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 # of 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.

_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 : 0
insert : 1
insert : 0 \rightarrow 0 1
insert : 0 1 \rightarrow 0 1 2
insert : 0 1 2 \rightarrow 0 1 2 3
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

\[
\begin{array}{c}
\text{start:} & \quad \begin{array}{c} \square \end{array} \\
\text{insert:} & \quad \begin{array}{c} \bullet \end{array} \\
\text{insert:} & \quad \begin{array}{c} \square \rightarrow \square \bullet \end{array} \\
\text{insert:} & \quad \begin{array}{c} \bullet \square \rightarrow \bullet \square \bullet \end{array} \\
\text{``} & \quad \begin{array}{c} \bullet \bullet \square \rightarrow \bullet \bullet \bullet \square \end{array} \\
\end{array}
\]

\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)
Dynamic Tables & Amortized Analysis

Suppose you want to use an array:

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

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

- Start: empty
- Insert: 1
- Insert: 1 is added to the array
- Insert: 2 is added to the array
- Insert: 3 is added to the array
- Insert: 4 is added to the array
- Insert: 5 is added to the array
Suppose you want to use an array:

You must support insertion but you don’t know max # of 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: $O(n)$
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. (Make a new larger array & copy)

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

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 the maximum number of elements.

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

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

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

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

Claim: for \( n \) inserts: also \( O(n) \)
Cost $c_i \begin{cases} & i \text{ if } i-1 \text{ is a power of 2} \\ & 1 \text{ otherwise} \end{cases}$

$i :$ copy $i$ elements  
(make array: $O(1)$)

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

$i$: copy $i$ elements

(size: $O(n)$)
cost $c_i \begin{cases} i & \text{if } i-1 \text{ is a power of 2} \\ 1 & \text{otherwise} \end{cases}$

$i :$ copy $i$ elements

$(\text{make array: } O(1))$
Cost \( c_i \) \( \begin{cases} i & \text{if } i-1 \text{ is a power of 2} \\ 1 & \text{otherwise} \end{cases} \)

\( i \): copy \( i \) elements
(make array: \( O(1) \))

If making an array of size \( j \) costs \( c \cdot j = \Theta(j) \), then we would just multiply by \( (c+1) \) here.
\[
\text{cost } c_i = \begin{cases} 
  i & \text{if } i-1 \text{ is a power of } 2 \\
  1 & \text{otherwise}
\end{cases}
\]

\(i: \) copy \(i\) elements

\((\text{make array: } O(1))\)

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

\( i \) : copy \( i \) elements  
\text{(make array: } O(1))

\[
\text{cost}(n) = \sum_{i=1}^{n} c_i = n + \, ?
\]
\[ \text{cost}(n) = \sum_{i=1}^{\log(n-1)} c_i = n + \sum_{j=0}^{\log(n-1)} 2^j \]
\[
\text{cost } c_i \begin{cases} 
i & \text{if } i-1 \text{ is a power of 2} \\ 1 & \text{otherwise} \end{cases}
\]

\(i: \text{ copy } i \text{ elements} \) 

(make array: 0(1))

\[
\begin{align*}
\text{CLRS says: } & \leq n + \sum_{j=0}^{\lfloor \log_2 n \rfloor} 2^j \\
\end{align*}
\]

\[
\text{cost}(n) = \sum_{i=1}^{n} c_i = n + \sum_{j=0}^{\lfloor \log_2(n-1) \rfloor} 2^j
\]
\[
\text{cost } c_i \begin{cases} 
i & \text{if } i-1 \text{ is a power of 2} \\ 1 & \text{otherwise} \end{cases}
\]

\(i:\) copy \(i\) elements (make array: \(O(1)\))

\[
\begin{align*}
\text{CHRS} & \text{ says: } \leq n + \sum_{j=0}^{\lceil \log_2(n-1) \rceil} 2^j \\
\text{size}_i & = 1 \quad 2 \quad 4 \quad 4 \quad 8 \quad 8 \quad 8 \quad 8 \quad 16 \quad 16 \\
C_i & = 1 \quad 2 \quad 3 \quad 1 \quad 5 \quad 1 \quad 1 \quad 1 \quad 9 \quad 1 \\
& \begin{cases} 
1 \quad i \quad i \quad i \quad i \quad i \quad i \quad i \quad i \quad i \quad i \\
-1 \quad 2 \quad -4 \quad - \quad - \quad - \quad 8 \quad - \\
2^0 \quad 2^1 \quad 2^2 \quad 2^3 \\
\end{cases}
\end{align*}
\]

\[
\text{cost}(n) = \sum_{i=1}^{n} c_i = n + \sum_{j=0}^{\lceil \log_2(n-1) \rceil} 2^j \leq n + ??
\]
\[ \text{cost } c_i = \begin{cases} i & \text{if } i-1 \text{ is a power of 2} \\ 1 & \text{otherwise} \end{cases} \]

\( i : \) copy \( i \) elements

(make array: \( O(1) \))

\[ (\text{CLRS says: } \leq n + \sum_{j=0}^{\log_2(n-1)} 2^j) \]

\[ \text{cost}(n) = \sum_{i=1}^{n} c_i = n + \sum_{j=0}^{\log_2(n-1)} 2^j \leq n + 2 \cdot 2^{\log n} \] (geom. series)
cost \( c_i \) = \begin{cases} i & \text{if } i-1 \text{ is a power of 2} \\ 1 & \text{otherwise} \end{cases}

i \cdot \text{copy } i \text{ elements (make array: } O(1))

\left( \text{CLRS says: } \leq n + \sum_{j=0}^{\log(n-1)} 2^j \right)

\text{cost}(n) = \sum_{i=1}^{n} c_i = n + \sum_{j=0}^{\log(n-1)} 2^j

\leq n + 2 \cdot 2^\log n \quad \text{(geom. series)}

\leq 3n \quad \text{-- -- -- --> if making an array of size } j \text{ costs } \Theta(j) \text{ then we would just get higher const.}
\[ \text{cost } c_i \begin{cases} 
  i & \text{if } i-1 \text{ is a power of 2} \\
  1 & \text{otherwise}
\end{cases} \]

\( i : \text{copy } i \text{ elements} \)

\((\text{make array: } O(1))\)

\[
\begin{align*}
\text{(CLRS says: } & \leq n + \sum_{j=0}^{\log_2(n-1)} 2^j \text{)}
\end{align*}
\]

\[
\text{cost}(n) = \sum_{i=1}^{n} c_i = n + \sum_{j=0}^{\log_2(n-1)} 2^j \leq n + 2 \cdot 2^{\log_2 n} \leq 3n
\]

\(\leq 3n \quad \Rightarrow \quad \text{if making an array of size } j \text{ costs } \Theta(j) \text{ then we would just get higher const.}\)

**aggregate analysis** : showing that even though worst case for individual action is \(f(n)\), total work = \(o(n \cdot f(n))\)
3 main ways for amortizing: aggregate, accounting, & potential method.

↓

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

Accounting: has an analogy to lending/borrowing $.
3 main ways for amortizing: aggregate, accounting, & potential method.

**Accounting**: has an analogy to lending/borrowing $.
Let simple operations cost more than they should. Ideally \( \Theta(\text{real cost}) \).
3 main ways for amortizing: aggregate, accounting, & potential method.

**Accounting**: has an analogy to lending/borrowing $.
Let simple operations cost more than they should. Ideally Θ(real cost)

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

Accounting: has an analogy to lending/borrowing $.

Let simple operations cost more than they should. Ideally $\Theta$(real cost)

\$ like borrowing $ \rightarrow$ spend the $\$ you borrowed.

"Complicated" operations (cost a lot) are paid for implicitly.
3 main ways for amortizing: aggregate, accounting, & potential method.

Accounting: has an analogy to lending/borrowing $.

Let simple operations cost more than they should. Ideally $\Theta(\text{real cost})$

\$ like borrowing $ \rightarrow$ spend the $\$ you borrowed.

"Complicated" operations (cost a lot) are paid for implicitly.

Rule: Never spend more than what you borrowed
3 main ways for amortizing: aggregate, accounting, & potential method.

Accounting: has an analogy to lending/borrowing $.

Let simple operations cost more than they should. Ideally \( \Theta(\text{real cost}) \)

\$ like borrowing $ \quad \Rightarrow \quad \text{spend the $ you borrowed.}

“Complicated” operations (cost a lot) are paid for implicitly.

Rule: Never spend more than what you borrowed

Goal: borrow as little as possible.
3 main ways for amortizing: aggregate, accounting, & potential method.

**Accounting**: has an analogy to lending/borrowing $.
Let simple operations cost more than they should. Ideally $\Theta(\text{real cost})$

$\xRightarrow{\text{like borrowing$}}$ spend the $\text{$ you borrowed.}$

“Complicated” operations (cost a lot) are paid for implicitly.

**Rule**: Never spend more than what you borrowed

**Goal**: borrow as little as possible.

Could have also said: simple $\rightarrow$ lend/deposit $\quad \Rightarrow$ try to lend as little as possible
complicated $\rightarrow$ ask for some back
3 main ways for amortizing: aggregate, accounting, & potential method.

**Accounting:** has an analogy to lending/borrowing $.

Let simple operations cost more than they should. Ideally $\Theta$(real cost)

\$ like borrowing $\rightarrow$ spend the $\$ you borrowed.

"Complicated" operations (cost a lot) are paid for implicitly.

**Rule:** Never spend more than what you borrowed

**Goal:** Borrow as little as possible.

Could have also said:

- simple $\rightarrow$ lend/deposit $\rightarrow$ try to lend as little as possible
- complicated $\rightarrow$ ask for some back

**Note:** this is only for analysis. Nothing to do with algorithm.
Amortized cost of operation $i$: $\hat{C}_i$
Amortized cost of operation $i$: $\hat{c}_i$

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

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

Back to dynamic tables:

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

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>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>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>
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

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

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>
</tbody>
</table>

\[
\begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{pmatrix}
\]

<table>
<thead>
<tr>
<th>size</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>4</th>
<th>8</th>
<th>8</th>
<th>8</th>
<th>8</th>
<th>16</th>
<th>16</th>
</tr>
</thead>
</table>

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

Back to dynamic tables:

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

implies $\sum c_i \leq 3n$ 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>9</td>
<td>1</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:

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

our banking game should always exaggerate true costs.

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></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>( \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>( \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**

1. Borrowed $\hat{C}_i = 3$, burned $\$1$.

   Paid 1 to insert, banked 1

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

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>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>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>

BURN $\$1$. Just this once.
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**

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

Added 2nd 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>Bank</th>
<th>i</th>
<th>$c_i$</th>
<th>$\hat{c}_i$</th>
<th>$\text{size}_i$</th>
<th>$\text{bank}_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>3</td>
<td>16</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>3</td>
<td>16</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>3</td>
<td>16</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

Spent $\$2$ from bank to double \rightarrow copy 2 items.

Item #3 inserted: 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>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>(\hat{C}_i)</td>
<td>(\bullet)</td>
<td></td>
<td></td>
<td>(\bullet)</td>
<td></td>
<td>(\bullet)</td>
<td>(\bullet)</td>
<td>(\bullet)</td>
<td>(\bullet)</td>
<td>(\bullet)</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>16</td>
<td>16</td>
</tr>
<tr>
<td>(\text{bank}_i)</td>
<td>1</td>
<td>2</td>
<td>2</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 \( \sum c_i \leq 3n \) for eventual doubling.

When table doubles, use 1 to copy each item.

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>$i$</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>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>size</td>
<td>1</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>

Amortized cost of operation $i$: $\hat{C}_i$ implies $\sum C_i \leq 3n$.

Back to dynamic tables:

When table doubles, use 1 to copy each item.

Spent 4 to copy 4.

Banked 2.

Bank: 1 2 2 2 2

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

Our banking game should always exaggerate true costs.

$\sum\hat{C}_i \leq \sum C_i$ for all $n$. 

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

\[
\begin{array}{cccccccccccc}
\text{i} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\hline
C_i & 1 & 2 & 3 & 1 & 5 & 1 & 1 & 1 & 1 & 1 \\
size_i & 1 & 2 & 4 & 4 & 8 & 8 & 8 & 8 & 16 & 16 \\
\hat{C}_i & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\
\text{bank}_i & 1 & 2 & 2 & 4 & 2 & 4
\end{array}
\]

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 \( \sum c_i \leq 3n \) 2 for eventual doubling

When table doubles, use 1 to copy each item.

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>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>
<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>2</td>
<td>4</td>
<td>2</td>
<td>4</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>16</td>
<td>16</td>
</tr>
</tbody>
</table>

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

BANK

\[
\begin{array}{cccccccccc}
\text{i} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
C_i & 1 & 2 & 3 & 1 & 5 & 1 & 1 & 1 & 9 & 1 \\
size_i & 1 & 2 & 4 & 4 & 8 & 8 & 8 & 8 & 16 & 16 \\
\hat{C}_i & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\
\text{bank}_i & 1 & 2 & 2 & 4 & 2 & 4 & 6 & 8 & & \\
\end{array}
\]

$\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{C}_i$ 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 \( \sum c_i \leq 3n \)

2 for eventual doubling

When table doubles, use 1 to copy each item.

**Bank**

last doubling was 4\(\rightarrow\)8

insert item 9

4 \[
\begin{array}{ccccccccc}
1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
\end{array}
\]

\[\text{cost} = 8 \text{ to copy}\]

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

our banking game should always exaggerate true costs.

\[
\begin{array}{cccccccccccc}
i & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
C_i & 1 & 2 & 3 & 1 & 5 & 1 & 1 & 1 & 9 & 1 \\
\hat{C}_i & & & & & & & & & & \\
\text{size}_i & 1 & 2 & 4 & 4 & 8 & 8 & 8 & 8 & 16 & 16 \\
\text{bank}_i & 1 & 2 & 2 & 2 & 4 & 2 & 4 & 6 & 8 & \end{array}
\]
Amortized cost of operation $i$: $\hat{C}_i$

Back to dynamic tables:

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

When table doubles, use 1 to copy each item.

**BANK**

last doubling was $4 \rightarrow 8$

insert item 9

$4 \rightarrow 4 \Rightarrow cost = 8$ to copy

back to initial conditions

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

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

Back to dynamic tables:

Let \( \hat{C}_i = 3 \longrightarrow 1 \) to cover insert cost.

Implies \( \sum C_i \leq 3n \) for eventual doubling.

When table doubles, use 1 to copy each item.

**Bank**

last doubling was 4 \( \rightarrow \) 8

insert item 9

\[ \begin{array}{c|c|c|c|c|c|c|c|c|c|c} i & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \hline C_i & 1 & 2 & 3 & 1 & 5 & 1 & 1 & 1 & 9 & 1 \\ \hline \end{array} \]

\[ \begin{array}{c|c|c|c|c|c|c|c|c|c|c} \text{size} & 1 & 2 & 4 & 4 & 8 & 8 & 8 & 8 & 16 & 16 \\ \hline \hat{C}_i & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ \hline \text{bank} & 1 & 2 & 2 & 4 & 2 & 4 & 6 & 8 & 2 & 4 \\ \end{array} \]

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

our banking game should always exaggerate true costs.
Summary of accounting method

Estimate a cost: \( \hat{c} \) ... higher than what you think average real cost \( \frac{1}{n} \Sigma c_i \) 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 $c_i$;

$\rightarrow$ 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} cost : $c_i$
Potential Method

Start with data structure $D_0$

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

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

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

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

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

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

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

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

$$= c_i + \Delta \Phi_i$$

If $\Delta \Phi_i > 0$, $\hat{c}_i > c_i : \text{storing potential}$

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

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

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

$= c_i + \Delta \Phi_i$

\[ \begin{cases} \text{If } \Delta \Phi_i > 0, \hat{c}_i > c_i : \text{storing potential} \\ \Delta \Phi_i < 0, \hat{c}_i < c_i : \text{release work.} \end{cases} \]
\( \hat{c}_i = c_i + \Delta \Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i \in \mathbb{Z}} (c_i + \Delta \Phi_i) \)
\[ \hat{c}_i = c_i + \Delta \Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{\hat{n}} (c_i + \Delta \Phi_i) \quad \text{telescoping series} \]

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

\[ = \phi_n - \phi_0 + \sum_{i}^{\tilde{n}} c_i \quad \Downarrow \quad \Downarrow \quad \Downarrow \quad \Downarrow \quad \Rightarrow \quad \sum_{i}^{\tilde{n}} c_i \]
\( \hat{c}_i = c_i + \Delta \Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{\tilde{n}} (c_i + \Delta \Phi_i) \quad \text{telescoping series} \)

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

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

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

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

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

Ideally this will give a good bound for total cost \((n \cdot \max \hat{c}_i)\)
\[ \hat{c}_i = c_i + \Delta \phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i \in i} (c_i + \Delta \phi_i) \quad \text{telescoping series} \]

\[ = \phi_n - \phi_0 + \sum c_i \quad \Rightarrow \quad \sum c_i \]

Back to dynamic tables:
\[ \phi_i = 2 \cdot (\# \text{items in table}) - (\text{size of table}) \]
\[ \hat{c}_i = c_i + \Delta \Phi_i \quad \Rightarrow \quad \sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta \Phi_i) \quad \text{telescoping series} \]

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

Back to dynamic tables:

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

\[ \Phi_0 = 0 \]

\[ \forall \Phi_i \geq 0 \]

\[ \text{always } \frac{1}{2} \text{ table} \]
\[
\hat{c}_i = c_i + \Delta \Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta \Phi_i) \quad \text{telescoping series}
\]

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

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

\[
\begin{cases} \Phi_0 = 0 \quad \text{always} > \frac{1}{2} \text{table} \\ \Phi_i > 0 \end{cases}
\]

\(\hat{c}_i\) type 1: \(c_i = 1\) (when element \(i\) doesn't trigger a doubling)

\(\hat{c}_i\) type 2: \(c_i = i\) (when element \(i\) does trigger a doubling)
\( \hat{c}_i = c_i + \Delta \Phi_i \Rightarrow \sum \hat{c}_i = \sum_{i=i'}^{n} (c_i + \Delta \Phi_i) \)  

= \Phi_n - \Phi_0 + \sum_{i}^n c_i \quad \geq \sum_i c_i

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

\( \Phi_0 = 0 \)

\( \Phi_i > 0 \)

\frac{1}{2} \text{ table}

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

\hat{c}_i = \frac{?}{?} \quad \frac{?}{?} \quad \frac{?}{?}

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

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

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

\[ \begin{cases} \Phi_0 = 0 \\ \Phi_i > 0 \\ \text{always } \geq \frac{1}{2} \text{table} \end{cases} \]

\( \hat{c}_i \) type 1: \( c_i = 1 \) (when element \( i \) doesn't trigger a doubling)

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

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

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

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

\[ \exists \Phi_0 = 0 \quad \\forall \Phi_i > 0 \]

\( \hat{c}_i \) type 1: \( c_i = 1 \) (when element \( i \) doesn't trigger a doubling)

\[ \hat{c}_i = 1 + [2i - \text{Size}_i] - [2(i-1) - \text{Size}_{i-1}] \]
\[ \hat{c}_i = c_i + \Delta \Phi_i \Rightarrow \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta \Phi_i) \text{ telescoping series} \]

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

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

\[ \begin{align*}
\Phi_0 &= 0 \\
\Phi_i &> 0 \quad \text{always \( \gg \frac{1}{2} \) table}
\end{align*} \]

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

\[ \hat{c}_i = 1 + [2i - \cancel{\text{Size}i}] - [2(i-1) - \cancel{\text{Size}i-1}] = 3 \]
\[ \hat{c}_i = c_i + \Delta \Phi_i \Rightarrow \sum \hat{c}_i = \sum_{i=1}^{\hat{n}} (c_i + \Delta \Phi_i) \]  

\[ \Delta \Phi_i = \Phi_n - \Phi_o + \sum_{i=1}^{\hat{n}} c_i \Rightarrow \sum_{i=1}^{\hat{n}} c_i \]

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

\( \Phi_0 = 0 \)

\( \Phi_i > 0 \)

\[ \frac{1}{2} \text{table} \]

\[ \hat{c}_i \text{ type 1: } c_i=1 \text{ (when element } i \text{ doesn't trigger a doubling)} \]

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

\[ \hat{c}_i \text{ type 2: } c_i=i \text{ (when element } i \text{ does trigger a doubling)} \]

\[ \hat{c}_i = ? \]
\[ \hat{c}_i = c_i + \Delta \Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum \left( c_i + \Delta \Phi_i \right) = \Phi_n - \Phi_0 + \sum c_i \quad \Rightarrow \quad \sum c_i \]

Back to dynamic tables: 
\[ \Phi_i = 2 \cdot (\text{#items in table}) - (\text{size of table}) \quad \begin{cases} \Phi_0 = 0 \\ \Phi_i > 0 \end{cases} \quad \text{always} \gg \frac{1}{2} \text{table} \]

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

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

\[ = \Phi_n - \Phi_0 + \sum_i c_i \quad \Rightarrow \quad \sum_i c_i \]

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

\[ \forall \Phi_0 = 0 \]

\[ \forall \Phi_i > 0 \quad \frac{1}{2} \text{table} \]

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

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

\[ \hat{c}_i \quad \text{type 2:} \quad c_i = i \quad \text{(when element } i \text{ does trigger a doubling)} \]

\[ \hat{c}_i = i + [2i - \text{Size}_i] - [2(i-1) - \text{Size}_{i-1}] \]
\[ \hat{c}_i = c_i + \Delta \phi_i \quad \Rightarrow \quad \sum_{i \in t} \hat{c}_i = \sum_{i \in t} (c_i + \Delta \phi_i) \quad \text{telescoping series} \]

\[ = \phi_n - \phi_0 + \sum_{i} c_i \quad \Rightarrow \quad \sum_{i} c_i \]

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

\[ \begin{cases} \phi_0 = 0 \\ \phi_i > 0 \end{cases} \]

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

\hat{c}_i \quad \text{type 2: } c_i = i \quad \text{(when element } i \text{ does trigger a doubling)} \\
\hat{c}_i = i + [2i - \text{Size}_i] - [2(i-1) - \text{Size}_{i-1}] \\
\downarrow \quad ? \\
\downarrow \quad ?
\[
\hat{c}_i = c_i + \Delta \Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i \in \mathbb{N}} (c_i + \Delta \Phi_i) \quad \text{telescoping series} \\
= \Phi_n - \Phi_0 + \sum_{i} c_i \quad \Rightarrow \quad \sum_{i} c_i
\]

Back to dynamic tables: \( \Phi_i = 2 \cdot (\# \text{items in table}) - (\text{size of table}) \) \( \Phi_0 = 0 \) \( \sum_{i} \Phi_i > 0 \) always > \( \frac{1}{2} \) table

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

\[\hat{c}_i \quad \text{type 2: } c_i=i \quad (\text{when element } i \text{ does trigger a doubling}) \]
\[
\hat{c}_i = i + \left[2i - \text{Size}_i\right] - \left[2(i-1) - \text{Size}_{i-1}\right] \\
= i + \left[2i - 2(i-1)\right] - \left[2(i-1) - (i-1)\right]
\]
\[ \hat{c}_i = c_i + \Delta \Phi_i \quad \Rightarrow \quad \sum \hat{c}_i = \sum_{i=1}^{n} (c_i + \Delta \Phi_i) \quad \text{telescoping series} \]

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

Back to dynamic tables: \[
\Phi_i = 2 \cdot (\# \text{items in table}) - (\text{size of table}) \quad \Rightarrow \quad \Phi_0 = 0 \]

\[ \Phi_i > 0 \]

\[ \text{always} \quad \frac{1}{2} \cdot \text{table} \]

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

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

\[ \hat{c}_i \quad \text{type 2: } \quad c_i = i \quad (\text{when element } i \text{ does trigger a doubling}) \]

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

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

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

Back to dynamic tables: \[ \Phi_i = 2 \cdot (\# \text{items in table}) - (\text{size of table}) \]
\[ \begin{align*}
\text{always } & \frac{1}{2} \text{ table} \\
\rightarrow \quad & \Phi_i > 0
\end{align*} \]

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

\[ \hat{c}_i \text{ type 2: } c_i = i \quad (\text{when element i does trigger a doubling}) \]
\[ \hat{c}_i = i + \left[ 2i - \text{Size}_i \right] - \left[ 2(i-1) - \text{Size}_{i-1} \right] \]
\[ = i + \left[ 2i - 2(i-1) \right] - \left[ 2(i-1) - (i-1) \right] \quad = 3i - 3(i-1) = 3 \]
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. (Conclude in worst type or analyze further)

⇒ or at least each “type” of operation

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

\( n \) 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"