

Algorithm and Data Structure Analysis (ADSA)

Lecture 6: Order Statistics

Review: Sorting Algorithms

Comparison Sorts

- Comparison sorts: $O(n \lg n)$ at best (decision tree with $n!$ leaves $\rightarrow O(n \lg n)$ height)
- Counting sort: $O(n+k) = O(n)$ for n inputs in the range $1..k$ ($k=O(n)$)
- Radix sort: $O(dn+dk) = O(n)$ for n numbers on d digits that range from $1..k$ (constant d , $k=O(n)$)
- Bucket sort:
 - Use n buckets (linked lists) to divide interval $[0,1)$ (range $k=O(n)$) into subintervals of size $1/n$ (k/n)
 - Uniform input distribution $\rightarrow O(1)$ bucket size \rightarrow expected total time $O(n)$

Order Statistics

- The i th *order statistic* in a set of n elements is the i th smallest element
- The *minimum* is thus the 1st order statistic
- The *maximum* is (duh) the n th order statistic
- The *median* is the $n/2$ order statistic
 - If n is even, there are 2 medians
- *How can we calculate order statistics?*
- *What is the running time?*

Order Statistics

- *How many comparisons are needed to find the minimum element in a set? The maximum?*

```
MINIMUM( $A, n$ )  
1   $min = A[1]$   
2  for  $i = 2$  to  $n$   
3      if  $min > A[i]$   
4           $min = A[i]$   
5  return  $min$ 
```

There is a lower bound of $n-1$ comparisons for this problem

Order Statistics

- *How many comparisons are needed to find the minimum element in a set? The maximum?*
- *Can we simultaneously find the minimum and maximum with less than twice the cost?*

Order Statistics

- *How many comparisons are needed to find the minimum element in a set? The maximum?*
- *Can we simultaneously find the minimum and maximum with less than twice the cost?*
- Yes:
 - Walk through elements **by pairs**
 - Compare each element in pair to the other
 - Compare the largest to maximum, smallest to minimum
 - Total cost: 3 comparisons per 2 elements = $O(3n/2)$

Finding Order Statistics: The Selection Problem

- A more interesting problem is *selection*:
finding the i th smallest element of a set

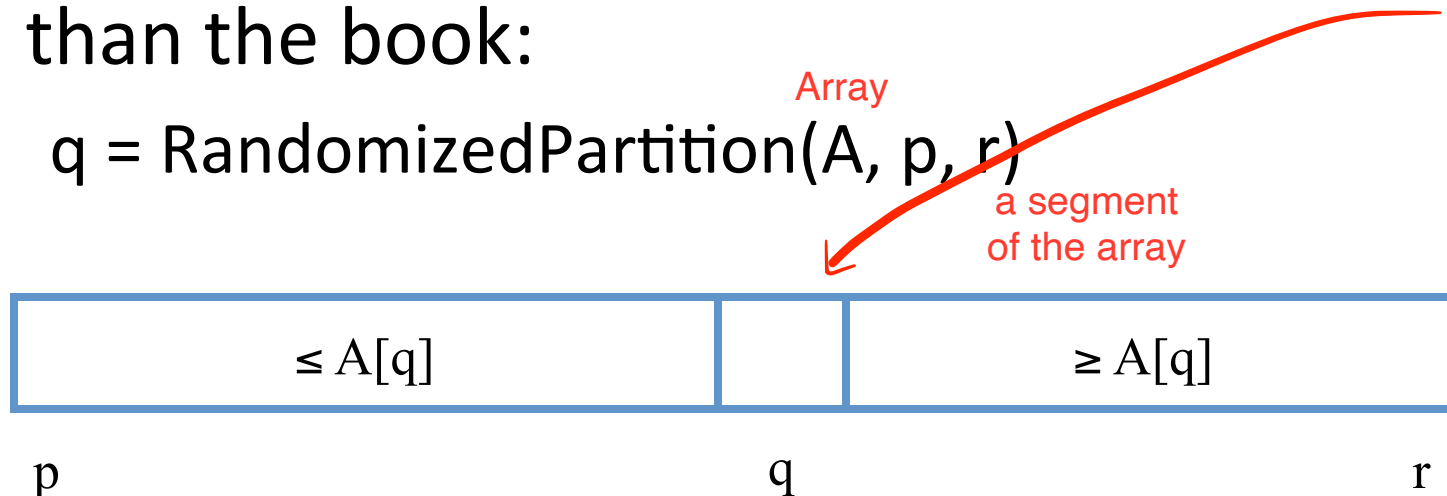
Finding Order Statistics: The Selection Problem

- A more interesting problem is *selection*: finding the i th smallest element of a set
- We will show:
 - A practical randomized algorithm with $O(n)$ expected running time
 - A cool algorithm of theoretical interest only with $O(n)$ worst-case running time

Randomized Selection

- Key idea: use `partition()` from quicksort
 - But, only need to examine one subarray
 - This savings shows up in running time: $O(n)$
- We will again use a slightly different partition than the book:

$q = \text{RandomizedPartition}(A, p, r)$

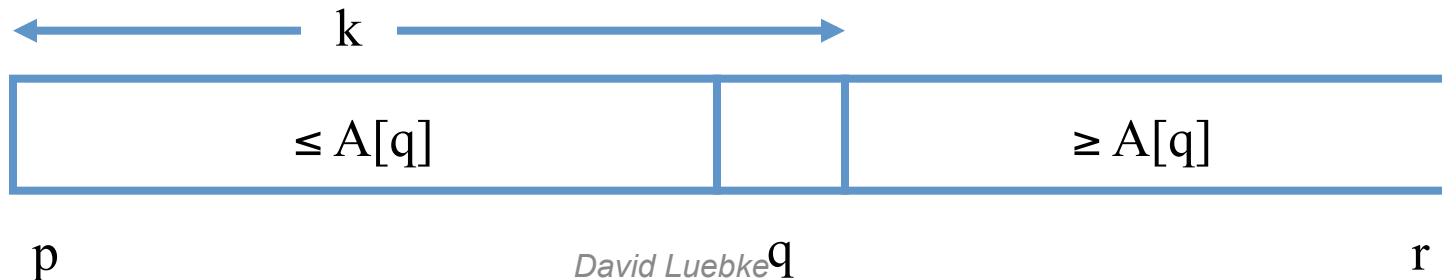


Randomized Selection

```

RandomizedSelect(A, p, r, i) start end array index index smallest segment among all the segments
    if (p == r) then return A[p];
    q = RandomizedPartition(A, p, r)
    k = q - p + 1;
    if (i == k) then return A[q];    // not in book
    if (i < k) then
        return RandomizedSelect(A, p, q-1, i);
    else
        return RandomizedSelect(A, q+1, r, i-k);

```



Randomized Selection

- Analyzing **RandomizedSelect()**
 - Worst case: partition always **0:n-1**

Randomized Selection

- Analyzing **RandomizedSelect()**
 - Worst case: partition always 0:n-1
$$T(n) = T(n-1) + O(n)$$
$$= O(n^2)$$
 - No better than sorting!

Randomized Selection

- Analyzing **RandomizedSelect()**
 - Worst case: partition always 0:n-1
$$T(n) = T(n-1) + O(n)$$
$$= O(n^2)$$
 - No better than sorting!
 - “Best” case: suppose a 9:1 partition

Randomized Selection

- Analyzing **RandomizedSelect()**
 - Worst case: partition always 0:n-1
$$T(n) = T(n-1) + O(n)$$
$$= O(n^2)$$
 - No better than sorting!
 - “Best” case: suppose a 9:1 partition
$$T(n) = T(9n/10) + O(n)$$

Randomized Selection

- Analyzing **RandomizedSelect()**
 - Worst case: partition always 0:n-1
$$T(n) = T(n-1) + O(n)$$
$$= O(n^2)$$
 - No better than sorting!
 - “Best” case: suppose a 9:1 partition
$$T(n) = T(9n/10) + O(n)$$
$$= O(n) \quad (\text{Master Theorem, case 3})$$
 - Better than sorting!

Randomized Selection

- Analyzing **RandomizedSelect()**
 - Worst case: partition always **0:n-1**
$$T(n) = T(n-1) + O(n)$$
$$= O(n^2)$$
 - No better than sorting!
 - “Best” case: suppose a **9:1 partition**
$$T(n) = T(9n/10) + O(n)$$
$$= O(n) \quad (\text{Master Theorem, case 3})$$
 - Better than sorting!
 - *What if this had been a 99:1 split?* still linear time

Randomized Selection

- Average case

- For upper bound, assume i th element always falls in larger side of partition:

when $k = 0$:

$$T(n-1)+T(n-2)+T(n-3)+T(n-4)+\dots T(n/2)$$

$$T(n) \leq \frac{1}{n} \sum_{k=0}^{n-1} T(\max(k, n-k-1)) + \Theta(n)$$

$$\dots T(n/2) \leq \frac{2}{n} \sum_{k=n/2}^{n-1} T(k) + \Theta(n)$$

What happened here?

- Let's show that $T(n) = O(n)$ by substitution

Randomized Selection

- Assume $T(n) \leq cn$ for sufficiently large c :

$$T(n) \leq \frac{2}{n} \sum_{k=n/2}^{n-1} T(k) + \Theta(n)$$

The recurrence we started with

$$\leq \frac{2}{n} \sum_{k=n/2}^{n-1} ck + \Theta(n)$$

Substitute $T(n) \leq cn$ for $T(k)$

$$= \frac{2c}{n} \left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{n/2-1} k \right) + \Theta(n)$$

“Split” the recurrence

$$= \frac{2c}{n} \left(\frac{1}{2}(n-1)n - \frac{1}{2} \left(\frac{n}{2} - 1 \right) \frac{n}{2} \right) + \Theta(n)$$

Expand arithmetic series

$$= c(n-1) - \frac{c}{2} \left(\frac{n}{2} - 1 \right) + \Theta(n)$$

Multiply it out

Randomized Selection

- Assume $T(n) \leq cn$ for sufficiently large c :

$$T(n) \leq c(n-1) - \frac{c}{2} \left(\frac{n}{2} - 1 \right) + \Theta(n) \quad \textit{The recurrence so far}$$

$$= cn - c - \frac{cn}{4} + \frac{c}{2} + \Theta(n) \quad \textit{Multiply it out}$$

$$= cn - \frac{cn}{4} - \frac{c}{2} + \Theta(n) \quad \textit{Subtract } c/2$$

$$= cn - \left(\frac{cn}{4} + \frac{c}{2} - \Theta(n) \right) \quad \textit{Rearrange the arithmetic}$$

$$\leq cn \quad (\text{if } c \text{ is big enough}) \quad \textit{What we set out to prove}$$

Worst-Case Linear-Time Selection

- Randomized algorithm works well in practice
- What follows is a worst-case linear time algorithm, really of theoretical interest only
- Basic idea:
 - Generate a good partitioning element
 - Call this element x

14	32	23	5	10	60	29
57	2	52	44	27	21	11
24	43	12	17	48	1	58
6	30	63	34	8	55	39
37	25	3	64	19	41	64

Group

6	2	3	5	8	1	11
14	25	12	17	10	21	29
24	30	23	34	19	41	39
37	32	52	44	27	55	58
57	43	63	64	48	60	64

Get group medians

8	3	6	2	5	11	1
10	12	14	25	17	29	21
19	23	24	30	34	39	41
27	52	37	32	44	58	55
48	63	57	43	64	64	60

Get median of medians

(Sorting of group medians is not really performed)

Figure 8: Choosing the Pivot. 30 is the final pivot.

14	32	23	5	10	60	29
57	2	52	44	27	21	11
24	43	12	17	48	1	58
6	30	63	34	8	55	39
37	25	3	64	19	41	64

Group

6	2	3	5	8	1	11
14	25	12	17	10	21	29
24	30	23	34	19	41	39
37	32	52	44	27	55	58
57	43	63	64	48	60	64

Get group medians

8	3	6	2	5	11	1
10	12	14	25	17	29	21
19	23	24	30	34	39	41
27	52	37	32	44	58	55
48	63	57	43	64	64	60

Get median of medians

(Sorting of group medians is not really performed)

Figure 8: Choosing the Pivot. 30 is the final pivot.

Worst-Case Linear-Time Selection

- The algorithm in words:
 1. Divide n elements into groups of 5
 2. Find median of each group (*How? How long?*)
 3. Use Select() recursively to find median x of the $\lfloor n/5 \rfloor$ medians
 4. Partition the n elements around x . Let $k = \text{rank}(x)$
 5. **if** ($i == k$) **then** return x
if ($i < k$) **then** use Select() recursively to find i th smallest element in first partition
else ($i > k$) use Select() recursively to find $(i-k)$ th smallest element in last partition

Worst-Case Linear-Time Selection

- (Sketch situation on the board)
- *How many of the 5-element medians are $\leq x$?*
 - At least $1/2$ of the medians = $\lfloor \lfloor n/5 \rfloor / 2 \rfloor = \lfloor n/10 \rfloor$
- *How many elements are $\leq x$?*
 - At least $3 \lfloor n/10 \rfloor$ elements
- For large n , $3 \lfloor n/10 \rfloor \geq n/4$ *(How large?)*
- So at least $n/4$ elements $\leq x$
- Similarly: at least $n/4$ elements $\geq x$

Worst-Case Linear-Time Selection

- Thus after partitioning around x , step 5 will call `Select()` on at most $3n/4$ elements

- The recurrence is therefore:

$$T(n) \leq T(\lfloor n/5 \rfloor) + T(3n/4) + \Theta(n)$$

$$\leq T(n/5) + T(3n/4) + \Theta(n) \quad \lfloor n/5 \rfloor \leq n/5$$

$$\leq cn/5 + 3cn/4 + \Theta(n) \quad \text{Substitute } T(n) = cn$$

$$= 19cn/20 + \Theta(n) \quad \text{Combine fractions}$$

$$= cn - (cn/20 - \Theta(n)) \quad \text{Express in desired form}$$

$$\leq cn \quad \text{if } c \text{ is big enough} \quad \text{What we set out to prove}$$

Worst-Case Linear-Time Selection

- Intuitively:
 - Work at each level is a constant fraction ($19/20$) smaller
 - Geometric progression!
 - Thus the $O(n)$ work at the root dominates

Linear-Time Median Selection

- Given a “black box” $O(n)$ median algorithm, what can we do?
 - i th order statistic:
 - Find median x
 - Partition input around x
 - if $(i \leq (n+1)/2)$ recursively find i th element of first half
 - else find $(i - (n+1)/2)$ th element in second half
 - $T(n) = T(n/2) + O(n) = O(n)$
 - *Can you think of an application to sorting?*

Linear-Time Median Selection

- Worst-case $O(n \lg n)$ quicksort
 - Find median x and partition around it
 - Recursively quicksort two halves
 - $T(n) = 2T(n/2) + O(n) = O(n \lg n)$

The End