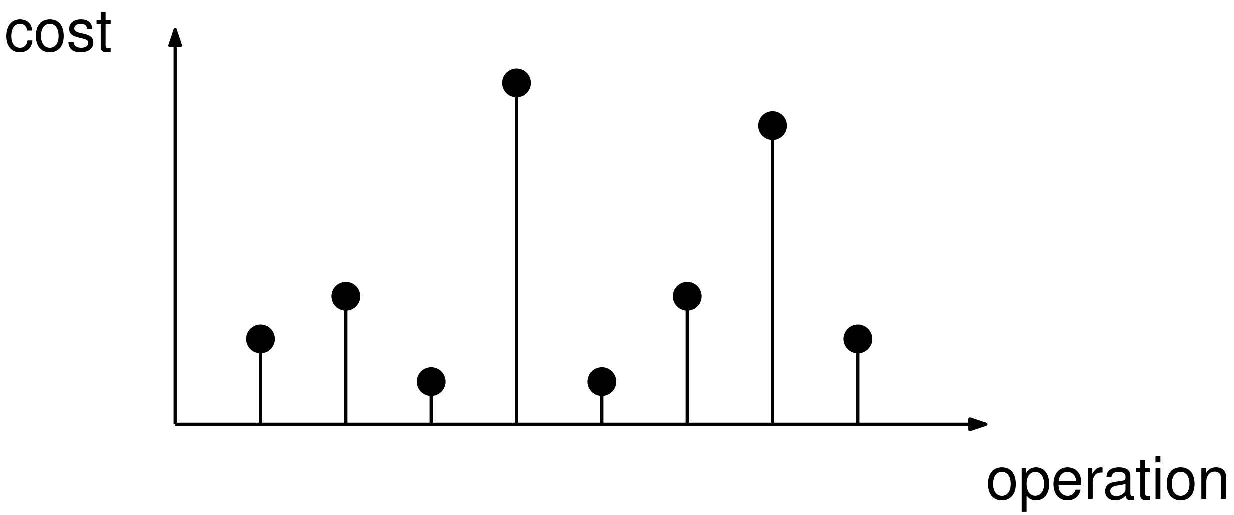
Amortized analysis

If i-th operation takes time (”costs”) c_i time steps, then total cost of an algorithm with n operations:

$$T(n) = \sum_{i=1}^{n} c_i$$

Amortized cost is the “average” cost of each operation:

$$\hat{c} = \frac{T(n)}{n}$$



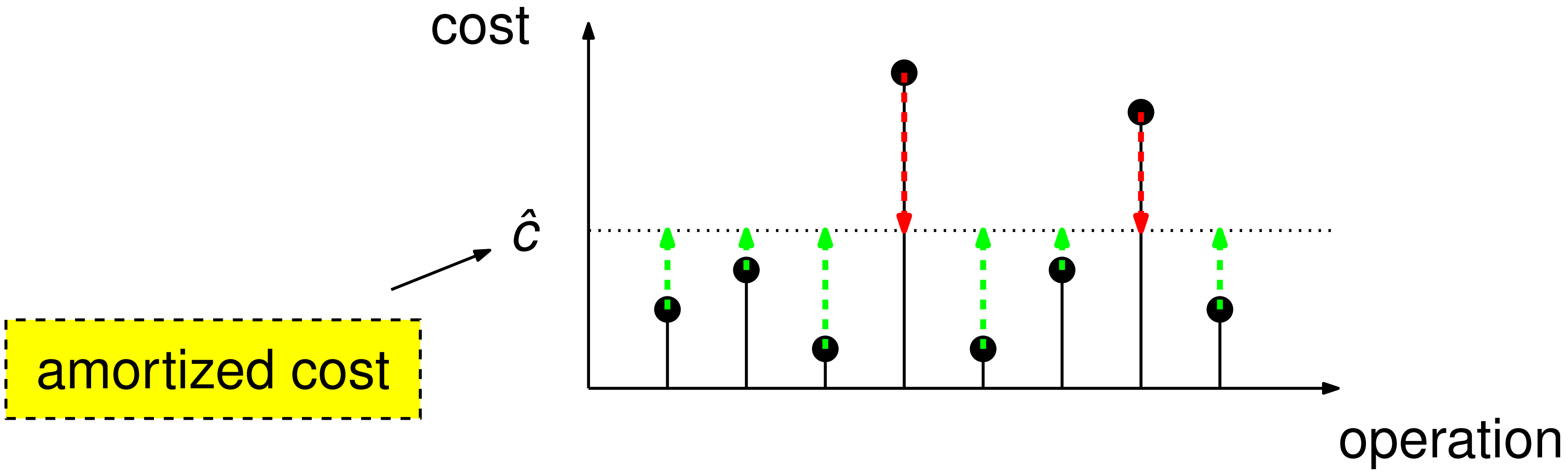
Amortized analysis

If i-th operation takes time (”costs”) c_i time steps, then total cost of an algorithm with n operations:

$$T(n) = \sum_{i=1}^{n} c_i$$

Amortized cost is the “average” cost of each operation:

$$\hat{c} = \frac{T(n)}{n}$$



Amortized cost $\neq$ average cost = expected cost

Amortized cost $\neq$ average cost = expected cost

Amortized cost is
worst-case upper bound

Expected cost is for
randomized algorithms

Amortized cost $\neq$ average cost = expected cost

Amortized cost is worst-case upper bound

Expected cost is for randomized algorithms

Examples

$$\boxed{\hat{c} = \Theta(\log n)}$$

$$T(n) \leq \sum_{i=1}^{n} \hat{c} = \Theta(n \log n)$$

worst-case

Amortized cost $\neq$ average cost = expected cost

Amortized cost is
worst-case upper bound

Expected cost is for
randomized algorithms

Examples

$\hat{c} = \Theta(\log n)$

$$T(n) \leq \sum_{i=1}^{n} \hat{c} = \Theta(n \log n)$$

worst-case

Quicksort

$$T(n) = \Theta(n \log n)$$

expected

Exist inputs for which
$T(n) = \Theta(n^2)$

Amortized analysis: Aggregate analysis

$$\boxed{\hat{c} = \frac{1}{n} \cdot \sum_{i=1}^{n} c_i}$$

Amortized analysis: Aggregate analysis

$$\hat{c} = \frac{1}{n} \cdot \sum_{i=1}^{n} c_i$$

Example: Binary Counter

B = array of k bits

Operation: INCREMENT(B, k)

000000

Cost: number of bits flipped

Amortized analysis: Aggregate analysis

$$\boxed{\hat{c} = \frac{1}{n} \cdot \sum_{i=1}^{n} c_i}$$

Example: Binary Counter

B = array of k bits

000000

Operation: INCREMENT(B, k)

```
procedure INCREMENT$(B, k)$
    $i = 0$
    while $i < k$ and $B[i] == 1$ do
        $B[i] = 0$
        $i = i + 1$
    if $i < k$ then
        $B[i] = 1$
```

Cost: number of bits flipped

Amortized analysis: Aggregate analysis

$$\hat{c} = \frac{1}{n} \cdot \sum_{i=1}^{n} c_i$$

Example: Binary Counter

B = array of k bits

Operation: INCREMENT(B, k)

000000 $\big\downarrow$ 1
000001

```
procedure INCREMENT($B$, $k$)
    $i = 0$
    while $i < k$ and $B[i] == 1$ do
        $B[i] = 0$
        $i = i + 1$
    if $i < k$ then
        $B[i] = 1$
```

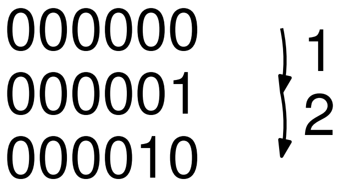
Cost: number of bits flipped

Amortized analysis: Aggregate analysis

$$\hat{c} = \frac{1}{n} \cdot \sum_{i=1}^{n} c_i$$

Example: Binary Counter

B = array of k bits



000000
000001
000010

Operation: INCREMENT(B, k)

```
procedure INCREMENT($B$, $k$)
    $i = 0$
    while $i < k$ and $B[i] == 1$ do
        $B[i] = 0$
        $i = i + 1$
    if $i < k$ then
        $B[i] = 1$
```

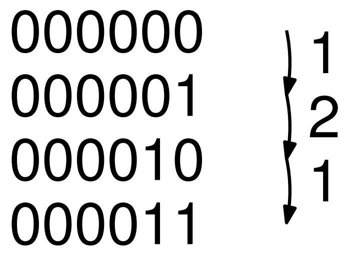
Cost: number of bits flipped

Amortized analysis: Aggregate analysis

$$\hat{c} = \frac{1}{n} \cdot \sum_{i=1}^{n} c_i$$

Example: Binary Counter

B = array of k bits



Operation: INCREMENT(B, k)

```
procedure INCREMENT($B$, $k$)
    $i = 0$
    while $i < k$ and $B[i] == 1$ do
        $B[i] = 0$
        $i = i + 1$
    if $i < k$ then
        $B[i] = 1$
```

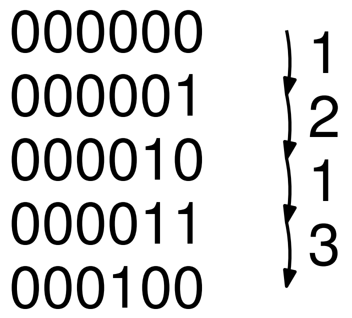
Cost: number of bits flipped

Amortized analysis: Aggregate analysis

$$\hat{c} = \frac{1}{n} \cdot \sum_{i=1}^{n} c_i$$

Example: Binary Counter

B = array of k bits



000000
000001
000010
000011
000100

Operation: INCREMENT(B, k)

```
procedure INCREMENT($B$, $k$)
    $i = 0$
    while $i < k$ and $B[i] == 1$ do
        $B[i] = 0$
        $i = i + 1$
    if $i < k$ then
        $B[i] = 1$
```

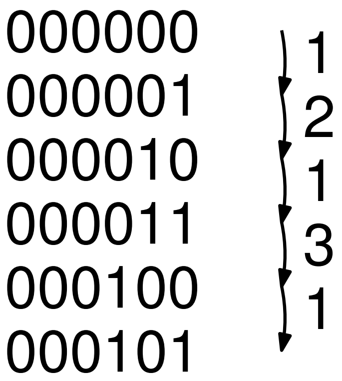
Cost: number of bits flipped

Amortized analysis: Aggregate analysis

$$\hat{c} = \frac{1}{n} \cdot \sum_{i=1}^{n} c_i$$

Example: Binary Counter

B = array of k bits



000000
000001
000010
000011
000100
000101

Operation: INCREMENT(B, k)

```
procedure INCREMENT($B$, $k$)
    $i = 0$
    while $i < k$ and $B[i] == 1$ do
        $B[i] = 0$
        $i = i + 1$
    if $i < k$ then
        $B[i] = 1$
```

Cost: number of bits flipped

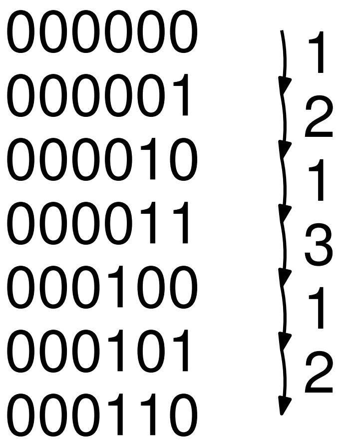
Amortized analysis: Aggregate analysis

$$\boxed{\hat{c} = \frac{1}{n} \cdot \sum_{i=1}^{n} c_i}$$

Example: Binary Counter

B = array of k bits

Operation: INCREMENT(B, k)



000000
000001
000010
000011
000100
000101
000110

```
procedure INCREMENT($B$, $k$)
    $i = 0$
    while $i < k$ and $B[i] == 1$ do
        $B[i] = 0$
        $i = i + 1$
    if $i < k$ then
        $B[i] = 1$
```

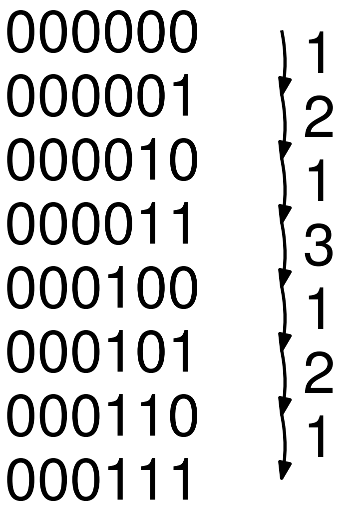
Cost: number of bits flipped

Amortized analysis: Aggregate analysis

$$\hat{c} = \frac{1}{n} \cdot \sum_{i=1}^{n} c_i$$

Example: Binary Counter

B = array of k bits



Operation: INCREMENT(B, k)

```
procedure INCREMENT($B$, $k$)
    $i = 0$
    while $i < k$ and $B[i] == 1$ do
        $B[i] = 0$
        $i = i + 1$
    if $i < k$ then
        $B[i] = 1$
```

Cost: number of bits flipped

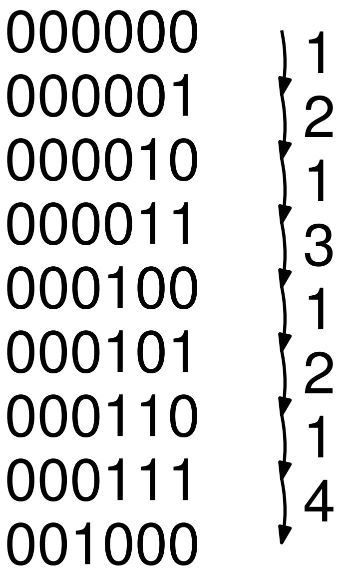
Amortized analysis: Aggregate analysis

$$\hat{c} = \frac{1}{n} \cdot \sum_{i=1}^{n} c_i$$

Example: Binary Counter

B = array of k bits

Operation: INCREMENT(B, k)



000000
000001
000010
000011
000100
000101
000110
000111
001000

```
procedure INCREMENT($B$, $k$)
    $i = 0$
    while $i < k$ and $B[i] == 1$ do
        $B[i] = 0$
        $i = i + 1$
    if $i < k$ then
        $B[i] = 1$
```

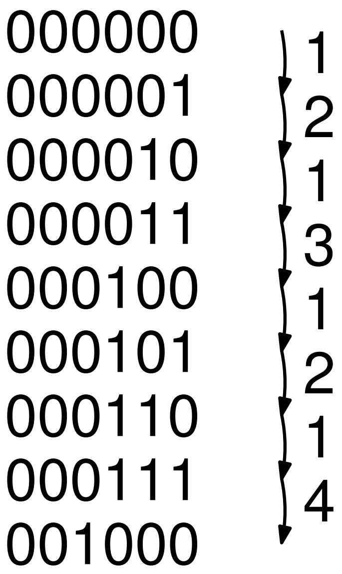
Cost: number of bits flipped

Amortized analysis: Aggregate analysis

What is the cost after *n* calls to INCREMENT(B, k)?

Example: Binary Counter

B = array of k bits



000000
000001
000010
000011
000100
000101
000110
000111
001000

Operation: INCREMENT(B, k)

```
procedure INCREMENT($B$, $k$)
    $i = 0$
    while $i < k$ and $B[i] == 1$ do
        $B[i] = 0$
        $i = i + 1$
    if $i < k$ then
        $B[i] = 1$
```

Cost: number of bits flipped

Amortized analysis: Aggregate analysis

What is the cost after n calls to INCREMENT(B, k)?

Example: Binary Counter

B = array of k bits

Operation: INCREMENT(B, k)

000000
↓ 1
000001
↓ 2
000010
↓ 1
000011
↓ 3
000100
↓ 1
000101
↓ 2
000110
↓ 1
000111
↓ 4
001000

$$T(n) \leq \sum_{i=1}^{n} k = nk = O(n \log n)$$

```
procedure INCREMENT($B, k$)
    $B[i] == 1$ do
        $i = i + 1$
    if $i < k$ then
        $B[i] = 1$
```

Cost: number of bits flipped

Example: Binary Counter

000000
000001
000010
000011
000100
000101
000110
000111
001000

```
procedure INCREMENT($B$, $k$)
    $i = 0$
    while $i < k$ and $B[i] == 1$ do
        $B[i] = 0$
        $i = i + 1$
    if $i < k$ then
        $B[i] = 1$
```

Example: Binary Counter

000000
000001
000010
000011
000100
000101
000110
000111
001000

```
procedure INCREMENT($B, k$)
    $i = 0$
    while $i < k$ and $B[i] == 1$ do
        $B[i] = 0$
        $i = i + 1$
    if $i < k$ then
        $B[i] = 1$
```

Observation: i-th bit is flipped every 2^i-th call to INCREMENT(B, k)

Example: Binary Counter

000000
000001
000010
000011
000100
000101
000110
000111
001000

```
procedure INCREMENT($B, k$)
    $i = 0$
    while $i < k$ and $B[i] == 1$ do
        $B[i] = 0$
        $i = i + 1$
    if $i < k$ then
        $B[i] = 1$
```

Observation: i-th bit is flipped every 2^i-th call to INCREMENT(B, k)

$$T(n) = \frac{n}{2^0} + \frac{n}{2^1} + \frac{n}{2^2} + \frac{n}{2^3} + \cdots + \frac{n}{2^i} + \cdots + \frac{n}{2^{k-1}}$$

Example: Binary Counter

000000
000001
000010
000011
000100
000101
000110
000111
001000

```
procedure INCREMENT($B, k$)
    $i = 0$
    while $i < k$ and $B[i] == 1$ do
        $B[i] = 0$
        $i = i + 1$
    if $i < k$ then
        $B[i] = 1$
```

Observation: i-th bit is flipped every 2^i-th call to INCREMENT(B, k)

$$\begin{aligned} T(n) &= \frac{n}{2^0} + \frac{n}{2^1} + \frac{n}{2^2} + \frac{n}{2^3} + \cdots + \frac{n}{2^i} + \cdots + \frac{n}{2^{k-1}} \\ &= n \cdot \left(\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^i} + \cdots + \frac{1}{2^{k-1}} \right) \end{aligned}$$

Example: Binary Counter

000000
000001
000010
000011
000100
000101
000110
000111
001000

```
procedure INCREMENT($B, k$)
    $i = 0$
    while $i < k$ and $B[i] == 1$ do
        $B[i] = 0$
        $i = i + 1$
    if $i < k$ then
        $B[i] = 1$
```

Observation: *i*-th bit is flipped every 2^i-th call to INCREMENT(B, k)

$$
\begin{aligned}
T(n) &= \frac{n}{2^0} + \frac{n}{2^1} + \frac{n}{2^2} + \frac{n}{2^3} + \cdots + \frac{n}{2^i} + \cdots + \frac{n}{2^{k-1}} \\
&= n \cdot \left(\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^i} + \cdots + \frac{1}{2^{k-1}} \right) \\
&= n \cdot \sum_{i=0}^{k-1} \frac{1}{2^i}
\end{aligned}
$$

Example: Binary Counter

000000
000001
000010
000011
000100
000101
000110
000111
001000

```
procedure INCREMENT($B, k$)
    $i = 0$
    while $i < k$ and $B[i] == 1$ do
        $B[i] = 0$
        $i = i + 1$
    if $i < k$ then
        $B[i] = 1$
```

Observation: i-th bit is flipped every 2^i-th call to INCREMENT(B, k)

$$\begin{aligned} T(n) &= \frac{n}{2^0} + \frac{n}{2^1} + \frac{n}{2^2} + \frac{n}{2^3} + \cdots + \frac{n}{2^i} + \cdots + \frac{n}{2^{k-1}} \\ &= n \cdot \left(\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^i} + \cdots + \frac{1}{2^{k-1}} \right) \\ &= n \cdot \sum_{i=0}^{k-1} \frac{1}{2^i} < n \cdot \sum_{i=0}^{\infty} \frac{1}{2^i} \end{aligned}$$

Example: Binary Counter

000000
000001
000010
000011
000100
000101
000110
000111
001000

```
procedure INCREMENT($B, k$)
    $i = 0$
    while $i < k$ and $B[i] == 1$ do
        $B[i] = 0$
        $i = i + 1$
    if $i < k$ then
        $B[i] = 1$
```

Observation: i-th bit is flipped every 2^i-th call to INCREMENT(B, k)

$$
\begin{aligned}
T(n) &= \frac{n}{2^0} + \frac{n}{2^1} + \frac{n}{2^2} + \frac{n}{2^3} + \cdots + \frac{n}{2^i} + \cdots + \frac{n}{2^{k-1}} \\
&= n \cdot \left(\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^i} + \cdots + \frac{1}{2^{k-1}} \right) \\
&= n \cdot \sum_{i=0}^{k-1} \frac{1}{2^i} < n \cdot \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n
\end{aligned}
$$

Example: Binary Counter

000000
000001
000010
000011
000100
000101
000110
000111
001000

```
procedure INCREMENT(B, k)
    i = 0
    while i < k and B[i] == 1 do
        B[i] = 0
        i = i + 1
    if i < k then
        B[i] = 1
```

Observation: *i*-th bit is flipped every 2^i-th call to INCREMENT(*B*, *k*)

$$
\begin{aligned}
T(n) &= \frac{n}{2^0} + \frac{n}{2^1} + \frac{n}{2^2} + \frac{n}{2^3} + \cdots + \frac{n}{2^i} + \cdots + \frac{n}{2^{k-1}} \\
&= n \cdot \left(\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^i} + \cdots + \frac{1}{2^{k-1}} \right) \\
&= n \cdot \sum_{i=0}^{k-1} \frac{1}{2^i} < n \cdot \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n
\end{aligned}
$$

$$\hat{c} = \frac{T(n)}{n} = 2$$

More precisely

When analyzing algorithms, we want an upper bound $\hat{C}$ on the total cost:

$$T(n) = \sum_{i=0}^{n} c_i \leq \hat{C}$$

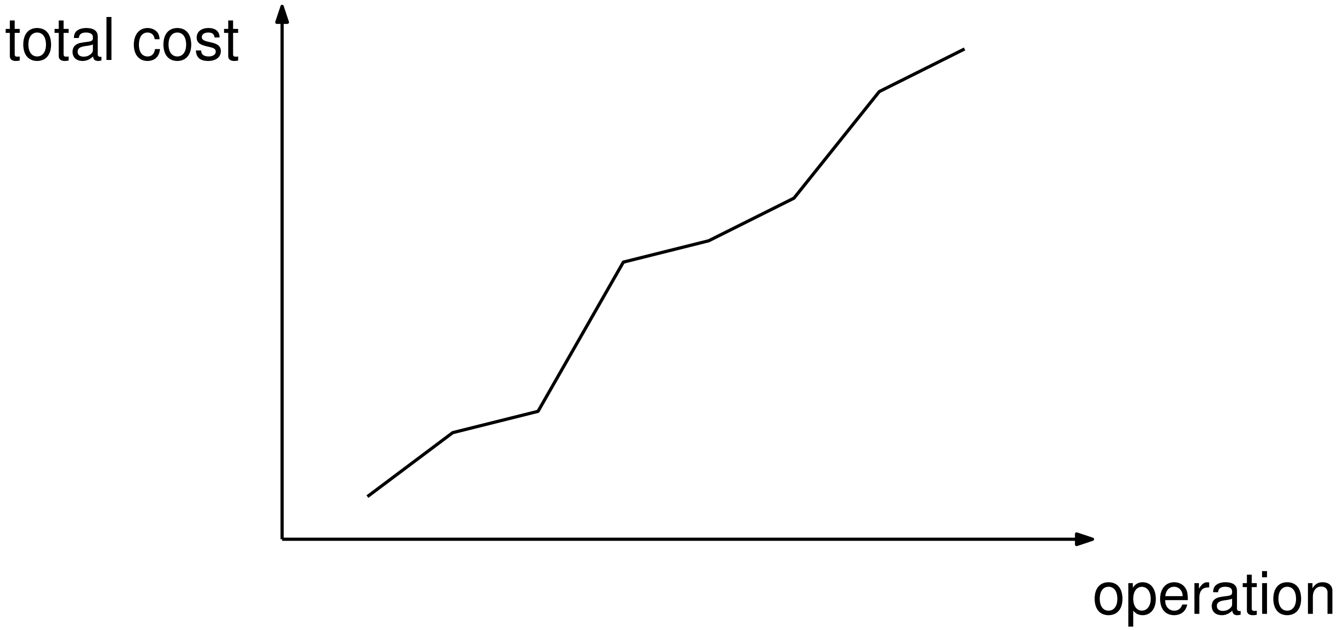
for **every** *n*.

More precisely

When analyzing algorithms, we want an upper bound $\hat{C}$ on the total cost:

$$T(n) = \sum_{i=0}^{n} c_i \leq \hat{C}$$

for **every** *n*.

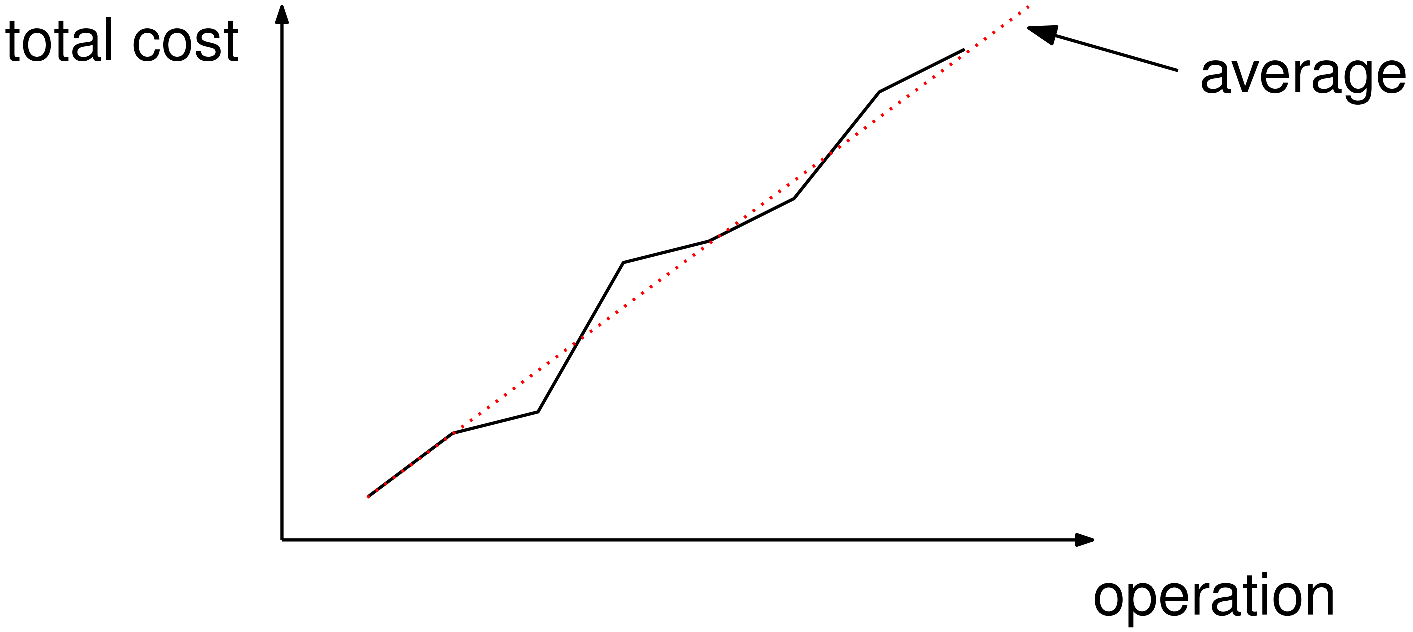


More precisely

When analyzing algorithms, we want an upper bound $\hat{C}$ on the total cost:

$$T(n) = \sum_{i=0}^{n} c_i \leq \hat{C}$$

for **every** *n*.

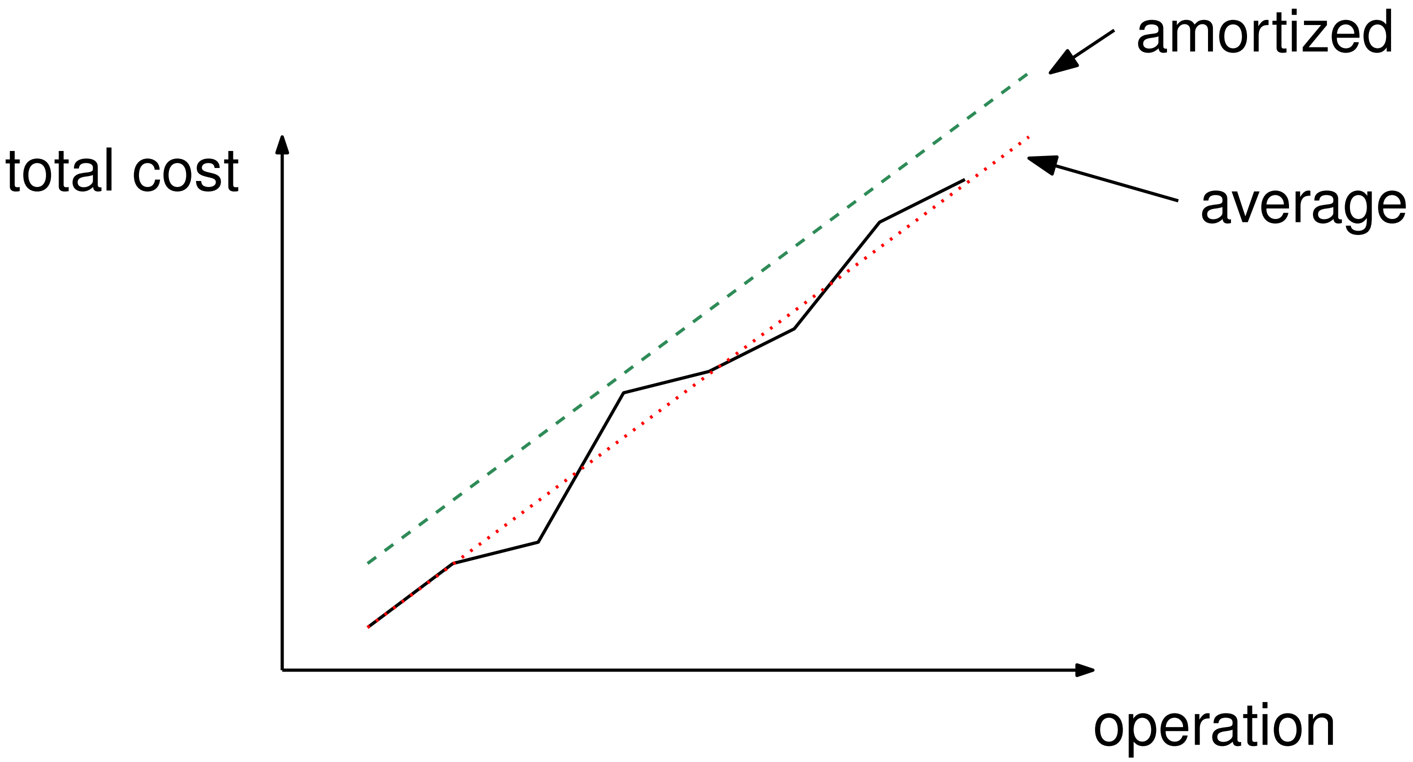


More precisely

When analyzing algorithms, we want an upper bound $\hat{C}$ on the total cost:

$$T(n) = \sum_{i=0}^{n} c_i \leq \hat{C}$$

for **every** *n*.

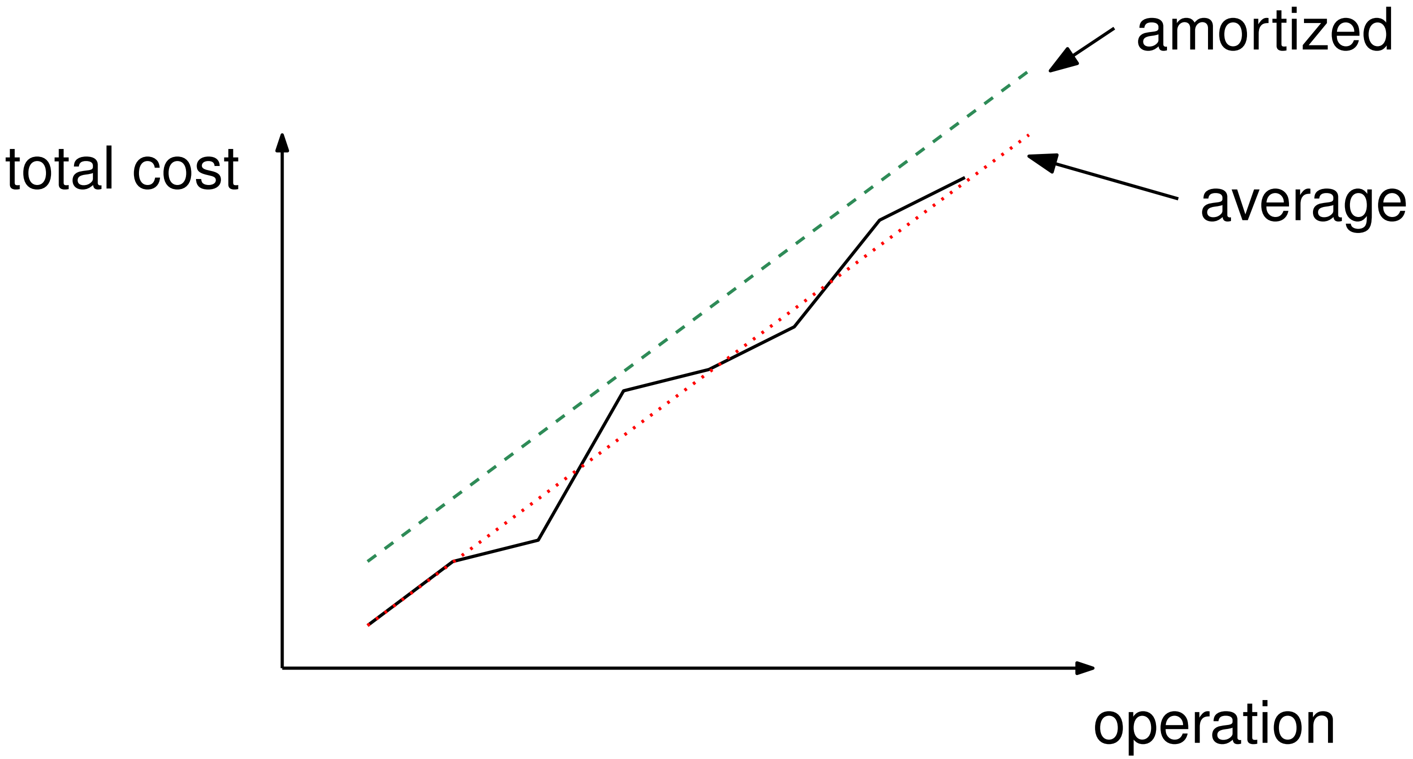


More precisely

When analyzing algorithms, we want an upper bound $\hat{C}$ on the total cost:

$$T(n) = \sum_{i=0}^{n} c_i \leq \hat{C} = \sum_{i=0}^{n} \hat{c}_i$$

for **every** *n*.



Amortized analysis: Accounting method

$$T(n) = \sum_{i=0}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i = \hat{C}$$

Amortized analysis: Accounting method

$$T(n) = \sum_{i=0}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i = \hat{C}$$

”Charge” $\$\hat{c}_i$
per operation

Amortized analysis: Accounting method

$$T(n) = \sum_{i=0}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i = \hat{C}$$

”Charge” \$$\hat{c}_i$ per operation

Pay \$$c_i$ for the actual cost of operation

Amortized analysis: Accounting method

$$T(n) = \sum_{i=0}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i = \hat{C}$$

”Charge” $\$\hat{c}_i$ per operation

Pay $\$c_i$ for the actual cost of operation

Save $\$(\hat{c}_i - c_i)$

Amortized analysis: Accounting method

$$T(n) = \sum_{i=0}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i = \hat{C}$$

"Charge" $\$\hat{c}_i$ per operation

Pay $\$c_i$ for the actual cost of operation

Save $\$(\hat{c}_i - c_i)$

Pay for more expensive operations using savings

Amortized analysis: Accounting method

$$T(n) = \sum_{i=0}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i = \hat{C}$$

”Charge” $\$\hat{c}_i$ per operation

Pay $\$c_i$ for the actual cost of operation

Save $\$(\hat{c}_i - c_i)$

Pay for more expensive operations using savings

$$\sum_{i=0}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i \quad \Rightarrow \quad \sum_{i=1}^{n} \hat{c}_i - \sum_{i=0}^{n} c_i = \sum_{i=1}^{n} (\hat{c}_i - c_i) \geq 0$$

Amortized analysis: Accounting method

$$T(n) = \sum_{i=0}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i = \hat{C}$$

"Charge" \$$\hat{c}_i$ per operation

Pay \$$c_i$ for the actual cost of operation

Save \$$(\hat{c}_i - c_i)$

Pay for more expensive operations using savings

Prove that $\sum_{i=0}^{n}(\hat{c}_i - c_i) \geq 0$

i.e., the balance in the bank is always non-negative

Example: Binary Counter

	Charge $\hat{c}_i$	Actual Cost c_i
000000		
000001		
000010		
000011		
000100		
000101		
000110		
000111		
001000		

Example: Binary Counter

	Charge $\hat{c}_i$	Actual Cost c_i	🏦
000000			
	\$2	\$1	\$1
000001			
000010			
000011			
000100			
000101			
000110			
000111			
001000			

Example: Binary Counter

	Charge $\hat{c}_i$	Actual Cost c_i	🏦
000000			
↓	\$2	\$1	\$1
000001			
↓	\$2	\$2	\$1
000010			
000011			
000100			
000101			
000110			
000111			
001000			

Example: Binary Counter

	Charge $\hat{c}_i$	Actual Cost c_i	🏛
000000			
	$2	$1	$1
000001			
	$2	$2	$1
000010			
	$2	$1	$2
000011			
000100			
000101			
000110			
000111			
001000			

Example: Binary Counter

	Charge $\hat{c}_i$	Actual Cost c_i	
000000			
	$2	$1	$1
000001			
	$2	$2	$1
000010			
	$2	$1	$2
000011			
	$2	$3	$1
000100			
000101			
000110			
000111			
001000			

Example: Binary Counter

	Charge $\hat{c}_i$	Actual Cost c_i	🏛
000000			
	\$2	\$1	\$1
000001			
	\$2	\$2	\$1
000010			
	\$2	\$1	\$2
000011			
	\$2	\$3	\$1
000100			
	\$2	\$1	\$2
000101			
000110			
000111			
001000			

Example: Binary Counter

	Charge $\hat{c}_i$	Actual Cost c_i	🏦
000000			
↓	\$2	\$1	\$1
000001			
↓	\$2	\$2	\$1
000010			
↓	\$2	\$1	\$2
000011			
↓	\$2	\$3	\$1
000100			
↓	\$2	\$1	\$2
000101			
↓	\$2	\$2	\$2
000110			
000111			
001000			

Example: Binary Counter

	Charge $\hat{c}_i$	Actual Cost c_i	🏦
000000			
↓	$2	$1	$1
000001			
↓	$2	$2	$1
000010			
↓	$2	$1	$2
000011			
↓	$2	$3	$1
000100			
↓	$2	$1	$2
000101			
↓	$2	$2	$2
000110			
↓	$2	$1	$3
000111			
001000			

Example: Binary Counter

	Charge $\hat{c}_i$	Actual Cost c_i	🏦
000000			
↓	$2	$1	$1
000001			
↓	$2	$2	$1
000010			
↓	$2	$1	$2
000011			
↓	$2	$3	$1
000100			
↓	$2	$1	$2
000101			
↓	$2	$2	$2
000110			
↓	$2	$1	$3
000111			
↓	$2	$4	$1
001000			

Example: Binary Counter

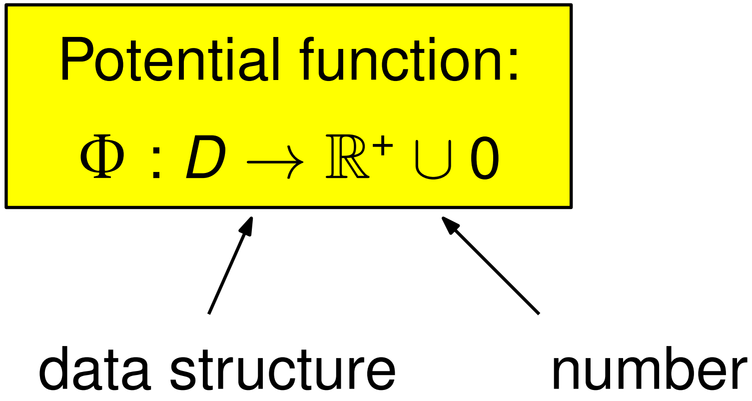
	Charge $\hat{c}_i$	Actual Cost c_i	🏦
000000			
	$2	$1	$1
$1 000001			
	$2	$2	$1
$ 000010			
	$2	$1	$2
$ $ 000011			
	$2	$3	$1
$ 000100			
	$2	$1	$2
$ $ 000101			
	$2	$2	$2
$ $ 000110			
	$2	$1	$3
$ $ $ 000111			
	$2	$4	$1
$ 001000			

Example: Binary Counter

	Charge $\hat{c}_i$	Actual Cost c_i	
000000			
↓	$2	$1	$1
$1 000001			
↓	$2	$2	$1
$ 000010			
↓	$2	$1	$2
$ $ 000011			
↓	$2		$1
$ 000100			
↓	$2		$2
$ $ 000101			
↓	$2	$2	$2
$ $ 000110			
↓	$2	$1	$3
$ $ $ 000111			
↓	$2	$4	$1
$ 001000			

Keep savings
on the data structure

Amortized analysis: Potential method



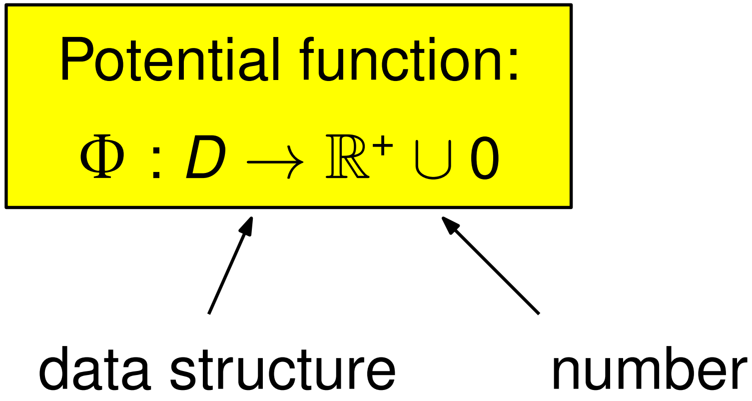
Potential function:

$$\Phi : D \rightarrow \mathbb{R}^{+} \cup 0$$

data structure

number

Amortized analysis: Potential method



Potential function:

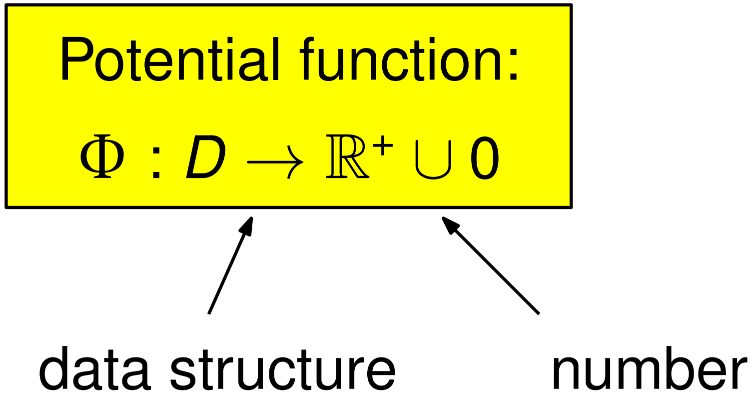
$\Phi : D \rightarrow \mathbb{R}^+ \cup 0$

data structure

number

E.g.: $\Phi(D_i)$ = number of 1’s in D_i

Amortized analysis: Potential method



Potential function:

$$\Phi : D \rightarrow \mathbb{R}^+ \cup 0$$

data structure

number

E.g.: $\Phi(D_i)$ = number of 1's in D_i

Amortized cost:

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = c_i + \Delta\Phi_i$$

Potential method: justification

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1}))$$

Potential method: justification

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1}))$$

$$= \sum_{i=1}^{n} c_i + \Phi(D_1) - \Phi(D_0) + \Phi(D_2) - \Phi(D_1) + \cdots + \Phi(D_{n-1}) - \Phi(D_{n-2}) + \Phi(D_n) - \Phi(D_{n-1})$$

Potential method: justification

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1}))$$

$$= \sum_{i=1}^{n} c_i + \cancel{\Phi(D_1)} - \Phi(D_0) + \Phi(D_2) - \cancel{\Phi(D_1)} + \cdots + \Phi(D_{n-1}) - \Phi(D_{n-2}) + \\ \Phi(D_n) - \Phi(D_{n-1})$$

Potential method: justification

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1}))$$

$$= \sum_{i=1}^{n} c_i + \cancel{\Phi(D_1)} - \Phi(D_0) + \cancel{\Phi(D_2)} - \cancel{\Phi(D_1)} + \cdots + \Phi(D_{n-1}) - \Phi(D_{n-2}) + \Phi(D_n) - \Phi(D_{n-1})$$

Potential method: justification

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n}(c_i + \Phi(D_i) - \Phi(D_{i-1}))$$

$$= \sum_{i=1}^{n} c_i + \cancel{\Phi(D_1)} - \Phi(D_0) + \cancel{\Phi(D_2)} - \cancel{\Phi(D_1)} + \cdots + \Phi(D_{n-1}) - \cancel{\Phi(D_{n-2})} + \Phi(D_n) - \Phi(D_{n-1})$$

Potential method: justification

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1}))$$

$$= \sum_{i=1}^{n} c_i + \cancel{\Phi(D_1)} - \Phi(D_0) + \cancel{\Phi(D_2)} - \cancel{\Phi(D_1)} + \cdots + \cancel{\Phi(D_{n-1})} - \cancel{\Phi(D_{n-2})} +$$

$$\Phi(D_n) - \cancel{\Phi(D_{n-1})}$$

Potential method: justification

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n}(c_i + \Phi(D_i) - \Phi(D_{i-1}))$$

$$= \sum_{i=1}^{n} c_i + \cancel{\Phi(D_1)} - \Phi(D_0) + \cancel{\Phi(D_2)} - \cancel{\Phi(D_1)} + \cdots + \cancel{\Phi(D_{n-1})} - \cancel{\Phi(D_{n-2})} +$$
$$\Phi(D_n) - \cancel{\Phi(D_{n-1})}$$

$$= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0)$$

Potential method: justification

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1}))$$

$$= \sum_{i=1}^{n} c_i + \Phi(\cancel{D_1}) - \Phi(D_0) + \Phi(\cancel{D_2}) - \Phi(\cancel{D_1}) + \cdots + \Phi(\cancel{D_{n-1}}) - \Phi(\cancel{D_{n-2}}) + \Phi(D_n) - \Phi(\cancel{D_{n-1}})$$

$$= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0)$$

$$\Rightarrow \sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i = \Phi(D_n) - \Phi(D_0)$$

Potential method: justification

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n}(c_i + \Phi(D_i) - \Phi(D_{i-1}))$$

$$= \sum_{i=1}^{n} c_i + \cancel{\Phi(D_1)} - \Phi(D_0) + \cancel{\Phi(D_2)} - \cancel{\Phi(D_1)} + \cdots + \cancel{\Phi(D_{n-1})} - \cancel{\Phi(D_{n-2})} + \Phi(D_n) - \cancel{\Phi(D_{n-1})}$$

$$= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0)$$

$$\Rightarrow \sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i = \Phi(D_n) - \Phi(D_0) \geq 0$$

Potential method: justification

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1}))$$

$$= \sum_{i=1}^{n} c_i + \cancel{\Phi(D_1)} - \Phi(D_0) + \cancel{\Phi(D_2)} - \cancel{\Phi(D_1)} + \cdots + \cancel{\Phi(D_{n-1})} - \cancel{\Phi(D_{n-2})} + \Phi(D_n) - \cancel{\Phi(D_{n-1})}$$

$$= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0)$$

$$\Rightarrow \sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i = \Phi(D_n) - \Phi(D_0) \geq 0$$

Satisfied if:

- $\Phi(D_0) = 0$, and
- $\Phi(D_n) \geq 0$ for all $n > 0$

Potential method: justification

Potential function:

$$\Phi : D \to \mathbb{R}^{+} \cup 0$$

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1}))$$

$$= \sum_{i=1}^{n} c_i + \cancel{\Phi(D_1)} - \Phi(D_0) + \cancel{\Phi(D_2)} - \cancel{\Phi(D_1)} + \cdots + \cancel{\Phi(D_{n-1})} - \cancel{\Phi(D_{n-2})} + \Phi(D_n) - \cancel{\Phi(D_{n-1})}$$

$$= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0)$$

$$\Rightarrow \sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i = \Phi(D_n) - \Phi(D_0) \geq 0$$

Satisfied if:

- $\Phi(D_0) = 0$, and
- $\Phi(D_n) \geq 0$ for all $n > 0$

Example: Binary counter

$\Phi(D_i)$ = # of 1's in the counter after the i-th $\textsc{Increment}(B, k)$

Example: Binary counter

$\Phi(D_i)$ = # of 1's in the counter after the i-th INCREMENT(B, k)

$\hat{c}_i = c_i + \Delta\Phi_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

0 0 1 1 0 1 1 0 1 1

INCREMENT

0 0 1 1 0 1 1 1 0 0

Example: Binary counter

$\Phi(D_i)$ = # of 1's in the counter after the i-th $\textsc{Increment}(B, k)$

$\hat{c}_i = c_i + \Delta\Phi_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Let d be # of bits flipped

$$\overbrace{0\ 0\ 1\ 1\ 0\ 1\ 1}^{k-d}\ \overbrace{0\ 1\ 1}^{d\,=\,3}$$

$\textsc{Increment}$

$$0\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 0$$

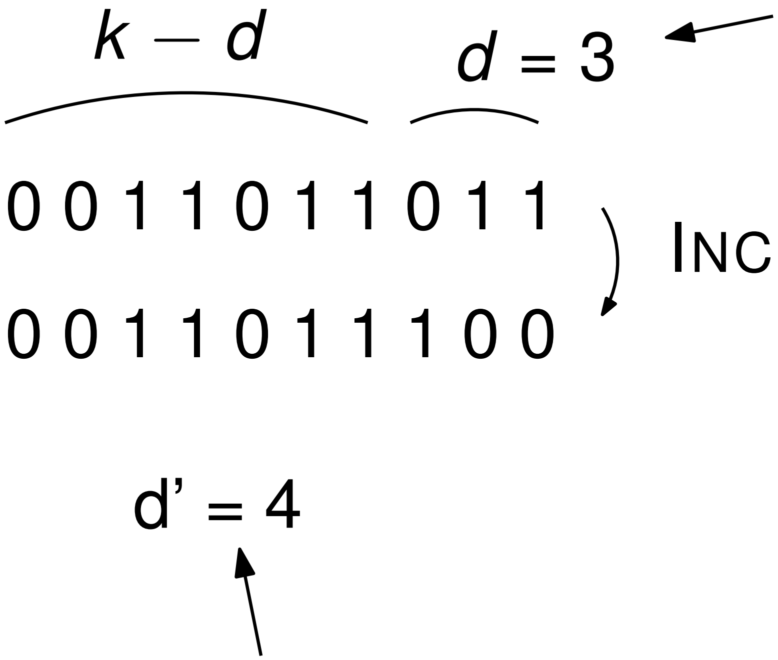
d' = 4

Let d' be # of 1's in $k - d$ MSBs

Example: Binary counter

$\Phi(D_i)$ = # of 1's in the counter after the i-th INCREMENT(B, k)

$$\hat{c}_i = c_i + \Delta\Phi_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$



Let d be # of bits flipped

$$\overbrace{0\ 0\ 1\ 1\ 0\ 1\ 1}^{k-d}\ \overbrace{0\ 1\ 1}^{d\,=\,3}$$

INCREMENT

$$0\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 0$$

$$\hat{c}_i = d + \Phi(D_i) - \Phi(D_{i-1})$$

d' = 4

Let d' be # of 1's in $k - d$ MSBs

Example: Binary counter

$\Phi(D_i)$ = # of 1's in the counter after the i-th INCREMENT(B, k)

$$\hat{c}_i = c_i + \Delta\Phi_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

Let d be # of bits flipped

$$\overbrace{0\;0\;1\;1\;0\;1\;1}^{k-d}\;\overbrace{0\;1\;1}^{d=3}$$

INCREMENT

$$0\;0\;1\;1\;0\;1\;1\;1\;0\;0$$

$$\hat{c}_i = d + \Phi(D_i) - \Phi(D_{i-1})$$

d' = 4

$$= d + (d' + 1) - (d'+(d-1))$$

Let d' be # of 1's in $k - d$ MSBs

Example: Binary counter

$\Phi(D_i)$ = # of 1's in the counter after the i-th INCREMENT(B, k)

$$\hat{c}_i = c_i + \Delta\Phi_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

Let d be # of bits flipped

$$\overbrace{0\ 0\ 1\ 1\ 0\ 1\ 1}^{k-d}\ \overbrace{0\ 1\ 1}^{d\,=\,3}$$

INCREMENT

$$0\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 0$$

d' = 4

Let d' be # of 1's in $k-d$ MSBs

$$\hat{c}_i = d + \Phi(D_i) - \Phi(D_{i-1})$$

$$= d + (\cancel{d'} + 1) - (\cancel{d'} + (d-1))$$

Example: Binary counter

$\Phi(D_i)$ = # of 1's in the counter after the i-th INCREMENT(B, k)

$$\hat{c}_i = c_i + \Delta\Phi_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

Let d be # of bits flipped

$$\overbrace{0\ 0\ 1\ 1\ 0\ 1\ 1}^{k-d}\ \overbrace{0\ 1\ 1}^{d\,=\,3}$$

INCREMENT

$$0\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 0$$

$d' = 4$

Let d' be # of 1's in $k - d$ MSBs

$$\hat{c}_i = d + \Phi(D_i) - \Phi(D_{i-1})$$

$$= \cancel{d} + (\cancel{d}' + 1) - (\cancel{d}' + (\cancel{d} - 1))$$

$$= 2$$

Amortized analysis: Summary

$$\hat{c} = \frac{1}{n} \cdot \sum_{i=1}^{n} c_i$$

- Aggregate method
- Accounting method
- Potential method

Potential function:

- $\Phi : D \to \mathbb{R}^+ \cup \{0\}$
- $\Phi(D_0) = 0$, and
- $\Phi(D_n) \geq 0$ for all $n > 0$

Keep savings
on the data structure