

Order Statistics

Data Structures and Algorithms (094224)

Tutorial 9

Winter 2022/23

1 Order Statistics

2 Algorithm Select

3 Questions

Order Statistics and Medians

- The i th *order statistic* (סטטיסטי הסדר) of a set of n elements is the i th smallest element
 - The minimum is the first order statistic ($i = 1$)
 - The maximum is the n th order statistic ($i = n$)
- A *median* (הציון), informally, is the "halfway point" of the set
- If n is odd, the median is unique, occurring at $i = \frac{n+1}{2}$
- If n is even, there are two medians
 - $i = \frac{n}{2}$ (lower median); and
 - $i = \frac{n}{2} + 1$ (upper median)
- Assumptions:
 - The phrase "the median" refers to lower median
 - Regardless of the parity of n , median occurs at $i = \lfloor \frac{n+1}{2} \rfloor$
 - The set of n elements resides in an array
 - The elements are distinct
 - Everything that we do extends to repeated values
 - Unless stated otherwise, if A is an array, then $n = A.length$

The Selection Problem

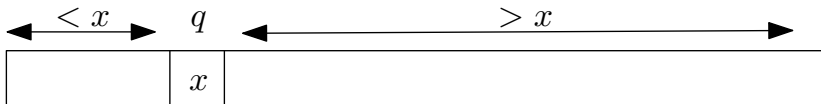
- The **selection** problem
 - **Input:** An array A of n (distinct) numbers and an integer i , with $1 \leq i \leq n$
 - **Output:** The i -th order statistic of A
- **Applications:**
 - Running time of `Quick_Sort` is $O(n \log(n))$ if the pivot element is the median and if it can be found in $O(n)$ time
 - Various uses in statistics and data base

The Selection Problem — cont.

- How can we find the minimum in $O(n)$ time?
 - Examine each element of the set in turn and keep track of the smallest element seen so far
- A naive solution for the selection problem:
`Naive_Select(A, i)`
 - ❶ `Merge_Sort($A, 1, n$)`
 - ❷ `return $A[i]$`
- `Naive_Select` first sorts the array, $T_{\text{Naive_Select}}(n) = \Omega(n \log(n))$
 - Regardless of the sorting algorithm being used. why?
- **Today:** algorithm for solving the selection problem in $O(n)$ time
 - There is hope since sorting the array does much more work than we are seeking
 - Solving for **every** i whereas we only need one

Procedure Partition

- Procedure Partition(A, p, r, x)
 - Let $n = \text{length}(A)$
 - **Precondition:** element x is part of array $A[p \dots r]$
 - Partition array $A[p \dots r]$ into subarrays $A[p \dots q]$ and $A[q + 1 \dots r]$ with respect to the pivot x so that $A[i] < x < A[j]$ for every $p \leq i < q$ and $q + 1 \leq j \leq r$. Set $A[q] = x$ and return q
 - $O(n)$ time complexity



Procedure Partition

Partition(A, p, r, x)

```
1:  $n = r - p + 1$ 
2: new array  $B[1 \dots n]$ 
3:  $left = 1, right = n$ 
4: for  $i = 0$  to  $n - 1$  do
5:     if  $A[i + p] > x$  then
6:          $B[right] = A[i + p]$ 
7:          $right = right - 1$ 
8:     else if  $A[i + p] < x$  then
9:          $B[left] = A[i + p]$ 
10:         $left = left + 1$ 
11:  $q = left$ 
12:  $B[q] = x$ 
13: for  $i = 1$  to  $n$  do
14:      $A[i - 1 + p] = B[i]$ 
15: return  $p + q - 1$ 
```

Question 1

Suppose that you have a “black-box” worst-case linear-time median subroutine, $\text{Median}(A)$. Give a simple, linear-time algorithm that solves the selection problem for an arbitrary order statistic.

Solution

Select_Given_Median(A, i)

```
1: if  $n == 1$  then
2:   return  $A[1]$ 
3:  $x = \text{Median}(A)$ 
4:  $q = \text{Partition}(A, 1, n, x)$ 
5: if  $i == q$  then
6:   return  $A[q]$ 
7: if  $i < q$  then
8:   new array  $B[1 \dots q - 1]$ 
9:   copy  $A[1 \dots q - 1]$  to  $B$ 
10:  return Select_Given_Median( $B, i$ )
11: else
12:  new array  $B[1 \dots n - q]$ 
13:  copy  $A[q + 1 \dots n]$  to  $B$ 
14:  return Select_Given_Median( $B, i - q$ )
```

Solution — Example

Running `Select_Given_Median(A, 7)`

$A =$

20	15	17	18	21	12	16	19	14	11	13
----	----	----	----	----	----	----	----	----	----	----

$n = 11, x = 16$

After `Partition(A, 1, 11, 16)`

$A =$

13	15	11	14	12	16	21	19	18	17	20
----	----	----	----	----	----	----	----	----	----	----

$q = 6 < 7$

$B =$

21	19	18	17	20
----	----	----	----	----

Call `Select_Given_Median(B, 1)`

$A =$

21	19	18	17	20
----	----	----	----	----

$n = 5, x = 19$

Solution — Example — cont.

After Partition($A, 1, 5, 19$)

$B =$

17	18	19	21	20
----	----	----	----	----

$q = 3 > 1$

$B =$

17	18
----	----

Call Select_Given_Median($B, 1$)

$A =$

17	18
----	----

$n = 2, x = 17$

After Partition($A, 1, 2, 17$)

$A =$

17	18
----	----

$q = 1$

The 7th order statistic is 17

- The run time function of `Select_Given_Median(A, i)`

$$T(n) \leq \begin{cases} c, & n = 1 \\ T(\lfloor n/2 \rfloor) + cn, & n > 1 \end{cases}$$

- Guess a solution $T(n) = O(n)$ and verify using substitution
- **Claim:** There exists a constant $c' > 0$ such that $T(n) \leq c'n$ for all $n \geq 1$
 - $\implies T(n) = O(n)$

- Base: $n = 1, c \leq c'$
 - $T(1) \leq c \leq c' \cdot 1$
 - Require $c' \geq c$
- Hypothesis: For all positive integer $m < n$ it holds $T(m) \leq c'm$
- Step:

$$T(n) \leq T(\lfloor n/2 \rfloor) + cn \leq c' \left\lfloor \frac{n}{2} \right\rfloor + cn \leq \frac{1}{2}c'n + cn$$

- Which is at most $c'n$ if $c' \geq 2c$
 - $c \leq \frac{1}{2}c' \implies \frac{1}{2}c'n + cn \leq \frac{1}{2}c'n + \frac{1}{2}c'n \leq c'n$
- Take any constant $c' \geq 2c$ to complete the proof

1 Order Statistics

2 Algorithm Select

3 Questions

Algorithm Select

Returns the i th order statistic of n distinct numbers in array A . If $n = 1$ simply returns the element $A[1]$ as the i th order statistic.

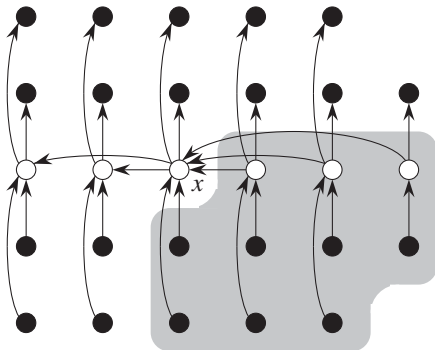
`Select(A, i)`

- ❶ Divide the n elements of A into $\lfloor \frac{n}{5} \rfloor$ groups of 5 elements each, and a residual group
- ❷ Find the median of each group in a straightforward way
- ❸ Invoke `Select` recursively to find the median of the $\lceil \frac{n}{5} \rceil$ medians from step 2 and denote it by x
- ❹ $q = \text{Partition}(A, 1, n, x)$. ($q - 1$ is the number of elements in the low side of the partition and $n - q$ is the number of elements on the high side of the partition)
- ❺
 - If $i = q$, then return x
 - Otherwise, if $i < q$, use `Select` recursively by calling `Select(copy of $A[1 \dots q - 1], i$)`
 - Otherwise, $i > q$, use `Select` recursively by calling `Select(copy of $A[q + 1 \dots n], i - q$)`

Algorithm Select — Run Time Analysis

- Denote by T the run time function of $\text{Select}(A, i)$
- Line 1 – $O(n)$
- Line 2 – $O(n)$
 - Find median of each group – $O(1)$, the group size is the constant 5
 - There are $\lceil \frac{n}{5} \rceil$ groups a total of $\lceil \frac{n}{5} \rceil \cdot O(1) = O(n)$
- Line 3 – $T(\lceil \frac{n}{5} \rceil)$
 - We recursively call Select with $\lceil \frac{n}{5} \rceil$ elements
- Line 4 – $O(n)$
- Line 5 – ?
 - In line 5 we recursively call Select with how many elements?

Algorithm Select — Run Time Analysis



- The n elements are represented by small circles
- Each one of the $\lceil \frac{n}{5} \rceil$ groups occupies a column
- The medians of the groups are whitened
- x is the median of medians
- Arrows goes from larger element to smaller
- The elements known to be grater then x appear on a shaded

Algorithm Select — Run Time Analysis — cont.

- At least in half of the $\lceil \frac{n}{5} \rceil$ groups there are 3 elements that greater than x except of the group containing x itself and the group that has fewer than 5 elements (if 5 does not divide n exactly)
- Thus,

$$\text{\#of elements greater than } x \geq 3 \left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 2 \right) \geq \frac{3}{10}n - 6$$

- Similarly,

$$\text{\#of elements lesser than } x \geq 3 \left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 2 \right) \geq \frac{3}{10}n - 6$$

- Consequently, Line 5 will recurse on at most $n - (\frac{3}{10}n - 6) = \frac{7}{10}n + 6$

Algorithm Select — Run Time Analysis — cont.

- T is monotonically not decreasing. Thus

$$T(n) \leq \begin{cases} O(1), & n = 1 \\ T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n), & n > 1 \end{cases}$$

- Since for any constant number of elements in array A the run time is a constant we can write

$$T(n) \leq \begin{cases} O(1), & n < 140 \\ T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n), & n \geq 140 \end{cases}$$

- The origin of the constant 140 will be clear shortly
- Using explicit constants

$$T(n) \leq \begin{cases} c, & n < 140 \\ T(\lceil n/5 \rceil) + T(7n/10 + 6) + cn, & n \geq 140 \end{cases}$$

Algorithm Select — Run Time Analysis — cont.

- Since Select recurse on a constant fraction of the elements (roughly $9n/10$) and discard the rest of the elements we guess $T(n) = O(n)$ and verify with substitution
- **Claim:** There exists a constant $c' > 0$ such that $T(n) \leq c'n$ for all $n \geq 1$
 - $\implies T(n) = O(n)$
- **Base:** For all $n < 140$ if we set $c' \geq c$ the claim holds
- **Hypothesis:** For all positive integer $m < n$ it holds $T(m) \leq c'm$
- **Step:**

$$\begin{aligned} T(n) &\leq T(\lceil n/5 \rceil) + T(7n/10 + 6) + cn \\ &\leq \underbrace{c' \lceil n/5 \rceil}_{i.h} + c'(7n/10 + 6) + cn \\ &\leq c'(n/5 + 1) + 7c'n/10 + 6c' + cn \\ &\leq 9c'n/10 + 7c' + cn = c'n - c'n/10 + 7c' + cn \end{aligned}$$

- Which is at most $c'n$ if $-c'n/10 + 7c' + cn \leq 0$

Algorithm Select — Run Time Analysis — cont.

- $-c'n/10 + 7c' + cn \leq 0$
$$\iff -c'n + 70c' + 10cn \leq 0$$
$$\iff 10cn \leq c'(n - 70)$$
$$\underbrace{\iff}_{n \geq 140} 10c \frac{n}{n - 70} \leq c'$$
- $n \geq 140 \implies \frac{n}{n-70} \leq 2$
- $10c \frac{n}{n-70} \leq 20c$
- Take any constant $c' \geq 20c$ to complete the proof
- There is nothing special about the constant 140 we could replace it by any constant > 70 and then choose c' accordingly

1 Order Statistics

2 Algorithm Select

3 Questions

Question 2

The k th *quantiles* of an n -element array of distinct numbers are the $k - 1$ order statistics that divide the sorted array into k equal-sized subarrays (to within 1). Give an $O(n \log k)$ -time algorithm to list the k th quantiles of an array.

- The k quantiles are defined by the following $k - 1$ order statistics $\lfloor \frac{n+1}{k} \rfloor, \lfloor 2 \cdot \frac{n+1}{k} \rfloor, \dots, \lfloor (k - 1) \cdot \frac{n+1}{k} \rfloor$

High Level:

- Maintain a list L of the requested order statistics
- Use `Select` to find the median order statistic of the order statistics, denote by x
 - x is the $\lfloor \lfloor \frac{k}{2} \rfloor \cdot \frac{n+1}{k} \rfloor$ order statistic
- Partition the array according to x
- Roughly half of the requested order statistics are on the right of x and roughly half are on the left of x
- Invoke recursively on both sides of the array

Return the $k - 1$ order statistics that divide an array $A[p \dots r]$ into k equal sized groups (to within 1). Initially L is an empty list

Quantiles(A, p, r, k, L)

- 1: **if** $k == 1$ **then**
- 2: **return**
- 3: $n = r - p + 1$
- 4: $i = \lfloor \lfloor \frac{k}{2} \rfloor \cdot \frac{n+1}{k} \rfloor$
- 5: $x = \text{Select}(\text{copy of } A[p \dots r], i)$
- 6: $q = \text{Partition}(A, p, r, x)$
- 7: Add x to L
- 8: **Quantiles**($A, p, q, \lfloor k/2 \rfloor, L$)
- 9: **Quantiles**($A, q + 1, r, \lceil k/2 \rceil, L$)

Runtime analysis (not a formal proof):

- Analyzing the run time using recursion tree
- Lines 1-6 takes $O(n)$ time
- Each internal node in the recursion tree has 2 children
- Each level of the recursion tree takes $O(n)$ time
- The depth of the recursion tree is $O(\log k)$
 - In each level the parameter k is divided by 2
- Thus the run time is $O(n) \cdot O(\log k) = O(n \log k)$

Question 3

Let $X[1 \dots n]$ and $Y[1 \dots n]$ be two arrays, each containing n numbers already in sorted order. Give an $O(\log n)$ -time algorithm to find the median of all $2n$ elements in arrays X and Y .

High level idea:

- The median of $2n$ elements is at location $\lfloor \frac{2n+1}{2} \rfloor = \lfloor \frac{2n}{2} + \frac{1}{2} \rfloor = n$ of the $2n$ sorted array
- If $X[1] \geq Y[n]$ then $Y[n]$ is the median
- If $Y[1] \geq X[n]$ then $X[n]$ is the median
- Otherwise, let $i = \lfloor \frac{n+1}{2} \rfloor$, $m_x = X[i]$ and $m_y = Y[i]$
 - The median of X and Y , respectively
- If $m_x = m_y$ then the median of the $2n$ elements is m_x
- If $m_x > m_y$ and n is odd then the median cannot be in locations $X[i+1 \dots n]$ and $Y[1 \dots i-1]$
- If $m_x > m_y$ and n is even then the median cannot be in locations $X[i+1 \dots n]$ and $Y[1 \dots i]$
- Similarly if $m_y > m_x$

Solution — cont.

Sorted_Arrays_Median($X, Y, x_\ell, x_r, y_\ell, y_r$)

```
1: if  $X[x_\ell] \geq Y[y_r]$  then
2:   return  $Y[y_r]$ 
3: if  $Y[y_\ell] \geq X[x_r]$  then
4:   return  $X[x_r]$ 
5:  $x_m = \lfloor \frac{x_\ell + x_r}{2} \rfloor, y_m = \lfloor \frac{y_\ell + y_r}{2} \rfloor$ 
6: if  $X[x_m] == Y[y_m]$  then
7:   return  $X[x_m]$ 
8:  $n = x_r - x_\ell + 1$ 
9:  $parity = n \bmod 2$ 
10: if  $X[x_m] > Y[y_m]$  then
11:   return Sorted_Arrays_Median( $X, Y, x_\ell, x_m, y_m + 1 - parity, y_r$ )
12: else
13:   return Sorted_Arrays_Median( $X, Y, x_m + 1 - parity, x_r, y_\ell, y_m$ )
```

Example

$X =$

15	16	17	18	19	20	21	22
----	----	----	----	----	----	----	----

 $n = 8$

$Y =$

10	11	12	13	14	15	16	17
----	----	----	----	----	----	----	----

$i = 4$, $X[4] > Y[4]$ call `Sorted_Arrays_Median($X[1 \dots 4]$, $Y[5 \dots 8]$)`

$X =$

15	16	17	18
----	----	----	----

 $n = 4$

$Y =$

14	15	16	17
----	----	----	----

$i = 2$, $X[2] > Y[2]$ call `Sorted_Arrays_Median($X[1 \dots 2]$, $Y[3 \dots 4]$)`

$X =$

15	16
----	----

 $n = 2$

$Y =$

16	17
----	----

$i = 1$, $Y[1] > X[1]$ call `Sorted_Arrays_Median($X[2]$, $Y[1]$)`

$X =$

16

 $n = 1$

$Y =$

16

$X[1] \geq Y[1]$ return $X[1] = 16$

Solution — Analysis

- If $m_x > m_y$ and n is odd then the median cannot be in $X[i + 1 \dots n]$
 - Let x be some element in $X[i + 1 \dots n]$
 - $i = \lfloor \frac{n+1}{2} \rfloor = \frac{n+1}{2}$ since n is odd
 - $x \geq X[j]$ and $x \geq Y[j]$ for $1 \leq j \leq i$
 - Thus, there are at least $2i = (n + 1)$ elements before x in the $2n$ sorted array
 - x is NOT the n th element in the $2n$ sorted array
- If $m_x > m_y$ and n is odd then the median cannot be in $Y[1 \dots i - 1]$
 - Let y be some element in $Y[1 \dots i - 1]$
 - $i = \lfloor \frac{n+1}{2} \rfloor = \frac{n+1}{2}$ since n is odd
 - $y \leq Y[j]$ and $y \leq X[j]$ for $i \leq j \leq n$
 - Thus, there are at least $2(n - i + 1) = n + 1$ elements after y in the $2n$ sorted array
 - y is NOT the n th element in the $2n$ sorted array
- Similarly for an even n
- For the case where $m_y > m_x$ replace the rule of X and Y
- **Run Time Analysis:** Since we eliminate, roughly, half of the elements the run time is $O(\log(2n)) = O(\log n)$ (NOT a FORMAL PROOF)