Sorting, Searching, 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

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

MSS$[i]$: the sum of the MS (MS$[i]$) in $A[1 \cdots i]$

$\mathsf{MSS}[i]$: the sum of the MS ($\mathsf{MS}[i]$) in $A[1 \cdots i]$

$$\mathsf{mss} = \mathsf{MSS}[n]$$

$\mathsf{MSS}[i]$: the sum of the MS ($\mathsf{MS}[i]$) in $A[1 \cdots i]$

$$\mathsf{mss} = \mathsf{MSS}[n]$$

Q : Is $a_i \in \mathsf{MS}[i]$?

$\mathsf{MSS}[i]$: the sum of the MS ($\mathsf{MS}[i]$) in $A[1 \cdots i]$

$$\mathsf{mss} = \mathsf{MSS}[n]$$

Q : Is $a_i \in \mathsf{MS}[i]$?

$$\mathsf{MSS}[i] = \max\{\mathsf{MSS}[i-1], ???\}$$

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

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

Q : where does the $\mathsf{MS}[i]$ start?

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

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

Q : where does the MS$[i]$ start?

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

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

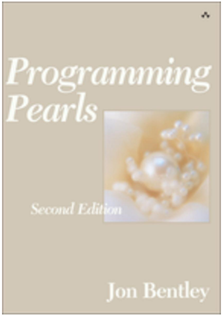
$$\boxed{\mathsf{mss} = \max_{1 \le i \le n} \mathsf{MSS}[i]}$$

Q : where does the $\mathsf{MS}[i]$ start?

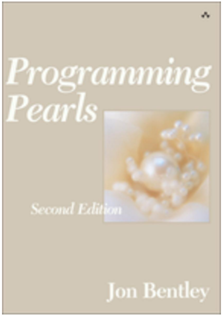
$$\boxed{\mathsf{MSS}[i] = \max\left\{\mathsf{MSS}[i-1] + a_i, 0\right\}}$$

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

```
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]
```

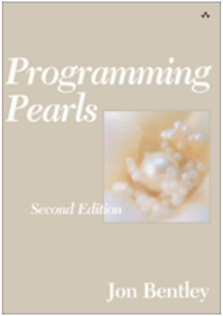


Ulf Grenander $O(n^3) \implies O(n^2)$



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

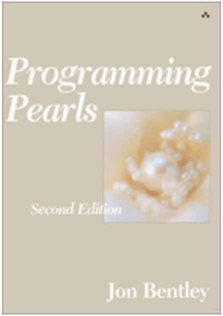
Michael Shamos $O(n \log n)$, onenight



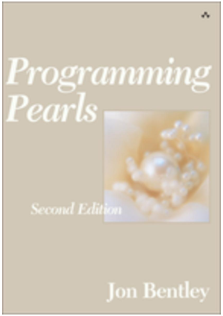
Ulf Grenander $O(n^3) \implies O(n^2)$

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

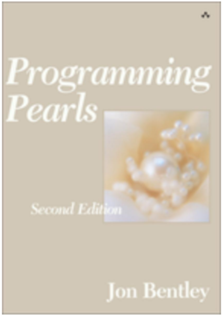
Jon Bentley Conjecture: $\Omega(n \log n)$



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

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 $\to$ 2-sorted

k-sorted

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

k-sorted

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

k-sorted

1-sorted $\rightarrow$ 2-sorted $\rightarrow$ 4-sorted $\rightarrow \cdots \rightarrow$ n-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.

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

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

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

$$L =$$

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

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

$\Omega(n \log k)$

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

$\Omega(n \log k)$

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

$$H \geq$$

$\Omega(n \log k)$

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

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

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

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

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

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

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, $\Theta(n \log n)$

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

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

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

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

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

Repeated elements (Problem 2.12)

$$R[1 \ldots n]$$

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

Repeated elements (Problem 2.12)

$$R[1 \ldots n]$$

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

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

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

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

An $O(n \log k)$ algorithm

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

An $O(n \log k)$ algorithm

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

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

An $O(n \log k)$ algorithm

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

L : $\#$ of leaves?

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

An $O(n \log k)$ algorithm

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

L : $\#$ of leaves?

"Finding Repeated Elements" by Misra & Gries, 1982

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.

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

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

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

The Summation Method for Array Doubling

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

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

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

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

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

The Accounting Method

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

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

$$\color{red}{a_1, a_2, \ldots, a_n}$$

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

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

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

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

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

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

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

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

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

The Accounting Method for Array Doubling

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

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

The Accounting Method for Array Doubling

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

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

	$\hat{c}_i$	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)

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

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

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

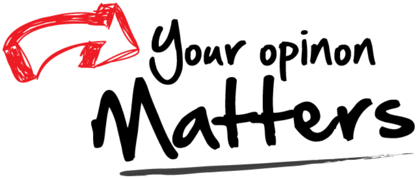
The Accounting Method for Queue Simulation

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

$$\#S_1 = t$$

	$\hat{c}_i$	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