Design and Analysis of Algorithms

CSE 5311

Lecture 13 Amortized Analysis

Song Jiang, Ph.D.

Department of Computer Science and Engineering

How large should a hash table be?

Goal: Make the table as small as possible, but large enough so that it won't overflow (or otherwise become inefficient).

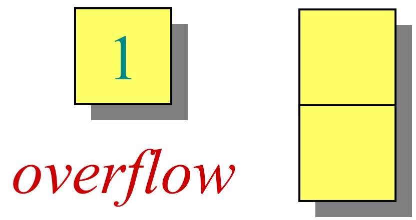
Problem: What if we don't know the proper size in advance?

Solution: ***Dynamic tables.***

IDEA: Whenever the table overflows, "grow" it by allocating (via **`malloc`** or **`new`**) a new, larger table. Move all items from the old table into the new one, and free the storage for the old table.

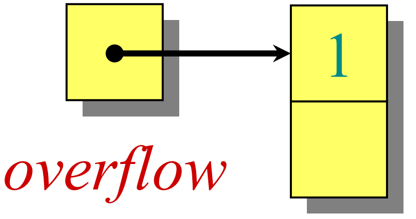
Example of a dynamic table

1. INSERT
2. INSERT



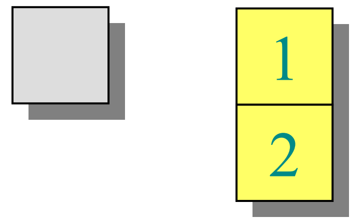
Example of a dynamic table

1. INSERT
2. INSERT



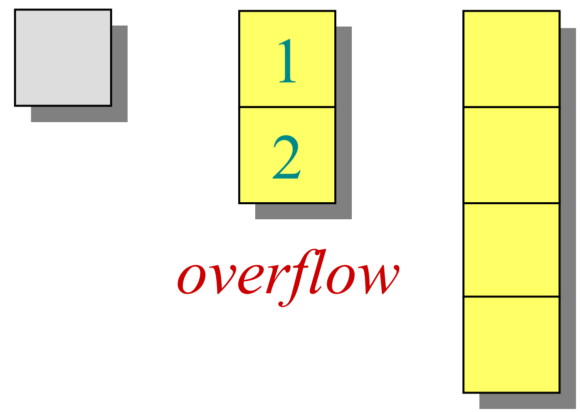
Example of a dynamic table

1. INSERT
2. INSERT



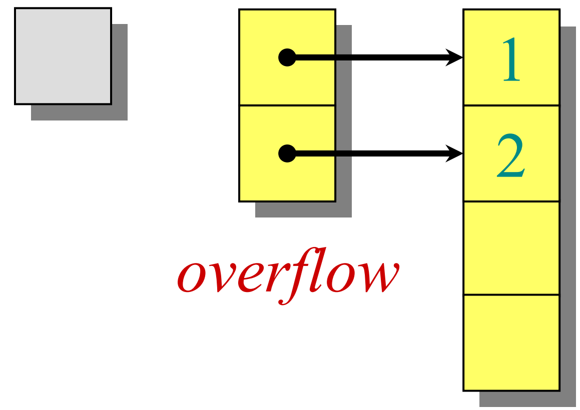
Example of a dynamic table

1. INSERT
2. INSERT
3. INSERT



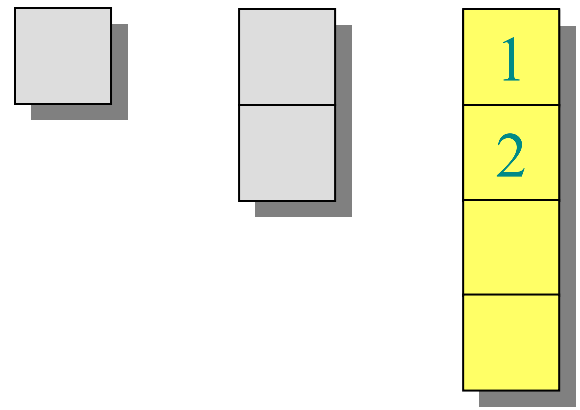
Example of a dynamic table

1. INSERT
2. INSERT
3. INSERT



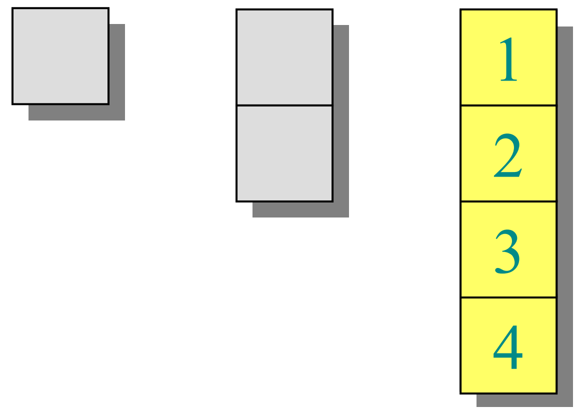
Example of a dynamic table

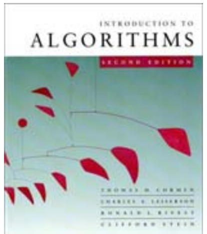
1. INSERT
2. INSERT
3. INSERT



Example of a dynamic table

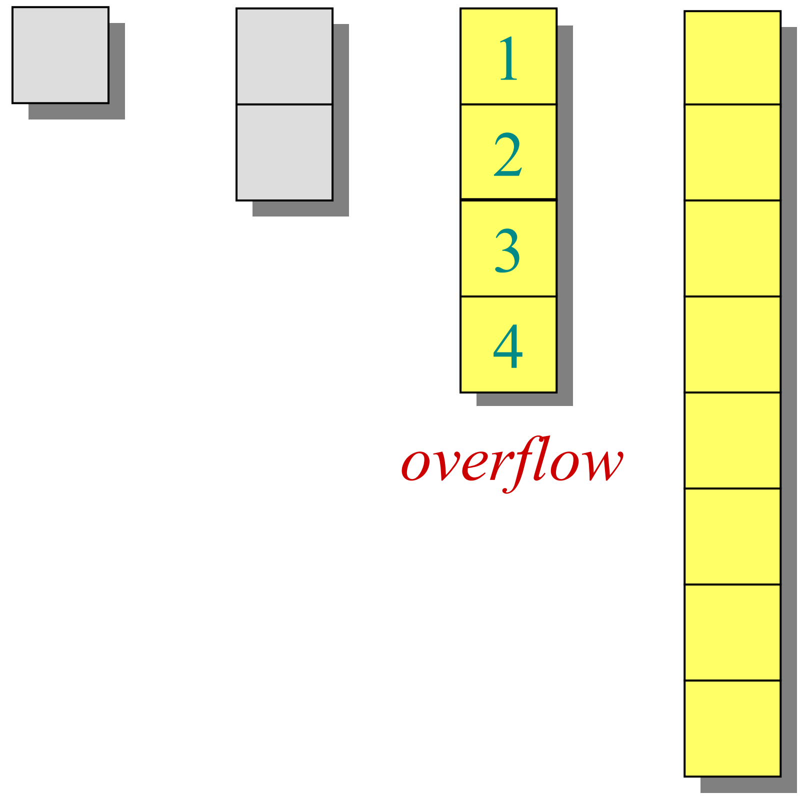
1. INSERT
2. INSERT
3. INSERT
4. INSERT





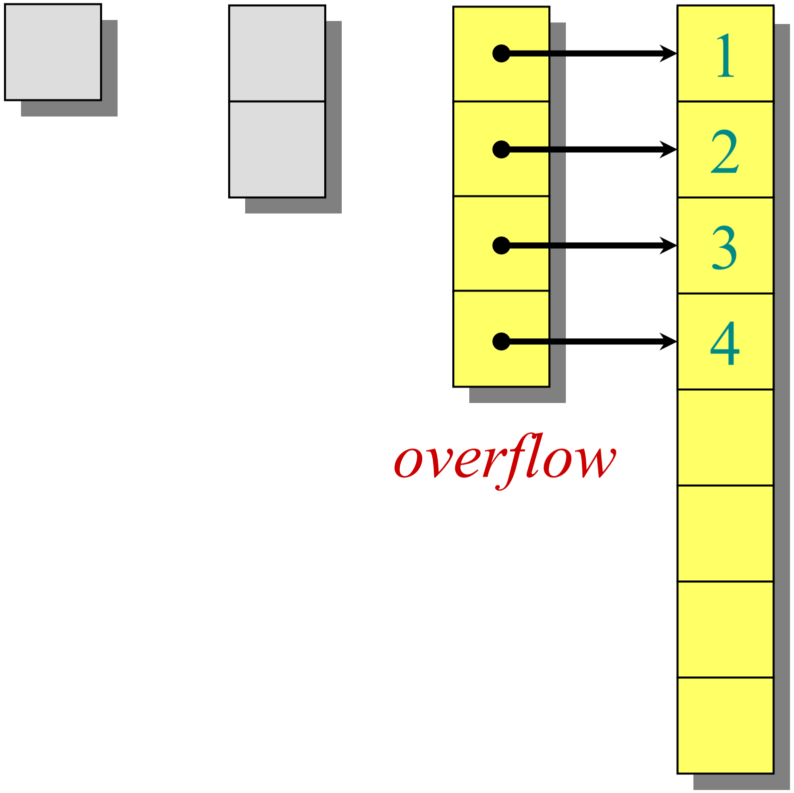
Example of a dynamic table

1. INSERT
2. INSERT
3. INSERT
4. INSERT
5. INSERT



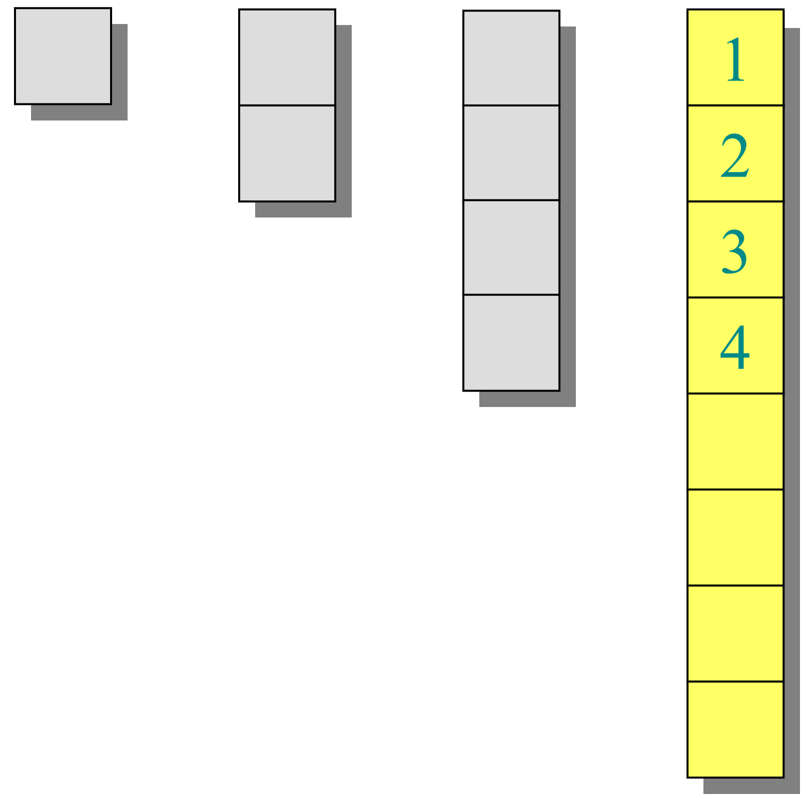
Example of a dynamic table

1. INSERT
2. INSERT
3. INSERT
4. INSERT
5. INSERT



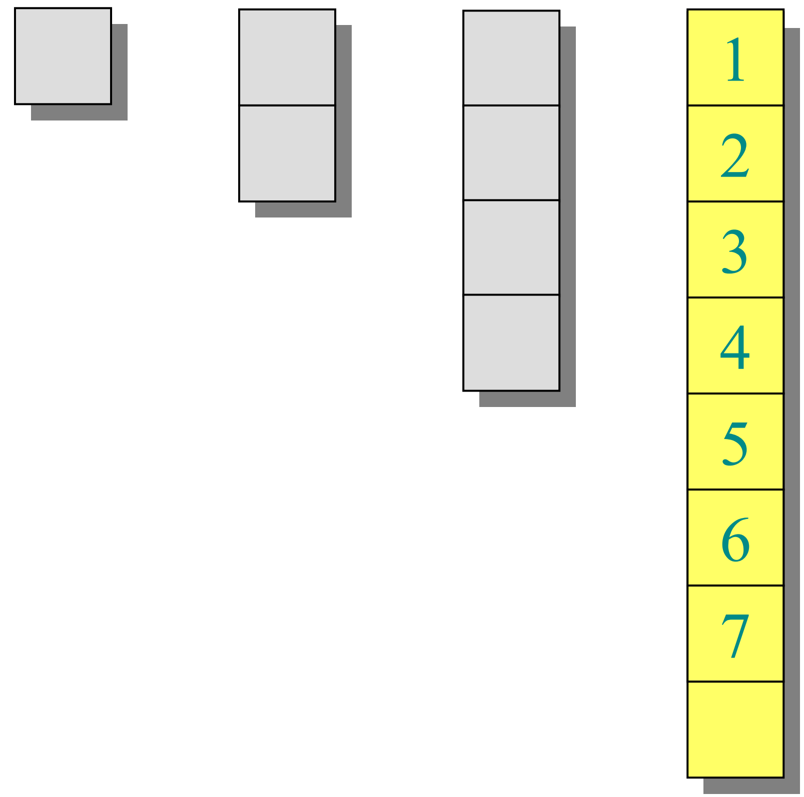
Example of a dynamic table

1. INSERT
2. INSERT
3. INSERT
4. INSERT
5. INSERT



Example of a dynamic table

1. INSERT
2. INSERT
3. INSERT
4. INSERT
5. INSERT
6. INSERT
7. INSERT



Worst-case analysis

Consider a sequence of n insertions. The worst-case time to execute one insertion is $\Theta(n)$. Therefore, the worst-case time for n insertions is $n \cdot \Theta(n) = \Theta(n^2)$.

WRONG! In fact, the worst-case cost for n insertions is only $\Theta(n) \ll \Theta(n^2)$.

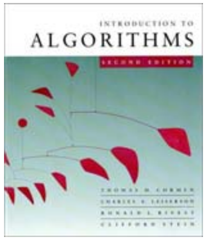
Let's see why.

Tighter analysis

Let $c_i =$ the cost of the ith insertion

$$= \begin{cases} i & \text{if } i-1 \text{ is an exact power of } 2, \\ 1 & \text{otherwise.} \end{cases}$$

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



Tighter analysis

Let c_i = the cost of the i th insertion

$$= \begin{cases} i & \text{if } i-1 \text{ is an exact power of } 2, \\ 1 & \text{otherwise.} \end{cases}$$

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

Tighter analysis (continued)

$$\text{Cost of } n \text{ insertions} = \sum_{i=1}^{n} c_i$$

$$\leq n + \sum_{j=0}^{\lfloor \lg(n-1) \rfloor} 2^j$$

$$\leq 3n$$

$$= \Theta(n).$$

Thus, the average cost of each dynamic-table operation is $\Theta(n)/n = \Theta(1)$.

Amortized analysis

An ***amortized analysis*** is any strategy for analyzing a sequence of operations to show that the average cost per operation is small, even though a single operation within the sequence might be expensive.

Even though we're taking averages, however, probability is not involved!

- An amortized analysis guarantees the average performance of each operation in the *worst case*.

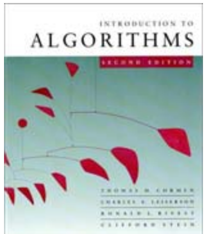
Types of amortized analyses

Three common amortization arguments:

- the ***aggregate*** method,
- the ***accounting*** method,
- the ***potential*** method.

We've just seen an aggregate analysis.

The aggregate method, though simple, lacks the precision of the other two methods. In particular, the accounting and potential methods allow a specific ***amortized cost*** to be allocated to each operation.

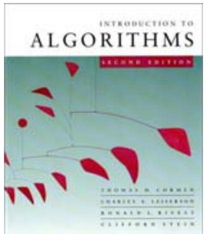


Accounting method

- Charge i th operation a fictitious ***amortized cost*** $\hat{c}_i$, where \$1 pays for 1 unit of work (*i.e.*, time).
- This fee is consumed to perform the operation.
- Any amount not immediately consumed is stored in the ***bank*** for use by subsequent operations.
- The bank balance must not go negative! We must ensure that

$$\sum_{i=1}^{n} c_i \le \sum_{i=1}^{n} \hat{c}_i$$

  for all n.
- Thus, the total amortized costs provide an upper bound on the total true costs.



Accounting analysis of dynamic tables

Charge an amortized cost of $\hat{c}_i = \$3$ for the ith insertion.

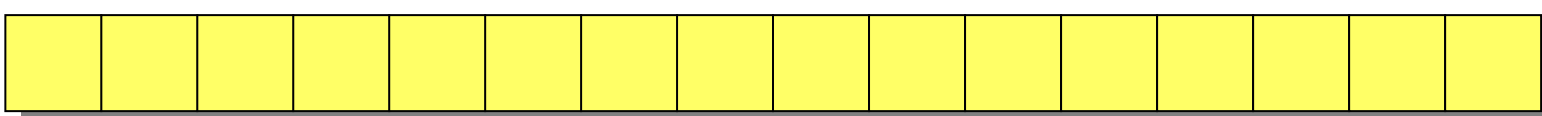
- $1 pays for the immediate insertion.
- $2 is stored for later table doubling.

When the table doubles, $1 pays to move a recent item, and $1 pays to move an old item.

Example:

$0	$0	$0	$0	$2	$2	$2	$2

overflow



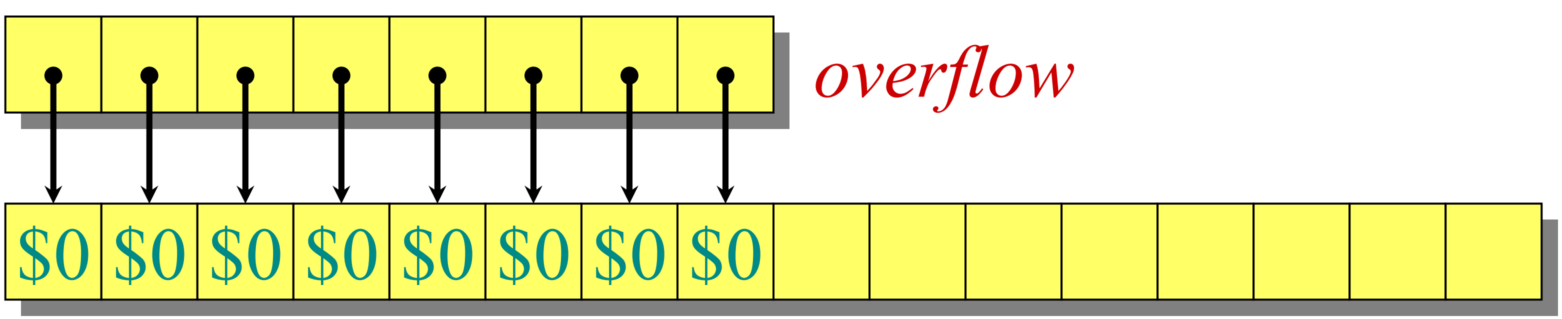
Accounting analysis of dynamic tables

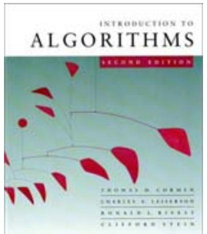
Charge an amortized cost of $\hat{c}_i = \$3$ for the i th insertion.

- \$1 pays for the immediate insertion.
- \$2 is stored for later table doubling.

When the table doubles, \$1 pays to move a recent item, and \$1 pays to move an old item.

Example:





Accounting analysis of dynamic tables

Charge an amortized cost of $\hat{c}_i = \$3$ for the i th insertion.

- $1 pays for the immediate insertion.
- $2 is stored for later table doubling.

When the table doubles, $1 pays to move a recent item, and $1 pays to move an old item.

Example:

$0	$0	$0	$0	$0	$0	$0	$0	$2	$2	$2					

Accounting analysis (continued)

Key invariant: Bank balance never drops below 0. Thus, the sum of the amortized costs provides an upper bound on the sum of the true costs.

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

*Okay, so I lied. The first operation costs only $2, not $3.

Potential method

IDEA: View the bank account as the potential energy (*à la* physics) of the dynamic set.

Framework:

- Start with an initial data structure D_0.
- Operation i transforms D_{i-1} to D_i.
- The cost of operation i is c_i.
- Define a ***potential function*** $\Phi : \{D_i\} \to \mathbb{R}$, such that $\Phi(D_0) = 0$ and $\Phi(D_i) \geq 0$ for all i.
- The ***amortized cost*** $\hat{c}_i$ with respect to Φ is defined to be $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$.

Understanding potentials

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

- If $\Delta\Phi_i > 0$, then $\hat{c}_i > c_i$. Operation i stores work in the data structure for later use.
- If $\Delta\Phi_i < 0$, then $\hat{c}_i < c_i$. The data structure delivers up stored work to help pay for operation i.

The amortized costs bound the true costs

The total amortized cost of n operations is

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

Summing both sides.

The amortized costs bound the true costs

The total amortized cost of n operations is

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

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

The series telescopes.

The amortized costs bound the true costs

The total amortized cost of n operations is

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

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

$$\ge \sum_{i=1}^{n} c_i$$

since $\Phi(D_n) \ge 0$ and $\Phi(D_0) = 0$.

Potential analysis of table doubling

Define the potential of the table after the ith insertion by $\Phi(D_i) = 2i - 2^{\lceil \lg i \rceil}$. (Assume that $2^{\lceil \lg 0 \rceil} = 0$.)

Note:

- $\Phi(D_0) = 0$,
- $\Phi(D_i) \geq 0$ for all i.

Example:

•	•	•	•	•	•		

$\Phi = 2 \cdot 6 - 2^3 = 4$

(

\$0	\$0	\$0	\$0	\$2	\$2		

accounting method)

Calculation of amortized costs

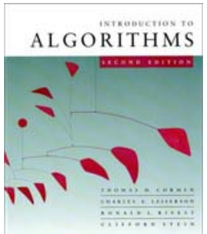
The amortized cost of the i th insertion is

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

Calculation of amortized costs

The amortized cost of the i th insertion is

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\
&= \left\{ \begin{array}{l} i \text{ if } i-1 \text{ is an exact power of } 2, \\ 1 \text{ otherwise;} \end{array} \right\} \\
&\quad + \left(2i - 2^{\lceil \lg i \rceil}\right) - \left(2(i-1) - 2^{\lceil \lg (i-1) \rceil}\right)
\end{aligned}$$



Calculation of amortized costs

The amortized cost of the i th insertion is

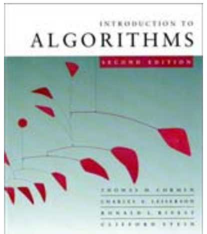
$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

$$= \begin{cases} i & \text{if } i-1 \text{ is an exact power of } 2, \\ 1 & \text{otherwise;} \end{cases}$$

$$+ \left(2i - 2^{\lceil \lg i \rceil}\right) - \left(2(i-1) - 2^{\lceil \lg (i-1) \rceil}\right)$$

$$= \begin{cases} i & \text{if } i-1 \text{ is an exact power of } 2, \\ 1 & \text{otherwise;} \end{cases}$$

$$+ 2 - 2^{\lceil \lg i \rceil} + 2^{\lceil \lg (i-1) \rceil} .$$



Calculation

Case 1: $i - 1$ is an exact power of 2.

$$\hat{c}_i = i + 2 - 2^{\lceil \lg i \rceil} + 2^{\lceil \lg (i-1) \rceil}$$

Calculation

Case 1: $i - 1$ is an exact power of 2.

$$\begin{aligned}\hat{c}_i &= i + 2 - 2^{\lceil \lg i \rceil} + 2^{\lceil \lg (i-1) \rceil} \\ &= i + 2 - 2(i - 1) + (i - 1)\end{aligned}$$

Calculation

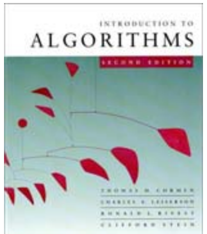
Case 1: $i - 1$ is an exact power of 2.

$$
\begin{aligned}
\hat{c}_i &= i + 2 - 2^{\lceil \lg i \rceil} + 2^{\lceil \lg (i-1) \rceil} \\
&= i + 2 - 2(i - 1) + (i - 1) \\
&= i + 2 - 2i + 2 + i - 1
\end{aligned}
$$

Calculation

Case 1: $i - 1$ is an exact power of 2.

$$\begin{aligned}\hat{c}_i &= i + 2 - 2^{\lceil \lg i \rceil} + 2^{\lceil \lg (i-1) \rceil} \\ &= i + 2 - 2(i-1) + (i-1) \\ &= i + 2 - 2i + 2 + i - 1 \\ &= 3\end{aligned}$$



Calculation

Case 1: $i - 1$ is an exact power of 2.

$$\begin{aligned}\hat{c}_i &= i + 2 - 2^{\lceil \lg i \rceil} + 2^{\lceil \lg (i-1) \rceil} \\ &= i + 2 - 2(i - 1) + (i - 1) \\ &= i + 2 - 2i + 2 + i - 1 \\ &= 3\end{aligned}$$

Case 2: $i - 1$ is *not* an exact power of 2.

$$\hat{c}_i = 1 + 2 - 2^{\lceil \lg i \rceil} + 2^{\lceil \lg (i-1) \rceil}$$

Calculation

Case 1: $i - 1$ is an exact power of 2.

$$\begin{aligned}\hat{c}_i &= i + 2 - 2^{\lceil \lg i \rceil} + 2^{\lceil \lg (i-1) \rceil} \\ &= i + 2 - 2(i - 1) + (i - 1) \\ &= i + 2 - 2i + 2 + i - 1 \\ &= 3\end{aligned}$$

Case 2: $i - 1$ is *not* an exact power of 2.

$$\begin{aligned}\hat{c}_i &= 1 + 2 - 2^{\lceil \lg i \rceil} + 2^{\lceil \lg (i-1) \rceil} \\ &= 3 \qquad\qquad \text{(since } 2^{\lceil \lg i \rceil} = 2^{\lceil \lg (i-1) \rceil}\text{)}\end{aligned}$$

Calculation

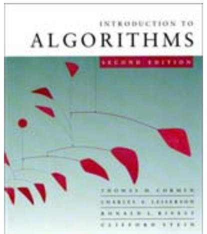
Case 1: $i - 1$ is an exact power of 2.

$$
\begin{aligned}
\hat{c}_i &= i + 2 - 2^{\lceil \lg i \rceil} + 2^{\lceil \lg (i-1) \rceil} \\
&= i + 2 - 2(i - 1) + (i - 1) \\
&= i + 2 - 2i + 2 + i - 1 \\
&= 3
\end{aligned}
$$

Case 2: $i - 1$ is *not* an exact power of 2.

$$
\begin{aligned}
\hat{c}_i &= 1 + 2 - 2^{\lceil \lg i \rceil} + 2^{\lceil \lg (i-1) \rceil} \\
&= 3
\end{aligned}
$$

Therefore, n insertions cost $\Theta(n)$ in the worst case.



Calculation

Case 1: $i - 1$ is an exact power of 2.

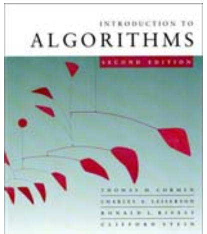
$$\begin{aligned}\hat{c}_i &= i + 2 - 2^{\lceil \lg i \rceil} + 2^{\lceil \lg (i-1) \rceil} \\ &= i + 2 - 2(i - 1) + (i - 1) \\ &= i + 2 - 2i + 2 + i - 1 \\ &= 3\end{aligned}$$

Case 2: $i - 1$ is *not* an exact power of 2.

$$\begin{aligned}\hat{c}_i &= 1 + 2 - 2^{\lceil \lg i \rceil} + 2^{\lceil \lg (i-1) \rceil} \\ &= 3\end{aligned}$$

Therefore, n insertions cost $\Theta(n)$ in the worst case.

Exercise: Fix the bug in this analysis to show that the amortized cost of the first insertion is only 2.



Conclusions

- Amortized costs can provide a clean abstraction of data-structure performance.
- Any of the analysis methods can be used when an amortized analysis is called for, but each method has some situations where it is arguably the simplest or most precise.
- Different schemes may work for assigning amortized costs in the accounting method, or potentials in the potential method, sometimes yielding radically different bounds.

