# DYNAMIC TABLES & AMORTIZED ANALYSIS

Suppose you want to use an array:

You must support insertion but you don't know max # elements

# DYNAMIC TABLES & AMORTIZED ANALYSIS

Suppose you want to use an array:

You must support insertion but you don't know max # elements

array doubling: start w/ array of size 1; every time it fills up, double the size

↳ (make a new larger array & copy)

# DYNAMIC TABLES & AMORTIZED ANALYSIS

Suppose you want to use an array:

You must support insertion but you don't know max # elements

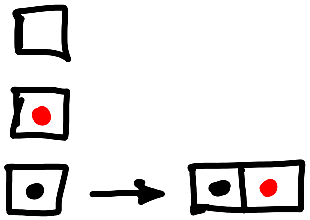
array doubling: start w/ array of size 1; every time it fills up, double the size

↳ (make a new larger array & copy)

start :

insert :

insert :



# DYNAMIC TABLES & AMORTIZED ANALYSIS

Suppose you want to use an array:

You must support insertion but you don't know max # elements

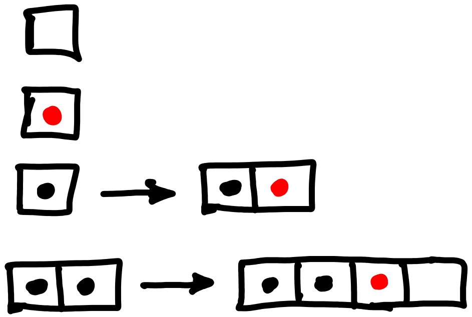
array doubling: start w/ array of size 1; every time it fills up, double the size
(make a new larger array & copy)

start :

insert :

insert :

insert :



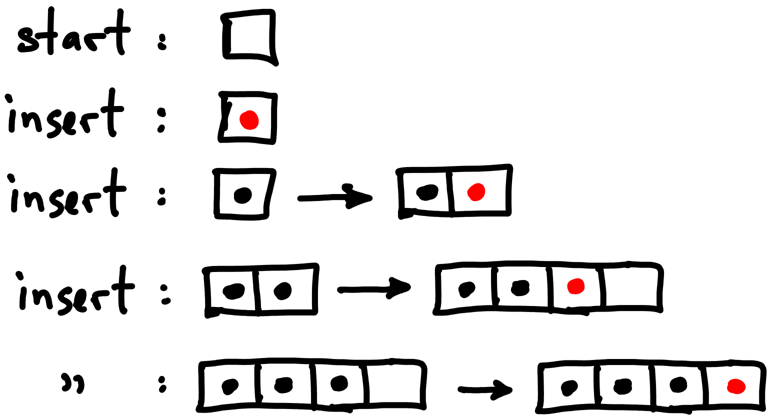
# DYNAMIC TABLES & AMORTIZED ANALYSIS

Suppose you want to use an array:

You must support insertion but you don't know max # elements

array doubling: start w/ array of size 1; every time it fills up, double the size

↳ (make a new larger array & copy)

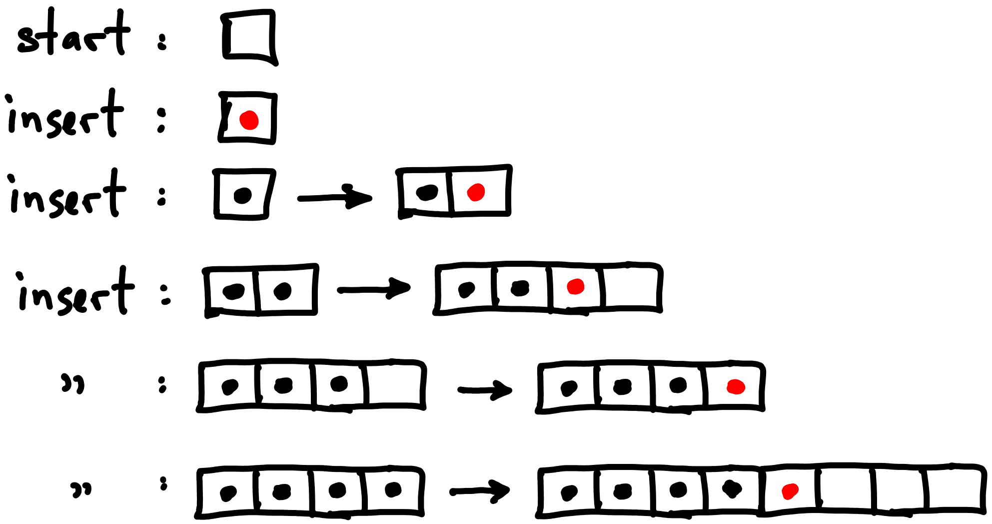


# DYNAMIC TABLES & AMORTIZED ANALYSIS

Suppose you want to use an array:

You must support insertion but you don't know max # elements

array doubling: start w/ array of size 1; every time it fills up, double the size
(make a new larger array & copy)



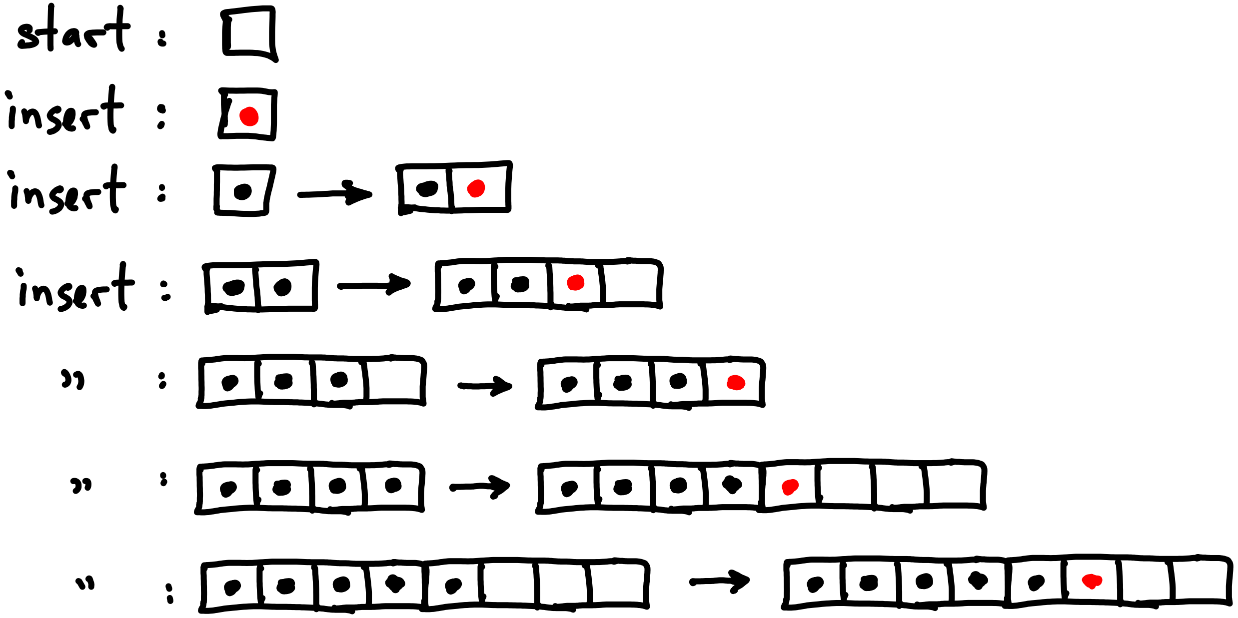
# DYNAMIC TABLES & AMORTIZED ANALYSIS

Suppose you want to use an array:

You must support insertion but you don't know max # elements

array doubling: start w/ array of size 1; every time it fills up, double the size

↳ (make a new larger array & copy)



# DYNAMIC TABLES & AMORTIZED ANALYSIS

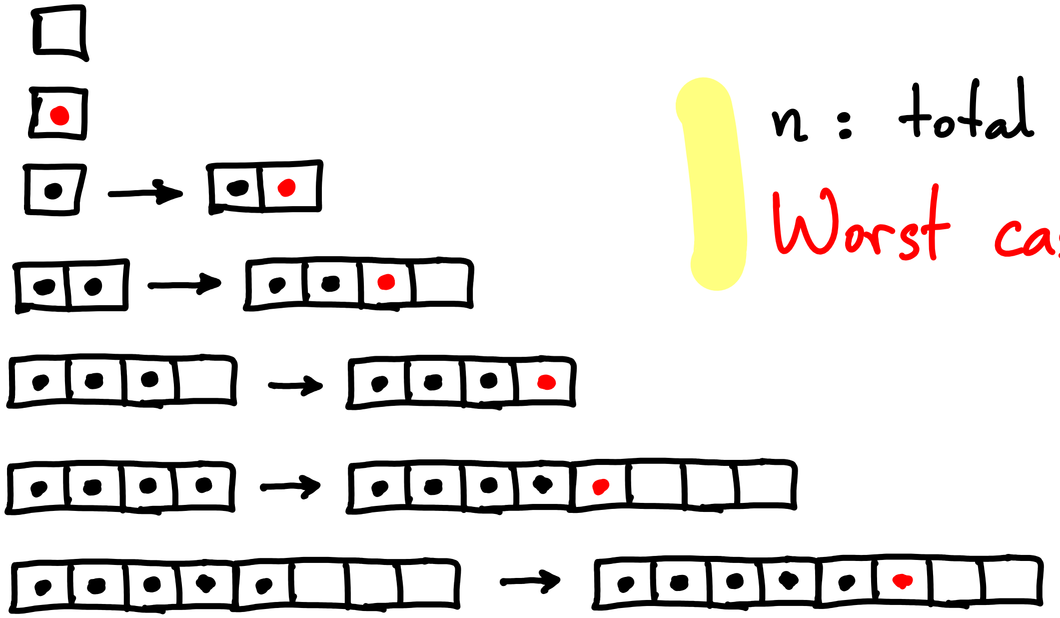
Suppose you want to use an array:

You must support insertion but you don't know max # elements

array doubling: start w/ array of size 1; every time it fills up, double the size

↳ (make a new larger array & copy)

start :
insert :
insert :
insert :
" :
" :
" :



n : total number of inserts

Worst case time of an insert : $O(n)$

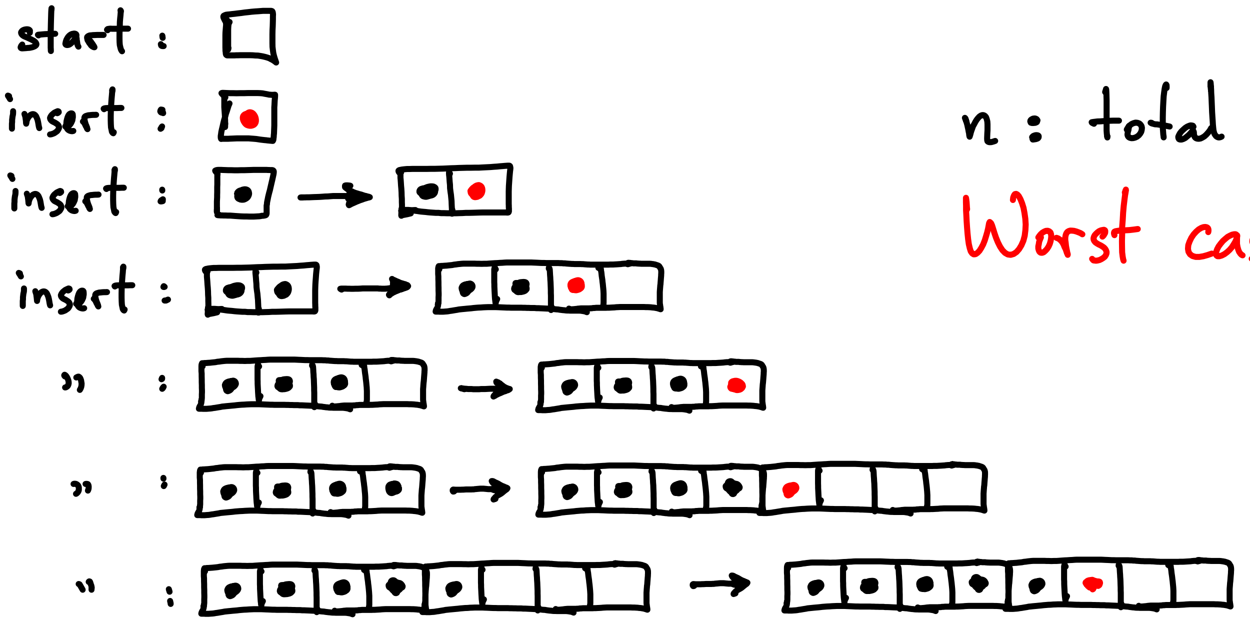
# DYNAMIC TABLES & AMORTIZED ANALYSIS

Suppose you want to use an array:

You must support insertion but you don't know max # elements

array doubling: start w/ array of size 1; every time it fills up, double the size

↳ (make a new larger array & copy)




$n$ : total number of inserts

Worst case time of an insert : $O(n)$

↳ for $n$ inserts : $O(n^2)$

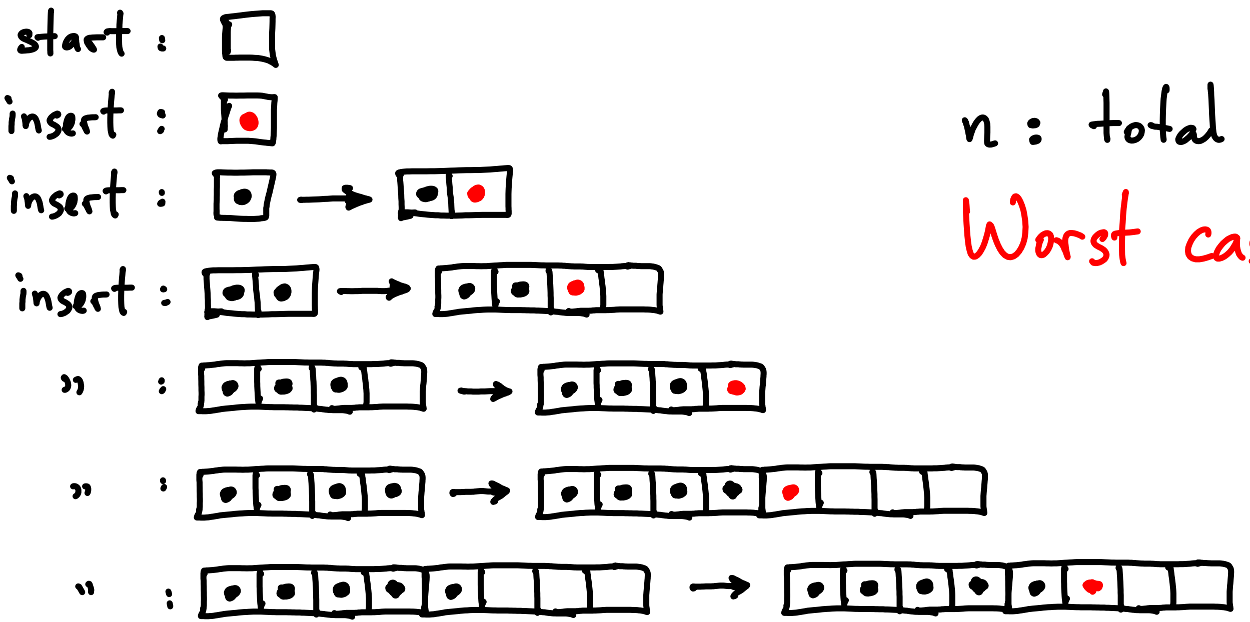
# Dynamic Tables & Amortized Analysis

Suppose you want to use an array:

You must support insertion but you don't know max # elements

array doubling: start w/ array of size 1; every time it fills up, double the size

(make a new larger array & copy)



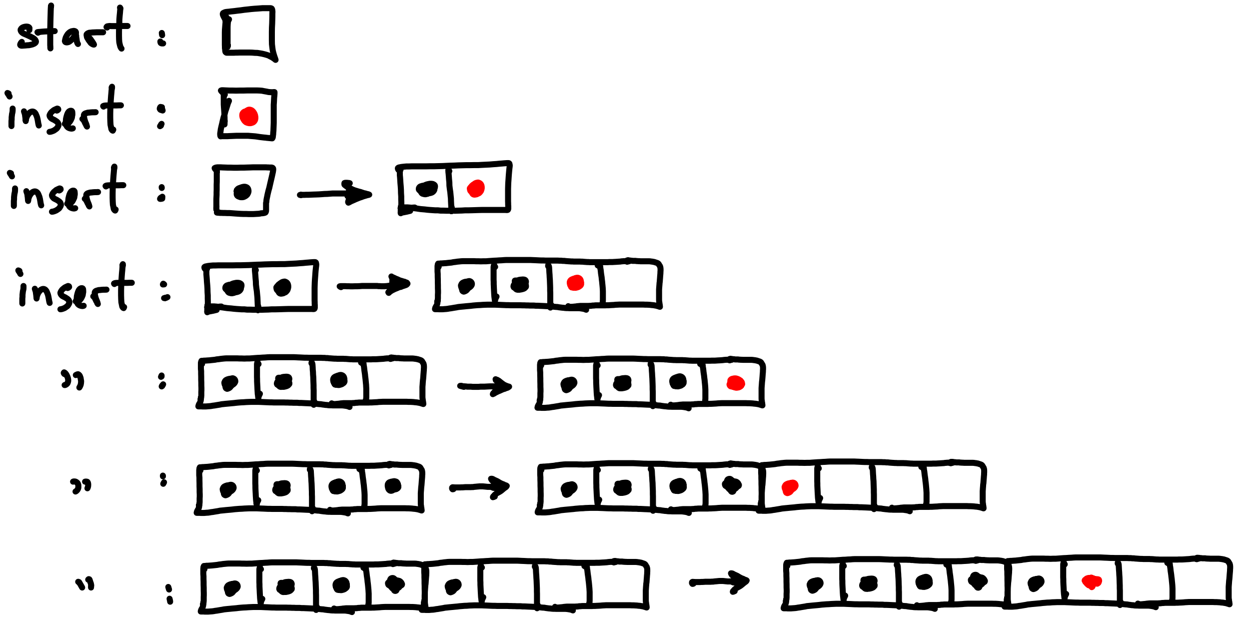

n : total number of inserts

Worst case time of an insert : $O(n)$

for n inserts : $O(n^2)$

Claim : for n inserts : also $O(n)$

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $size_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |



| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| | $2^0$ | $2^1$ | $2^{1+}$ | $2^2$ | $2^{2+}$ | $2^{2+}$ | $2^{2+}$ | $2^3$ | $2^{3+}$ | $2^{3+}$ |
| $size_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |

$$\text{cost } c_i \begin{cases} i & \text{if } i-1 \text{ is a power of } 2 \\ 1 & \text{otherwise} \end{cases}$$

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | $2^0$ | $2^1$ | $2^{1+}$ | $2^2$ | $2^{2+}$ | $2^{2+}$ | $2^{2+}$ | $2^3$ | $2^{3+}$ | $2^{3+}$ |
| $size_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $c_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |

$$\text{cost } c_i \begin{cases} i & \text{if } i-1 \text{ is a power of } 2 \\ 1 & \text{otherwise} \end{cases}$$

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | $2^0$ | $2^1$ | $2^{1+}$ | $2^2$ | $2^{2+}$ | $2^{2+}$ | $2^{2+}$ | $2^3$ | $2^{3+}$ | $2^{3+}$ |
| $size_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $c_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | – | 1 | 2 | – | 4 | – | – | – | 8 | – |

$$\text{cost } c_i \begin{cases} i & \text{if } i-1 \text{ is a power of } 2 \\ 1 & \text{otherwise} \end{cases}$$

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | $2^0$ | $2^1$ | $2^{1+}$ | $2^2$ | $2^{2+}$ | $2^{2+}$ | $2^{2+}$ | $2^3$ | $2^{3+}$ | $2^{3+}$ |
| $size_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $c_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | – | 1 | 2 | – | 4 | – | – | – | 8 | – |
| | | $2^0$ | $2^1$ | | $2^2$ | | | | $2^3$ | |

$$\text{cost } c_i \begin{cases} i & \text{if } i-1 \text{ is a power of } 2 \\ 1 & \text{otherwise} \end{cases}$$

$c_i$ = copy $i$ elements

(make array: free)

if making an array of size $2i$ costs $\Theta(i)$ then we would multiply by a constant

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| | $2^0$ | $2^1$ | $2^{1+}$ | $2^2$ | $2^{2+}$ | $2^{2+}$ | $2^{2+}$ | $2^3$ | $2^{3+}$ | $2^{3+}$ |
| $size_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $c_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | – | 1 | 2 | – | 4 | – | – | – | 8 | – |
| | | $2^0$ | $2^1$ | | $2^2$ | | | | $2^3$ | |

$$\text{cost } c_i \begin{cases} i & \text{if } i-1 \text{ is a power of } 2 \\ 1 & \text{otherwise} \end{cases}$$

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

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| | $2^0$ | $2^1$ | $2^{1+}$ | $2^2$ | $2^{2+}$ | $2^{2+}$ | $2^{2+}$ | $2^3$ | $2^{3+}$ | $2^{3+}$ |
| $size_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $c_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | – | 1 | 2 | – | 4 | – | – | – | 8 | – |
| | | $2^0$ | $2^1$ | | $2^2$ | | | | $2^3$ | |

$$\text{cost } c_i \begin{cases} i & \text{if } i-1 \text{ is a power of } 2 \\ 1 & \text{otherwise} \end{cases}$$

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| | $2^0$ | $2^1$ | $2^{1+}$ | $2^2$ | $2^{2+}$ | $2^{2+}$ | $2^{2+}$ | $2^3$ | $2^{3+}$ | $2^{3+}$ |
| $size_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $c_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | – | 1 | 2 | – | 4 | – | – | – | 8 | – |
| | | $2^0$ | $2^1$ | | $2^2$ | | | | $2^3$ | |

$$\text{cost}(n) = \sum_{i=1}^{n} c_i = n + ?$$

$$\text{cost } c_i \begin{cases} i & \text{if } i-1 \text{ is a power of } 2 \\ 1 & \text{otherwise} \end{cases}$$

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | $2^0$ | $2^1$ | $2^{1+}$ | $2^2$ | $2^{2+}$ | $2^{2+}$ | $2^{2+}$ | $2^3$ | $2^{3+}$ | $2^{3+}$ |
| $size_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $c_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | – | 1 | 2 | – | 4 | – | – | – | 8 | – |
| | | $2^0$ | $2^1$ | | $2^2$ | | | | $2^3$ | |

$$\text{cost}(n) = \sum_{i=1}^{n} c_i = n + ?$$

$$\underbrace{2^0 \quad 2^1 \quad 2^2}_{< 2^3}$$

$$\text{cost } c_i \begin{cases} i & \text{if } i-1 \text{ is a power of } 2 \\ 1 & \text{otherwise} \end{cases}$$

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| | $2^0$ | $2^1$ | $2^{1+}$ | $2^2$ | $2^{2+}$ | $2^{2+}$ | $2^{2+}$ | $2^3$ | $2^{3+}$ | $2^{3+}$ |
| $size_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $c_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | – | 1 | 2 | – | 4 | – | – | – | 8 | – |
| | | $2^0$ | $2^1$ | | $2^2$ | | | | $2^3$ | |

$$\text{cost}(n) = \sum_{i=1}^{n} c_i = n + ?$$

$\underbrace{2^0 \quad 2^1 \quad 2^2}_{< 2^3}$

$2^3 \downarrow$

$< n \leftarrow < n$

$$\text{cost } c_i \begin{cases} i & \text{if } i-1 \text{ is a power of } 2 \\ 1 & \text{otherwise} \end{cases}$$

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | $2^0$ | $2^1$ | $2^{1+}$ | $2^2$ | $2^{2+}$ | $2^{2+}$ | $2^{2+}$ | $2^3$ | $2^{3+}$ | $2^{3+}$ |
| $size_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $c_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | – | 1 | 2 | – | 4 | – | – | – | 8 | – |

$$\underbrace{2^0 \quad 2^1 \quad 2^2}_{< 2^3} \qquad 2^3$$

$< n \quad \leftarrow \quad < n$

$$\text{cost}(n) = \sum_{i=1}^{n} c_i \leq n + 2n$$

AGGREGATE ANALYSIS

# AMORTIZATION (analyzing cost)

Applies to some problems that involve many operations.

If worst case time of operation $k$ is $O(f(k))$,
try to show that $n$ operations cost $o(n \cdot f(n))$

3 main ways for amortizing: aggregate, accounting, & potential method.

just did this

3 main ways for amortizing: aggregate, accounting, & potential method.

---

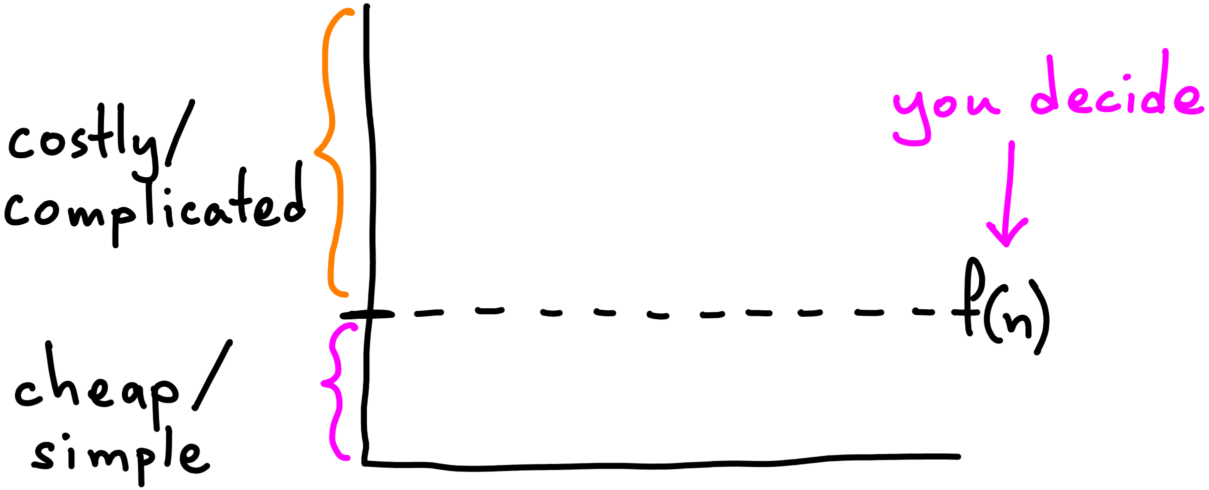
Accounting: saving for a rainy day

3 main ways for amortizing: aggregate, accounting, & potential method.

Accounting: saving for a rainy day

"simple" operations

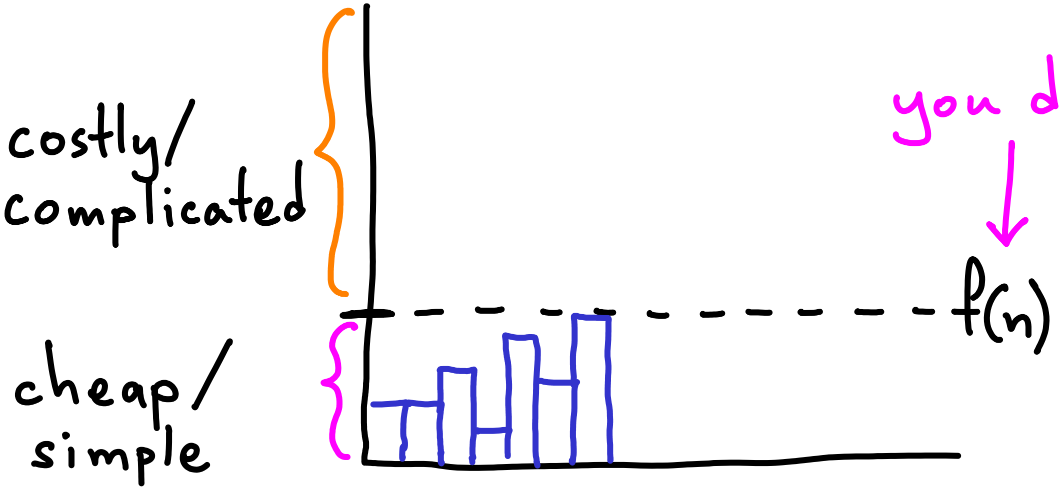
"Complicated/costly" operations



3 main ways for amortizing: aggregate, accounting, & potential method.

## ACCOUNTING: saving for a rainy day

"simple" operations



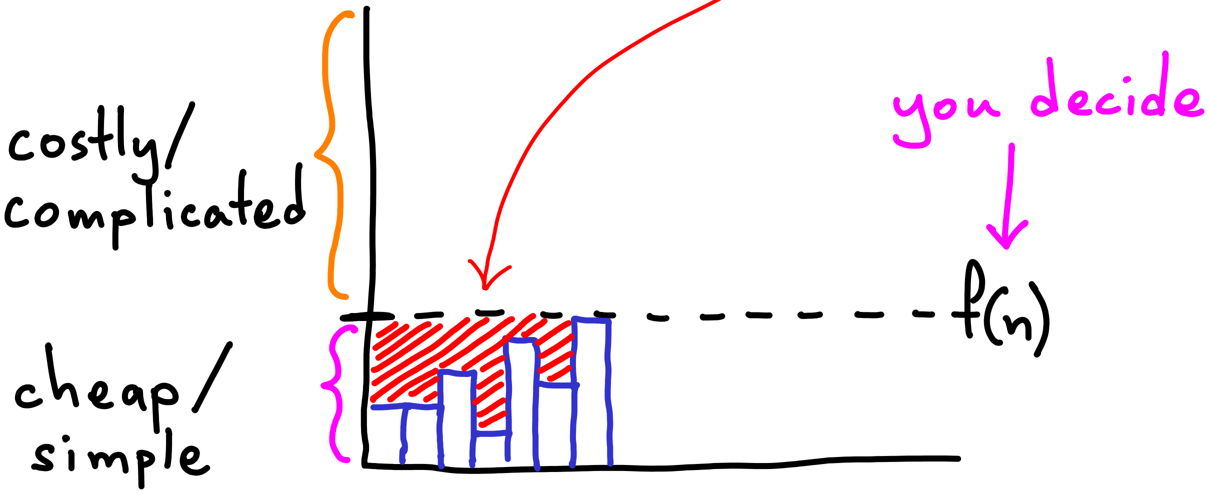
3 main ways for amortizing: aggregate, accounting, & potential method.

---

ACCOUNTING: saving for a rainy day

Pretend "simple" operations cost more than they do. Ideally $\Theta$(real cost)

↳ "save" the difference



3 main ways for amortizing: aggregate, accounting, & potential method.

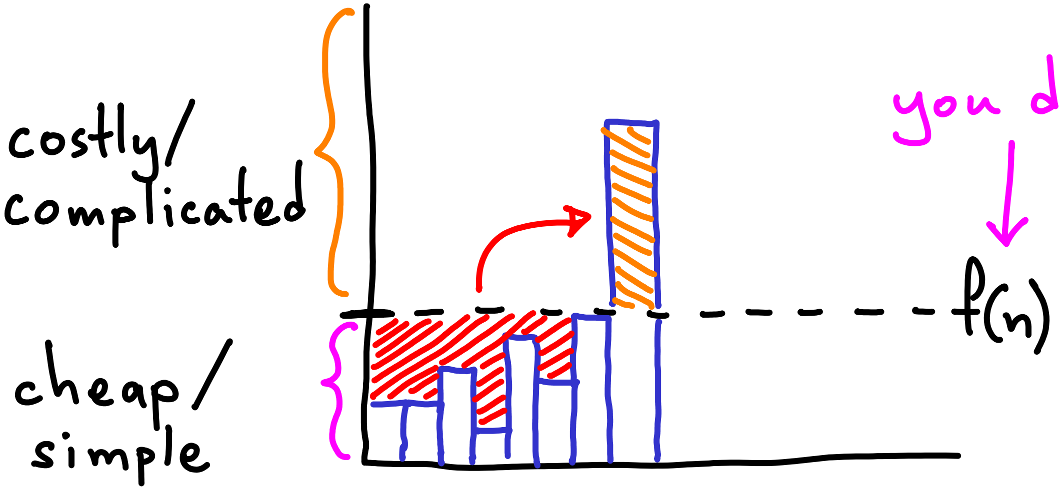
---

**Accounting:** saving for a rainy day

Pretend "simple" operations cost more than they do. Ideally $\Theta$(real cost)

↳ "save" the difference → "spend" what you saved up.

"Complicated/costly" operations: pretend they cost less; pay excess via savings



3 main ways for amortizing: aggregate, accounting, & potential method.

---

ACCOUNTING: saving for a rainy day

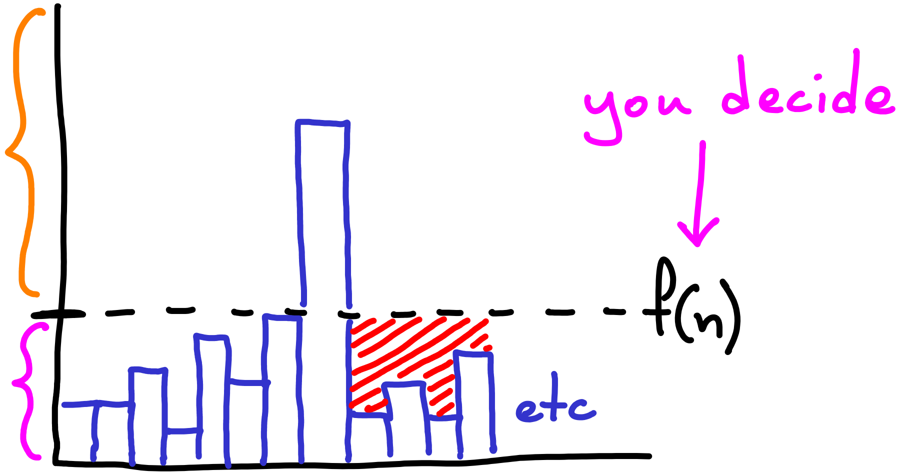
Pretend "simple" operations cost more than they do. Ideally $\Theta$(real cost)

↳ "save" the difference → "spend" what you saved up.

"Complicated/costly" operations: pretend they cost less; pay excess via savings

costly/ complicated

cheap/ simple



3 main ways for amortizing: aggregate, accounting, & potential method.

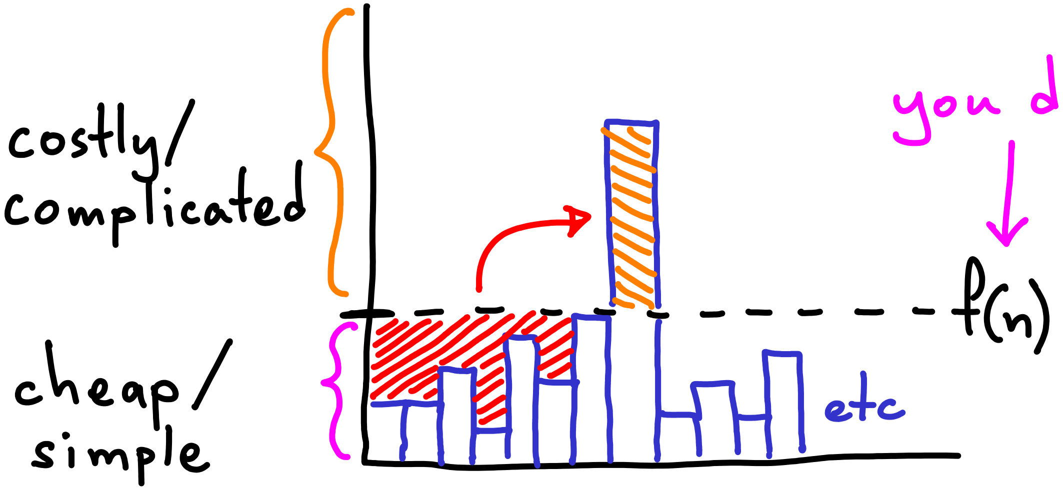
---

**ACCOUNTING:** saving for a rainy day

Pretend "simple" operations cost more than they do. Ideally $\Theta$(real cost)

↳ "save" the difference → "spend" what you saved up.

"Complicated/costly" operations: pretend they cost less; pay excess via savings




Rule: Never spend more than what you saved.

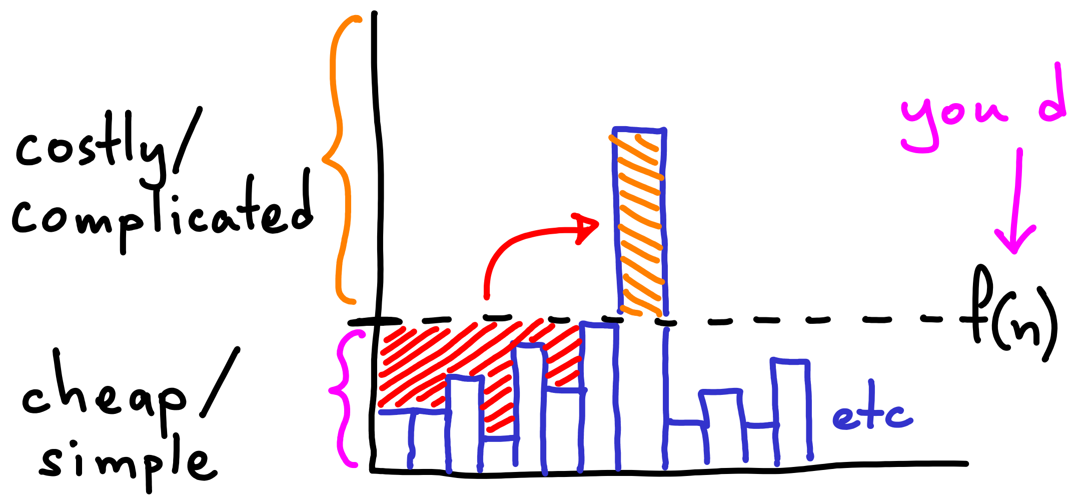
3 main ways for amortizing: aggregate, accounting, & potential method.

# ACCOUNTING: saving for a rainy day

Pretend "simple" operations cost more than they do. Ideally $\Theta$(real cost)

↳ "save" the difference → "spend" what you saved up.

"Complicated/costly" operations: pretend they cost less; pay excess via savings




Rule: Never spend more than what you saved.

Amortized cost: $n \cdot f(n) \geq$ true cost

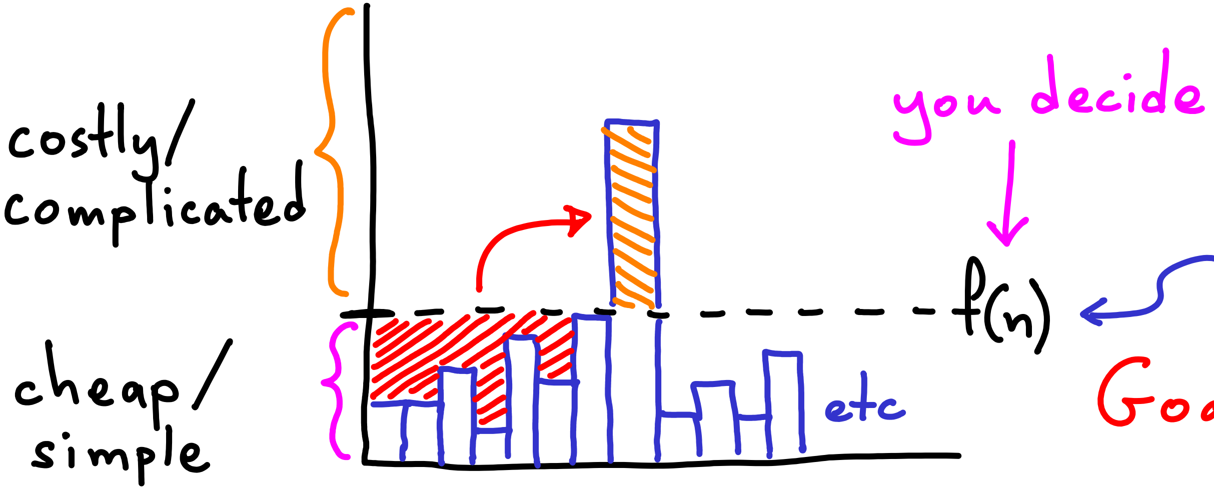
# 3 main ways for amortizing: aggregate, accounting, & potential method.

## ACCOUNTING: saving for a rainy day

Pretend "simple" operations cost more than they do. Ideally $\Theta$(real cost)

↳ "save" the difference → "spend" what you saved up.

"Complicated/costly" operations: pretend they cost less; pay excess via savings




Rule: Never spend more than what you saved.

Amortized cost: $n \cdot f(n) \geq$ true cost

Goal: exaggerate / save as little as possible. } minimize $f(n)$

Amortized cost of operation $i$: $\hat{c}_i$

Amortized cost of operation $i$: $\hat{c}_i$

$$\underbrace{\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i \quad \text{for all } n}$$

our banking game should always exagerrate true costs.

Amortized cost of operation $i$: $\hat{c}_i$

$$\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i \quad \text{for all } n$$

our banking game should always exagerrate true costs.

Back to dynamic tables:

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $c_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | – | 1 | 2 | – | 4 | – | – | – | 8 | – |
| $\text{size}_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |

Amortized cost of operation $i$: $\hat{c}_i$

$$\sum_{i=1}^{n} c_i \le \sum_{i=1}^{n} \hat{c}_i \quad \text{for all } n$$

our banking game should always exagerrate true costs.

Back to dynamic tables:

let $\hat{c}_i = 3 \longrightarrow$ 1 to cover insert cost
2 for eventual doubling

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $c_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | – | 1 | 2 | – | 4 | – | – | – | 8 | – |
| $\text{size}_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $\hat{c}_i$ | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |

Amortized cost of operation $i$: $\hat{c}_i$

$$\underbrace{\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i \quad \text{for all } n}$$

our banking game should always exagerrate true costs.

Back to dynamic tables:

let $\hat{c}_i = 3$ $\longrightarrow$ 1 to cover insert cost
2 for eventual doubling

implies $\sum c_i \leq 3n$

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $c_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | – | 1 | 2 | – | 4 | – | – | – | 8 | – |
| $\text{size}_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $\hat{c}_i$ | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |

Amortized cost of operation $i$: $\hat{c}_i$

$$\sum_{i=1}^{n} c_i \le \sum_{i=1}^{n} \hat{c}_i \quad \text{for all } n$$

our banking game should always exagerrate true costs.

Back to dynamic tables:

let $\hat{c}_i = 3$ $\longrightarrow$ 1 to cover insert cost

2 for eventual doubling

implies $\sum c_i \le 3n$

When table doubles, use 1 to copy each item.

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $c_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | – | 1 | 2 | – | 4 | – | – | – | 8 | – |
| $size_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $\hat{c}_i$ | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |

Amortized cost of operation $i$: $\hat{c}_i$

$$\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i \quad \text{for all } n$$

our banking game should always exagerrate true costs.

Back to dynamic tables:

let $\hat{c}_i = 3 \longrightarrow$ 1 to cover insert cost

2 for eventual doubling

implies $\sum c_i \leq 3n$

When table doubles, use 1 to copy each item.

BANK (savings per iteration)

$\boxed{2}$

Pretend cost is 3

pay 1 to insert, save/bank 2

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $c_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | – | 1 | 2 | – | 4 | – | – | – | 8 | – |
| $size_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $\hat{c}_i$ | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |

Amortized cost of operation $i$: $\hat{c}_i$

$$\sum_{i=1}^{n} c_i \le \sum_{i=1}^{n} \hat{c}_i \quad \text{for all } n$$

our banking game should always exagerrate true costs.

Back to dynamic tables:

let $\hat{c}_i = 3$ ⟶ 1 to cover insert cost
2 for eventual doubling

implies $\sum c_i \le 3n$

When table doubles, use 1 to copy each item.

BANK (savings per iteration)

$\boxed{1}$

Pretend cost is ~~3~~ 2

pay 1 to insert, save/bank ~~2~~ 1

We can even give \$ to charity

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| $c_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | – | 1 | 2 | – | 4 | – | – | – | 8 | – |
| $size_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $\hat{c}_i$ | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |

Just this one time

Amortized cost of operation $i$: $\hat{c}_i$

$$\sum_{i=1}^{n} c_i \le \sum_{i=1}^{n} \hat{c}_i \quad \text{for all } n$$

our banking game should always exagerrate true costs.

Back to dynamic tables:

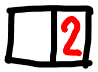
let $\hat{c}_i = 3$ → 1 to cover insert cost
2 for eventual doubling

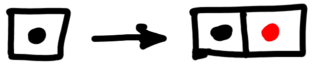
implies $\sum c_i \le 3n$

When table doubles, use 1 to copy each item.

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $c_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | – | 1 | 2 | – | 4 | – | – | – | 8 | – |
| $\text{size}_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $\hat{c}_i$ | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| $\text{bank}_i$ | 1 | 2 | | | | | | | | |

BANK (savings per iteration)





Spent 1 to copy 1 item into new array. → left ∅ in bank.

Added 2nd item & banked \$2

Amortized cost of operation $i$: $\hat{c}_i$

$$\sum_{i=1}^{n} c_i \le \sum_{i=1}^{n} \hat{c}_i \quad \text{for all } n$$

our banking game should always exagerrate true costs.

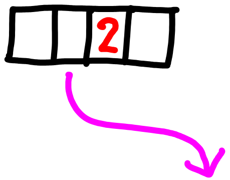
Back to dynamic tables:

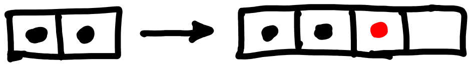
let $\hat{c}_i = 3 \longrightarrow$ 1 to cover insert cost
2 for eventual doubling

implies $\sum c_i \le 3n$

When table doubles, use 1 to copy each item.

BANK (savings per iteration)





Spent \$2 from bank to copy 2 items.

Item #3 inserted: bank \$2

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| $c_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | – | 1 | 2 | – | 4 | – | – | – | 8 | – |
| $size_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $\hat{c}_i$ | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| $bank_i$ | 1 | 2 | 2 | | | | | | | |

Amortized cost of operation $i$: $\hat{c}_i$

$$\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i \quad \text{for all } n$$

our banking game should always exagerrate true costs.

Back to dynamic tables:

let $\hat{c}_i = 3 \longrightarrow$ 1 to cover insert cost
2 for eventual doubling

implies $\sum c_i \leq 3n$

When table doubles, use 1 to copy each item.

BANK (savings per iteration)


Finally, bank account is growing

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $c_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | – | 1 | 2 | – | 4 | – | – | – | 8 | – |
| $size_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $\hat{c}_i$ | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| $bank_i$ | 1 | 2 | 2 | 4 | | | | | | |

Amortized cost of operation $i$: $\hat{c}_i$

$$\sum_{i=1}^{n} c_i \le \sum_{i=1}^{n} \hat{c}_i \quad \text{for all } n$$

our banking game should always exagerrate true costs.

Back to dynamic tables:

let $\hat{c}_i = 3$ $\longrightarrow$ 1 to cover insert cost

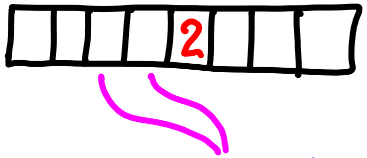
2 for eventual doubling

implies $\sum c_i \le 3n$

When table doubles, use 1 to copy each item.

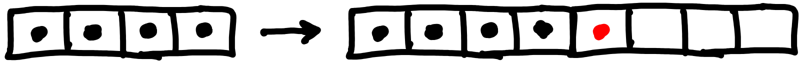
| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $c_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | – | 1 | 2 | – | 4 | – | – | – | 8 | – |
| $size_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $\hat{c}_i$ | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| $bank_i$ | 1 | 2 | 2 | 4 | 2 | | | | | |

BANK (savings per iteration)




Spent 4 to copy 4.

Banked 2.



Amortized cost of operation $i$: $\hat{c}_i$

$$\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i \quad \text{for all } n$$

our banking game should always exagerrate true costs.

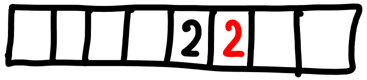
Back to dynamic tables:

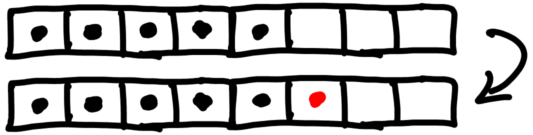
let $\hat{c}_i = 3$ → 1 to cover insert cost
2 for eventual doubling

implies $\sum c_i \leq 3n$

When table doubles, use 1 to copy each item.

BANK (savings per iteration)



| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $c_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | – | 1 | 2 | – | 4 | – | – | – | 8 | – |
| $size_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $\hat{c}_i$ | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| $bank_i$ | 1 | 2 | 2 | 4 | 2 | 4 | | | | |

Amortized cost of operation $i$: $\hat{c}_i$

$$\sum_{i=1}^{n} c_i \le \sum_{i=1}^{n} \hat{c}_i \quad \text{for all } n$$

our banking game should always exagerrate true costs.

Back to dynamic tables:

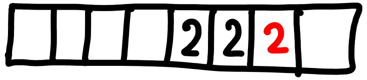
let $\hat{c}_i = 3$ $\longrightarrow$ 1 to cover insert cost
2 for eventual doubling

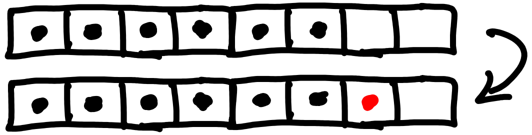
implies $\sum c_i \le 3n$

When table doubles, use 1 to copy each item.

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $c_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | – | 1 | 2 | – | 4 | – | – | – | 8 | – |
| $\text{size}_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $\hat{c}_i$ | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| $\text{bank}_i$ | 1 | 2 | 2 | 4 | 2 | 4 | 6 | | | |

BANK (savings per iteration)



Amortized cost of operation $i$: $\hat{c}_i$

$$\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i \quad \text{for all } n$$

our banking game should always exagerrate true costs.

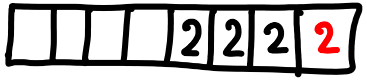
Back to dynamic tables:

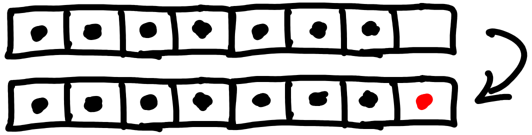
let $\hat{c}_i = 3 \longrightarrow$ 1 to cover insert cost
2 for eventual doubling

implies $\sum c_i \leq 3n$

When table doubles, use 1 to copy each item.

BANK (savings per iteration)



| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $c_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | – | 1 | 2 | – | 4 | – | – | – | 8 | – |
| $size_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $\hat{c}_i$ | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| $bank_i$ | 1 | 2 | 2 | 4 | 2 | 4 | 6 | 8 | | |

Amortized cost of operation $i$: $\hat{c}_i$

$$\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i \quad \text{for all } n$$

our banking game should always exagerrate true costs.

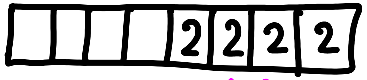
Back to dynamic tables:

let $\hat{c}_i = 3 \longrightarrow$ 1 to cover insert cost
2 for eventual doubling

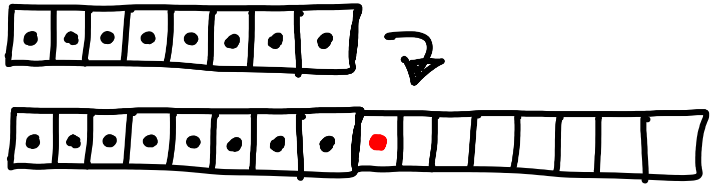
implies $\sum c_i \leq 3n$

When table doubles, use 1 to copy each item.

BANK (savings per iteration)



use all savings to copy 8 items



insert item 9

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $c_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | – | 1 | 2 | – | 4 | – | – | – | 8 | – |
| $size_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $\hat{c}_i$ | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| $bank_i$ | 1 | 2 | 2 | 4 | 2 | 4 | 6 | 8 | | |

Amortized cost of operation $i$: $\hat{c}_i$

$$\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i \quad \text{for all } n$$

our banking game should always exagerrate true costs.

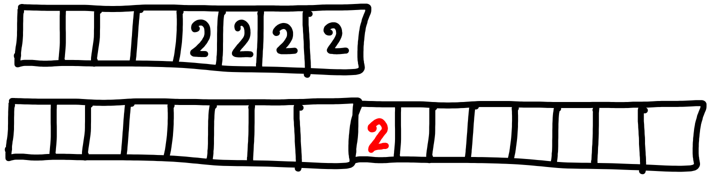
Back to dynamic tables:

let $\hat{c}_i = 3$ ⟶ 1 to cover insert cost
2 for eventual doubling

implies $\sum c_i \leq 3n$

When table doubles, use 1 to copy each item.

BANK (savings per iteration)




restore the condition:

after doubling we have \$2 in bank

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| $c_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | – | 1 | 2 | – | 4 | – | – | – | 8 | – |
| $size_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $\hat{c}_i$ | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| $bank_i$ | 1 | 2 | 2 | 4 | 2 | 4 | 6 | 8 | 2 | |

Amortized cost of operation $i$: $\hat{c}_i$

$$\underbrace{\sum_{i=1}^{n} c_i \le \sum_{i=1}^{n} \hat{c}_i \quad \text{for all } n}$$

our banking game should always exagerrate true costs.

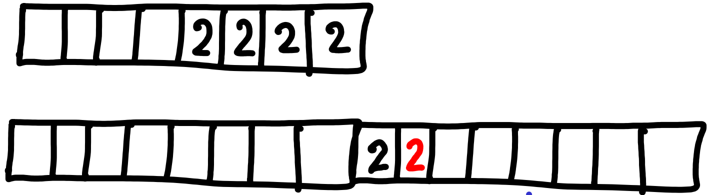
Back to dynamic tables:

let $\hat{c}_i = 3 \longrightarrow$ 1 to cover insert cost
2 for eventual doubling

implies $\sum c_i \le 3n$

When table doubles, use 1 to copy each item.

BANK (savings per iteration)




insert 10th etc

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| $c_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | – | 1 | 2 | – | 4 | – | – | – | 8 | – |
| $size_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $\hat{c}_i$ | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| $bank_i$ | 1 | 2 | 2 | 4 | 2 | 4 | 6 | 8 | 2 | 4 |

# Summary of accounting method

Estimate a cost: $\hat{c}_i$ ... higher than what you think average real cost $\frac{1}{n}\sum c_i$ will be

# Summary of accounting method

Estimate a cost: $\hat{c}_i$ ... higher than what you think average real cost $\frac{1}{n}\sum c_i$ will be

Prove that $\hat{c}_i$ is an overestimate of average $c_i$

↳ get bounds on how much you "save" & "spend"

Really does involve already having intuition.

# POTENTIAL METHOD

Start with data structure $D_0$

Operation $i$ : $D_{i-1} \rightarrow D_i$ cost : $c_i$

# POTENTIAL METHOD

Start with data structure $D_0$

Operation $i$ : $D_{i-1} \to D_i$ cost : $c_i$

Potential function $\Phi_i$ maps $D_i \to \mathbb{R}$ : potential value.

$\Phi_0 = 0$ $\Phi_i \geq 0$ $\Rightarrow$ 2 conditions that help.

# POTENTIAL METHOD

Start with data structure $D_0$

Operation $i$ : $D_{i-1} \to D_i$ cost : $c_i$

Potential function $\Phi_i$ maps $D_i \to \mathbb{R}$ : potential value.

$\Phi_0 = 0$ $\Phi_i \geq 0$ $\Rightarrow$ 2 conditions that help.

Let $\hat{c}_i = c_i + \Phi_i - \Phi_{i-1}$

$= c_i + \Delta\Phi_i$

# POTENTIAL METHOD

Start with data structure $D_0$

Operation $i$ : $D_{i-1} \to D_i$ cost : $c_i$

Potential function $\Phi_i$ maps $D_i \to \mathbb{R}$ : potential value.

$\Phi_0 = 0$ $\Phi_i \geqslant 0$ $\Rightarrow$ 2 conditions that help.

Let $\hat{c}_i = c_i + \Phi_i - \Phi_{i-1}$

$= c_i + \Delta\Phi_i$

If $\Delta\Phi_i > 0$, $\hat{c}_i > c_i$ : storing potential ... "work" in $D_i$

# POTENTIAL METHOD

aka Physicist's method

Start with data structure $D_0$

Operation $i$ : $D_{i-1} \to D_i$ cost : $c_i$

Potential function $\Phi_i$ maps $D_i \to \mathbb{R}$ : potential value.

$\Phi_0 = 0$ $\Phi_i \geq 0$ $\Rightarrow$ 2 conditions that help.

Let $\hat{c}_i = c_i + \Phi_i - \Phi_{i-1}$

$= c_i + \Delta\Phi_i$

If $\Delta\Phi_i > 0$, $\hat{c}_i > c_i$ : storing potential ... "work" in $D_i$

$\Delta\Phi_i < 0$, $\hat{c}_i < c_i$ : release work.

$$\hat{c}_i = c_i + \Delta\Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i) \quad = \; ?$$

$$\hat{c}_i = c_i + \Delta\Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i)$$

telescoping series

$$= \Phi_n - \Phi_0 + \sum_{1}^{n} c_i$$

$$\hat{c}_i = c_i + \Delta\Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i)$$

telescoping series

$$= \underset{\substack{\downarrow \\ \geq 0}}{\Phi_n} - \underset{\substack{\downarrow \\ 0}}{\Phi_0} + \sum_1^n c_i \geq \sum_1^n c_i$$

$$\hat{c}_i = c_i + \Delta\Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i)$$

telescoping series

$$= \underbrace{\underset{\downarrow \atop \geq 0}{\Phi_n} - \underset{\downarrow \atop 0}{\Phi_0} + \sum_{1}^{n} c_i \geq \sum_{1}^{n} c_i}_{}$$

so we know that the amortized cost will not underestimate real cost.

$\hat{c}_i = c_i + \Delta\Phi_i \Rightarrow \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i)$ telescoping series

$= \underbrace{\Phi_n}_{\geq 0} - \underbrace{\Phi_0}_{0} + \sum_1^n c_i \geq \sum_1^n c_i$

so we know that the amortized cost will not underestimate real cost.

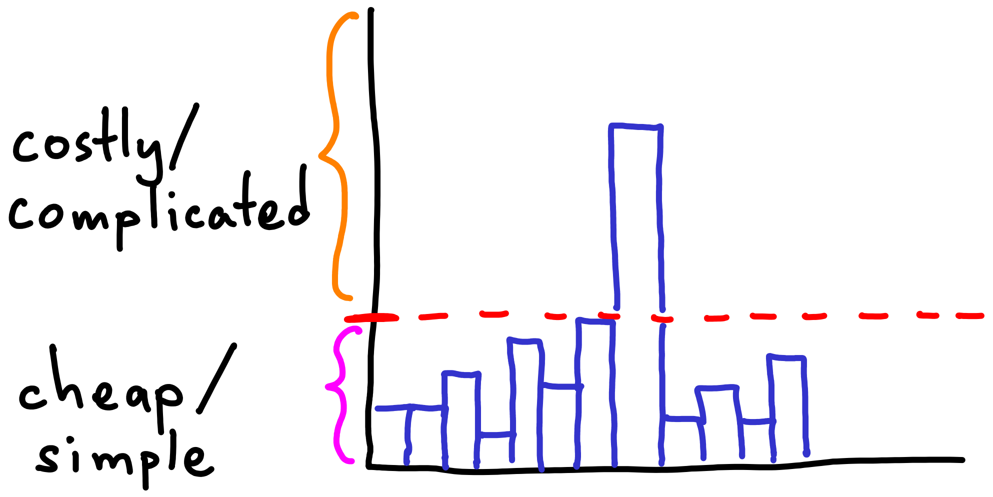
now figure out worst case for any individual $\hat{c}_i$

↳ Ideally this will give a good (and easy) bound for total cost

$$\sum c_i \leq \sum \hat{c}_i \leq n \cdot \max \hat{c}_i$$

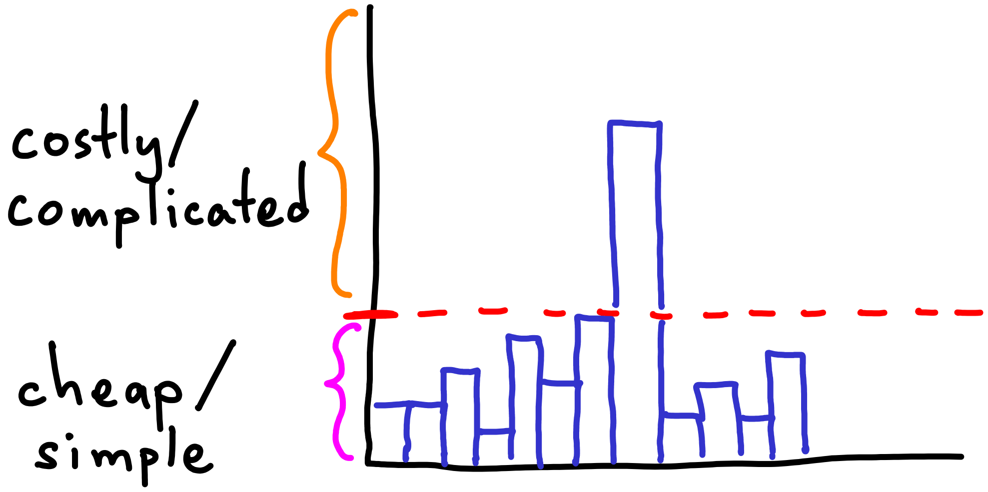
# How this works

- (Subjectively) define what a complicated / costly operation ($c_i$) is.



# How this works

- (Subjectively) define what a complicated / costly operation ($c_i$) is.
- Find something that changes a lot in the data structure in such cases.

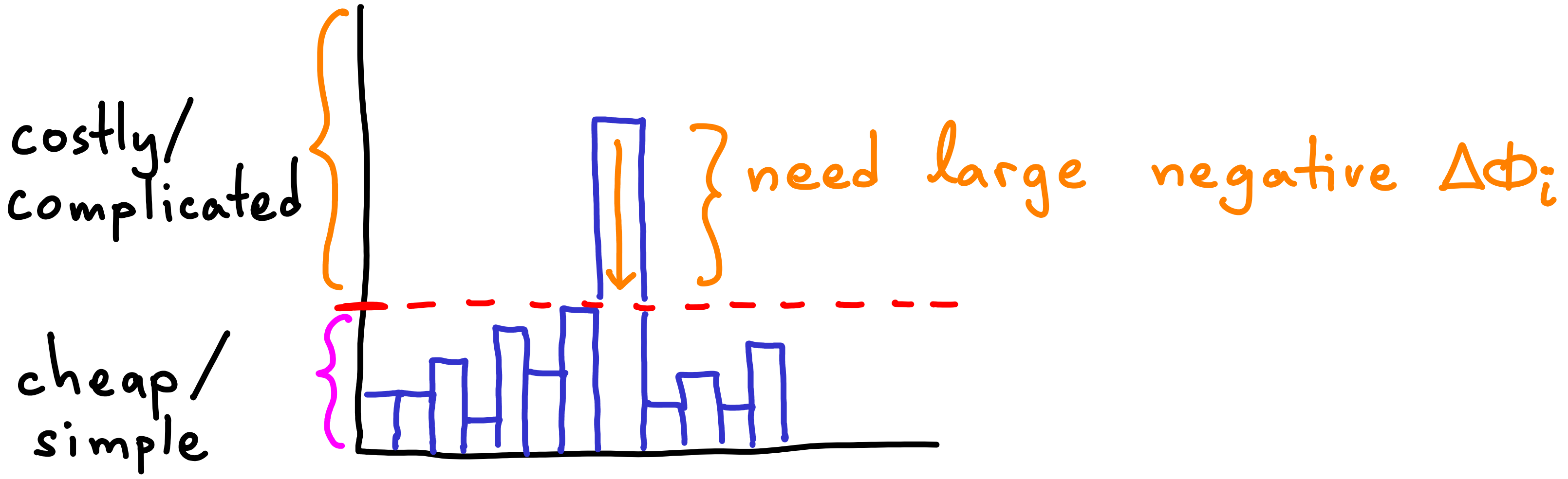


# How this works

- (Subjectively) define what a complicated / costly operation ($c_i$) is.
- Find something that changes a lot in the data structure in such cases.

  ↪ Quantify this change as $\Delta\Phi_i$ : let it "kill" $c_i$ : $\hat{c}_i = c_i + \Delta\Phi_i$

  ↪ obtain low $\hat{c}_i$

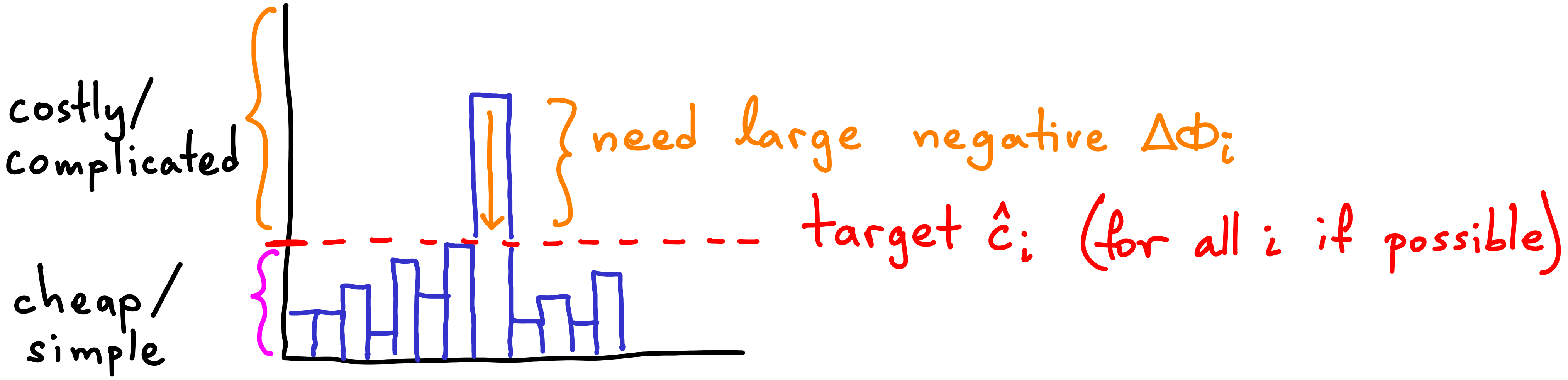


# How this works

- (Subjectively) define what a complicated / costly operation ($c_i$) is.
- Find something that changes a lot in the data structure in such cases.

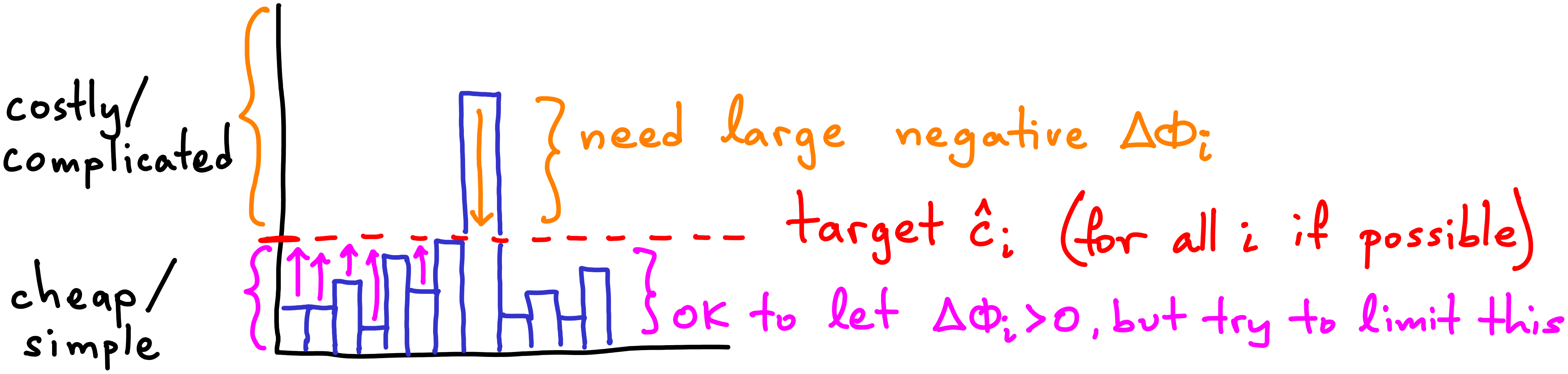
↳ Quantify this change as $\Delta\Phi_i$ : let it "kill" $c_i$ : $\hat{c}_i = c_i + \Delta\Phi_i$

↳ obtain low $\hat{c}_i$



# How this works

- (Subjectively) define what a complicated / costly operation ($c_i$) is.
- Find something that changes a lot in the data structure in such cases.
  - ↪ Quantify this change as $\Delta\Phi_i$ : let it "kill" $c_i$ : $\hat{c}_i = c_i + \Delta\Phi_i$
  - ↪ Invent your $\Phi$ accordingly — ↪ obtain low $\hat{c}_i$
  - ↪ Make sure $\Delta\Phi_i$ doesn't add much to $c_i$, when $c_i$ is cheap.



Back to dynamic tables:

$$\boxed{\hat{c}_i = c_i + \Delta\Phi_i}$$

- (Subjectively) define what a complicated / costly operation ($c_i$) is.

?

$$\hat{c}_i = c_i + \Delta\Phi_i$$

Back to dynamic tables:

- (Subjectively) define what a complicated / costly operation ($c_i$) is.
  - → whenever we insert on a full array: $c_i = \#\text{items}$

Back to dynamic tables:

$\hat{c}_i = c_i + \Delta\Phi_i$

- (Subjectively) define what a complicated / costly operation ($c_i$) is.
  ↳ whenever we insert on a full array: $c_i = \#\text{items}$
- Find something that changes a lot in the data structure in such cases.

?

$$\hat{c}_i = c_i + \Delta\Phi_i$$

# Back to dynamic tables:

- (Subjectively) define what a complicated / costly operation ($c_i$) is.
  - ↳ whenever we insert on a full array: $c_i = \#\text{items}$
- Find something that changes a lot in the data structure in such cases.
  - ↳ size of array

$$\hat{c}_i = c_i + \Delta\Phi_i$$

Back to dynamic tables:

- (Subjectively) define what a complicated / costly operation ($c_i$) is.
  - ↳ whenever we insert on a full array: $c_i = \#\text{items}$
- Find something that changes a lot in the data structure in such cases.
  - ↳ size of array → try $\Phi = -\text{size}$? → $\Delta\Phi_i \sim -\#\text{items}$

$$\hat{c}_i = c_i + \Delta\Phi_i$$

# Back to dynamic tables:

- (Subjectively) define what a complicated / costly operation ($c_i$) is.
  - ↳ whenever we insert on a full array: $c_i = \#\text{items}$
- Find something that changes a lot in the data structure in such cases.
  - ↳ size of array → try $\Phi = -\text{size}$? → $\Delta\Phi_i \sim -\#\text{items}$ ☺
    - "kills" costly $c_i$
    - doesn't hurt cheap $c_i$

$$\hat{c}_i = c_i + \Delta\Phi_i$$

# Back to dynamic tables:

- (Subjectively) define what a complicated / costly operation ($c_i$) is.

  ↪ whenever we insert on a full array: $c_i = \#\text{items}$

- Find something that changes a lot in the data structure in such cases.

  ↪ size of array → try $\Phi = -\text{size}$? → $\Delta\Phi_i \sim -\#\text{items}$ :)

  - "kills" costly $c_i$
  - doesn't hurt cheap $c_i$

  but

Back to dynamic tables:

$$\hat{c}_i = c_i + \Delta\Phi_i$$

- (Subjectively) define what a complicated / costly operation ($c_i$) is.
  - ↳ whenever we insert on a full array: $c_i = \#\text{items}$
- Find something that changes a lot in the data structure in such cases.
  - ↳ size of array → try $\Phi = -\text{size}$? → $\Delta\Phi_i \sim -\#\text{items}$ :)
    - "kills" costly $c_i$
    - doesn't hurt cheap $c_i$

but

$\Phi_i \leq 0$ :(

$\hat{c}_i = c_i + \Delta\Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i)$ telescoping series

$$= \Phi_n - \Phi_0 + \sum_{1}^{n} c_i \geq \sum_{1}^{n} c_i$$

dynamic tables: $\Phi_i = 2 \cdot (\#\text{ items in table}) - (\text{size of table})$

$\hat{c}_i = c_i + \Delta\Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i)$ telescoping series

$= \Phi_n - \Phi_0 + \sum_1^n c_i \geqslant \sum_1^n c_i$

dynamic tables: $\Phi_i = 2 \cdot (\underbrace{\#\text{ items in table}}_{\text{always } \geqslant \frac{1}{2} \text{ table}}) - (\text{size of table})$ $\left.\right\}$ $\Phi_0 = 0$, $\Phi_i \geqslant 0$

(and doesn't change rapidly) ← always $\geqslant \frac{1}{2}$ table

$\hat{c}_i = c_i + \Delta\Phi_i \Rightarrow \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i)$ telescoping series

$= \Phi_n - \Phi_0 + \sum_1^n c_i \geqslant \sum_1^n c_i$

dynamic tables: $\Phi_i = 2 \cdot \underbrace{(\#\text{ items in table)}}_{\text{always} \geqslant \frac{1}{2}\text{ table}} - (\text{size of table})$ $\left.\right\}$ $\Phi_0 = 0$, $\Phi_i \geqslant 0$

(and doesn't change rapidly) ← always $\geqslant \frac{1}{2}$ table

---

type 1 : $c_i = 1$ (when element $i$ doesn't trigger a doubling)

type 2 : $c_i = i$ (when element $i$ does trigger a doubling)

$$\hat{c}_i = c_i + \Delta\Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i)$$

telescoping series

$$= \Phi_n - \Phi_0 + \sum_1^n c_i \geq \sum_1^n c_i$$

dynamic tables: $\Phi_i = 2 \cdot \underbrace{(\#\text{ items in table})}_{\text{always } \geq \frac{1}{2} \text{ table}} - (\text{size of table})$ $\left.\right\}$ $\Phi_0 = 0$, $\Phi_i \geq 0$

(and doesn't change rapidly) ←

---

type 1 : $c_i = 1$ (when element i doesn't trigger a doubling)

$$\hat{c}_i = \; ? \quad ? \quad ?$$

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

$\hat{c}_i = c_i + \Delta\Phi_i \Rightarrow \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i)$ telescoping series

$= \Phi_n - \Phi_0 + \sum_1^n c_i \geq \sum_1^n c_i$

dynamic tables: $\Phi_i = 2 \cdot (\underbrace{\#\text{ items in table}}_{\text{always} \geq \frac{1}{2} \text{table}}) - (\text{size of table})$ $\left.\right\}$ $\Phi_0 = 0$, $\Phi_i \geq 0$

(and doesn't change rapidly) ← always $\geq \frac{1}{2}$ table

---

type 1 : $c_i = 1$ (when element $i$ doesn't trigger a doubling)

$$\hat{c}_i = \underset{c_i}{1} + \underset{\Phi_i}{\left[2i - \text{Size}_i\right]} \underset{-}{-} \underset{\Phi_{i-1}}{\left[2(i-1) - \text{Size}_{i-1}\right]}$$

$\hat{c}_i = c_i + \Delta\Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i)$ telescoping series

$$= \Phi_n - \Phi_0 + \sum_1^n c_i \geq \sum_1^n c_i$$

dynamic tables: $\Phi_i = 2 \cdot \underbrace{(\#\text{ items in table})}_{\text{always} \geq \frac{1}{2}\text{ table}} - (\text{size of table})$ $\Big\}$ $\Phi_0 = 0$, $\Phi_i \geq 0$

(and doesn't change rapidly) ← always $\geq \frac{1}{2}$ table

---

type 1 : $c_i = 1$ (when element $i$ doesn't trigger a doubling)

$$\hat{c}_i = 1 + [2i - \cancel{\text{Size}_i}] - [2(i-1) - \cancel{\text{Size}_{i-1}}]$$

$\hat{c}_i = c_i + \Delta\Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i)$ telescoping series

$= \Phi_n - \Phi_0 + \sum_1^n c_i \geq \sum_1^n c_i$

dynamic tables: $\Phi_i = 2 \cdot (\underbrace{\#\text{items in table}}_{\text{always} \geq \frac{1}{2}\text{ table}}) - (\text{size of table})$ $\left.\begin{matrix} \Phi_0 = 0 \\ \Phi_i \geq 0 \end{matrix}\right.$

(and doesn't change rapidly) ←

---

type 1 : $c_i = 1$ (when element $i$ doesn't trigger a doubling)

$\hat{c}_i = 1 + [2i - \cancel{\text{Size}_i}] - [2(i-1) - \cancel{\text{Size}_{i-1}}] = 3$

$\hat{c}_i = c_i + \Delta\Phi_i \Rightarrow \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i)$ telescoping series

$= \Phi_n - \Phi_0 + \sum_1^n c_i \geq \sum_1^n c_i$

dynamic tables: $\Phi_i = 2 \cdot (\underbrace{\#\text{ items in table}}_{\text{always} \geq \frac{1}{2} \text{ table}}) - (\text{size of table})$ $\left.\right\}$ $\Phi_0 = 0$, $\Phi_i \geq 0$

(and doesn't change rapidly) ←

---

type 1 : $c_i = 1$ (when element i doesn't trigger a doubling)

$\hat{c}_i = 1 + [2i - \cancel{\text{Size}_i}] - [2(i-1) - \cancel{\text{Size}_{i-1}}] = 3$

type 2 : $c_i = i$ (when element i does trigger a doubling)

$\hat{c}_i = ?$

$\hat{c}_i = c_i + \Delta\Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i)$ telescoping series

$$= \Phi_n - \Phi_0 + \sum_1^n c_i \geq \sum_1^n c_i$$

dynamic tables: $\Phi_i = 2 \cdot \underbrace{(\#\text{ items in table})}_{\text{always } \geq \frac{1}{2} \text{ table}} - (\text{size of table})$ $\left.\right\}$ $\Phi_0 = 0$, $\Phi_i \geq 0$

(and doesn't change rapidly) ←

---

type 1 : $c_i = 1$ (when element $i$ doesn't trigger a doubling)

$$\hat{c}_i = 1 + [2i - \cancel{\text{Size}_i}] - [2(i-1) - \cancel{\text{Size}_{i-1}}] = 3$$

type 2 : $c_i = i$ (when element $i$ does trigger a doubling)

$$\hat{c}_i = c_i + \Phi_i - \Phi_{i-1}$$

$\hat{c}_i = c_i + \Delta\Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i)$ telescoping series

$= \Phi_n - \Phi_0 + \sum_1^n c_i \geq \sum_1^n c_i$

dynamic tables: $\Phi_i = 2 \cdot \underbrace{(\#\text{ items in table})}_{\text{always} \geq \frac{1}{2}\text{ table}} - (\text{size of table})$ $\left.\begin{matrix}\Phi_0 = 0 \\ \Phi_i \geq 0\end{matrix}\right.$

(and doesn't change rapidly) ←

---

type 1 : $c_i = 1$ (when element $i$ doesn't trigger a doubling)

$\hat{c}_i = 1 + [2i - \cancel{\text{Size}_i}] - [2(i-1) - \cancel{\text{Size}_{i-1}}] = 3$

type 2 : $c_i = i$ (when element $i$ does trigger a doubling)

$\hat{c}_i = i + [2i - \text{Size}_i] - [2(i-1) - \text{Size}_{i-1}]$

$\hat{c}_i = c_i + \Delta\Phi_i \Rightarrow \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i)$ telescoping series

$= \Phi_n - \Phi_0 + \sum_1^n c_i \geq \sum_1^n c_i$

dynamic tables: $\Phi_i = 2 \cdot (\underbrace{\#\text{ items in table}}_{\text{always} \geq \frac{1}{2}\text{ table}}) - (\text{size of table})$ $\left.\begin{matrix} \Phi_0 = 0 \\ \Phi_i \geq 0 \end{matrix}\right\}$

(and doesn't change rapidly) ← always $\geq \frac{1}{2}$ table

---

type 1 : $c_i = 1$ (when element $i$ doesn't trigger a doubling)

$$\hat{c}_i = 1 + [2i - \cancel{Size_i}] - [2(i-1) - \cancel{Size_{i-1}}] = 3$$

type 2 : $c_i = i$ (when element $i$ does trigger a doubling)

$$\hat{c}_i = i + [2i - \underbrace{Size_i}_{?}] - [2(i-1) - \underbrace{Size_{i-1}}_{?}]$$

$$\hat{c}_i = c_i + \Delta\Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i)$$ telescoping series

$$= \Phi_n - \Phi_0 + \sum_1^n c_i \geq \sum_1^n c_i$$

dynamic tables: $\Phi_i = 2 \cdot \underbrace{(\#\text{ items in table})}_{\text{always } \geq \frac{1}{2} \text{ table}} - (\text{size of table})$ $\Big\}$ $\Phi_0 = 0$, $\Phi_i \geq 0$

(and doesn't change rapidly) ← always $\geq \frac{1}{2}$ table

---

type 1 : $c_i = 1$ (when element $i$ doesn't trigger a doubling)

$$\hat{c}_i = 1 + [2i - \cancel{\text{Size}_i}] - [2(i-1) - \cancel{\text{Size}_{i-1}}] = 3$$

type 2 : $c_i = i$ (when element $i$ does trigger a doubling)

$$\hat{c}_i = i + [2i - \text{Size}_i] - [2(i-1) - \text{Size}_{i-1}]$$

$$= i + [2i - 2(i-1)] - [2(i-1) - (i-1)]$$

$$\hat{c}_i = c_i + \Delta\Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i)$$ telescoping series

$$= \Phi_n - \Phi_0 + \sum_1^n c_i \geq \sum_1^n c_i$$

dynamic tables: $\Phi_i = 2\cdot\underbrace{(\#\text{items in table})}_{\text{always} \geq \frac{1}{2}\text{ table}} - (\text{size of table})$ $\left.\begin{matrix}\Phi_0 = 0 \\ \Phi_i \geq 0\end{matrix}\right\}$

(and doesn't change rapidly) ←

---

type 1 : $c_i = 1$ (when element $i$ doesn't trigger a doubling)

$$\hat{c}_i = 1 + [2i - \cancel{\text{Size}_i}] - [2(i-1) - \cancel{\text{Size}_{i-1}}] = 3$$

type 2 : $c_i = i$ (when element $i$ does trigger a doubling)

$$\hat{c}_i = i + [2i - \text{Size}_i] - [2(i-1) - \text{Size}_{i-1}]$$

$$= i + [2i - 2(i-1)] - [2(i-1) - (i-1)] = 3i - 3(i-1)$$

$\hat{c}_i = c_i + \Delta\Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i)$ telescoping series

$$= \Phi_n - \Phi_0 + \sum_{1}^{n} c_i \geq \sum_{1}^{n} c_i$$

dynamic tables: $\Phi_i = 2 \cdot \underbrace{(\#\text{items in table})}_{\text{always} \geq \frac{1}{2}\text{table}} - (\text{size of table}) \left\} \begin{array}{l} \Phi_0 = 0 \\ \Phi_i \geq 0 \end{array} \right.$

(and doesn't change rapidly) ← always $\geq \frac{1}{2}$ table

---

type 1 : $c_i = 1$ (when element $i$ doesn't trigger a doubling)

$$\hat{c}_i = 1 + [2i - \cancel{\text{Size}_i}] - [2(i-1) - \cancel{\text{Size}_{i-1}}] = 3$$

type 2 : $c_i = i$ (when element $i$ does trigger a doubling)

$$\hat{c}_i = i + [2i - \text{Size}_i] - [2(i-1) - \text{Size}_{i-1}]$$

$$= i + [2i - 2(i-1)] - [2(i-1) - (i-1)] = 3i - 3(i-1) = 3$$

$$\hat{c}_i = c_i + \Delta\Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta\Phi_i)$$

telescoping series

$$= \Phi_n - \Phi_0 + \sum_{1}^{n} c_i \geq \sum_{1}^{n} c_i$$

dynamic tables: $\Phi_i = 2 \cdot \underbrace{(\#\text{ items in table})}_{\text{always} \geq \frac{1}{2}\text{ table}} - (\text{size of table})$ $\left.\right\}$ $\Phi_0 = 0$, $\Phi_i \geq 0$

(and doesn't change rapidly) ← always $\geq \frac{1}{2}$ table

---

type 1 : $c_i = 1$ (when element i doesn't trigger a doubling)

$$\hat{c}_i = 1 + [2i - \cancel{Size_i}] - [2(i-1) - \cancel{Size_{i-1}}] = 3$$

type 2 : $c_i = i$ (when element i does trigger a doubling)

$$\hat{c}_i = i + [2i - Size_i] - [2(i-1) - Size_{i-1}]$$

$$= i + [2i - 2(i-1)] - [2(i-1) - (i-1)] = 3i - 3(i-1) = 3$$

$$\boxed{\sum \hat{c}_i = 3n}$$

Never needed to know how many of each type, or order of operations

# Summary of potential method

Amortized cost is calculated for each operation.
↳ or at least each "type" of operation

Conclude
$n \cdot$ worst type
OR
analyze further

# Summary of potential method

Amortized cost is calculated for each operation.
↳ or at least each "type" of operation

Conclude
n · worst type
OR
analyze further

Requires an inspired choice of function $\Phi$
↳ typically something that changes a lot when you have those rare expensive operations.

# Summary of potential method

Amortized cost is calculated for each operation.
↳ or at least each "type" of operation

Conclude $n \cdot$ worst type OR analyze further

Requires an inspired choice of function $\Phi$
↳ typically something that changes a lot when you have those rare expensive operations.

Once you have $\Phi$, the rest can be easy.

Find $\Delta\Phi$ once for each "type"