

# Sorting, Searching, Selection, and Amortized Analysis

Hengfeng Wei

hfwei@nju.edu.cn

April 24, 2018



## Maximal-sum Subarray (Problem 3.7)

- ▶ Array  $A[1 \cdots n], a_i \geq 0$
- ▶ To find (the sum of) an MS in  $A$

$$A[-2, 1, -3, \boxed{4, -1, 2, 1}, -5, 4]$$

$$O(n)$$

$MSS[i]$ : the sum of the MS *ending with*  $a_i$  or 0

$MSS[i]$ : the sum of the MS *ending with*  $a_i$  or 0

$$mss = \max_{1 \leq i \leq n} MSS[i]$$

$MSS[i]$ : the sum of the MS *ending with*  $a_i$  or 0

$$mss = \max_{1 \leq i \leq n} MSS[i]$$

$$MSS[i] = \max \{MSS[i-1] + a_i, 0\}$$

*Q* : Where does the  $MSS[i]$  start?

$MSS[i]$ : the sum of the MS *ending with*  $a_i$  or 0

$$mss = \max_{1 \leq i \leq n} MSS[i]$$

$$MSS[i] = \max \{MSS[i-1] + a_i, 0\}$$

*Q* : Where does the  $MSS[i]$  start?

$$MSS[0] = 0$$

---

```
1: procedure MSS( $A[1 \cdots n]$ )
2:   MSS[0]  $\leftarrow$  0
3:   for  $i \leftarrow 1$  to  $n$  do
4:     MSS[ $i$ ]  $\leftarrow$   $\max \{ \text{MSS}[i - 1] + A[i], 0 \}$ 
5:   return  $\max_{1 \leq i \leq n} \text{MSS}[i]$ 
```

---

---

```
1: procedure MSS( $A[1 \cdots n]$ )
2:    $\text{MSS}[0] \leftarrow 0$ 
3:   for  $i \leftarrow 1$  to  $n$  do
4:      $\text{MSS}[i] \leftarrow \max \{ \text{MSS}[i-1] + A[i], 0 \}$ 
5:   return  $\max_{1 \leq i \leq n} \text{MSS}[i]$ 
```

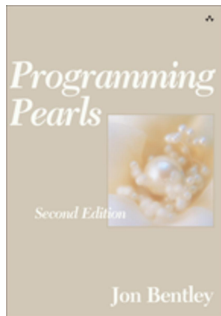
---

---

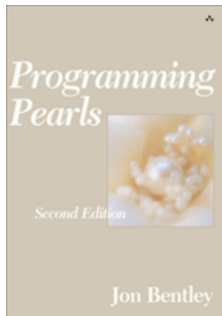
```
1: procedure MSS( $A[1 \cdots n]$ )
2:    $\text{mss} \leftarrow 0$ 
3:    $\text{MSS} \leftarrow 0$ 
4:   for  $i \leftarrow 1$  to  $n$  do
5:      $\text{MSS} \leftarrow \max \{ \text{MSS} + A[i], 0 \}$ 
6:      $\text{mss} \leftarrow \max \{ \text{mss}, \text{MSS} \}$ 
7:   return  $\text{mss}$ 
```

---



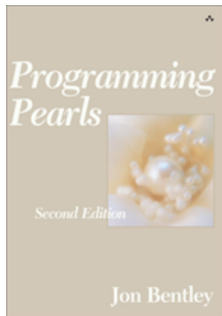


Ulf Grenander  $O(n^3) \implies O(n^2)$



Ulf Grenander  $O(n^3) \implies O(n^2)$

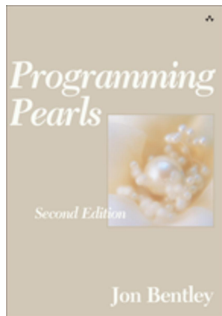
Michael Shamos  $O(n \log n)$ , onenight



Ulf Grenander  $O(n^3) \implies O(n^2)$

Michael Shamos  $O(n \log n)$ , onenight

Jon Bentley Conjecture:  $\Omega(n \log n)$

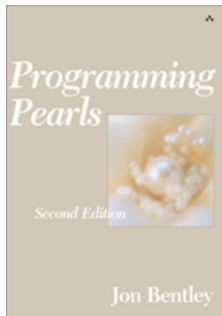


Ulf Grenander  $O(n^3) \implies O(n^2)$

Michael Shamos  $O(n \log n)$ , onenight

Jon Bentley Conjecture:  $\Omega(n \log n)$

Michael Shamos Carnegie Mellon seminar



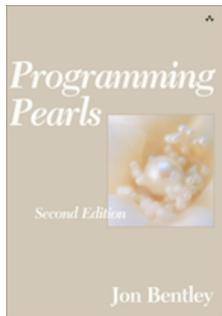
Ulf Grenander  $O(n^3) \implies O(n^2)$

Michael Shamos  $O(n \log n)$ , onenight

Jon Bentley Conjecture:  $\Omega(n \log n)$

Michael Shamos Carnegie Mellon seminar

Jay Kadane  $O(n)$ ,



Ulf Grenander  $O(n^3) \implies O(n^2)$

Michael Shamos  $O(n \log n)$ , onenight

Jon Bentley Conjecture:  $\Omega(n \log n)$

Michael Shamos Carnegie Mellon seminar

Jay Kadane  $O(n)$ ,  $\leq 1$  minute

# Sorting

### Definition ( $K$ -sorting (Problem 6.8))

An array  $A[1 \cdots n]$  is  *$k$ -sorted* if it can be divided into  $k$  blocks, each of size  $n/k$  (we assume that  $n/k \in \mathbb{N}$ ), such that the elements in each block are larger than the elements in earlier blocks and smaller than elements in later blocks. The elements within each block need *not* be sorted.

$$n = 16, k = 4, \frac{n}{k} = 4$$

1, 2, 4, 3;    7, 6, 8, 5;    10, 11, 9, 12;    15, 13, 16, 14



$k$ -sorted

$k$ -sorted

1-sorted

$k$ -sorted

1-sorted  $\rightarrow$  2-sorted

$k$ -sorted

1-sorted  $\rightarrow$  2-sorted  $\rightarrow$  4-sorted

## $k$ -sorted

1-sorted  $\rightarrow$  2-sorted  $\rightarrow$  4-sorted  $\rightarrow \dots \rightarrow n$ -sorted

## $k$ -sorted

1-sorted  $\rightarrow$  2-sorted  $\rightarrow$  4-sorted  $\rightarrow \dots \rightarrow n$ -sorted

Quicksort (with median as pivot) stops after the  $\log k$  recursions.

$k$ -sorted

1-sorted  $\rightarrow$  2-sorted  $\rightarrow$  4-sorted  $\rightarrow \dots \rightarrow n$ -sorted

Quicksort (with median as pivot) stops after the  $\log k$  recursions.

$$\Theta(n \log k)$$

$$\Omega(n \log k)$$



$$\Omega(n \log k)$$

$$L \geq$$

$$\Omega(n \log k)$$

$$L \geq \binom{n}{n/k, \dots, n/k}$$

$$\Omega(n \log k)$$

$$L \geq \binom{n}{n/k, \dots, n/k} = \frac{n!}{((\frac{n}{k})!)^k}$$

$$\Omega(n \log k)$$

$$L \geq \binom{n}{n/k, \dots, n/k} = \frac{n!}{((\frac{n}{k})!)^k}$$

$$H \geq \log \left( \frac{n!}{((\frac{n}{k})!)^k} \right)$$

$$\Omega(n \log k)$$

$$L \geq \binom{n}{n/k, \dots, n/k} = \frac{n!}{((\frac{n}{k})!)^k}$$

$$H \geq \log \left( \frac{n!}{((\frac{n}{k})!)^k} \right)$$

$$n! \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \implies \log n! \sim n \log n$$

## Bolts and Nuts (Problem 6.9)



## Bolts and Nuts (Problem 6.9)



Quicksort

## Bolts and Nuts (Problem 6.9)



Quicksort

$$A(n) = O(n \log n)$$



## Bolts and Nuts (Problem 6.9)



### Quicksort

$$A(n) = O(n \log n)$$

In the worst case:

- ▶ “Matching Nuts and Bolts” by Alon *et al.*,  $\Theta(n \log^4 n)$
- ▶ “Matching Nuts and Bolts Optimality” by Bradford, 1995,  $\Theta(n \log n)$



$$\Omega(n \log n)$$



$$\Omega(n \log n)$$

$$3^H \geq L \geq n!$$



$$\Omega(n \log n)$$

$$3^H \geq L \geq n! \implies H \geq \log n! \implies H = \Omega(n \log n)$$

# Searching

## Repeated Elements (Problem 2.12)

$$R[1 \dots n]$$

$$\# > \lfloor \frac{n}{13} \rfloor$$

To find all  $\frac{n}{13}$ -repeated elements

$\text{check}(R[i], R[j])$

## Repeated Elements (Problem 2.12)

$$R[1 \dots n]$$

$$\# > \lfloor \frac{n}{13} \rfloor$$

To find all  $\frac{n}{13}$ -repeated elements

$$\text{check}(R[i], R[j])$$

$$\# \text{ of } \frac{n}{13}\text{-repeated elements} \leq 13$$

## Repeated Elements (Problem 2.12)

$$R[1 \dots n]$$

$$\# > \lfloor \frac{n}{13} \rfloor$$

To find all  $\frac{n}{13}$ -repeated elements

$\text{check}(R[i], R[j])$

$\#$  of  $\frac{n}{13}$ -repeated elements  $\leq 13$

$x$  is a  $\frac{n}{13}$ -repeated element

$\implies x$  is a  $\frac{n}{26}$ -repeated element of  $R[1 \dots \frac{n}{2}]$  or  $R[\frac{n}{2} + 1 \dots n]$



# Divide and Conquer Algorithm

## Divide and Conquer Algorithm

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

## Divide and Conquer Algorithm

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

$$Q : 13 \rightarrow k$$

## Divide and Conquer Algorithm

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

$$Q : 13 \rightarrow k$$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(kn)$$

## Divide and Conquer Algorithm

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

$$Q : k \leftarrow 2$$

$$Q : 13 \rightarrow k$$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(kn)$$

## Divide and Conquer Algorithm

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

$$Q : k \leftarrow 2$$

$$Q : 13 \rightarrow k$$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(kn)$$



## Divide and Conquer Algorithm

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

$$Q : k \leftarrow 2$$

$$Q : 13 \rightarrow k$$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(kn)$$



$$k : O(n \log k)$$

$$\Omega(n \log k)$$



$$\Omega(n \log k)$$

$Q : \exists$  a repeated element?

$$\Omega(n \log k)$$

$Q : \exists$  a repeated element?

$L : \#$  of leaves?

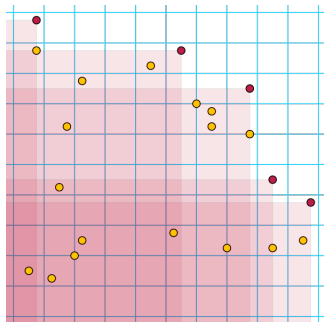
$$\Omega(n \log k)$$

$Q : \exists$  a repeated element?

$L : \#$  of leaves?

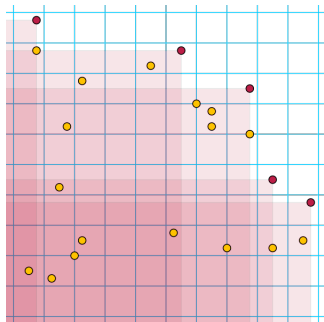
*“Finding Repeated Elements” by Misra & Gries, 1982*

## Maxima of a Point Set (Problem 6.15)



$$(x_1, y_1) \succ (x_2, y_2) \iff x_1 > x_2 \wedge y_1 > y_2$$

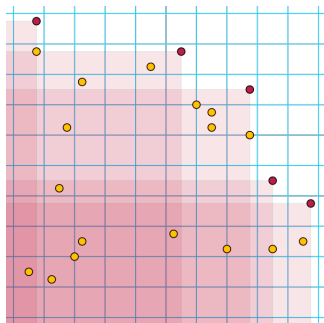
## Maxima of a Point Set (Problem 6.15)



$$(x_1, y_1) \succ (x_2, y_2) \iff x_1 > x_2 \wedge y_1 > y_2$$

$x$ -sorting,  $\max y$

## Maxima of a Point Set (Problem 6.15)



$$(x_1, y_1) \succ (x_2, y_2) \iff x_1 > x_2 \wedge y_1 > y_2$$

$$x\text{-sorting, } \max y \implies O(n \log n)$$

## Wrong Divide and Conquer Algorithms

$x$ -median &  $y$ -median

$$T(n) = T\left(\frac{3}{4}n\right) + O(n) \implies T(n) = O(n)$$

$$T(n) = 3T\left(\frac{1}{4}n\right) + O(n) \implies T(n) = O(n)$$

# Divide and Conquer Algorithm

$x$ -median



## Divide and Conquer Algorithm

$x$ -median

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

## Divide and Conquer Algorithm

$x$ -median

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$



$$\Omega(n \log n)$$

## Smallest Missing Positive Integer (Problem 9.6)

Sorted array  $A[1 \dots n]$

$$a_i \in \mathbb{Z}^+$$

$$\forall i \neq j : a_i \neq a_j$$

$$A = [1, 2, 4, 5] \implies 3$$

## Smallest Missing Positive Integer (Problem 9.6)

Sorted array  $A[1 \dots n]$

$$a_i \in \mathbb{Z}^+$$

$$\forall i \neq j : a_i \neq a_j$$

$$A = [1, 2, 4, 5] \implies 3$$

$$T(n) = O(n)$$

## Smallest Missing Positive Integer (Problem 9.6)

Sorted array  $A[1 \dots n]$

$$a_i \in \mathbb{Z}^+$$

$$\forall i \neq j : a_i \neq a_j$$

$$A = [1, 2, 4, 5] \implies 3$$

$$T(n) = O(n)$$

$$O(\log n)$$

## Smallest Missing Positive Integer (Problem 9.6)

Sorted array  $A[1 \dots n]$

$$a_i \in \mathbb{Z}^+$$

$$\forall i \neq j : a_i \neq a_j$$

$$A = [1, 2, 4, 5] \implies 3$$

$$T(n) = O(n)$$

$$O(\log n)$$

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

## Local Minimum (Problem 9.12)

$$A[1 \cdots n]$$

$$A[0] \geq A[1] \wedge A[n-2] \leq A[n-1]$$

$$A[i-1] \geq A[i] \leq A[i+1]$$

## Local Minimum (Problem 9.12)

$$A[1 \cdots n]$$

$$A[0] \geq A[1] \wedge A[n-2] \leq A[n-1]$$

$$A[i-1] \geq A[i] \leq A[i+1]$$

$\exists$



## Local Minimum (Problem 9.12)

$$A[1 \cdots n]$$

$$A[0] \geq A[1] \wedge A[n-2] \leq A[n-1]$$

$$A[i-1] \geq A[i] \leq A[i+1]$$

$\exists$

$$\text{Scan} : T(n) = O(n)$$

## Local Minimum (Problem 9.12)

$$A[1 \cdots n]$$

$$A[0] \geq A[1] \wedge A[n-2] \leq A[n-1]$$

$$A[i-1] \geq A[i] \leq A[i+1]$$

$\exists$

$$\text{Scan} : T(n) = O(n)$$

$$\min A : T(n) = O(n)$$

## Local Minimum (Problem 9.12)

$$A[1 \cdots n]$$

$$A[0] \geq A[1] \wedge A[n-2] \leq A[n-1]$$

$$A[i-1] \geq A[i] \leq A[i+1]$$

## Local Minimum (Problem 9.12)

$$A[1 \cdots n]$$

$$A[0] \geq A[1] \wedge A[n-2] \leq A[n-1]$$

$$A[i-1] \geq A[i] \leq A[i+1]$$

$$T(n) = O(\log n)$$

## Local Minimum (Problem 9.12)

$$A[1 \cdots n]$$

$$A[0] \geq A[1] \wedge A[n-2] \leq A[n-1]$$

$$A[i-1] \geq A[i] \leq A[i+1]$$

$$T(n) = O(\log n)$$

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

## Local Minimum (Problem 9.12)

$$A[1 \cdots n]$$

$$A[0] \geq A[1] \wedge A[n-2] \leq A[n-1]$$

$$A[i-1] \geq A[i] \leq A[i+1]$$

$$m = \frac{n}{2}$$

$$T(n) = O(\log n)$$

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

## Local Minimum (Problem 9.12)

$$A[1 \cdots n]$$

$$A[0] \geq A[1] \wedge A[n-2] \leq A[n-1]$$

$$A[i-1] \geq A[i] \leq A[i+1]$$

$$m = \frac{n}{2}$$

$$T(n) = O(\log n)$$

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

$$A[m-1] \geq A[m] \leq A[m+1]$$

$$A[m-1] < A[m] \vee A[m+1] < A[m]$$

## Local Minimum (Problem 9.12)

$$A[1 \cdots n]$$

$$A[0] \geq A[1] \wedge A[n-2] \leq A[n-1]$$

$$A[i-1] \geq A[i] \leq A[i+1]$$

$$m = \frac{n}{2}$$

$$T(n) = O(\log n)$$

$$A[m-1] \geq A[m] \leq A[m+1]$$

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

$$A[m-1] < A[m] \vee A[m+1] < A[m]$$

$$n = 1 \quad \vee \quad n = 2$$



## Searching in Matrix (Problem 9.8)

$$M : m \times n$$

Row: increasing from left to right

Col: increasing from top to down

$$x \in M?$$

## Searching in Matrix (Problem 9.8)

$$M : m \times n$$

Row: increasing from left to right

Col: increasing from top to down

$$x \in M?$$

Divide & Conquer

## Searching in Matrix (Problem 9.8)

$$M : m \times n$$

Row: increasing from left to right

Col: increasing from top to down

$$x \in M?$$

Divide & Conquer

$$T(m, n) = 3T\left(\frac{m}{2}, \frac{n}{2}\right) + 1$$

## Searching in Matrix (Problem 9.8)

$$M : m \times n$$

Row: increasing from left to right

Col: increasing from top to down

$$x \in M?$$

Divide & Conquer

$$T(m, n) = 3T\left(\frac{m}{2}, \frac{n}{2}\right) + 1$$

Always checking the lower left corner.

## Searching in Matrix (Problem 9.8)

$$M : m \times n$$

Row: increasing from left to right

Col: increasing from top to down

$$x \in M?$$

Divide & Conquer

$$T(m, n) = 3T\left(\frac{m}{2}, \frac{n}{2}\right) + 1$$

Always checking the lower left corner.

$$T(m, n) = m + n - 1$$

Assume  $M : n \times n$

$$W(n) \leq 2n - 1$$

Assume  $M : n \times n$

$$W(n) \leq 2n - 1$$

$$W(n) \geq 2n - 1$$

Assume  $M : n \times n$

$$W(n) \leq 2n - 1$$

$$W(n) \geq 2n - 1$$

By Adversary Argument!



Assume  $M : n \times n$

$$W(n) \leq 2n - 1$$

$$W(n) \geq 2n - 1$$

By Adversary Argument!

Diagonals:  $i + j = n - 1$  &  $i + j = n$

Assume  $M : n \times n$

$$W(n) \leq 2n - 1$$

$$W(n) \geq 2n - 1$$

By Adversary Argument!

Diagonals:  $i + j = n - 1$  &  $i + j = n$

No particular ordering requirements on these two diagonals!

Assume  $M : n \times n$

$$W(n) \leq 2n - 1$$

$$W(n) \geq 2n - 1$$

By Adversary Argument!

Diagonals:  $i + j = n - 1$  &  $i + j = n$

No particular ordering requirements on these two diagonals!

$$i + j \leq n - 1 \implies x > M_{ij}$$

$$i + j > n - 1 \implies x < M_{ij}$$

$A + B = c$  (Problem 9.9)

*Sorted*  $S[1 \cdots n]$ ,  $c$

$\exists A, B : A + B = c?$

$O(n)$

$A + B = c$  (Problem 9.9)

*Sorted*  $S[1 \cdots n]$ ,  $c$

$\exists A, B : A + B = c?$

$O(n)$

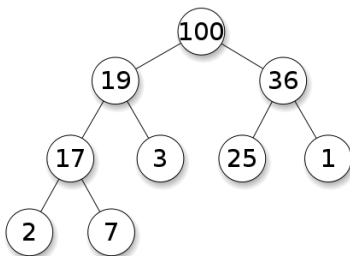
Left-right-pointers iteration

$S_i + S_j \geq < c$

# Selection

## The $k$ -th Largest Elements in a Heap (Problem 7.2)

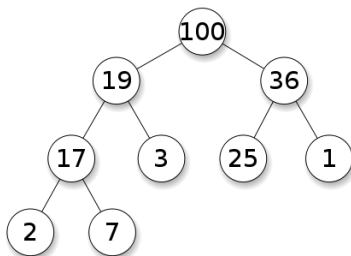
$$k \ll n$$



$$T(k)$$

## The $k$ -th Largest Elements in a Heap (Problem 7.2)

$$k \ll n$$



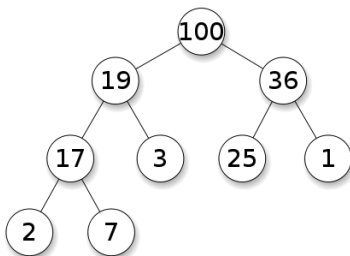
$$T(k)$$

Must be in the first  $k$  layers



## The $k$ -th Largest Elements in a Heap (Problem 7.2)

$$k \ll n$$



$$T(k)$$

Must be in the first  $k$  layers  $\implies O(2^k)$

## The Largest $k$ Elements (Problem 8.5)

$$O(n \log n)$$

$$O(n + k \log n)$$

$$O(n + k \log k)$$

## The Largest $k$ Elements (Problem 8.5)

$$O(n \log n)$$

sorting

$$O(n + k \log n)$$

$$O(n + k \log k)$$

## The Largest $k$ Elements (Problem 8.5)

$$O(n \log n)$$

sorting

$$O(n + k \log n)$$

max-heap

$$O(n + k \log k)$$

## The Largest $k$ Elements (Problem 8.5)

$$O(n \log n)$$

sorting

$$O(n + k \log n)$$

max-heap

$$O(n + k \log k)$$

$k$ -selection + **partition** + sorting

## Selecting $k$ Elements Close to the Median (Problem 8.6)

$$S = \{800, 6, 900, 50, 7\}, \quad k = 2 \implies \{6, 7\}$$

## Selecting $k$ Elements Close to the Median (Problem 8.6)

$$S = \{800, 6, 900, 50, 7\}, \quad k = 2 \implies \{6, 7\}$$

$$O(n \log n + k)$$

$$O(n + k \log k)$$

## Selecting $k$ Elements Close to the Median (Problem 8.6)

$$S = \{800, 6, 900, 50, 7\}, \quad k = 2 \implies \{6, 7\}$$

$$O(n \log n + k)$$

sorting +

$$O(n + k \log k)$$



## Selecting $k$ Elements Close to the Median (Problem 8.6)

$$S = \{800, 6, 900, 50, 7\}, \quad k = 2 \implies \{6, 7\}$$

$$O(n \log n + k)$$

sorting + left-right iteration

$$O(n + k \log k)$$

## Selecting $k$ Elements Close to the Median (Problem 8.6)

$$S = \{800, 6, 900, 50, 7\}, \quad k = 2 \implies \{6, 7\}$$

$$O(n \log n + k)$$

sorting + left-right iteration

$$O(n + k \log k)$$

median-selection +

## Selecting $k$ Elements Close to the Median (Problem 8.6)

$$S = \{800, 6, 900, 50, 7\}, \quad k = 2 \implies \{6, 7\}$$

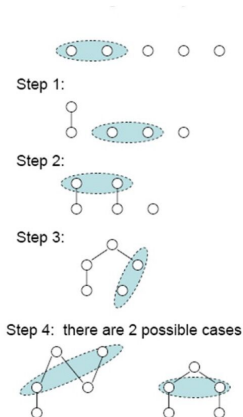
$$O(n \log n + k)$$

sorting + left-right iteration

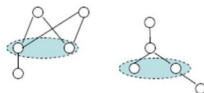
$$O(n + k \log k)$$

median-selection + the smallest  $k$  elements

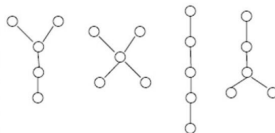
# Selecting the Median of 5 Elements using 6 Comparisons (Problem 8.2)



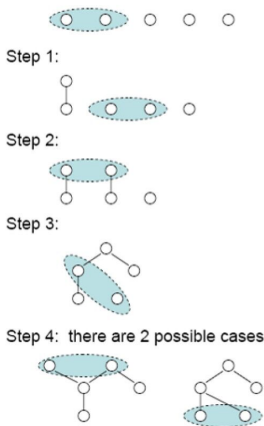
Step 5: there are 2 possible cases



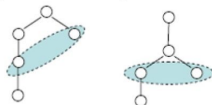
Step 6: there are 4 possible cases, done



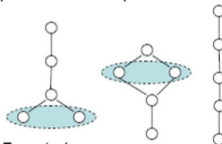
# Sorting 5 Elements using 7 Comparisons



Step 5: there are 2 possible cases



Step 6: there are 3 possible cases



Step 7: sorted



*Amortized analysis* is  
an algorithm analysis technique for  
analyzing a sequence of operations  
irrespective of the input to show that  
the average cost per operation is small, even though  
a single operation within the sequence might be expensive.

## The Summation Method

$$o_1, o_2, \dots, o_n$$

$$c_1, c_2, \dots, c_n$$

## The Summation Method

$$o_1, o_2, \dots, o_n$$

$$c_1, c_2, \dots, c_n$$

$$\forall i, \hat{c}_i = \frac{\left( \sum_{i=1}^n c_i \right)}{n}$$



# The Summation Method for Array Doubling

## The Summation Method for Array Doubling

On **any sequence** of  $n$  INSERTs on an **initially empty** array.

## The Summation Method for Array Doubling

On **any sequence** of  $n$  INSERTs on an **initially empty** array.

$o_i :$	$o_1$	$o_2$	$o_3$	$o_4$	$o_5$	$o_6$	$o_7$	$o_8$	$o_9$	$o_{10}$
$c_i :$	1	2	3	1	5	1	1	1	9	1

## The Summation Method for Array Doubling

On **any sequence** of  $n$  INSERTs on an **initially empty** array.

$o_i :$	$o_1$	$o_2$	$o_3$	$o_4$	$o_5$	$o_6$	$o_7$	$o_8$	$o_9$	$o_{10}$
$c_i :$	1	2	3	1	5	1	1	1	9	1

$$c_i = \begin{cases} (i-1) + 1 = i & \text{if } i-1 \text{ is an exact power of 2} \\ 1 & \text{o.w.} \end{cases}$$

## The Summation Method for Array Doubling

On **any sequence** of  $n$  INSERTs on an **initially empty** array.

$$\begin{array}{cccccccccc} o_i : & o_1 & o_2 & o_3 & o_4 & o_5 & o_6 & o_7 & o_8 & o_9 & o_{10} \\ c_i : & 1 & 2 & 3 & 1 & 5 & 1 & 1 & 1 & 9 & 1 \end{array}$$

$$c_i = \begin{cases} (i-1) + 1 = i & \text{if } i-1 \text{ is an exact power of 2} \\ 1 & \text{o.w.} \end{cases}$$

$$\sum_{i=1}^n c_i \leq n + \sum_{j=0}^{\lceil \lg n \rceil - 1} 2^j = n + (2^{\lceil \lg n \rceil} - 1) \leq n + 2n = 3n$$

## The Summation Method for Array Doubling

On **any sequence** of  $n$  INSERTs on an **initially empty** array.

$$\begin{array}{cccccccccc} o_i : & o_1 & o_2 & o_3 & o_4 & o_5 & o_6 & o_7 & o_8 & o_9 & o_{10} \\ c_i : & 1 & 2 & 3 & 1 & 5 & 1 & 1 & 1 & 9 & 1 \end{array}$$

$$c_i = \begin{cases} (i-1) + 1 = i & \text{if } i-1 \text{ is an exact power of 2} \\ 1 & \text{o.w.} \end{cases}$$

$$\sum_{i=1}^n c_i \leq n + \sum_{j=0}^{\lceil \lg n \rceil - 1} 2^j = n + (2^{\lceil \lg n \rceil} - 1) \leq n + 2n = 3n$$

$$\boxed{\forall i, \hat{c}_i = 3}$$

# The Accounting Method

$$o_1, o_2, \dots, o_n$$

$$c_1, c_2, \dots, c_n$$

$$a_1, a_2, \dots, a_n$$

## The Accounting Method

$$o_1, o_2, \dots, o_n$$

$$c_1, c_2, \dots, c_n$$

$$a_1, a_2, \dots, a_n$$

$$\hat{c}_i = c_i + a_i, \quad a_i \geq 0$$

Amortized Cost = Actual Cost + Accounting Cost



## The Accounting Method

$$O_1, O_2, \dots, O_n$$

$$C_1, C_2, \dots, C_n$$

$$a_1, a_2, \dots, a_n$$

$$\hat{c}_i = c_i + a_i, a_i \geq 0$$

Amortized Cost = Actual Cost + Accounting Cost

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

## The Accounting Method

$$O_1, O_2, \dots, O_n$$

$$C_1, C_2, \dots, C_n$$

$$a_1, a_2, \dots, a_n$$

$$\hat{c}_i = c_i + a_i, a_i \geq 0$$

Amortized Cost = Actual Cost + Accounting Cost

$$\forall n, \sum_{i=1}^n c_i \leq \sum_{i=1}^n \hat{c}_i \implies \forall n, \sum_{i=1}^n a_i \geq 0$$

## The Accounting Method

$$O_1, O_2, \dots, O_n$$

$$c_1, c_2, \dots, c_n$$

$$a_1, a_2, \dots, a_n$$

$$\hat{c}_i = c_i + a_i, a_i \geq 0$$

Amortized Cost = Actual Cost + Accounting Cost

$$\forall n, \sum_{i=1}^n c_i \leq \sum_{i=1}^n \hat{c}_i \implies \forall n, \sum_{i=1}^n a_i \geq 0$$

Key Point: Put the accounting cost on specific objects.

## The Accounting Method for Array Doubling

$$Q : \hat{c}_i = 3 \text{ vs. } \hat{c}_i = 2$$

## The Accounting Method for Array Doubling

$Q : \hat{c}_i = 3$  vs.  $\hat{c}_i = 2$

$$\hat{c}_i = 3 = \underbrace{1}_{\text{insert}} + \underbrace{1}_{\text{move itself}} + \underbrace{1}_{\text{help move another}}$$

## The Accounting Method for Array Doubling

$Q : \hat{c}_i = 3 \text{ vs. } \hat{c}_i = 2$

$$\hat{c}_i = 3 = \underbrace{1}_{\text{insert}} + \underbrace{1}_{\text{move itself}} + \underbrace{1}_{\text{help move another}}$$

	$\hat{c}_i$	$c_i$ (actual cost)	$a_i$ (accounting cost)
INSERT ( <i>normal</i> )	3	1	2
INSERT ( <i>expansion</i> )	3	$1 + t$	$-t + 2$

## Simulating a queue $Q$ using two stacks $S_1, S_2$ (Problem E3)

---

---

**procedure** ENQ( $x$ )

*Push*( $S_1, x$ )

**procedure** DEQ()

**if**  $S_2 = \emptyset$  **then**

**while**  $S_1 \neq \emptyset$  **do**

*Push*( $S_2, \text{Pop}(S_1)$ )

*Pop*( $S_2$ )

---

## The Summation Method for Queue Simulation

$$\frac{\left( \sum_{i=1}^n c_i \right)}{n}$$



## The Summation Method for Queue Simulation

$$\frac{\left( \sum_{i=1}^n c_i \right)}{n}$$

The operation sequence is *NOT* known.

## The Accounting Method for Queue Simulation

<i>item:</i>	Push into $S_1$	Pop from $S_1$	Push into $S_2$	Pop from $S_2$
	1	1	1	1

## The Accounting Method for Queue Simulation

<i>item:</i>	Push into $S_1$	Pop from $S_1$	Push into $S_2$	Pop from $S_2$
	1	1	1	1

$$\hat{c}_{\text{ENQ}} = 3$$

$$\hat{c}_{\text{DEQ}} = 1$$

## The Accounting Method for Queue Simulation

<i>item:</i>	Push into $S_1$	Pop from $S_1$	Push into $S_2$	Pop from $S_2$
	1	1	1	1

$$\hat{c}_{\text{ENQ}} = 3$$

$$\hat{c}_{\text{DEQ}} = 1$$

$$\sum_{i=1}^n a_i \geq 0$$

## The Accounting Method for Queue Simulation

*item:*    Push into  $S_1$     Pop from  $S_1$     Push into  $S_2$     Pop from  $S_2$   
                 1                    1                    1                    1

$$\hat{c}_{\text{ENQ}} = 3$$

$$\hat{c}_{\text{DEQ}} = 1$$

$$\sum_{i=1}^n a_i \geq 0 \iff \sum_{i=1}^n a_i = \#S_1 \times 2$$

## The Accounting Method for Queue Simulation

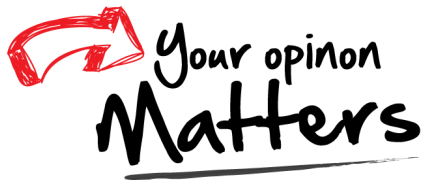
$$\hat{c}_{\text{ENQ}} = 3$$

$$\hat{c}_{\text{DEQ}} = 1$$

$$\#S_1 = t$$

	$\hat{c}_i$	$c_i$ ( <i>actual cost</i> )	$a_i$ ( <i>accounting cost</i> )
ENQUEUE	3	1	2
DEQUEUE ( $S_2 = \emptyset$ )	1	1	0
DEQUEUE ( $S_2 \neq \emptyset$ )	1	$1 + 2t$	$-2t$

Thank  
You!



Office 302

Mailbox: H016

hfwei@nju.edu.cn