Suppose you want to use an array:

You must support insertion but you don't know maximal 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

\[ \text{start: } \square \]
\[ \text{insert: } \square \rightarrow \square \square \]
\[ \text{(make a new larger array & copy)} \]
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

\$(\text{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 the maximum number of elements.

Array doubling: start with an array of size 1; every time it fills up, double the size.

Start: \[
\begin{array}{c}
\emptyset
\end{array}
\]

Insert: \[
\begin{array}{c}
\bullet
\end{array}
\]

Insert: \[
\begin{array}{c}
\bullet
\end{array}
\rightarrow
\begin{array}{c}
\bullet \bullet
\end{array}
\]

Insert: \[
\begin{array}{c}
\bullet \bullet
\end{array}
\rightarrow
\begin{array}{c}
\bullet \bullet \bullet
\end{array}
\]

\[
\begin{array}{c}
\bullet \bullet \bullet \bullet
\end{array}
\rightarrow
\begin{array}{c}
\bullet \bullet \bullet \bullet \bullet
\end{array}
\]

\(\text{(make a new larger array & copy)}\)
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: 
- insert: 
- insert: 
- insert: 
- insert: 
- insert: 
- insert: 
- insert: 
- insert: 
- insert: 
- insert:
DYNAMIC TABLES & AMORTIZED ANALYSIS

Suppose you want to use an array:

You must support insertion but you don't know the maximum number of elements.

Array doubling: start with an array of size 1; every time it fills up, double the size and make a new larger array & copy.

\[
\begin{align*}
\text{start:} & \quad [] \\
\text{insert:} & \quad [\ ] \\
\text{insert:} & \quad [\ 1\ ] \\
\text{insert:} & \quad [\ 1\ 2\ ] \\
\text{insert:} & \quad [\ 1\ 2\ 3\ ] \\
\text{insert:} & \quad [\ 1\ 2\ 3\ 4\ ] \\
\text{insert:} & \quad [\ 1\ 2\ 3\ 4\ 5\ ] \\
\text{insert:} & \quad [\ 1\ 2\ 3\ 4\ 5\ 6\ ] \\
\text{insert:} & \quad [\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ ] \\
\end{align*}
\]
**Dynamic Tables & Amortized Analysis**

Suppose you want to use an array:

You must support insertion but you don’t know the maximum number of elements.

**Array Doubling:** start with an array of size 1; every time it fills up, double the size and make a new larger array & copy.

\[ n : \text{total number of inserts} \]

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

\[ \text{(make a new larger array & copy)} \]

\[ n \text{: total number of inserts} \]

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

\( \text{for } n \text{ inserts: } O(n^2) \)
Suppose you want to use an array:

You must support insertion but you don’t know the maximum number of elements.

Array doubling: start with an array of size 1; every time it fills up, double the size.

\[
\text{(make a new larger array & copy)}
\]

\( n \): total number of inserts.

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

\( \Rightarrow \) for \( n \) inserts: \( O(n^2) \)

Claim: for \( n \) inserts: also \( O(n) \)
start: □
insert: ◯
insert: □ → □
insert: □□ → □□

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

\[
\begin{array}{cccccccccccc}
i & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
2^0 & 2^1 & 2^{1+} & 2^2 & 2^{2+} & 2^3 & 2^{3+} & 2^4 & 2^{4+} & 2^5 & 2^{5+} & 2^6 & 2^{6+} \\
\text{Size}_i & 1 & 2 & 4 & 4 & 8 & 8 & 8 & 8 & 16 & 16 \\
C_i & 1 & 2 & 3 & 1 & 5 & 1 & 1 & 1 & 1 & 9 & 1
\end{array}
\]
\[ \text{cost } c_i \begin{cases} i & \text{if } i-1 \text{ is a power of 2} \\ 1 & \text{otherwise} \end{cases} \]
\[
\text{cost } c_i \begin{cases} 
  i & \text{if } i-1 \text{ is a power of 2} \\
  1 & \text{otherwise}
\end{cases}
\]

\[C_i = \text{copy } i \text{ elements}\]

(make array: free)

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

\[
\begin{array}{ccccccccc}
  i & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
  2^0 & 2^1 & 2^2 & 2^2 & 2^2 & 2^2 & 2^3 & 2^3 & 2^3 & 2^3 \\
  \text{size}_i & 1 & 2 & 4 & 4 & 8 & 8 & 8 & 8 & 16 & 16 \\
  C_i & 1 & 2 & 3 & 1 & 5 & 1 & 1 & 1 & 9 & 1 \\
  \{ & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]
\[ \text{cost } c_i \begin{cases} i & \text{if } i-1 \text{ is a power of 2} \\ 1 & \text{otherwise} \end{cases} \]

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

\[ i \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10 \]

\[ 2^0 \quad 2^1 \quad 2^1 \quad 2^2 \quad 2^2 \quad 2^2 \quad 2^2 \quad 2^3 \quad 2^3 \quad 2^3 \]

\[ \text{Size}_i \]

\[ 1 \quad 2 \quad 4 \quad 4 \quad 8 \quad 8 \quad 8 \quad 8 \quad 16 \quad 16 \]

\[ C_i \]

\[ 1 \quad 2 \quad 3 \quad 1 \quad 5 \quad 1 \quad 1 \quad 1 \quad 9 \quad 1 \]

\[ \begin{cases} -1 \quad 2 \quad -4 \quad -8 \quad - \quad - \quad 2^0 \quad 2^1 \quad 2^2 \quad 2^3 \end{cases} \]
\[ \text{cost } c_i \begin{cases} i & \text{if } i-1 \text{ is a power of 2} \\ 1 & \text{otherwise} \end{cases} \]

\[ \text{cost}(n) = \sum_{i=1}^{n} c_i = n + \]
\[
\text{cost } c_i = \begin{cases} i & \text{if } i-1 \text{ is a power of } 2 \\ 1 & \text{otherwise} \end{cases}
\]

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

\[
\begin{array}{cccccccccccc}
  i & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
 2^i & 2 & 2^2 & 2 & 2^4 & 2^2 & 2^4 & 2^2 & 2^8 & 2^2^3 & 2^3^3
\end{array}
\]

\[
\begin{array}{cccccccccccc}
  \text{size}_i & 1 & 2 & 4 & 4 & 8 & 8 & 8 & 8 & 16 & 16 \\
  C_i & 1 & 2 & 3 & 1 & 5 & 1 & 1 & 1 & 9 & 1
\end{array}
\]

\[
\begin{array}{cccccccccccc}
  2^i & 2^2 & 2^3 & \text{---} & \text{---} & \text{---} & \text{---} & \text{---} & \text{---} & \text{---} & \text{---} \\
  2^3
\end{array}
\]

\[
2^3 < 2^3
\]
The cost $c_i$ is defined as:

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

The cost function $\text{cost}(n)$ is given by:

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

The table shows:

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^{i-1}$</td>
<td>$2^0$</td>
<td>$2^1$</td>
<td>$2^1$</td>
<td>$2^2$</td>
<td>$2^2$</td>
<td>$2^2$</td>
<td>$2^2$</td>
<td>$2^3$</td>
<td>$2^3$</td>
<td>$2^3$</td>
</tr>
</tbody>
</table>

| $\text{Size}_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |

| $C_i$ | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9 | 1 |

The values in the table are connected by arrows indicating dependencies or relationships. The image also includes a diagram with arrows pointing to specific values, which seem to illustrate the relationships between different elements in the problem.
\[ \text{cost } c_i \begin{cases} i & \text{if } i-1 \text{ is a power of 2} \\ 1 & \text{otherwise} \end{cases} \]

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

**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

costly/complicated \{ \} \rightarrow \text{you decide} \rightarrow f(n)

cheap/simple \{ \}
3 main ways for amortizing: aggregate, accounting, & potential method.

Accounting: saving for a rainy day

"simple" operations

costly/complicated

cheap/simple

you decide

f(n)
3 main ways for amortizing: aggregate, accounting, & potential method.

**Accounting:** saving for a rainy day

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

$\bowtie$ "save" the difference

```
<table>
<thead>
<tr>
<th>costly/complicated</th>
<th>cheap/simple</th>
</tr>
</thead>
<tbody>
<tr>
<td>┌─────┐</td>
<td>┌─────┐</td>
</tr>
<tr>
<td>└─────┘</td>
<td>└─────┘</td>
</tr>
</tbody>
</table>
```

you decide

$f(n)$
3 main ways for amortizing: aggregate, accounting, & potential method.

Accounting: saving for a rainy day

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

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

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

costly/complicated \{ \}

cheap/simple \{ \}

\[ f(n) \]

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

**Accounting:** saving for a rainy day

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

↓ "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 $O(\text{real cost})$.

↓ "save" the difference  $\rightarrow$ "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 $\Theta(\text{real cost})$

 gzip “save” the difference $\rightarrow$ “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: $n \cdot f(n) \geq \text{true cost}$
3 main ways for amortizing: aggregate, accounting, & potential method.

**Accounting:** saving for a rainy day

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

ług "save" the difference $\Rightarrow$ "spend" what you saved up.

“Complicated/costly” operations: pretend they cost less; pay excess via savings.

**costly/complicated**

{ 
- you decide
- $f(n)$

**cheap/simple**

{ 
- Rule: Never spend more than what you saved.
- Amortized cost: $n \cdot f(n) \geq \text{true cost}$
- Goal: exaggerate/save as little as possible. minimize $f(n)$
- etc
Amortized cost of operation $i$: $\hat{c}_i$
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 exaggerate true costs.
Amortized cost of operation $i$: $\hat{C}_i$

Back to dynamic tables:

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

our banking game should always exaggerate true costs.

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_i$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>$\hat{C}_i$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
</tr>
<tr>
<td>$\sum_{i=1}^{\infty} c_i$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
</tr>
<tr>
<td>$\sum_{i=1}^{\infty} \hat{c}_i$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
</tr>
<tr>
<td>size $i$</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>16</td>
<td>16</td>
</tr>
</tbody>
</table>
Amortized cost of operation $i$: $\hat{C}_i$

Back to dynamic tables:

Let $\hat{C}_i = 3 \rightarrow 1$ to cover insert cost
2 for eventual doubling

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

Our banking game should always exaggerate true costs.

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_i$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$\hat{C}_i$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$\text{size}_i$</td>
<td>1</td>
<td>2</td>
<td>-</td>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>8</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$\hat{C}_i$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Amortized cost of operation $i$: $\hat{C}_i$

Back to dynamic tables:

Let $\hat{C}_i = 3 \rightarrow 1$ to cover insert cost implies $\sum c_i \leq 3n$

2 for eventual doubling

\[ \sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{C}_i \] for all $n$

our banking game should always exaggerate true costs.

\[
\begin{array}{ccccccccccc}
& 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\hline
i & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
C_i & 1 & 2 & 3 & 1 & 5 & 1 & 1 & 1 & 9 & 1 \\
\text{size}_i & 1 & 1 & i & i & i & i & i & i & i & i \\
\hat{C}_i & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\
\end{array}
\]
Amortized cost of operation $i$: $\hat{c}_i$

Back to dynamic tables:

let $\hat{c}_i = 3 \rightarrow 1$ to cover insert cost
implies $\sum c_i \leq 3n$ 2 for eventual doubling

When table doubles, use 1 to copy each item.

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

- our banking game should always exaggerate true costs.

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_i$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>$\hat{c}_i$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$size_i$</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>$\hat{c}_i$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Amortized cost of operation $i$: $\hat{C}_i$

Back to dynamic tables:

Let $\hat{C}_i = 3 \rightarrow 1$ to cover insert cost implies $\sum C_i \leq 3n$ for eventual doubling when table doubles, use 1 to copy each item.

**BANK** (savings per iteration)

Pretend cost is 3
pay 1 to insert, save/bank 2

\[
\sum_{i=1}^{n} C_i \leq \sum_{i=1}^{n} \hat{C}_i \quad \text{for all } n
\]

Our banking game should always exaggerate true costs.

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_i$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>$\hat{C}_i$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>size</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>16</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

\[ \hat{C}_i = \boxed{3} \]

3 3 3 3 3 3 3 3 3 3
Amortized cost of operation $i$: $\hat{C}_i$

Back to dynamic tables:

let $\hat{C}_i = 3 \rightarrow 1$ to cover insert cost implies $\sum c_i \leq 3n$ for eventual doubling

When table doubles, use 1 to copy each item.

BANK (savings per iteration)

\[ \begin{array}{ccccccccccccc}
\text{i} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
C_i & 1 & 2 & 3 & 1 & 5 & 1 & 1 & 1 & 9 & 1 \\
size_i & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array} \]

Pretend cost is $\mathcal{O}(2)$

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

We can even give $\$5$ to charity

\[ \sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{C}_i \quad \text{for all } n \]

our banking game should always exagerrate true costs.

Just this one time
Amortized cost of operation $i$: $\hat{C}_i$

Back to dynamic tables:

let $\hat{C}_i = 3 \rightarrow 1$ to cover insert cost implies $\Sigma c_i \leq 3n$ for eventual doubling

When table doubles, use $1$ to copy each item.

**BANK** (savings per iteration)

Spent $1$ to copy $1$ item into new array. $\rightarrow$ left $\emptyset$ in bank.

Added $2$nd item & banked $\$2$

\[
\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{C}_i \quad \text{for all } n
\]

our banking game should always exaggerate true costs.

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_i$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>$\hat{C}_i$</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$\text{size}_i$</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>$\text{bank}_i$</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Amortized cost of operation $i$: $\hat{C}_i$

Back to dynamic tables:

let $\hat{C}_i = 3 \rightarrow 1$ to cover insert cost
implies $\sum c_i \leq 3n$ 2 for eventual doubling
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 $2$
Amortized cost of operation $i$: $\hat{C}_i$

Back to dynamic tables:

let $\hat{C}_i = 3 \rightarrow 1$ to cover insert cost
implies $\sum c_i \leq 3n$ 2 for eventual doubling
When table doubles, use 1 to copy each item.

BANK (savings per iteration)

Finally, bank account is growing

\[
\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{C}_i \quad \text{for all } n
\]

our banking game should always exaggerate true costs.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_i$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>$\hat{C}_i$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{size}_i &= 1, 2, 4, 4, 8, 8, 8, 8, 16, 16 \\
\hat{C}_i &= 2, 3, 3, 3, 3, 3, 3, 3, 3, 3 \\
\text{bank}_i &= 1, 2, 2, 4
\end{align*}
\]
Amortized cost of operation $i$: $\hat{C}_i$

Back to dynamic tables:

let $\hat{C}_i = 3 \rightarrow 1$ to cover insert cost
implies $\sum C_i \leq 3n$ for eventual doubling

When table doubles, use 1 to copy each item.

$\mathbf{BANK}$ (savings per iteration)

Spent 4 to copy 4.
Banked 2.

\[
\sum_{i=1}^{n} C_i \leq \sum_{i=1}^{n} \hat{C}_i \quad \text{for all } n
\]

our banking game should always exagerrate true costs.

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4-5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_i$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>bank$_i$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>size$_i$</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>$\hat{C}_i$</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{Bank:} & & 1 & 2 & 2 & 4 & 2
\end{align*}
\]
Amortized cost of operation $i$: $\hat{C}_i$

Back to dynamic tables:

Let $\hat{C}_i = 3 \rightarrow 1$ to cover insert cost

implies $\sum C_i \leq 3n$ for eventual doubling

When table doubles, use 1 to copy each item.

**BANK** (savings per iteration)

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4-5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_i$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$\hat{C}_i$</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$\text{bank}_i$</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$\sum_{i=1}^{n} C_i \leq \sum_{i=1}^{n} \hat{C}_i$ for all $n$

our banking game should always exaggerate true costs.
Amortized cost of operation $i$: $\hat{C}_i$

Back to dynamic tables:

Let $\hat{C}_i = 3 \rightarrow 1$ to cover insert cost implies $\sum C_i \leq 3n$ for eventual doubling.

When table doubles, use 1 to copy each item.

**BANK (savings per iteration)**

Our banking game should always exaggerate true costs.

\[
\sum_{i=1}^{n} C_i \leq \sum_{i=1}^{n} \hat{C}_i \quad \text{for all } n
\]
Amortized cost of operation $i$: $\hat{C}_i$

Back to dynamic tables:

let $\hat{C}_i = 3 \rightarrow 1$ to cover insert cost

implies $\sum c_i \leq 3n$ 2 for eventual doubling

When table doubles, use 1 to copy each item.

```
<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_i$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>1</td>
</tr>
</tbody>
</table>
```

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

our banking game should always exaggerate true costs.

```
<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{size}_i$</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>$\hat{C}_i$</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>bank</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Amortized cost of operation $i$: $\hat{C}_i$

Back to dynamic tables:

Let $\hat{C}_i = 3 \rightarrow 1$ to cover insert cost implies $\sum c_i \leq 3n$ for eventual doubling.

When table doubles, use 1 to copy each item.

**BANK (savings per iteration)**

use all savings to copy 8 items

insert item 9

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

our banking game should always exaggerate true costs.

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_i$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>$i$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$\times$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>size</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>$\hat{C}_i$</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>bank</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>
Amortized cost of operation $i$: $\hat{C}_i$

Back to dynamic tables:

Let $\hat{C}_i = 3 \rightarrow 1$ to cover insert cost implies $\sum c_i \leq 3n$ for eventual doubling.

When table doubles, use 1 to copy each item.

**BANK (savings per iteration)**

restore the condition:

after doubling we have $\$2$ in bank

\[
\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{C}_i \quad \text{for all } n
\]

our banking game should always exaggerate true costs.
Amortized cost of operation $i$: $\hat{C}_i$

Back to dynamic tables:

Let $\hat{C}_i = 3 \rightarrow 1$ to cover insert cost implies $\Sigma c_i \leq 3n$ 2 for eventual doubling.

When table doubles, use 1 to copy each item.

**Bank** (savings per iteration)

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_i$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>size_i</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>$\hat{C}_i$</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>bank_i</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

$\Sigma_{i=1}^{n} C_i \leq \Sigma_{i=1}^{n} \hat{C}_i$ for all $n$

Our banking game should always exaggerate true costs.
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 $i : D_{i-1} \rightarrow D_i$  \hspace{1cm} \text{cost} : c_i
**Potential Method**

Start with data structure $D_0$

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

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

$\Phi_0 = 0 \quad \Phi_i > 0 \quad \Rightarrow \ 2 \ \text{conditions that help.}$
**Potential Method**

Start with data structure $D_0$

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

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

$\Phi_0 = 0 \quad \Phi_i \geq 0 \quad \Rightarrow \text{2 conditions that help}$

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 $i: D_{i-1} \rightarrow D_i$ \hspace{1cm} \text{cost: } c_i$

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

$\Phi_0 = 0 \hspace{1cm} \Phi_i \geq 0 \hspace{1cm} \Rightarrow 2$ conditions that help.

Let $\hat{c}_i = c_i + \Phi_i - \Phi_{i-1} \bigg\{ \begin{array}{ll} 1 & \text{if } \Delta \Phi_i > 0, \hat{c}_i > c_i : \text{storing potential} \\ = c_i + \Delta \Phi_i & \end{array}$

... "work" in $D_i$
**Potential Method** aka Physicists' method

Start with data structure $D_0$

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

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

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

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

$\begin{align*}
= c_i + \Delta \Phi_i \quad \left\{ \begin{array}{ll}
\text{if } \Delta \Phi_i > 0, & \hat{c}_i > c_i: \text{ storing potential} \\
\text{if } \Delta \Phi_i < 0, & \hat{c}_i < c_i: \text{ release work.}
\end{array} \right.
\end{align*}$
\[ \hat{c}_i = c_i + \Delta \phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \hat{\sum}_{i \in \mathbb{I}} (c_i + \Delta \phi_i) = ? \]
\[ \hat{c}_i = c_i + \Delta \phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta \phi_i) \quad \text{telescoping series} \]
\[ = \phi_n - \phi_0 + \sum_i^\infty c_i \]
\[ \hat{c}_i = c_i + \Delta \Phi_i \implies \sum \hat{c}_i = \sum_{i=1}^{\hat{n}} (c_i + \Delta \Phi_i) \quad \text{telescoping series} \]

\[ = \Phi_n - \Phi_0 + \sum_{i}^{\hat{n}} c_i \quad \Downarrow \quad \Downarrow \quad \Downarrow \quad 0 \quad 0 \quad 0 \]
\( \hat{c}_i = c_i + \Delta \Phi_i \) \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta \Phi_i) \quad \text{telescoping series}

\[
= \Phi_n - \Phi_0 + \sum_{i}^{n} c_i \\
\gg 0 \quad \gg 0
\]

so we know that the amortized cost will not underestimate real cost.
\[ \hat{c}_i = c_i + \Delta \Phi_i \implies \sum \hat{c}_i = \sum_{i=1}^{\hat{n}} (c_i + \Delta \Phi_i) \text{ telescoping series} \]

\[ = \Phi_n - \Phi_0 + \sum c_i \implies \sum c_i \]

\[ \geq 0 \quad \Downarrow \quad \Rightarrow \]

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

\[ \sum c_i \leq \sum \hat{c}_i \leq n \cdot \max \hat{c}_i \]
(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.

\[ \Delta \Phi_i: \text{let it "kill" } c_i: \quad \hat{c}_i = c_i + \Delta \Phi_i \]

\( \\text{obtain low } \hat{c}_i \)

- Need large negative \( \Delta \Phi_i \) for costly/complicated operations.
- Need small positive \( \Delta \Phi_i \) for cheap/simple operations.
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.

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

$\Rightarrow$ obtain low $\hat{c}_i$

\begin{align*}
\left\{ \text{costly/complicated} \right\} & \rightarrow \{ \text{need large negative } \Delta \Phi_i \} \\
\left\{ \text{cheap/simple} \right\} & \rightarrow \text{target } \hat{c}_i \text{ (for all } i \text{ if possible)}
\end{align*}
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.
  
  \[ \text{Quantify this change as } \Delta \Phi_i : \text{let it "kill" } c_i : \hat{c}_i = c_i + \Delta \Phi_i \]

  \[\downarrow \quad \text{Invent your } \Phi \text{ accordingly} \quad \hat{c}_i \text{ } \text{obtain low } \hat{c}_i \]

  \[\downarrow \quad \text{Make sure } \Delta \Phi_i \text{ doesn't add much to } c_i \text{, when } c_i \text{ is cheap.} \]

\[\begin{cases} \text{costly/complicated} \\ \text{cheap/simple} \end{cases} \quad \text{\{need large negative } \Delta \Phi_i \text{\}} \quad \text{\{target } \hat{c}_i \text{ (for all } i \text{ if possible)\}} \quad \text{\{ok to let } \Delta \Phi_i > 0 \text{, but try to limit this\}} \]
Back to dynamic tables:

\[ \hat{c}_i = c_i + \Delta \Phi_i \]

- (Subjectively) define what a complicated/costly operation \((c_i)\) is.
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: \(c_i = \#\text{items}\).
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.
Back to dynamic tables:

\[ \hat{c}_i = c_i + \Delta \Phi_i \]

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

- Find something that changes a lot in the data structure in such cases.
  
  \(\downarrow\) size of array
Back to dynamic tables:

\[
\hat{c}_i = c_i + \Delta \Phi_i
\]

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

- Find something that changes a lot in the data structure in such cases.
  \(\downarrow\) size of array \(\rightarrow\) try \(\Phi = -\text{size?} \rightarrow \Delta \Phi_i \sim -\#\text{items}\)
Back to dynamic tables:

\[ \hat{c}_i = c_i + \Delta \phi_i \]

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

- Find something that changes a lot in the data structure in such cases.
  \( \leftarrow \) size of array \( \rightarrow \) try \( \phi = -\text{size} \)?
  \( \rightarrow \Delta \phi_i \sim -\# \text{items} \)
  - "kills" costly \( c_i \)
  - doesn't hurt cheap \( c_i \)
Back to dynamic tables:

\[ \hat{c}_i = c_i + \Delta \Phi_i \]

• (Subjectively) define what a complicated/costly operation \( c_i \) is.
  \[\rightarrow\text{ whenever we insert on a full array: } c_i = \#\text{items}\]

• Find something that changes a lot in the data structure in such cases.
  \[\rightarrow\text{ size of array } \rightarrow \text{ try } \Phi = -\text{size?} \rightarrow \Delta \Phi_i \sim -\#\text{items} \]
  • "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.
  \( \Downarrow \) whenever we insert on a full array: \( c_i = \# \text{items} \)

- Find something that changes a lot in the data structure in such cases.
  \( \Downarrow \) size of array \rightarrow \text{try} \Phi = -\text{size?} \rightarrow \Delta \Phi_i \sim -\# \text{items} \:
  \begin{align*}
    & \text{"kills" costly } c_i \\
    & \text{doesn't hurt cheap } c_i \\
  \end{align*}

but

\[ \Phi_i \leq 0 \]
\( \hat{c}_i = c_i + \Delta \Phi_i \Rightarrow \sum \hat{c}_i = \sum_i (c_i + \Delta \Phi_i) \) \quad \text{telescoping series}

\[ = \Phi_n - \Phi_0 + \sum c_i \Rightarrow \sum c_i \]

dynamic tables: \( \Phi_i = 2 \cdot (\# \text{items in table}) - (\text{size of table}) \)
\[ \hat{c}_i = c_i + \Delta \Phi_i \implies \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta \Phi_i) \text{ telescopong series} \]

\[ = \Phi_n - \Phi_0 + \sum_{i} c_i \gg \sum_{i} c_i \]

Dynamic tables: \[ \Phi_i = 2 \cdot (\text{# items in table}) - (\text{size of table}) \]

\( \Phi_0 = 0 \)

(and doesn't change rapidly) \( \Rightarrow \) \text{always} \( \gg \) \text{half table}

\[ \bigg\{ \Phi_i > 0 \bigg\} \]
\[ \hat{c}_i = c_i + \Delta \Phi_i \Rightarrow \sum \hat{c}_i = \sum_{i=1}^{\hat{n}} (c_i + \Delta \Phi_i) \text{ telescoping series} \]
\[ = \Phi_n - \Phi_0 + \sum c_i \geq \gamma \sum c_i \]

**Dynamic tables:**  
\[ \Phi_i = 2 \cdot (\text{# items in table}) - (\text{size of table}) \]
\[ \Rightarrow \Phi_0 = 0 \]
\[ \Phi_i > 0 \]
(\text{and doesn't change rapidly})\[\overbrace{\text{always } > \frac{1}{2} \text{ table}}\]

- **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)
\( \hat{c}_i = c_i + \Delta \Phi_i \implies \sum \hat{c}_i = \sum_{i=1}^{\hat{n}} (c_i + \Delta \Phi_i) \) \text{ telescoping series}

\[
= \Phi_n - \Phi_0 + \sum_{i}^\hat{n} c_i \geq \sum_{i}^\hat{n} c_i
\]

dynamic tables: \( \Phi_i = 2 \cdot (\# \text{items in table}) - (\text{size of table}) \)
(\text{and doesn't change rapidly}) \( \Phi_0 = 0 \)
\( \Phi_i > 0 \)

(always \( > \frac{1}{2} \) table)

\underline{type 1:} \( c_i = 1 \) \text{ (when element i doesn't trigger a doubling)}

\( \hat{c}_i = ? \quad ? \quad ? \quad ? \)

\[ c_i + \Phi_i - \Phi_{i-1} \]
\[ \hat{c}_i = c_i + \Delta \Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=i'}^{n} (c_i + \Delta \Phi_i) \quad \text{telescoping series} \]
\[ = \Phi_n - \Phi_0 + \sum c_i \quad \Rightarrow \quad \sum c_i \]

**Dynamic Table:**

\[ \Phi_i = 2 \cdot (\# \text{items in table}) - (\text{size of table}) \Rightarrow \Phi_0 = 0 \]
\[ \Phi_i > 0 \quad \text{(and doesn't change rapidly)} \quad \text{always } \geq \frac{1}{2} \text{table} \]

**Type 1:**

\[ c_i = 1 \quad \text{(when element i doesn't trigger a doubling)} \]

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

\[ c_i + \Phi_i = \Phi_{i-1} \]
\[ \hat{c}_i = c_i + \Delta \Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i \in i} (c_i + \Delta \Phi_i) \quad \text{telescoping series} \]
\[ = \Phi_n - \Phi_0 + \sum c_i \quad \Rightarrow \quad \sum c_i \]

**Dynamic Tables:**
\[ \Phi_i = 2 \cdot (\text{# items in table}) - (\text{size of table}) \quad \exists \Phi_0 = 0 \]
\[ \Phi_i > 0 \quad \text{(and doesn't change rapidly)} \quad \text{always} \gg \frac{1}{2} \text{ table} \]

**Type 1:**
\[ c_i = 1 \quad \text{(when element i doesn't trigger a doubling)} \]
\[ \hat{c}_i = 1 + [2i - \text{Size}_i] - [2(i-1) - \text{Size}_{i-1}] \]
\[ \hat{c}_i = c_i + \Delta \phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{\hat{\nu}} (c_i + \Delta \phi_i) \quad \text{telescoping series} \]

\[ = \phi_n - \phi_0 + \sum_{i} \hat{c}_i \quad \Rightarrow \quad \sum \hat{c}_i \]

**Dynamic Tables:**
\[ \phi_i = 2 \cdot (\text{# items in table}) - (\text{size of table}) \]
\[ \phi_0 = 0 \]
\[ \phi_i \geq 0 \]

(and doesn't change rapidly) \[ \overset{\text{always} \geq \frac{1}{2} \text{table}}{\longleftrightarrow} \]

**Type 1:**
\[ c_i = 1 \quad \text{(when element i doesn't trigger a doubling)} \]
\[ \hat{c}_i = 1 + [2i - \text{Size}_i] - [2(i-1) - \text{Size}_{i-1}] = 3 \]
\[ \hat{c}_i = c_i + \Delta \phi_i \Rightarrow \sum \hat{c}_i = \sum_{i \in \mathbb{Z}} (c_i + \Delta \phi_i) \quad \text{telescoping series} \]
\[ = \phi_n - \phi_0 + \sum_i \hat{c}_i \Rightarrow \sum_i \hat{c}_i \]

**Dynamic Tables:**
\[ \phi_i = 2 \cdot (\# \text{items in table}) - (\text{size of table}) \]
\[ \begin{array}{c}
\phi_0 = 0 \\
\phi_i > 0 \\
\text{(and doesn't change rapidly)} \quad \left\{ \begin{array}{c}
\text{always} > \frac{1}{2} \text{table}
\end{array} \right.
\end{array} \]

**Type 1:** \( c_i = 1 \) (when element \( i \) doesn't trigger a doubling)
\[ \hat{c}_i = 1 + [2i - \text{Size}_i] - [2(i-1) - \text{Size}_{i-1}] = 3 \]

**Type 2:** \( c_i = i \) (when element \( i \) does trigger a doubling)
\[ \hat{c}_i = ? \]
\[ \hat{c}_i = c_i + \Delta \Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta \Phi_i) \quad \text{telescoping series} \]
\[ = \Phi_n - \Phi_0 + \sum_{i} c_i \quad \Rightarrow \quad \sum c_i \]

**Dynamic Tables:**

\[ \Phi_i = 2 \cdot (\# \text{items in table}) - (\text{size of table}) \]

\[ \Phi_0 = 0 \quad \bigg\{ \begin{array}{c}
\Phi_i > 0 \\
(\text{and doesn't change rapidly}) \leftarrow \text{always} \gg \frac{1}{2} \text{table}
\end{array} \]

**Type 1:** \( c_i = 1 \) (when element \( i \) doesn't trigger a doubling)

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

**Type 2:** \( c_i = i \) (when element \( i \) does trigger a doubling)

\[ \hat{c}_i = c_i + \Phi_i - \Phi_{i-1} \]
\[ \hat{c}_i = c_i + \Delta \Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i \in i} (c_i + \Delta \Phi_i) \quad \text{telescoping series} \]
\[ = \Phi_n - \Phi_0 + \sum_i c_i \quad \gg \sum_i c_i \]

**Dynamic tables:**
\[ \Phi_i = 2 \cdot (\# \text{items in table}) - (\text{size of table}) \]
\[ \begin{cases} \Phi_0 = 0 \\ \Phi_i > 0 \end{cases} \]
(and doesn't change rapidly) \[ \Rightarrow \text{always } \geq \frac{1}{2} \text{ table} \]

**Type 1:** \( c_i = 1 \) (when element i doesn't trigger a doubling)
\[ \hat{c}_i = 1 + [2i - \text{Size}_i] - [2(i-1) - \text{Size}_{i-1}] = 3 \]

**Type 2:** \( c_i = i \) (when element i does trigger a doubling)
\[ \hat{c}_i = i + [2i - \text{Size}_i] - [2(i-1) - \text{Size}_{i-1}] \]
\[ \hat{c}_i = c_i + \Delta \Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i \neq i'} (c_i + \Delta \Phi_i) \quad \text{telescoping series} \]

\[ = \Phi_n - \Phi_0 + \sum_{i} \hat{c}_i \quad \Rightarrow \sum_{i} \hat{c}_i \]

**Dynamic Tables:**

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

\( \Phi_0 = 0 \)

\( \Phi_i > 0 \)

(and doesn't change rapidly) \[ \text{always } \gg \frac{1}{2} \text{ table} \]

---

**Type 1:** \( c_i = 1 \) (when element \( i \) doesn't trigger a doubling)

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

**Type 2:** \( c_i = i \) (when element \( i \) does trigger a doubling)

\[ \hat{c}_i = i + \left[ 2i - \text{Size}_i \right] - \left[ 2(i-1) - \text{Size}_{i-1} \right] \]
\[ \hat{c}_i = c_i + \Delta \Phi_i \] \Rightarrow \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta \Phi_i) \quad \text{telescoping series} \\
= \Phi_n - \Phi_0 + \sum_{i} \hat{c}_i \Rightarrow \sum_{i} \hat{c}_i \\
\text{dynamic tables: } \Phi_i = 2 \cdot (\# \text{items in table}) - (\text{size of table}) \quad \Phi_0 = 0 \\
(\text{and doesn't change rapidly}) \quad \text{always } \frac{1}{2} \text{ table} \\
\text{type 1: } c_i = 1 \quad (\text{when element i doesn't trigger a doubling}) \quad \hat{c}_i = 1 + [2i - \text{Size}_i] - [2(i-1) - \text{Size}_{i-1}] = 3 \\
\text{type 2: } c_i = i \quad (\text{when element i does trigger a doubling}) \quad \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)]
\[ \hat{c}_i = c_i + \Delta \Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i \in T} (c_i + \Delta \Phi_i) \quad \text{telescoping series} \]

\[ = \Phi_n - \Phi_0 + \sum_i \hat{c}_i \quad \Rightarrow \quad \sum_i \hat{c}_i \]

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

\[ \Phi_0 = 0 \]

(and doesn’t change rapidly) \( \leftarrow \) always > \( \frac{1}{2} \) table

**type 1**: \( c_i = 1 \)  (when element \( i \) doesn’t trigger a doubling)

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

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

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

\[ = i + [2i - 2(i-1)] - [2(i-1) - (i-1)] = 3i - 3(i-1) \]
\[ \hat{c}_i = c_i + \Delta \Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum \left( c_i + \Delta \Phi_i \right) \text{ telescoping series} \]
\[ = \Phi_0 - \Phi_0 + \sum c_i \quad \Rightarrow \quad \sum c_i \]

**Dynamic Tables:**  
\[ \Phi_i = 2 \cdot (\# \text{items in table}) - (\text{size of table}) \]
\[ \left\{ \begin{array}{c} \Phi_0 = 0 \\ \Phi_0 > 0 \end{array} \right. \]
(and doesn't change rapidly) \[ \left\{ \begin{array}{c} \Phi_0 = 0 \\ \Phi_0 > 0 \end{array} \right. \]
(always \[ \frac{1}{2} \text{table} \])

**Type 1:** \[ c_i = 1 \] (when element \( i \) doesn't trigger a doubling)
\[ \hat{c}_i = 1 + \left[ 2i - \text{Size}_i \right] - \left[ 2(i-1) - \text{Size}_{i-1} \right] = 3 \]

**Type 2:** \[ c_i = i \] (when element \( i \) does trigger a doubling)
\[ \hat{c}_i = i + \left[ 2i - \text{Size}_i \right] - \left[ 2(i-1) - \text{Size}_{i-1} \right] \]
\[ = i + \left[ 2i - 2(i-1) \right] - \left[ 2(i-1) - (i-1) \right] = 3i - 3(i-1) = 3 \]


\[ \hat{c}_i = c_i + \Delta \Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=i'}^{n} (c_i + \Delta \Phi_i) \quad \text{telescoping series} \]

\[ = \Phi_n - \Phi_0 + \sum_{i} c_i \quad \Rightarrow \quad \sum c_i \]

**Dynamic tables:**

\[ \Phi_i = 2 \cdot (\# \text{items in table}) - \text{(size of table)} \]

\( \Phi_0 = 0 \)

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

---

**Type 1:** \( c_i = 1 \) (when element i doesn't trigger a doubling)

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

**Type 2:** \( c_i = i \) (when element i does trigger a doubling)

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

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

\[ = 3i - 3(i-1) = 3 \]

\[ \sum c_i = 3n \]

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. worst type
or
analyze further
Summary of potential method

Amortized cost is calculated for each operation.

\[ \Phi \text{ or at least each "type" of operation} \]

\[ \Rightarrow \text{Conclude n. worst type or analyze further} \]

Requires an inspired choice of function $\Phi$

\[ \Rightarrow \text{typically something that changes a lot when you have those rare expensive operations.} \]
Summary of potential method

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

Conclude in 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"