CMPS 6610/4610 – Fall 2016

Amortized Analysis

Carola Wenk

Slides courtesy of Charles Leiserson
with 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.

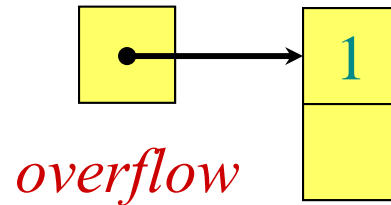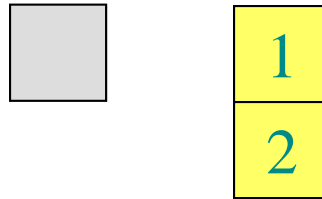# Example of a dynamic table
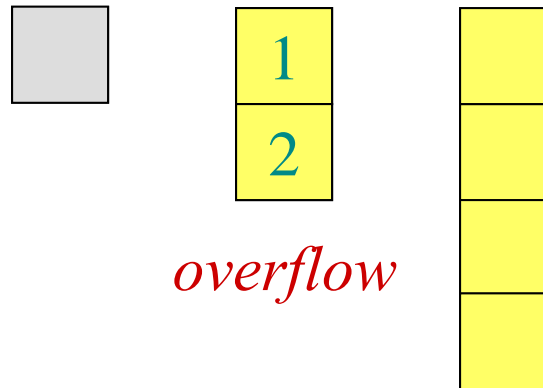
1. INSERT

2. INSERT

1

overflow

Example of a dynamic table

1. INSERT
2. INSERT

overflow

Example of a dynamic table

1. INSERT
2. INSERT

Example of a dynamic table

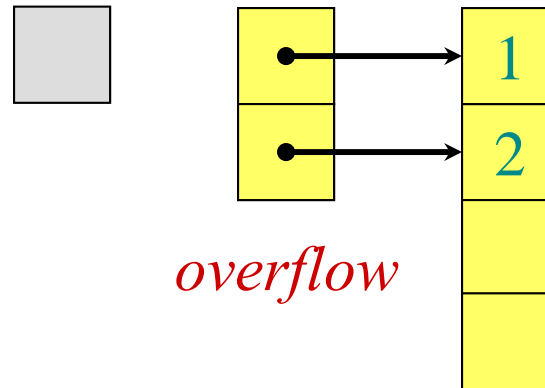1. INSERT
2. INSERT
3. INSERT

overflow

Example of a dynamic table

1. INSERT
2. INSERT
3. INSERT

overflow

Example of a dynamic table

1. INSERT
2. INSERT
3. INSERT

Example of a dynamic table

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

1
2
3
4

Example of a dynamic table

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

1
2
3
4

overflow

Example of a dynamic table

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

overflow
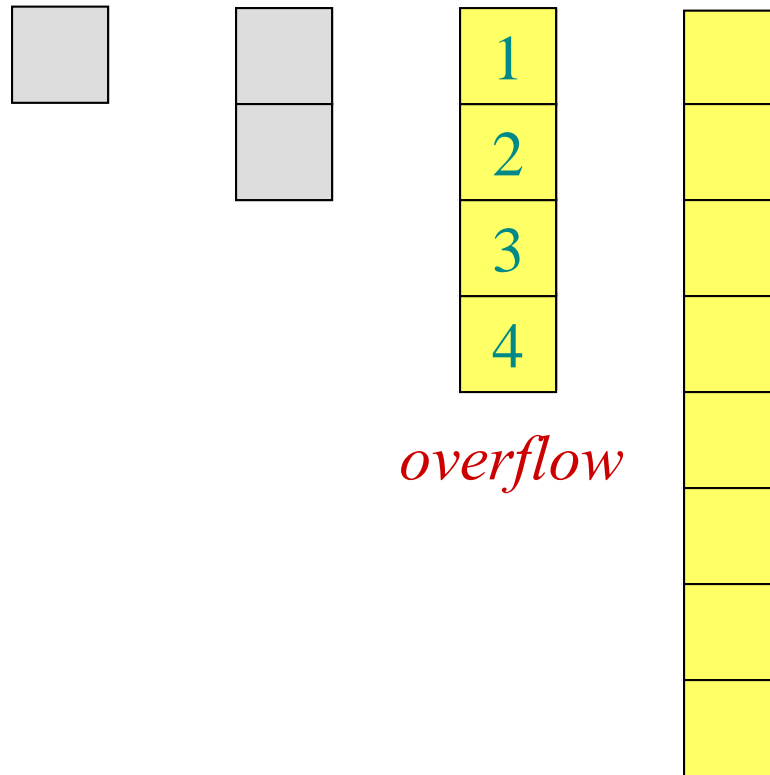
Example of a dynamic table

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

Example of a dynamic table
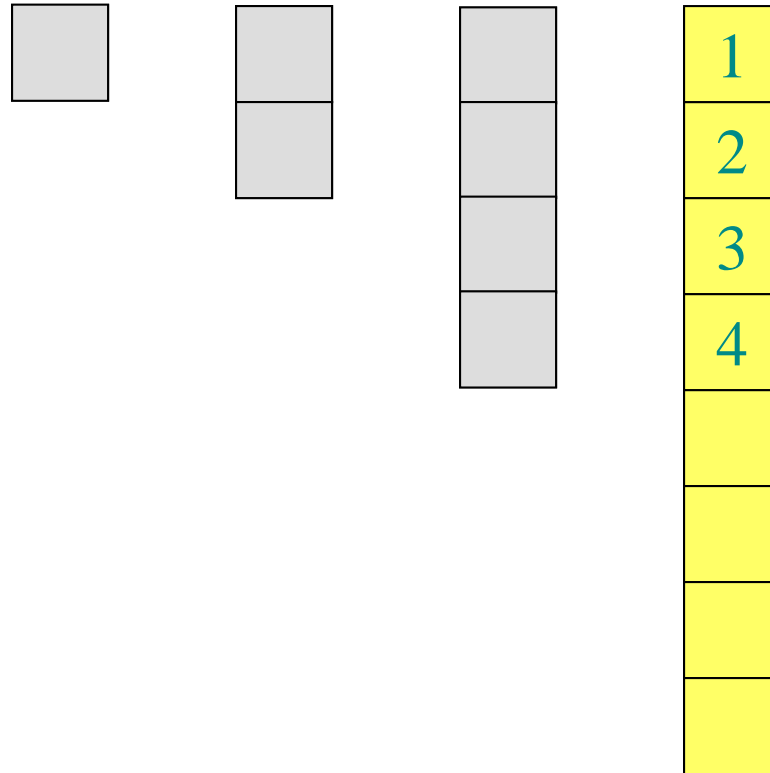
1. INSERT
2. INSERT
3. INSERT
4. INSERT
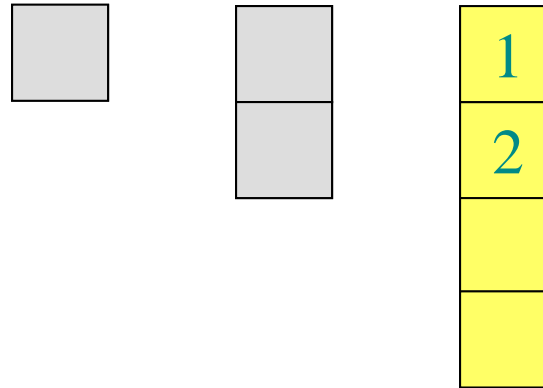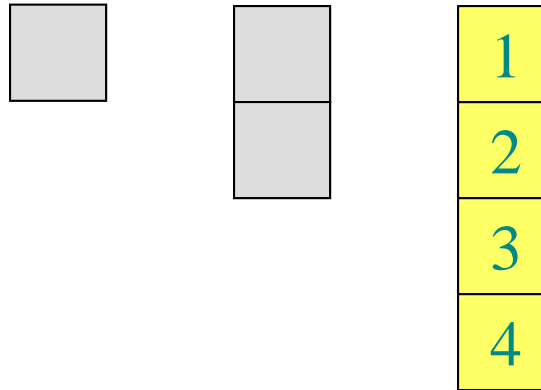5. INSERT
6. INSERT
7. INSERT
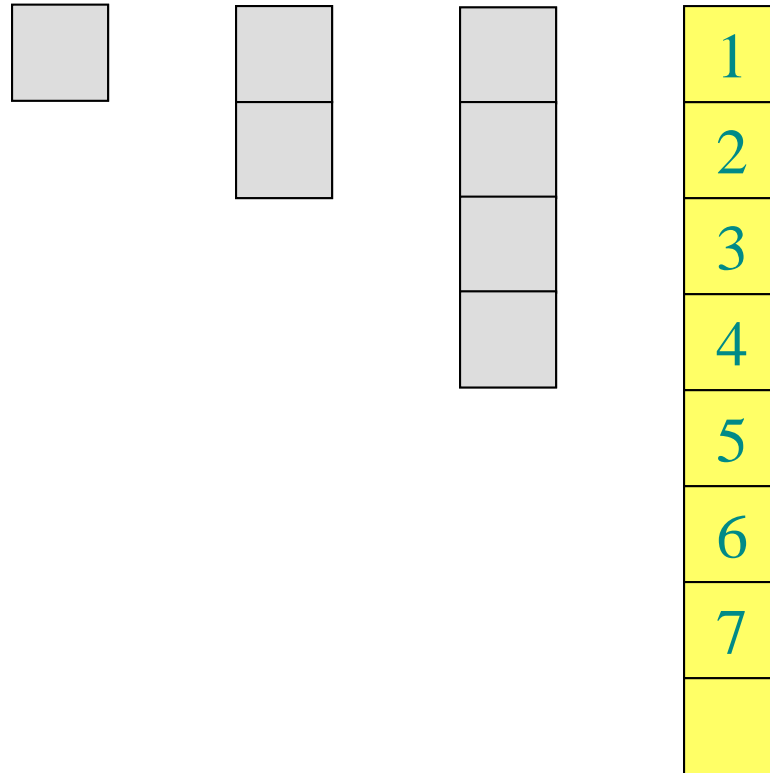
Worst-case analysis

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 $n \cdot O(n) = O(n^2)$.

NO! 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										

Tighter analysis

Let $c_i = $ the cost of the ith insertion

$$= 1 + \text{cost to double array size}$$

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 ith insertion

$$= 1 + \text{cost to double array size}$$

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 ith insertion

$\qquad = 1 +$ 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	$1\brace 0$ }1	$1\brace 1$ }2	$1\brace 2$ }3	$1\brace 0$ }1	$1\brace 4$ }5	$1\brace 0$ }1	$1\brace 0$ }1	$1\brace 0$ }1	$1\brace 8$ }9	$1\brace 0$ }1

Tighter analysis (continued)

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

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

$$\leq 3n$$

$$= \Theta(n).$$

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:

• 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.

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:

$0	$0	$0	$0	$2	$2	$2	$2	*overflow*
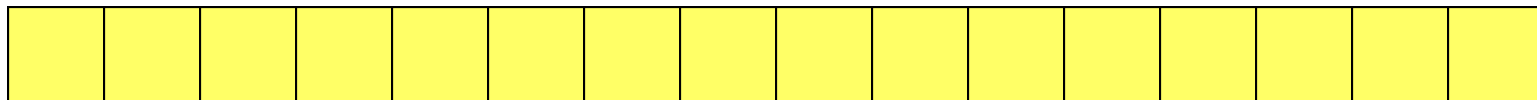
Accounting analysis of dynamic tables

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:

overflow

$0 $0 $0 $0 $0 $0 $0 $0

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					
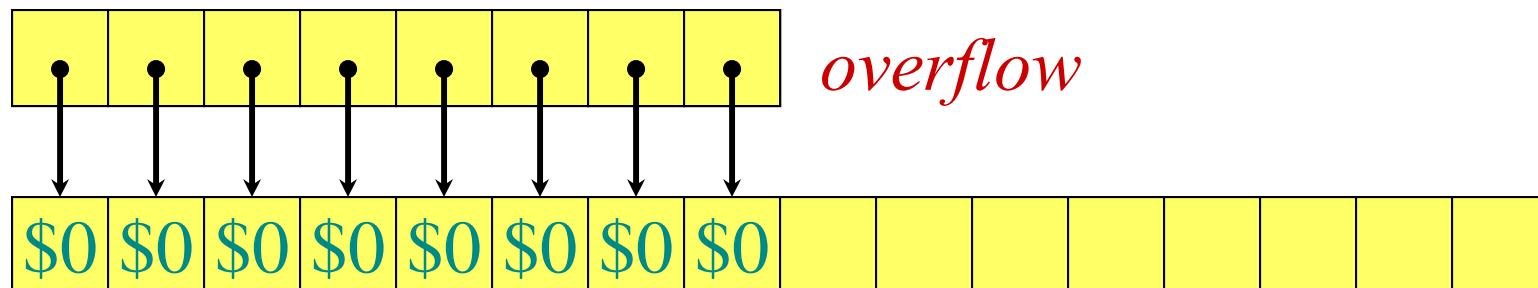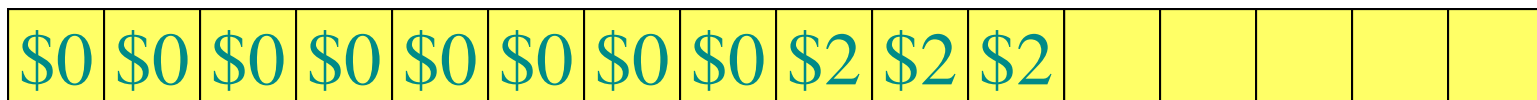
Accounting analysis (continued)

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*** $\Phi : \{D_i\} \to \mathsf{R}$, 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

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

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

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

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

$$\geq \sum_{i=1}^{n} c_i \qquad \text{since } \Phi(D_n) \geq 0 \text{ and}$$
$$\Phi(D_0) = 0.$$

Potential analysis of table doubling

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

$\lceil \log i \rceil$	0	1	2	2	3	3	3	3	4	4
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
$\Phi(D_i)$ $bank_i$	1	2	2	4	2	4	6	8	2	4
$\Delta\Phi_i$		1	0	2	-2	2	2	2	-6	2

Potential analysis of table doubling

- Potential after ith insertion is $\Phi(D_i) = 2i - \text{size}_i$
- Immediately before an $\boxed{\text{expansion}}$, $i = \text{size}_i$, and thus $\Phi(D_i) = i$.
- After an expansion, $i - 1 = \text{size}_i / 2$, and thus $\Phi(D_i) = 0 + 2$ after insertion of one element.

$\lceil \log i \rceil$	0	1	2	2	3	3	3	3	4	4
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
$\Phi(D_i)$ $bank_i$	1	2	2	4	2	4	6	8	2	4
$\Phi(D_i) - \Phi(D_i)$		1	0	2	-2	2	2	2	-6	2

Potential analysis of table doubling

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

- Immediately before an expansion, $i = \text{size}_i$, and thus $\Phi(D_i) = i$.
- After an expansion, $i - 1 = \text{size}_i / 2$, and thus $\Phi(D_i) = 0 + 2$ after insertion of one element.

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

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*** $\Phi : \{D_i\} \rightarrow \mathsf{R}$, 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 $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$.

Calculation of amortized costs

The amortized cost of the ith insertion is

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

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

$$= \begin{cases} i + \left(2i - 2^{\lceil \log i \rceil}\right) - \left(2(i-1) - 2^{\lceil \log (i-1) \rceil}\right), \\ \quad \text{if } i - 1 \text{ is an exact power of } 2 \\ 1 + \left(2i - 2^{\lceil \log i \rceil}\right) - \left(2(i-1) - 2^{\lceil \log (i-1) \rceil}\right), \\ \quad \text{otherwise.} \end{cases}$$

for all $i \geq 1$

Calculation (Case 1)

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

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

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

$$= i + 2 - \left(2^{\log (i-1)+1} - 2^{\log (i-1)}\right)$$

$$= i + 2 - \left(2(i - 1) - (i - 1)\right)$$

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

$$= 3$$

Calculation (Case 2)

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

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

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

$$= 3$$

And for $i = 1$:

$$\hat{c}_1 = 3 - \left(2^{\lceil \log i \rceil} - 2^{\lceil \log (i-1) \rceil}\right) = 3 - (1 - 0) = 2$$

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

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 INCREMENT 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		$1
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 count

$i \leftarrow 0$
while $i<$length(A) and $A[i]$=1 do
$\quad A[i] \leftarrow 0$
$\quad i$++
if $i<$length(A) then
$\quad A[i] \leftarrow 1$

- Charge $2 to set a bit to 1 (0→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→0 flip)

  ➢ Every 1 in the counter has $1 credit on it, which is used to pay for this flip

- The bank balance is never negative, since each 0→1 flip pays for its own cost, and each 1→0 flip is prepaid by the $1 credit on it.

Binary Counter Example

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

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

Actual cost **Amortized cost**

Accounting Method

$\Rightarrow$ Since each INCREMENT operation is composed of one $0 \rightarrow 1$ flip and possibly multiple $1 \rightarrow 0$ flips, the amortized 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.