Sorting, Searching, Selection, and Amortized Analysis

Hengfeng Wei

hfwei@nju.edu.cn

April 24, 2018

Maximal-sum Subarray (Problem 3.7)

- Array $A[1 \cdots n], a_i >=< 0$
- 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

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

$$\boxed{\mathsf{mss} = \max_{1 \le i \le n} \mathsf{MSS}[i]}$$

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

$$\boxed{\mathsf{mss} = \max_{1 \le i \le n} \mathsf{MSS}[i]}$$

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

Q : Where does the $\mathsf{MSS}[i]$ start?

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

$$\boxed{\text{mss} = \max_{1 \le i \le n} \text{MSS}[i]}$$

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

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

$$\text{MSS}[0] = 0$$

```
procedure MSS(A[1 ··· n])
    MSS[0] ← 0
    for i ← 1 to n do
        MSS[i] ← max {MSS[i − 1] + A[i], 0}
    return max_{1≤i≤n} MSS[i]
```

procedure $\mathrm{MSS}(A[1 \cdots n])$

  $\textsf{MSS}[0] \leftarrow 0$

  for $i \leftarrow 1$ **to** n **do**

    $\textsf{MSS}[i] \leftarrow \max \{\textsf{MSS}[i-1] + A[i], 0\}$

  return $\max\limits_{1 \le i \le n} \textsf{MSS}[i]$

---

procedure $\mathrm{MSS}(A[1 \cdots n])$

  $\textsf{mss} \leftarrow 0$

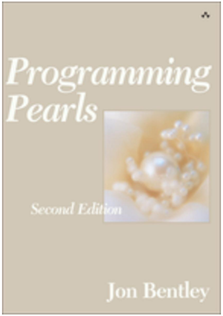
  $\textsf{MSS} \leftarrow 0$

  for $i \leftarrow 1$ **to** n **do**

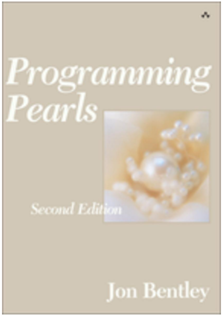
    $\textsf{MSS} \leftarrow \max \{\textsf{MSS} + A[i], 0\}$

    $\textsf{mss} \leftarrow \max \{\textsf{mss}, \textsf{MSS}\}$

  return $\textsf{mss}$

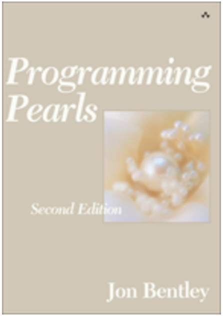


Ulf Grenander $O(n^3) \implies O(n^2)$



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

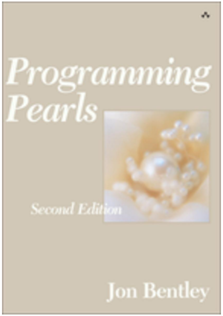
Michael Shamos $O(n \log n)$, onenight



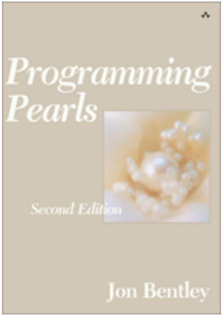
Ulf Grenander $O(n^3) \implies O(n^2)$

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

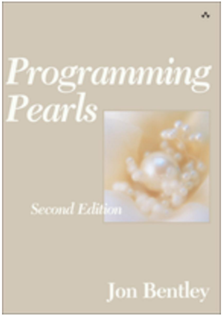
Jon Bentley Conjecture: $\Omega(n \log n)$



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	$O(n)$, ≤ 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.

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

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

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

$$1\text{-sorted} \rightarrow 2\text{-sorted} \rightarrow 4\text{-sorted} \rightarrow \cdots \rightarrow n\text{-sorted}$$

k-sorted

$$1\text{-sorted} \to 2\text{-sorted} \to 4\text{-sorted} \to \cdots \to n\text{-sorted}$$

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

k-sorted

1-sorted $\to$ 2-sorted $\to$ 4-sorted $\to \cdots \to$ n-sorted

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

$$\boxed{\Theta(n \log k)}$$

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

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

$$L =$$

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

$$L = \binom{n}{n/k, \ldots, n/k}$$

$\Omega(n \log k)$

$$L = \binom{n}{n/k, \ldots, n/k} = \frac{n!}{\left(\left(\frac{n}{k}\right)!\right)^k}$$

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

$$L = \binom{n}{n/k, \ldots, n/k} = \frac{n!}{\left((\frac{n}{k})!\right)^k}$$

$$H \geq$$

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

$$L = \binom{n}{n/k, \ldots, n/k} = \frac{n!}{\left((\frac{n}{k})!\right)^k}$$

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

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

$$L = \binom{n}{n/k, \ldots, n/k} = \frac{n!}{\left((\frac{n}{k})!\right)^k}$$

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

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

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)

$$R[1 \ldots n]$$

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

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

check$(R[i], R[j])$

Repeated Elements (Problem 2.12)

$$R[1 \dots n]$$

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

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

check$(R[i], R[j])$

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

Repeated Elements (Problem 2.12)

$$R[1 \dots n]$$

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

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

check$(R[i], R[j])$

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

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

$\implies x$ is a $\dfrac{n}{13}$-repeated element of $R[1 \cdots \frac{n}{2}]$ or $R[\frac{n}{2} + 1 \cdots n]$

Divide and Conquer Algorithm

Divide and Conquer Algorithm

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

Divide and Conquer Algorithm

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

$Q : 13 \to k$

Divide and Conquer Algorithm

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

$Q : 13 \to k$

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

Divide and Conquer Algorithm

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

$Q : k \leftarrow 2$

$Q : 13 \rightarrow k$

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

Divide and Conquer Algorithm

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

$Q : k \leftarrow 2$

$Q : 13 \rightarrow k$

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

Divide and Conquer Algorithm

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

$Q : k \leftarrow 2$

$Q : 13 \rightarrow k$

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

$k : O(n \log k)$

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

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

$Q : \exists$ a repeated element?

$\Omega(n \log k)$

$Q : \exists$ a repeated element?

$L : \#$ of leaves?

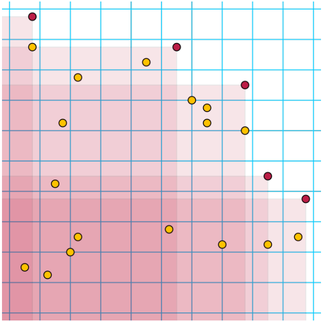
$$\Omega(n \log k)$$

$Q : \exists$ a repeated element?

$L : \#$ of leaves?

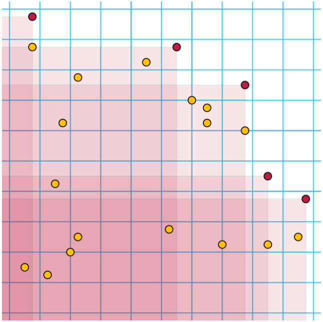
"Finding Repeated Elements" by Misra & Gries, 1982

Maxima of a Point Set (Problem 6.15)



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

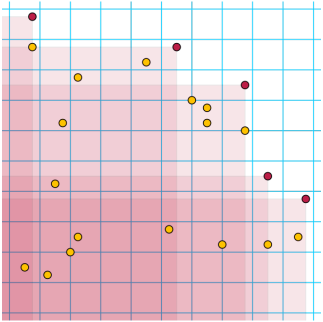
Maxima of a Point Set (Problem 6.15)



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

x-sorting, $\quad \max y$

Maxima of a Point Set (Problem 6.15)



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

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

Wrong Divide and Conquer Algorithms

x-median & y-median

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

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

Divide and Conquer Algorithm

x-median

Divide and Conquer Algorithm

x-median

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

Divide and Conquer Algorithm

x-median

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

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

Smallest Missing Positive Integer (Problem 9.6)

Sorted array $A[1 \ldots n]$

$$a_i \in \mathbb{Z}^+$$
$$\forall i \neq j : a_i \neq a_j$$

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

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 $A[1 \ldots n]$

$$a_i \in \mathbb{Z}^+$$
$$\forall i \neq j : a_i \neq a_j$$

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

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

$$O(\log n)$$

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

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)

$$A[1 \cdots n]$$

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

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

$$\exists$$

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

Local Minimum (Problem 9.12)

$$A[1 \cdots n]$$

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

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

$$\exists$$

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

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

Local Minimum (Problem 9.12)

$$A[1 \cdots n]$$

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

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

Local Minimum (Problem 9.12)

$$A[1 \cdots n]$$

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

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

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

Local Minimum (Problem 9.12)

$$A[1 \cdots n]$$

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

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

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

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

Local Minimum (Problem 9.12)

$$A[1 \cdots n]$$

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

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

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

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

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

Local Minimum (Problem 9.12)

$$A[1 \cdots n]$$

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

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

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

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

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

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

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

Local Minimum (Problem 9.12)

$$A[1 \cdots n]$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Always checking the lower left corner.

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

Assume $M : n \times n$

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

Assume $M : n \times n$

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

$$\boxed{W(n) \geq 2n - 1}$$

Assume $M : n \times n$

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

$$\boxed{W(n) \geq 2n - 1}$$

By Adversary Argument!

Assume $M : n \times n$

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

$$\boxed{W(n) \geq 2n - 1}$$

By Adversary Argument!

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

Assume $M : n \times n$

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

$$\boxed{W(n) \geq 2n - 1}$$

By Adversary Argument!

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

No particular ordering requirements on these two diagonals!

Assume $M : n \times n$

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

$$\boxed{W(n) \geq 2n - 1}$$

By Adversary Argument!

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

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)

$$\textit{Sorted } S[1 \cdots n], \quad c$$

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

$$O(n)$$

$A + B = c$ (Problem 9.9)

$$\textit{Sorted } S[1 \cdots n], \quad c$$

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

$$O(n)$$

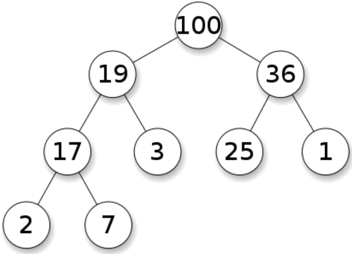
Left-right-pointers iteration

$$S_i + S_j >=< c$$

Selection

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

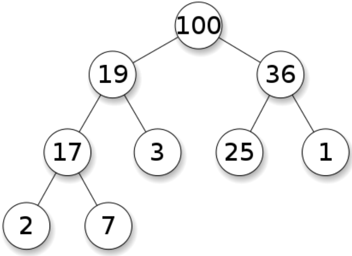
$$k \ll n$$



$$T(k)$$

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

$$k \ll n$$

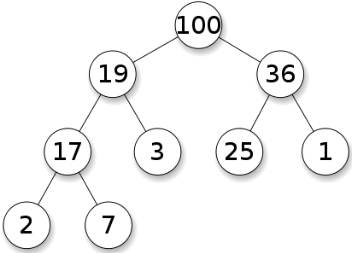


$$\color{red}{T(k)}$$

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 +

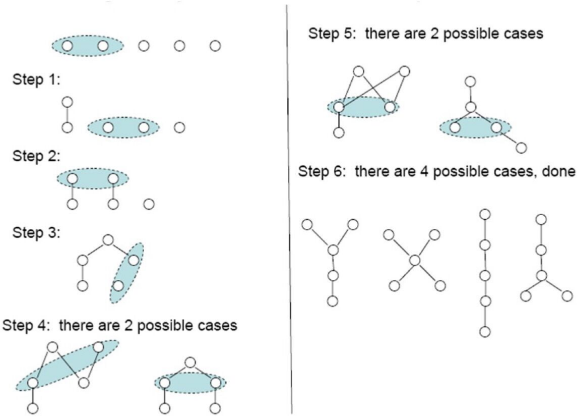
Selecting k Elements Close to the Median (Problem 8.6)

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

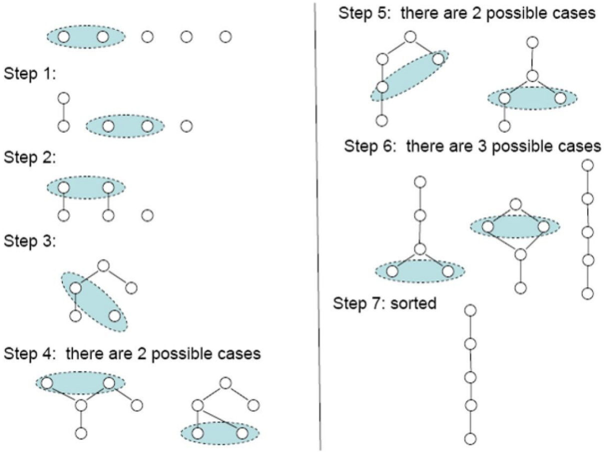
$O(n \log n + k)$ sorting + left-right iteration

$O(n + k \log k)$ median-selection + the smallest k elements

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



Sorting 5 Elements using 7 Comparisons



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

$$o_1, o_2, \ldots, o_n$$

$$c_1, c_2, \ldots, c_n$$

The Summation Method

$$o_1, o_2, \ldots, o_n$$

$$c_1, c_2, \ldots, c_n$$

$$\boxed{\forall i,\ \hat{c_i} = \frac{\left(\sum\limits_{i=1}^{n} c_i\right)}{n}}$$

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.

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

The Summation Method for Array Doubling

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

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

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

The Summation Method for Array Doubling

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

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

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

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

The Summation Method for Array Doubling

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

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

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

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

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

The Accounting Method

$$o_1, o_2, \ldots, o_n$$

$$c_1, c_2, \ldots, c_n$$

$$a_1, a_2, \ldots, a_n$$

The Accounting Method

$$o_1, o_2, \ldots, o_n$$

$$c_1, c_2, \ldots, c_n$$

$$a_1, a_2, \ldots, a_n$$

$$\boxed{\hat{c_i} = c_i + a_i,\ a_i >=< 0}$$

Amortized Cost $=$ Actual Cost $+$ Accounting Cost

The Accounting Method

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

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

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

$$\boxed{\hat{c}_i = c_i + a_i, \; a_i >=< 0}$$

Amortized Cost $=$ Actual Cost $+$ Accounting Cost

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

The Accounting Method

$$o_1, o_2, \ldots, o_n$$

$$c_1, c_2, \ldots, c_n$$

$$a_1, a_2, \ldots, a_n$$

$$\boxed{\hat{c}_i = c_i + a_i, \; a_i >=< 0}$$

Amortized Cost $=$ Actual Cost $+$ Accounting Cost

$$\forall n, \; \sum_{i=1}^{n} c_i \le \sum_{i=1}^{n} \hat{c}_i \implies \boxed{\forall n, \; \sum_{i=1}^{n} a_i \ge 0}$$

The Accounting Method

$$o_1, o_2, \ldots, o_n$$

$$c_1, c_2, \ldots, c_n$$

$$a_1, a_2, \ldots, a_n$$

$$\boxed{\hat{c}_i = c_i + a_i, \; a_i >=< 0}$$

Amortized Cost $=$ Actual Cost $+$ Accounting Cost

$$\forall n, \; \sum_{i=1}^{n} c_i \le \sum_{i=1}^{n} \hat{c}_i \implies \boxed{\forall n, \; \sum_{i=1}^{n} a_i \ge 0}$$

Key Point: Put the accounting cost on specific objects.

The Accounting Method for Array Doubling

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

The Accounting Method for Array Doubling

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

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

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 *(normal)*	3	1	2
INSERT *(expansion)*	3	$1 + t$	$-t + 2$

Simulating a queue Q using two stacks S_1, S_2 (Problem $\mathbb{E}3$)

procedure $\textsc{Enq}(x)$
  Push(S_1, x)

procedure $\textsc{Deq}()$
  if $S_2 = \emptyset$ **then**
    while $S_1 \neq \emptyset$ **do**
      Push$(S_2,$ *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

item:	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

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

$$\hat{c}_{\text{ENQ}} = 3$$
$$\hat{c}_{\text{DEQ}} = 1$$

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

$$\hat{c}_{\text{ENQ}} = 3$$
$$\hat{c}_{\text{DEQ}} = 1$$

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

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

$$\hat{c}_{\text{ENQ}} = 3$$
$$\hat{c}_{\text{DEQ}} = 1$$

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

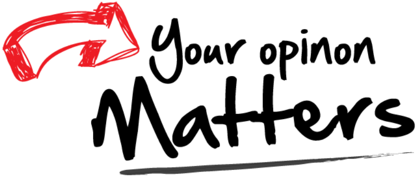
The Accounting Method for Queue Simulation

$$\hat{c}_{\text{ENQ}} = 3$$
$$\hat{c}_{\text{DEQ}} = 1$$

$$\#S_1 = t$$

	$\hat{c}_i$	c_i *(actual cost)*	a_i *(accounting cost)*
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