Name: Zefeng Song Gwid: G22237721 Gmail:zefengsong@gwmail.gwu.edu

I. Question A :

Given a set of N unordered numbers, find the Kth smallest number, using median finding (Median of medians) to find a good partition deterministically.

II. Analysis

A simple solution is to sort the array then return the Kth smallest number, which will take O(nlgn) time, but use Divide&Conquer strategy allow us to find Kth smallest element in O(n) time in worst case.

Steps:

1. If n<6,just sort and return the Kth smallest number in constant time O(1);
2. If n>=6,divide the n elements of the input array into [n/5] groups of 5 elements each and at most one group made up of the remaining n mod 5 elements.
3. Find the median of each of the [n/5] groups by first quicksort the elements of each group (of which there are at most 5) and then picking the median from the sorted list of group elements.
4. Call the function in step3 recursively to find the median of [n/5] medians, call it m. T(n/5)
5. Partition the array on the median-of-medians m using the modified version of PARTITION and determine the sets L and R, where L contains all elements <m, and R contains all elements >m. Clearly, the rank of m is r=|L|+1 (|L| is the size of L).

   "Depending upon value of K, either the left side of the partition or the right side will be discarded. In either case, we eliminate at least 30% of data."(discussed in class)
6. If k=r, then return m; If k<r, then return k th smallest of the set L; If k>r, then return (k-r) th smallest of the set R.

For the function we used in step2~4 to find the median of medians, which is the m, we can prove it's time complexity is O(n):

- In step 2, we divide the array into [n/5] groups, or [n/5]+1 groups consider the remaining elements, but it doesn't matter to the time complexity, let's just say we get [n/5]groups.
- Then we find the median of each group using quicksort, it takes constant time to sort each group (of which there are at most 5),O(1)(or more specifically, O(5lg5)), and there are [n/5] groups, so we have to sort [n/5] times, therefore, the time complexity is: O(n/5)*O(5lg5)=O(nlg5)=O(n)

Combine step1~6,we get the overall recurrence: T(n)=T(n/5)+T(7n/10)+cn;

Either using master theorem or mathematical induction we can all prove it's time complexity is O(n). (Mathmatical Induction presented in class:We claim that T(n)<=10cn for all values of n<N;T(N/5)<=2cN,T(7N/10)<=7cN;T(N)<=2cN+7cN+cN=10cN;Therefore,T(n)=O(n))

III. Coding

My program consists of 5 functions:

1. QUICKSORT function used in step3 to sort the elements in each [n/5] groups.
2. PARTITION function which is a part of QUICKSORT.
3. A modified PARTITION, in my program I named it partition_using_m:

```
int partition_using_m(int* A, int p, int r, int m) {int x = m;  //pivot
    int i = p - 1;...//the same as in PARTITION
    if (A[r] <= m) { swap(A[i + 1], A[r]); return i + 1+1;}
     else return i + 1;}
```

it is used in step5 to find the rank of m(median of medians) in the array. In origin version of

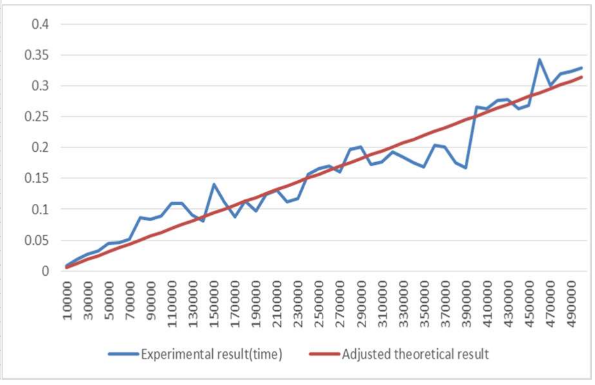
PARTITION, it uses the last element of array(A[r]) as a pivot to divide the array into 2 groups,then swap the A[i+1] with A[r],but now we use m to divide the array, so it is a little difference and we need to compare the A[r] with m, if A[r] is smaller than m, then the rank of m should plus 1.

4. `int find_median_of_medians(int* A, int left, int right);`which implements step2~4,it's function follows as `//divide the array//call QUICKSORT function//find medians//call itself recursively//return m-median of medians;`
5. `int Find_Kth_smallest(int* A, int left, int right, int k);` which implements step5~6,it's function is `//call find_median_of_medians to find the m//partition the array using m//call itself recursively//return m//`

```
int r = partition_using_m(A, left, right - 1,m );r = r - left;
 if (k == r) return m;
else if (k < r) return Find_Kth_smallest(A, left, r+left, k);
else return Find_Kth_smallest(A, r+left, right, k - r);
```

IV. Results

n	result(median)	Experimental Result, in s	Theoretical Result	Scaling Constant	Adjusted Theoretical Result
10000	5000	0.009	0.01	0.62847	0.006285
20000	10000	0.019	0.02	0.62847	0.012569
30000	15000	0.027	0.03	0.62847	0.018854
40000	20000	0.032	0.04	0.62847	0.025139
50000	25000	0.045	0.05	0.62847	0.031424
60000	30000	0.046	0.06	0.62847	0.037708
70000	35000	0.052	0.07	0.62847	0.043993
80000	40000	0.086	0.08	0.62847	0.050278
90000	45000	0.084	0.09	0.62847	0.056562
100000	50000	0.089	0.1	0.62847	0.062847
110000	55000	0.109	0.11	0.62847	0.069132
120000	60000	0.109	0.12	0.62847	0.075416
130000	65000	0.091	0.13	0.62847	0.081701
140000	70000	0.081	0.14	0.62847	0.087986
150000	75000	0.14	0.15	0.62847	0.094271
160000	80000	0.112	0.16	0.62847	0.100555
170000	85000	0.088	0.17	0.62847	0.10684
180000	90000	0.114	0.18	0.62847	0.113125
190000	95000	0.097	0.19	0.62847	0.119409
200000	100000	0.124	0.2	0.62847	0.125694
210000	105000	0.131	0.21	0.62847	0.131979
220000	110000	0.112	0.22	0.62847	0.138263
230000	115000	0.117	0.23	0.62847	0.144548
240000	120000	0.157	0.24	0.62847	0.150833
250000	125000	0.166	0.25	0.62847	0.157118
260000	130000	0.17	0.26	0.62847	0.163402
270000	135000	0.161	0.27	0.62847	0.169687
280000	140000	0.197	0.28	0.62847	0.175972
290000	145000	0.201	0.29	0.62847	0.182256
300000	150000	0.173	0.3	0.62847	0.188541
310000	155000	0.177	0.31	0.62847	0.194826
320000	160000	0.193	0.32	0.62847	0.20111
330000	165000	0.185	0.33	0.62847	0.207395
340000	170000	0.175	0.34	0.62847	0.21368
350000	175000	0.169	0.35	0.62847	0.219965
360000	180000	0.204	0.36	0.62847	0.226249
370000	185000	0.201	0.37	0.62847	0.232534
380000	190000	0.175	0.38	0.62847	0.238819
390000	195000	0.167	0.39	0.62847	0.245103
400000	200000	0.265	0.4	0.62847	0.251388
410000	205000	0.263	0.41	0.62847	0.257673
420000	210000	0.276	0.42	0.62847	0.263957
430000	215000	0.278	0.43	0.62847	0.270242
440000	220000	0.263	0.44	0.62847	0.276527
450000	225000	0.268	0.45	0.62847	0.282812
460000	230000	0.343	0.46	0.62847	0.289096
470000	235000	0.3	0.47	0.62847	0.295381
480000	240000	0.32	0.48	0.62847	0.301666
490000	245000	0.323	0.49	0.62847	0.30795
500000	250000	0.329	0.5	0.62847	0.314235
Average		0.16026	0.255	0.62847	



Conclusion: From the graph, we can see the Experimental result, which is the time it takes to run the program in seconds, matches well with the adjusted theoretical result, which is a straight line, so the time complexity of the algorithm is O(n).

Comparison: As was written in first page, a simple solution to find the Kth smallest number is to sort the array then return the Kth smallest number, which will take O(nlgn) time, but use Divide&Conquer strategy as **quick select using median of medians** is a more efficient algorithm which takes O(n) time in worst case.