Analysis of Algorithms
CS 477/677

Instructor: Monica Nicolescu

Lecture 10

Selection

- General Selection Problem:
  - select the i-th smallest element form a set of n distinct numbers
  - that element is larger than exactly $i - 1$ other elements
- The selection problem can be solved in $O(n\lg n)$ time
  - Sort the numbers using an $O(n\lg n)$-time algorithm, such as merge sort
  - Then return the i-th element in the sorted array

Medians and Order Statistics

*Def.: The i-th **order statistic** of a set of n elements is the i-th smallest element.*

- The minimum of a set of elements:
  - The first order statistic $i = 1$
- The maximum of a set of elements:
  - The n-th order statistic $i = n$
- The median is the "halfway point" of the set
  - $i = (n+1)/2$, is unique when n is odd
  - $i = \lfloor (n+1)/2 \rfloor = n/2$ (lower median) and $\lceil (n+1)/2 \rceil = n/2+1$ (upper median), when n is even

Finding Minimum or Maximum

Alg.: MINIMUM(A, n)

```
min ← A[1]
for i ← 2 to n
    do if min > A[i]
        then min ← A[i]
return min
```

- How many comparisons are needed?
  - n – 1: each element, except the minimum, must be compared to a smaller element at least once
  - The same number of comparisons are needed to find the maximum
  - The algorithm is optimal with respect to the number of comparisons performed

Simultaneous Min, Max

- Find min and max independently
  - Use n – 1 comparisons for each ⇒ total of **2n – 2**
- However, we can do better: at most **3n/2** comparisons
  - Process elements in pairs
  - Maintain the minimum and maximum of elements seen so far
  - Don't compare each element to the minimum and maximum separately
  - Compare the elements of a pair to each other
  - Compare the larger element to the maximum so far, and compare the smaller element to the minimum so far
  - This leads to only 3 comparisons for every 2 elements

Analysis of Simultaneous Min, Max

- Setting up initial values:
  - n is odd: set both **min** and **max** to the first element
  - n is even: compare the first two elements, assign the smallest one to **min** and the largest one to **max**

- Total number of comparisons:
  - n is odd: we do $3(n-1)/2$ comparisons
  - n is even: we do 1 initial comparison + $3(n-2)/2$ more comparisons = $3n/2 - 2$ comparisons

Example: Simultaneous Min, Max

- n = 5 (odd), array A = {2, 7, 1, 3, 4}

1. Set **min** = **max** = 2
2. Compare elements in pairs:
   - 1 < 7 ⇒ compare 1 with **min** and 7 with **max**

     ⇒ **min** = 1, **max** = 7

     3 comparisons
   - 3 < 4 ⇒ compare 3 with **min** and 4 with **max**

     ⇒ **min** = 1, **max** = 7

     3 comparisons

We performed: 3(n-1)/2 = 6 comparisons

Example: Simultaneous Min, Max

- n = 6 (even), array A = {2, 5, 3, 7, 1, 4}

1. Compare 2 with 5: 2 < 5 — 1 comparison
2. Set **min =** 2, **max** = 5
3. Compare elements in pairs:
   - 3 < 7 ⇒ compare 3 with **min** and 7 with **max**

     ⇒ **min** = 2, **max** = 7 — 3 comparisons
   - 1 < 4 ⇒ compare 1 with **min** and 4 with **max**

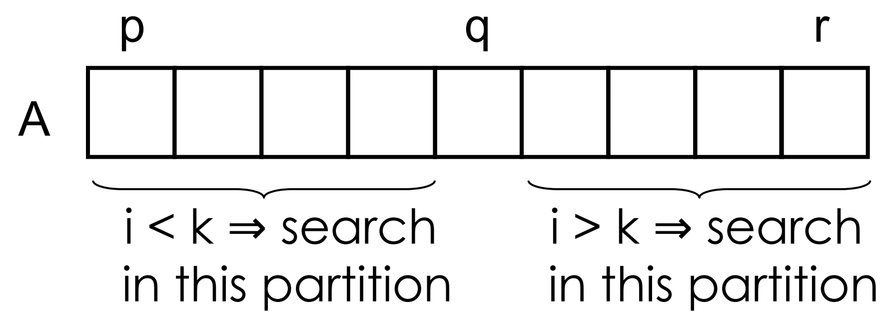
     ⇒ **min** = 1, **max** = 7 — 3 comparisons

We performed: 3n/2 - 2 = 7 comparisons

General Selection Problem

- Select the i-th order statistic (i-th smallest element) form a set of n distinct numbers

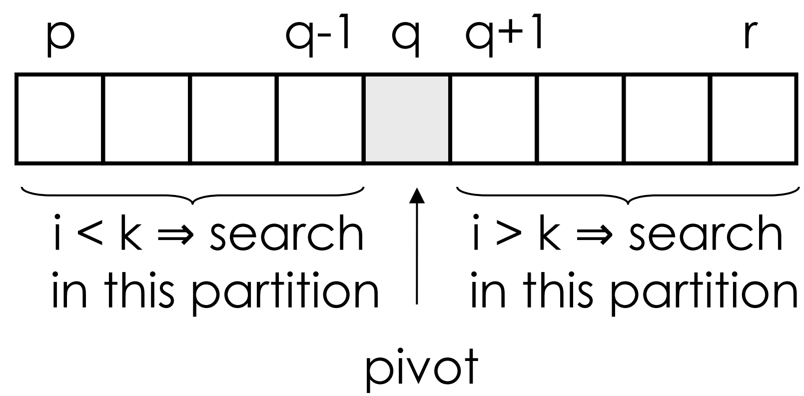
$k = q - p + 1$



- Idea:
  - Partition the input array similarly with the approach used for Quicksort (use RANDOMIZED-PARTITION)
  - Recurse on one side of the partition to look for the i-th element depending on where i is with respect to the pivot

- We will show that selection of the **i**-th smallest element of the array *A* can be done in $\Theta(n)$ time

Randomized Select



Alg.: RANDOMIZED-SELECT(*A*, p, r, i)

if p = r

  then return *A*[p]

q ←RANDOMIZED-PARTITION(*A*, p, r)

k ← q - p + 1

if i = k   pivot value is the answer

  then return *A*[q]

elseif i < k

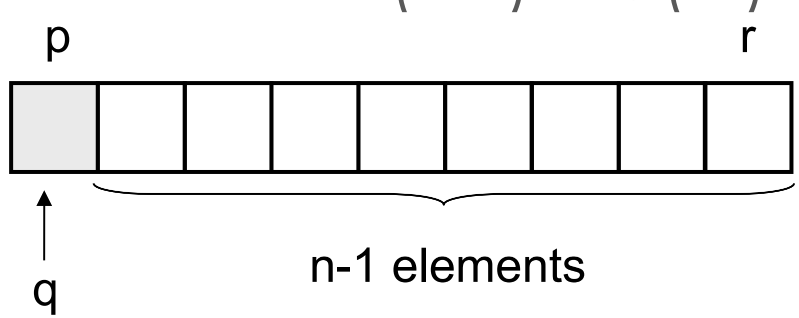
    then return RANDOMIZED-SELECT(*A*, p, q-1, i)

else return RANDOMIZED-SELECT(*A*, q + 1, r, i-k)

Analysis of Running Time

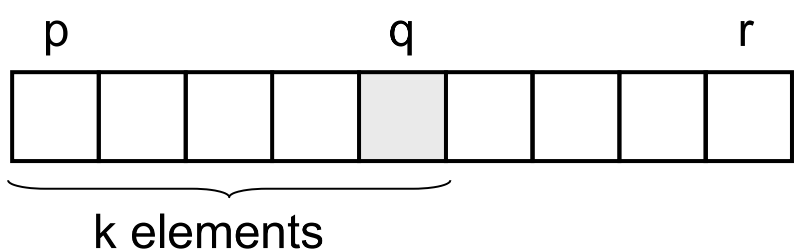
- **Worst case** running time: **$\Theta(n^2)$**
  - If we always partition around the largest/smallest remaining element
  - Partition takes $\Theta(n)$ time
  - $T(n) = \Theta(1)$ (compute k) $+ \Theta(n)$ (partition) $+ T(n-1)$

    $= 1 + n + T(n-1) = \Theta(n^2)$



Analysis of Running Time

- **Expected** running time (on **average**)
  - Let T(n) be a random variable denoting the running time of RANDOMIZED-SELECT



  - RANDOMIZED-PARTITION is equally likely to return any element of *A* as the pivot ⇒
  - For each k such that $1 \le k \le n$, the subarray $A[p\,..\,q]$ has k elements (all ≤ pivot) with probability $1/n$

Analysis of Running Time

- When we call RANDOMIZED-SELECT we could have three situations:
  - The algorithm terminates with the answer (**$i = k$**), or
  - The algorithm recurses on the subarray **$A[p..q-1]$**, or
  - The algorithm recurses on the subarray **$A[q+1..r]$**
- The decision depends on where the **i**-th smallest element falls relative to **$A[q]$**
- To obtain an upper bound for the running time **$T(n)$**:
  - assume the **i**-th smallest element is always in the larger subarray

Analysis of Running Time (cont.)

$$E[T(n)] = \underbrace{\text{Probability that T(n) takes a value} \times \text{The value of the random variable T(n)}}_{\text{Summed over all possible values}}$$

+ +…+

since select recurses only on the larger partition

PARTITION

$$\text{¿}\frac{1}{n}\left[T(n-1)+T(n-2)+T(n-3)+\ldots+T\left(\frac{n}{2}\right)+\ldots+T(n-3)+T(n-2)+T(n-1)\right]+O(n)$$

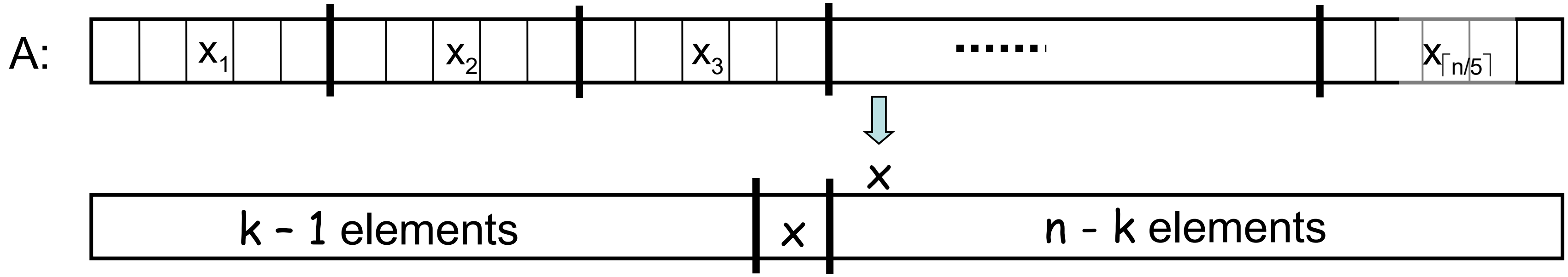
$$E[T(n)]=\frac{2}{n}\sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k)+O(n)$$

T(n) = O(n) (prove by substitution)

A Better Selection Algorithm

- Can perform Selection in $O(n)$ Worst Case
- Idea: guarantee a good split on partitioning
  - Running time is influenced by how "balanced" are the resulting partitions
- Use a modified version of PARTITION
  - Takes as input the element around which to partition

Selection in $O(n)$ Worst Case



1. Divide the n elements into groups of 5 ⇒ $\lceil n/5 \rceil$ groups
2. Find the median of each of the $\lceil n/5 \rceil$ groups
   - Use insertion sort, then pick the median
3. Use SELECT recursively to find the median x of the $\lceil n/5 \rceil$ medians
4. Partition the input array around x, using the modified version of PARTITION
   - There are $k-1$ elements on the low side of the partition and $n-k$ on the high side
5. If $i = k$ then return x. Otherwise, use SELECT recursively:
   - Find the i-th smallest element on the low side if $i < k$
   - Find the $(i-k)$-th smallest element on the high side if $i > k$

Example

- Find the 11th smallest element in the array:

  A = {12, 34, 0, 3, 22, 4, 17, 32, 3, 28, 43, 82, 25, 27, 34, 2 ,19 ,12 ,5 ,18 ,20 ,33, 16, 33, 21, 30, 3, 47}

1. Divide the array into groups of 5 elements

12	4	43	2	20	30
34	17	82	19	33	3
0	32	25	12	16	47
3	3	27	5	33	
22	28	34	18	21	

Example (cont.)

2. Sort the groups and find their medians

0	4	25	2	20	3
3	3	27	5	16	30
12	17	34	12	21	47
34	32	43	19	33	
22	28	82	18	33	

3. Find the median of the medians

12, 12, 17, 21, 34, 30

Example (cont.)

4. Partition the array around the median of medians (17)

   First partition:

   {12, 0, 3, 4, 3, 2, 12, 5, 16, 3}

   Pivot:

   17 (position of the pivot is q = 11)

   Second partition:

   {34, 22, 32, 28, 43, 82, 25, 27, 34, 19, 18, 20, 33, 33, 21, 30, 47}

To find the 6-th smallest element we would have to recurse our search in the first partition.

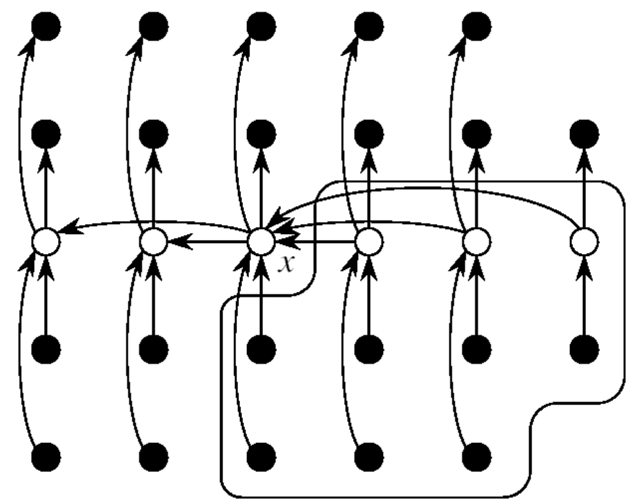
Analysis of Running Time

- Step 1: making groups of 5 elements takes $O(n)$
- Step 2: sorting $n/5$ groups in $O(1)$ time each takes $O(n)$
- Step 3: calling SELECT on $\lceil n/5 \rceil$ medians takes time $T(\lceil n/5 \rceil)$
- Step 4: partitioning the n-element array around x takes $O(n)$
- Step 5: recursion on one partition takes

depends on the size of the partition!!

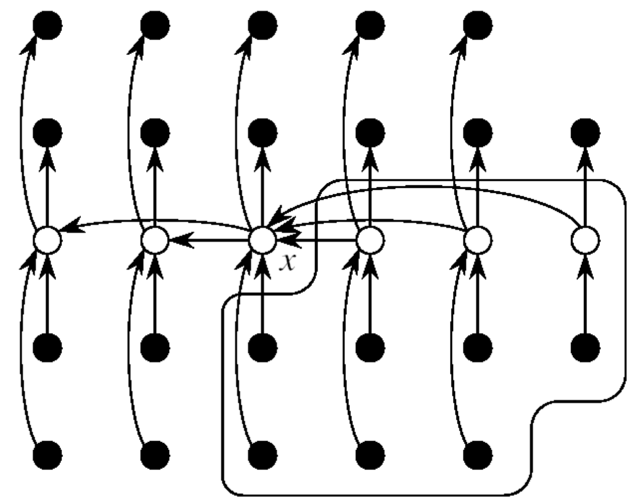
Analysis of Running Time

- First determine an upper bound for the sizes of the partitions
  - See how bad the split can be
- Consider the following representation
  - Each column represents one group of 5 (elements in columns are sorted)
  - Columns are sorted by their medians



Analysis of Running Time

- At least half of the medians found in step 2 are ≥ **x**: $\lceil \frac{1}{2} \lceil \frac{n}{5} \rceil \rceil$
- All but two of these groups contribute 3 elements > **x**



$\lceil \frac{1}{2} \lceil \frac{n}{5} \rceil \rceil - 2$ groups with 3 elements > **x**

- At least $3\left(\lceil \frac{1}{2} \lceil \frac{n}{5} \rceil \rceil - 2\right) \geq \frac{3n}{10} - 6$ elements greater than x
- SELECT is called on at most $n - \left(\frac{3n}{10} - 6\right) = \frac{7n}{10} + 6$ elements

Recurrence for the Running Time

- Step 1: making groups of 5 elements takes $O(n)$
- Step 2: sorting $n/5$ groups in $O(1)$ time each takes $O(n)$
- Step 3: calling SELECT on $\lceil n/5 \rceil$ medians takes time $T(\lceil n/5 \rceil)$
- Step 4: partitioning the n-element array around x takes $O(n)$
- Step 5: recursion on one partition takes time $\leq T(7n/10 + 6)$
- $T(n) = T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n)$
- We will show that $T(n) = O(n)$

Substitution

- $T(n) = T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n)$
  Show that $T(n) \leq cn$ for some constant $c > 0$ and all $n \geq n_0$

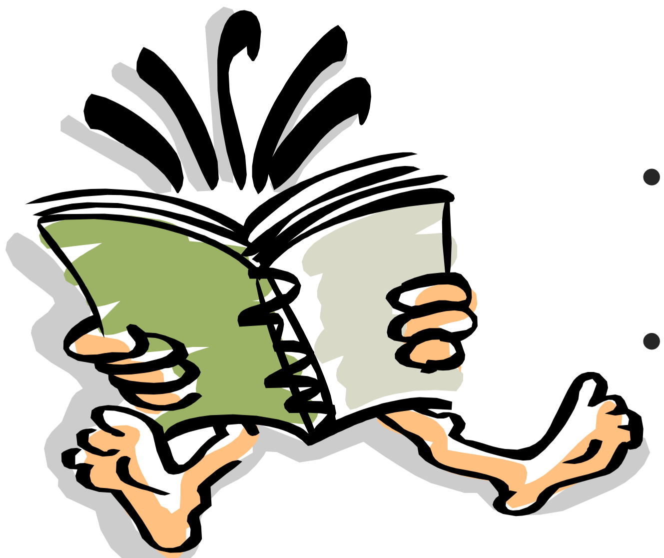
  $$\begin{aligned} T(n) &\leq c\lceil n/5 \rceil + c\,(7n/10 + 6) + an \\ &\leq cn/5 + c + 7cn/10 + 6c + an \\ &= 9cn/10 + 7c + an \\ &= cn + (-cn/10 + 7c + an) \\ &\leq cn \qquad \text{if: } -cn/10 + 7c + an \leq 0 \end{aligned}$$

- $c \geq 10a(n/(n-70))$
  - choose $n_0 > 70$ and obtain the value of c

How Fast Can We Sort?

- Insertion sort, Bubble Sort, Selection Sort $\Theta(n^2)$
- Merge sort $\Theta(n\lg n)$
- Quicksort $\Theta(n\lg n)$
- What is common to all these algorithms?
  - These algorithms sort by making comparisons between the input elements
- To sort n elements, comparison sorts must make $\Omega(n\lg n)$ comparisons in the worst case

Readings



- For this lecture
  - Section 9.3, 8.1, 8.2
- Coming next
  - Section 8.3, 8.4
  - Chapter 6