Subject 6
Spring 2016

MEDIAN AND ORDER STATISTICS

Chapter 4

DISCLAIMER: These abbreviated notes DO NOT substitute the textbook for this class. They should be used IN CONJUNCTION with the textbook and the material presented in class. If there is a discrepancy between these notes and the textbook, ALWAYS consider the textbook to be correct. Report such a discrepancy to the instructor so that he resolves it. These notes are only distributed to the students taking this class with A. Gerbessiotis in Spring 2016; distribution outside this group of students is NOT allowed.
Selection
Review

When an input sequence of keys is sorted, as a byproduct of sorting we also know what the maximum and minimum keys of the sorted sequence are. They are elements $A[0]$ (minimum) and $A[n-1]$ (maximum) of the sorted output array $A[0..n-1]$.

We also know the $k$-th smallest key: it is $A[k-1]$. This is also the $(n-k+1)$-st largest key (the 1-st smallest is the minimum and 1-st largest is the maximum).

The $k$-th smallest of a set of keys is also known as the $k$-th order statistic.

One of the interesting statistics is what is called the median. The lower median or median is the $\lfloor \frac{n+1}{2} \rfloor$ statistic. The upper median is the $\lceil \frac{n+1}{2} \rceil$ statistic. We have a lower-median and upper-median to disambiguate the case where $n$ is even.

The selection problem is defined as follows.

**Input.** A set $S$ of $n$ (distinct) keys and a number $k$ such that $1 \leq k \leq n$.

**Output** The element $s \in S$ such that $s$ is larger than $k-1$ other elements of $S$.

One way to solve the Selection problem is by sorting and then reading off the $k$-th sorted element (from the left, where leftmost is the smallest of the $n$ elements).
Problem 1. Find the minimum of \( n \) keys?

Answer: An elementary upper bound on the number of comparisons is \( n - 1 \). This is also a lower bound as all the other keys need to be read and ruled out before a conclusion on the minimum can be drawn. This can also be shown by a tournament argument where every key but the winner must lose a match (i.e. a comparison must result in this element not being the minimum).

Problem 2. Solve the MINMAX problem i.e. find both the minimum and the maximum at the same time.

Answer: A relevant question is how long it takes to find both the minimum and the maximum. An easy upper bound is \( 2n - 2 \) comparisons by using the previous algorithm twice. Under this algorithm every key is compared twice, once with the current minimum and once with the current maximum.

A better algorithm is the following

Take the first two elements of the set and make the smallest of the two the current minimum and the largest the current maximum.

Take the next two elements, compare them, and then compare the minimum with the current minimum and the maximum of the two with the current maximum. The total number of comparisons performed is 3 instead of 4 (2 per key).

Problem View the same problem as a divide-and-conquer problem.

Question How many comparisons do you need to find simultaneously the min and max of \( n \) keys?

Answer. At most \( 3n/2 - 2 \).

Question. Suppose we have found the minimum. How many additional comparisons are required to find the second smallest key of the original \( n \) keys?

Answer. \( \lceil \lg n \rceil - 1 \).

or Question. Find the second smallest of \( n \) keys is \( n + \lceil \lg n \rceil - 2 \) comparisons.
Selection  
Expected linear time algorithm

The idea of the algorithm is to use recursive partitioning (divide and conquer) just as quick sort did. The difference from quick sort is that in the recursive step only one of the two partitions needs to be further searched. This difference leads to an $\Theta(n)$ expected time algorithm rather than $\Theta(n \lg n)$.

The algorithm uses **Randomized-Partition** which is similar to **Partition** except that the splitter is chosen at random (instead of being the first element of the subsequence passed as an argument). Then **Randomized-Select** works as follows.

**Randomized-QuickSelect** ($A, l, r, k$) //Split $A[l...r]$ into two sets: Left, Right
1. if ($l==r$) the return $A[l]$ //containing all keys $\leq$ and $\geq$ splitter
2. $m=$Randomized-Partition($A, l, r$); //Splitter is chosen uniformly at random
3. $L=m-l+1$; // Line 3 computes the size Left
4. ($k<=L$) //Is statistic on the left side?
5. return Randomized-Select($A, l, m, k$) // yes it is: search for statistic in Left
6. else // otherwise in Right
8. return Randomized-Select($A, m+1, r, k-L$) //no it is not: When you search $A[m+1..r]$
   //do not ignore the $L$ keys of the left set $A[l...m]$

Note: In GT set **Left** is sets $L$ and $E$ and set **Right** is sets $G$ and $E$.
The worst case running time is $\Theta(n^2)$. For the average case the recurrence to solve is

$$T(n) \leq 1/n(T(\max(1, n - 1)) + \sum_{k=1}^{n-1} T(\max(k, n - k))) + O(n)$$

whose solution is $T(n) = O(n)$.

This is so because if splitter is the minimum, then the left partition is of size 1 (prob $1/n$ this happening). If otherwise rank(splitter) $\geq 2$, the left set is $1, 2, \ldots n-1$ with probability $1/n$ for each case as well. Therefore probability left set is 1 is $2/n$ (two cases this happening).
We describe an algorithm whose worst case running time is $O(n)$. We achieve this by splitting the input set more accurately than using Randomized-Partition. The algorithm below describes procedure $\text{Select}(A,i)$ which is Exercise C-4.24 on page 254 of GT.

$\text{Select}(A,i)$ // Find $i$-th smallest in set $A$ of $n$ keys.
1. Group $n$ keys into groups of 5 keys.
2. Sort the keys in each group with insertion sort.
3. Identify the median of each group.
4. Find the median $x$ of the medians of the groups recursively using Select.
5. Partition $A$ into
   $A_{left} = \{ a: a \text{ belongs to } A \text{ and } a < x \}$
   $A_{right} = \{ a: a \text{ belongs to } A \text{ and } a \geq x \}$
6. Let $|A_{left}| = k$ and $|A_{right}| = n-k$.
7. if $(i \leq k)$
8. $\text{Select}(A_{left},i)$;
9. else
10. $\text{Select}(A_{right},i-k)$;

1. Split the $n$ keys into groups of 5 keys. There are $\text{ceiling}(n/5)$ many groups of which $\text{floor}(n/5)$ have 5 keys and the remainder the remaining keys of the input set (less than 5).
2. Find the median of each group of 5 keys by sorting each group with say, insertion sort. This requires constant time $O(5^2) = O(1)$ per group and $O((n/5))$ overall, since insertion sort on 5 keys requires $O(1)$ time per group and thus $O(n)$ for all groups. The median of 5 keys is the third smallest (or largest) of the 5 keys.
4. Use Select recursively to find the median $x$ of the ceiling($n/5$) medians of step 3.
5. Partition the input keys around $x$. Let $k$ be the number of elements less than $x$ and $n-k$ the number of elements greater than or equal to $x$ forming respectively sets $A_{left}$ and $A_{right}$.
7-10. Use Select recursively to find the $i$-th smallest key in $A_{left}$ if $i \leq k$ or the $(i-k)$-th smallest key in $A_{right}$ otherwise.
The first step in the analysis of Select is the finding of the size of the set of elements which are larger than $x$. At least half of the medians in step 3 are greater than or equal to $x$. For each such median there are three elements (including the median element) out of 5 that are greater than $x$ except for the last group (which may have fewer than 5 elements, i.e. fewer than 3 greater than $x$) and the group containing $x$ (which has 2 keys greater than $x$), plus the two elements greater than $x$. Therefore THE NUMBER OF KEYS GREATER THAN $x$ is AT LEAST

$$3(\lfloor 1/2 \rfloor \lceil n/5 \rceil - 2) + 2 \geq 3n/10 - 4$$

Then, the NUMBER OF KEYS LESS THAN $x$ is AT MOST

$$n - (3n/10 - 4) \leq 7n/10 + 4$$

Similarly, THE NUMBER OF KEYS LESS THAN $x$ is AT LEAST $3n/10 - 4$ and NUMBER OF KEYS GREATER THAN $x$ is AT MOST $7n/10 + 4$. Therefore in step 5 and in steps 8-10 Select is called on a set whose maximum size is at most $7n/10 + 4$. 
**Linear Selection**

**Graphical View of the keys**

Assume we have sorted the medians of the groups of 5 left to right (smallest-to-largest) and also the keys within a group bottom-to-top (smallest to largest). Then the $n$ keys look like.

| <=x | 29 10 70 | /5 10 4 5 |
| <=x | 19 5 20 | / 1 2 3 4 |

Call the median of the medians $x(=100)$

| >=x | 100 300 1010 | 140 122 144 162 |
| >=x | 70 200 1000 | 110 121 143 161 |

The top right area is the set of keys $>=x$

The bottom left area is the set of keys $<=x$

#keys ($<=x$) $>= 3n/10$ [bottom-left]

#keys ($>=x$) $>= 3n/10$ [top-right]

HOWEVER WE WANT TO KNOW AN UPPER BOUND
ON THE NUMBER OF KEYS not the $3n/10$ lower bound.

If we use [bottom-left] we establish the upper-bound for the [top-right] area

Use $\#\text{keys} (<= x) + \#\text{keys} (> x) = n$ i.e. $\#\text{keys} (> x) = n - \#\text{keys}(<=x) <= n - 3n/10 = 7n/10$

If we use [top-right] we establish the upper-bound for the [bottom-left] area

Use $\#\text{keys} (<= x) + \#\text{keys} (> x) = n$ i.e. $\#\text{keys} (<= x) = n - \#\text{keys}(>=x) <= n - 3n/10 = 7n/10$
Selection in linear time

Performance Analysis

We then analyze the performance of Algorithm Select in detail.

Steps 1, 2 and 3 require $O(n)$ steps. Step 4 takes $T(ceiling(n/5))$ steps and step 5-10 requires $T(7n/10 + 4)$ steps. Therefore

If $n > 100$,

$$ T(n) = T(n/5 + 1) + T(7n/10 + 4) + n \quad , \quad n \geq 100 $$

else if $n < 100$, $T(n) = \Theta(1)$ i.e. it takes constant time to find the median or other statistic of a constant number (up to 100) number of keys. We solve the recurrence using the substitution method. Let us guess $T(n) \leq cn$ for $n \geq 100$. Then

$$ T(n) = T(n/5 + 1) + T(7n/10 + 4) + n \leq c(n/5 + 1) + c(7n/10 + 4) + n $$

$$ = 9cn/10 + 5c + n $$

We observe that $5c + n \leq cn/10$ for say $c = 20$ as long as $n \geq 100$. Large values of $c$ are OK as well. Therefore, for $n \geq 100$

$$ T(n) \leq 9cn/10 + 5c + n \leq 20n $$

which implies that $T(n) = O(n)$, as needed.

**Question.** What if we had used groups of 3 keys? How about groups of 7? How about groups of 11?

**Question.** How fast can we sort 5 keys? Although the algorithms suggests the use of insertion sort, one could utilize an optimal method whose running time (one 5 keys, not asymptotically) cannot be improved upon. This sorting determines the constant of the linear term (more or less) hidden inside the big-Oh, and is the base case of the recurrence and the algorithm.