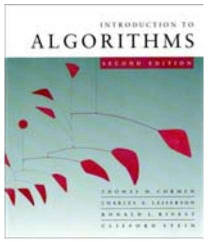
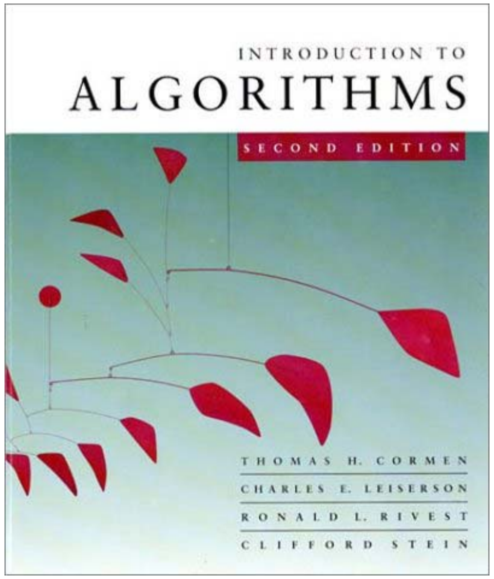


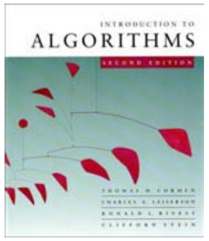
CS 5633 -- Spring 2008



Amortized Analysis

Carola Wenk

Slides courtesy of Charles Leiserson with small changes by Carola Wenk

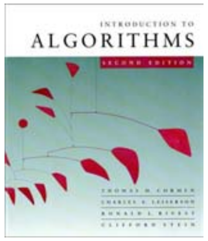


Dynamic tables

Task: Store a dynamic set in a table/array. Elements can only be inserted, and all inserted elements are stored in one contiguous part in the array. The table should be as small as possible, but large enough so that it won't overflow.

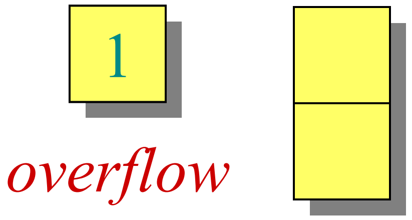
Problem: We may not know the proper size in advance!

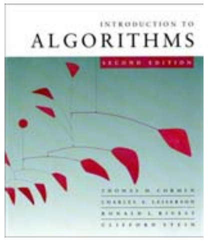
Solution: ***Dynamic tables.***
IDEA: Whenever the table overflows, "grow" it by allocating (via **`malloc`** or **`new`**) a new, larger table. Move all items from the old table into the new one, and free the storage for the old table.



Example of a dynamic table

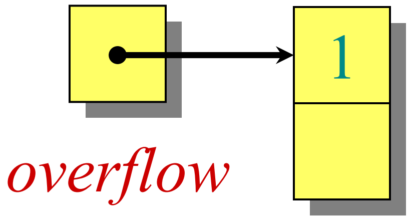
1. INSERT
2. INSERT

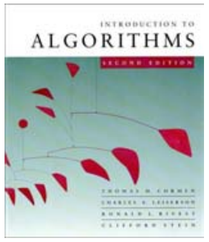




Example of a dynamic table

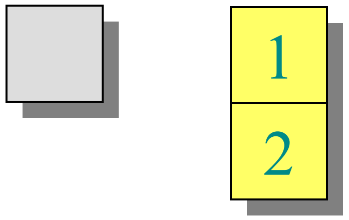
1. INSERT
2. INSERT

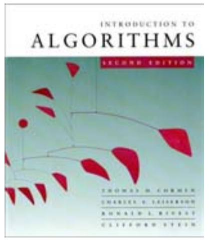




Example of a dynamic table

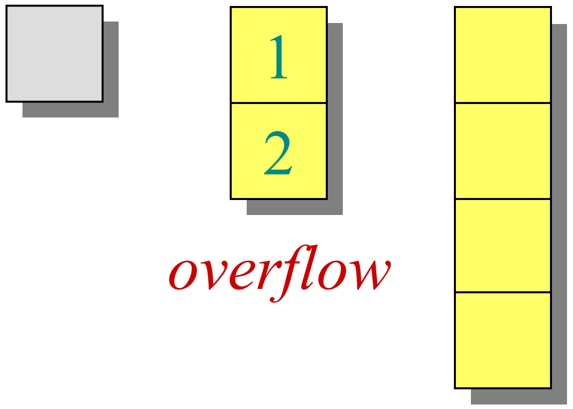
1. INSERT
2. INSERT

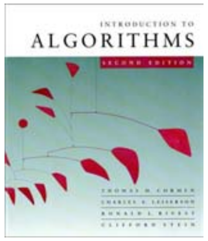




Example of a dynamic table

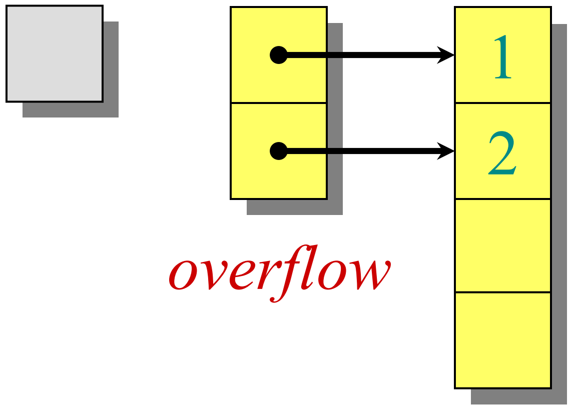
1. INSERT
2. INSERT
3. INSERT

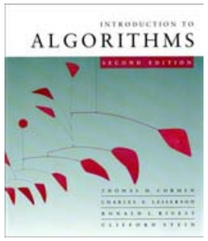




Example of a dynamic table

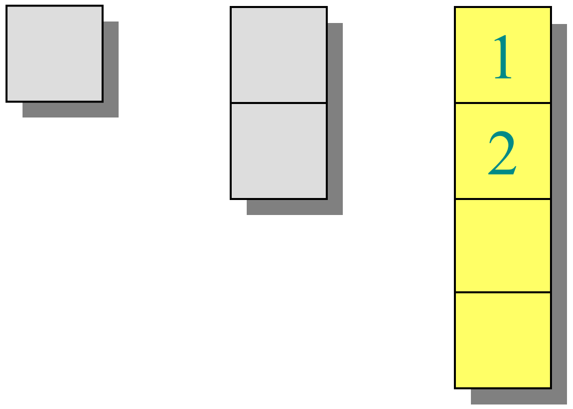
1. INSERT
2. INSERT
3. INSERT

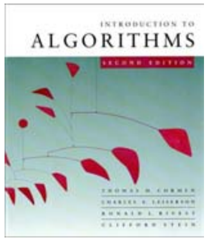




Example of a dynamic table

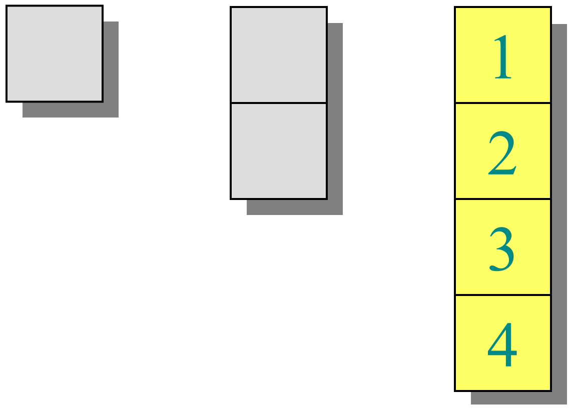
1. INSERT
2. INSERT
3. INSERT

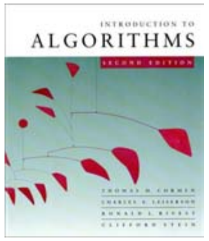




Example of a dynamic table

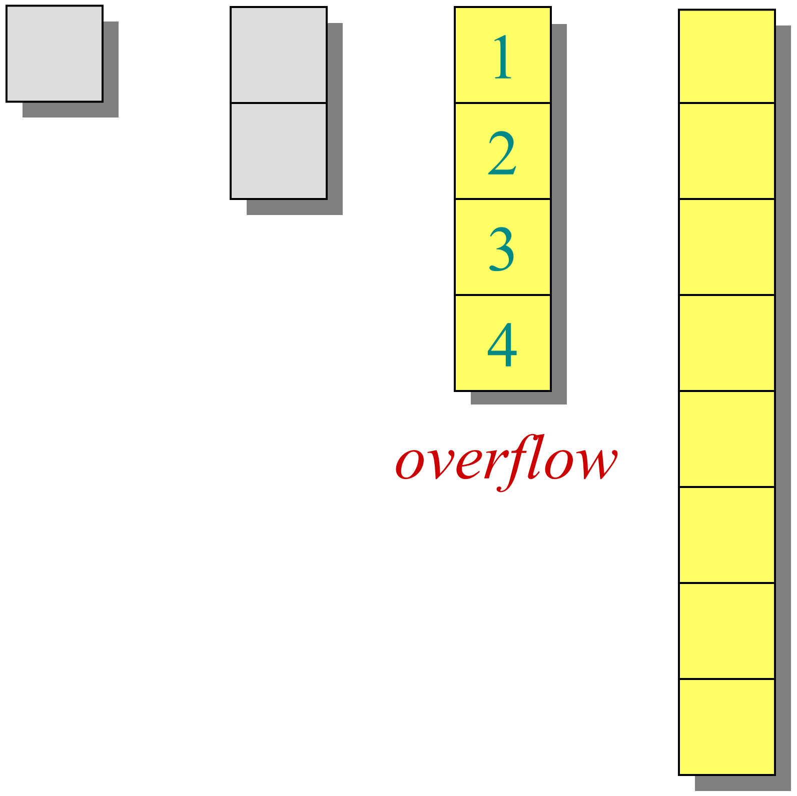
1. INSERT
2. INSERT
3. INSERT
4. INSERT

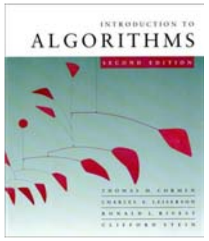




Example of a dynamic table

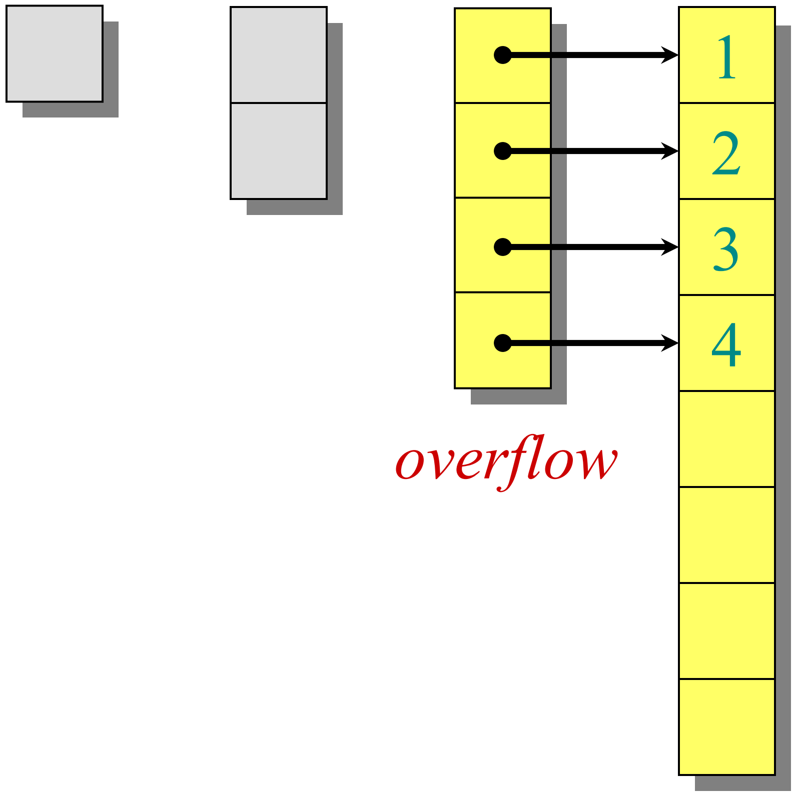
1. INSERT
2. INSERT
3. INSERT
4. INSERT
5. INSERT

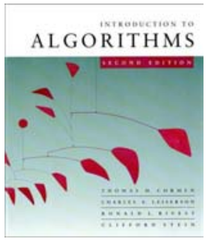




Example of a dynamic table

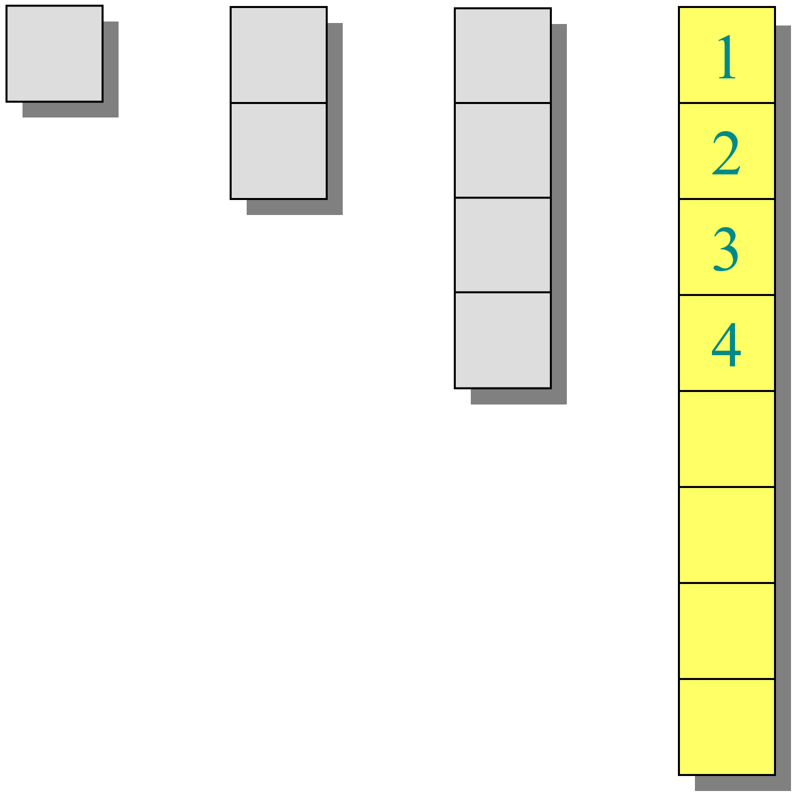
1. INSERT
2. INSERT
3. INSERT
4. INSERT
5. INSERT

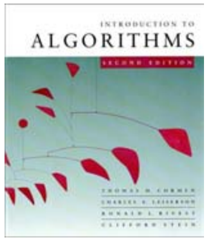




Example of a dynamic table

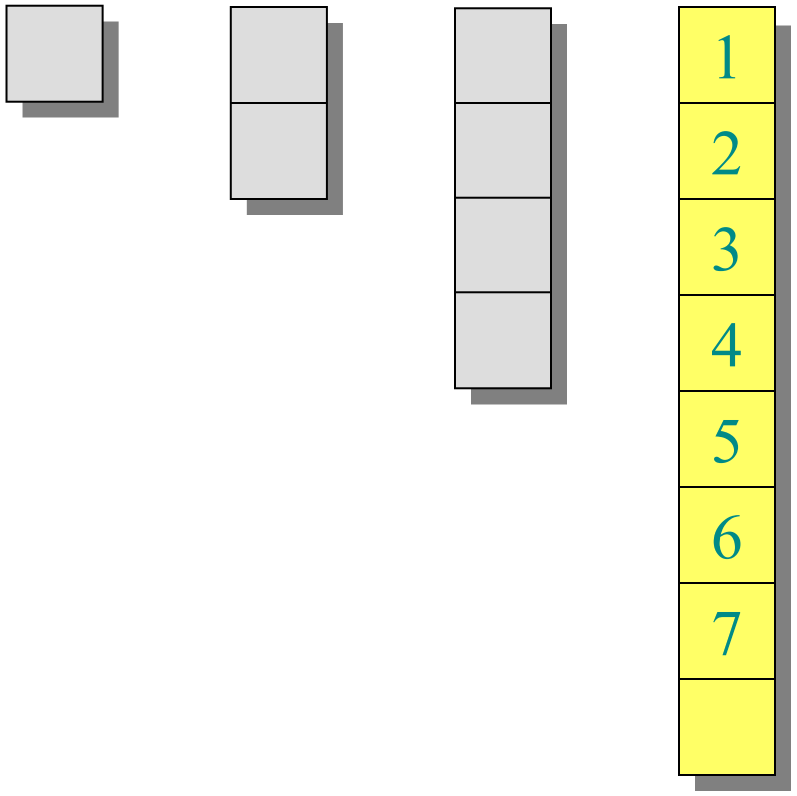
1. INSERT
2. INSERT
3. INSERT
4. INSERT
5. INSERT

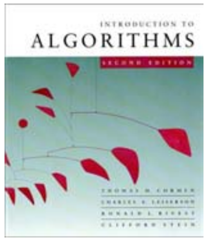




Example of a dynamic table

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



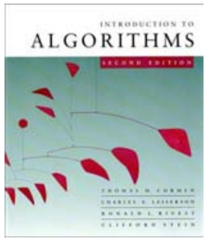


Worst-case analysis

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

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

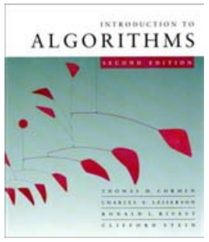
Let's see why.



Tighter analysis

Let $c_i =$ the cost of the ith insertion

i	1	2	3	4	5	6	7	8	9	10
$size_i$	1	2	4	4	8	8	8	8	16	16
$c_i \left\{ \right.$										

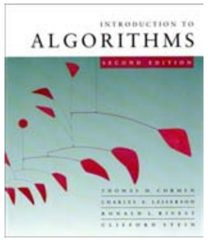


Tighter analysis

Let $c_i =$ the cost of the i th insertion

$= 1 +$ cost to double array size

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
	?	?	?	?	?	?	?	?	?	?

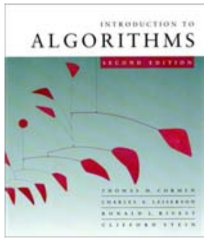


Tighter analysis

Let $c_i =$ the cost of the i th insertion

$=$ 1 + cost to double array size

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
	0	1	2	0	4	0	0	0	8	0

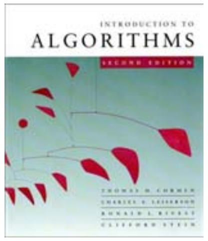


Tighter analysis

Let $c_i =$ the cost of the i th insertion

$= \ 1 +$ cost to double array size

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	$\left.\begin{matrix}1\\0\end{matrix}\right\}1$	$\left.\begin{matrix}1\\1\end{matrix}\right\}2$	$\left.\begin{matrix}1\\2\end{matrix}\right\}3$	$\left.\begin{matrix}1\\0\end{matrix}\right\}1$	$\left.\begin{matrix}1\\4\end{matrix}\right\}5$	$\left.\begin{matrix}1\\0\end{matrix}\right\}1$	$\left.\begin{matrix}1\\0\end{matrix}\right\}1$	$\left.\begin{matrix}1\\0\end{matrix}\right\}1$	$\left.\begin{matrix}1\\8\end{matrix}\right\}9$	$\left.\begin{matrix}1\\0\end{matrix}\right\}1$



Tighter analysis (continued)

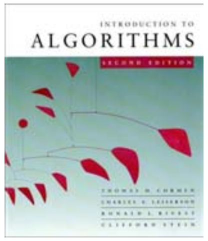
$$\text{Cost of } n \text{ insertions} = \sum_{i=1}^{n} c_i$$

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

$$\le 3n$$

$$= \Theta(n).$$

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



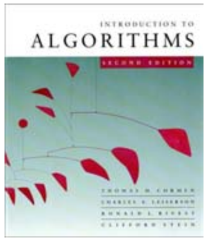
Amortized analysis

An ***amortized analysis*** is any strategy for analyzing a **sequence** of operations:

- compute the total cost of the sequence, OR

- amortized cost of an operation = average cost per operation, averaged over the number of operations in the sequence

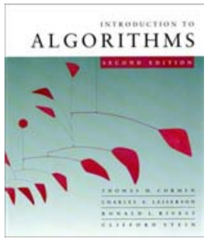
- amortized cost can be small, even though a single operation within the sequence might be expensive



Amortized analysis

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

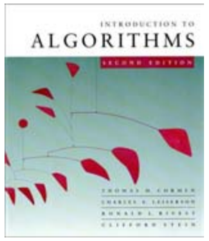
Three common amortization arguments:

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

Won't cover in class

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.

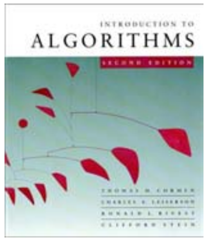


Accounting method

- Charge ith 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, and
- 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

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

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



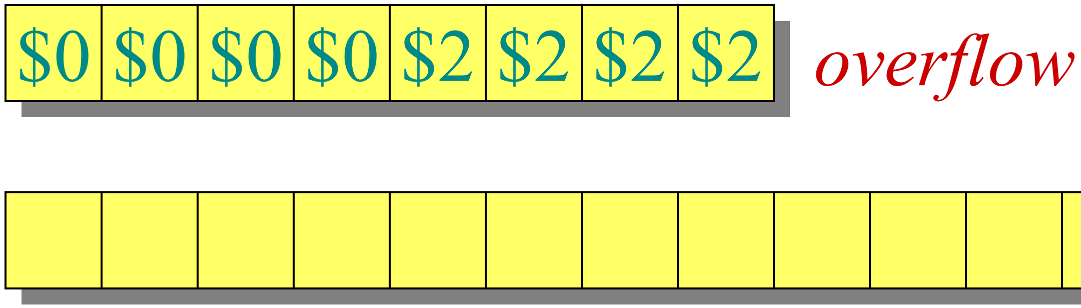
Accounting analysis of dynamic tables

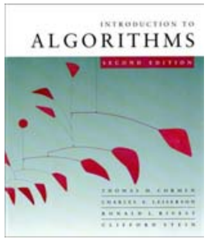
Charge an amortized cost of $\hat{c}_i = \$3$ for the ith insertion.

- \$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:





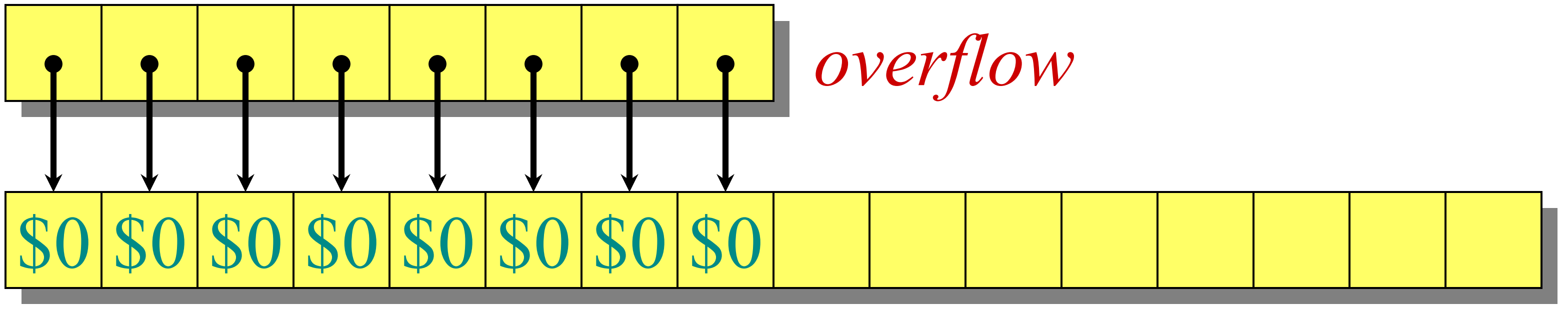
Accounting analysis of dynamic tables

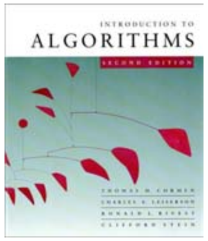
Charge an amortized cost of $\hat{c}_i = \$3$ for the i th insertion.

- $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:





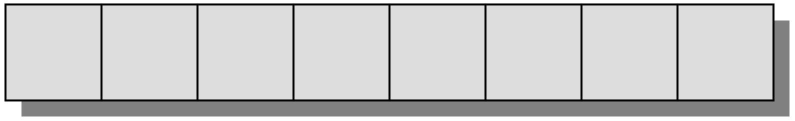
Accounting analysis of dynamic tables

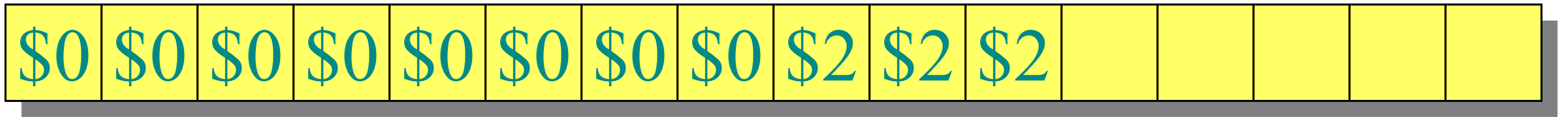
Charge an amortized cost of $\hat{c}_i = \$3$ for the ith insertion.

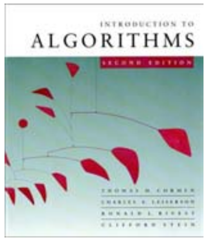
- \$1 pays for the immediate insertion.
- \$2 is stored for later table doubling.

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

Example:





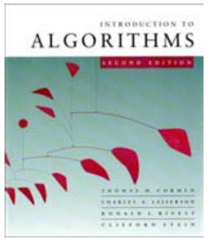


Accounting analysis (continued)

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.

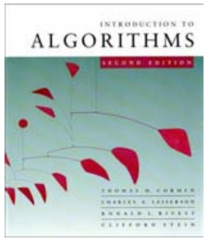


Incrementing a Binary Counter

Given: A k-bit binary counter $A[0,1,\ldots,k-1]$, initialized with $0,0,\ldots,0$. The counter supports the following INREMENT operation:

```
INCREMENT($A$) // increases counter by 1
    $i \leftarrow 0$
    while $i$<length($A$) and $A[i]=1$ do
        $A[i] \leftarrow 0$
        $i$++
    if $i$<length($A$) then
        $A[i] \leftarrow 1$
```

- **Question:** In a sequence of n INCREMENT operations, what is the amortized runtime of one INCREMENT operation?

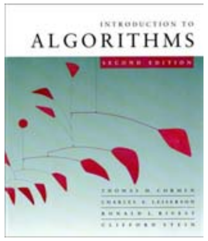


Binary Counter Example

Example for k=8 and n=9:

		1→0 flip	0→1 flip
Initial counter	0 0 0 0 0 0 0 0		
After 1 increment	0 0 0 0 0 0 0 1		$1
After 2 increments	0 0 0 0 0 0 1 0	$1	$1
After 3 increments	0 0 0 0 0 0 1 1		$1
After 4 increments	0 0 0 0 0 1 0 0	$2	$1
After 5 increments	0 0 0 0 0 1 0 1		$1
After 6 increments	0 0 0 0 0 1 1 0	$1	$1
After 7 increments	0 0 0 0 0 1 1 1		$1
After 8 increments	0 0 0 0 1 0 0 0	$3	$1
After 9 increments	0 0 0 0 1 0 0 1		$1

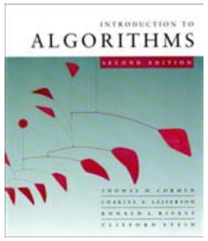
- The worst-case runtime of one INCREMENT operation is O(k)
- For n operations the total is O(nk)



Accounting Method

```
INCREMENT($A$) // increases counte
$i \leftarrow 0$
while $i$<length($A$) and $A[i]=1$ do
        $A[i] \leftarrow 0$
        $i$++
if $i$<length($A$) then
        $A[i] \leftarrow 1$
```

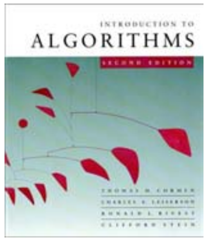
- Charge \$2 to set a bit to 1 ($0 \rightarrow 1$ flip)
  - ➢ \$1 pays for the actual flip
  - ➢ Store \$1 on the bit as credit to be used later when this bit is flipped back to 0
- Charge \$0 to set a bit to 0 ($1 \rightarrow 0$ flip)
  - ➢ Every 1 in the counter has \$1 credit on it, which is used to pay for this flip



Binary Counter Example

Example for $k=8$ and $n=9$:

		$1 \to 0$ flip	$0 \to 1$ flip
Initial counter	0 0 0 0 0 0 0 0		
After 1 increment	0 0 0 0 0 0 0 1		\$1
After 2 increments	0 0 0 0 0 0 1 0	\$1	\$1
After 3 increments	0 0 0 0 0 0 1 1		\$1
After 4 increments	0 0 0 0 0 1 0 0	\$2	\$1
After 5 increments	0 0 0 0 0 1 0 1		\$1
After 6 increments	0 0 0 0 0 1 1 0	\$1	\$1
After 7 increments	0 0 0 0 0 1 1 1		\$1
After 8 increments	0 0 0 0 1 0 0 0	\$3	\$1
After 9 increments	0 0 0 0 1 0 0 1		\$1

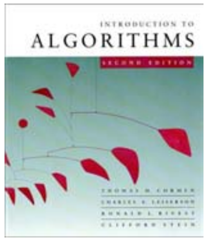


Accounting Method

```
INCREMENT($A$) // increases counte
$i \leftarrow 0$
while $i$<length($A$) and $A[i]$=1 do
        $A[i] \leftarrow 0$
        $i$++
if $i$<length($A$) then
        $A[i] \leftarrow 1$
```

- Charge $2 to set a bit to 1 ($0 \rightarrow 1$ flip)
    - ➢ $1 pays for the actual flip
    - ➢ Store $1 on the bit as credit to be used later when this bit is flipped back to 0
- Charge $0 to set a bit to 0 ($1 \rightarrow 0$ flip)
    - ➢ Every 1 in the counter has $1 credit on it, which is used to pay for this flip

$\Rightarrow$ Since each INCREMENT operation is composed of one $0 \rightarrow 1$ flip and possibly multiple $1 \rightarrow 0$ flips, the asymptotic runtime of one INCREMENT operation is O(1).



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.
- Different schemes may work for assigning amortized costs in the accounting method, sometimes yielding radically different bounds.