DSAD Assignment I

Bishwajit Prasad Gond
222CS3113

Master of Technology
222cs3113@nitrkl.ac.in

Department of Computer Science & Engineering
NIT, Rourkela

September 28, 2022

Contents

1 Question

k^{th} **Smallest Element** Suppose you have a group of n numbers and would like to determine the k^{th} smallest. This is known as the selection problem.

Present atleast 6 different algorithms to find the k^{th} smallest element in an array having n unordered elements? Comment on the complexity of all algorithms suggested by you? Is it possible to design an optimal algorithm for this problem? Discuss on the possibility to design different randomized algorithm to determine the k^{th} smallest element in an array.

Alternate representation of the problem

- A problem closely related to, but simpler than sorting is that of the selection (also referred to as the order statistics) problem.
- The Selection problem: Given a sequence $A = (a1, a2, ..., an)$ of n elements on which a linear ordering is defined, and an integer k, $1 \leq k \leq n$, find the k^{th} smallest element in the sequence.
- The selection problem is the problem of computing, given a set A of n distinct numbers and a number k, 1 k n, the k^{th} order statistics (i.e., the k^{th} smallest number) of A.

1.1 Algorithm 1: Sort the elements in an array and return the element in the k^{th} position

```
1  Algorithm KthSmallest (A,n,k)
2  {
3  //A is an array of size n
4  // K represent kth  order of smallest element
5      //Sort the given array
6      sort(A)
7      // Return k'th element in the
8      //sorted array
9      return A[k-1]
10 }
```

In line 6, we have used a simple sorting algorithm which has time complexity of $O(n^2)$ so the time complexity of entire algorithms is $O(n^2)$ and if we use merge or heap sort we will get time complexity of $O(n \log n)$

1.2 Algorithm 2: To find k^{th} Smallest element using min heap data structure implementation

```
1  Algorithm with smallest (A,n,k)
2  {
3      //A is an array of size n
4      // K represent kth  order of smallest element
5      Create MinHeap(A,n).
6      for i:=1 to i<k do
7      {
8          Extract min(A);
9      }
10     return root;
11 }
```

In the above algorithm, we can observe lime no. 5 "Create MinHeap(A,n)" function is called on the array of n elements in linear time. So build min heap produce a min heap from unordered liner array in $O(n)$ time. In line no 8, we extract min element from heap in $\log_2(n)$ time (time taken to search the tree by height)

Total time = Building the min heap + extracting min element (k-1) times

$= O(n) + (K-1)\log n = O(n + K\log n)$

1.3 Algorithm 3: To find k^{th} Smallest element using max heap data structure implementation

```
1  Algorithm with smallest (A,n,k)
2  {
3      //A is an array of size n
4      // K represent kth  order of smallest element
5      Create MaxHeap(A,n).
6      for i:=1 to i<n-k do
7      {
8          if(A[i]<A[0])
9          {
10             A[0]=A[i];      //A[i] is new root
11             Heapify(A,0);  //heapify function to buildheap
                  property from root node
12         }
13     }
14     return A[0];
15 }
```

In the above algorithm, we can observe lime no. 5 "Create MaxHeap(A,n)" takes $O(k)$ time to build a max heap from first k elements, line no 11 heapify function healps us to maintain heap property at root into $O(\log_2 k)$ times.

So, total time = Building max heap of k elements + inserting (n+k) elements into heap (Worst case (n-k)times heapify)
$=O(k)+O((n-k)logk = O(k+(n-k)logk)$

1.4 Algorithm 4: To find k^{th} Smallest element by duplicating an array and removing k-1 elements which is smaller

```
1  Algorithm with smallest (A,k,x)
2  {
3      //A is an array of size A[1;x]
4      // K represent kth  order of smallest element
5      for i:=1 to x do
6      {
7          temp[i]:= A[i]
8      }
9      for i:=1 to k do
10     {
11         e:=min[temp]
12         delete(temp,e)
13     }
14     return e
15 }
```

In the above algorithm, we can observe replication of an array will take time $O(n)$ form line number 5 to 8. While "for" loop at line 9 will take only $O(k)$ and at line 11 it will take $O(n)$ for one call, so there are n such call,so it will take $O(nk)$. From line 9 to 13 in worst case k might be equal to n. so we can conclude in worst case the algorithm will take $O(n^2)$

1.5 Algorithm 5: To find k^{th} Smallest element using binary search tree data structure implementation

```
1  Algorithm Searchk(k)
2  {
3      found:= false; t := tree;
4      while((t!=0) and not found) do
5      {
6          if (k = (t->leftsize)) then found := true;
7          else if (k< (t->leftsize)) then t:= (t-> Ichild);
8              else
9              {
10                 k:= k- (t->leftsize);
11                 t:= (t->rchild);
12             }
13     }
14     if (not found) then return 0;
```

```
15      else return t;
16  }
```

In the above algorithm, we can observe in the line number 4 the "while" loop will run for $1/2+1/4+1/8$..... as binary at each level get divided into two branch so the time complexity will be $O(n \log n)$

1.6 Algorithm 6: Using Quick Sort partition

```
1  Algorithm KthSmallest_QuickSelect (A,l,r,k)
2  {
3  //A is an array of size A[l..r] containing r-l-1 element we have
      to find an element belongs to A which is greater than k-1
      elements of the array
4  if(l==r) //contains only one element
5      return A[l]
6  pos:= Partition(A,l,r)
7  // partition the array A[l..r] into two subarrays A[l... pos-1]
      and A[pos+1 .... r] pos is the index of pivot returned by
      partition
8  i:= pos- l + 1;
9  if(i=k) then
10     return A[pos];
11 else if (i>k) then
12     return KthSmallest_QuickSelect (A,l,pos-1,k)
13 else
14     return KthSmallest_QuickSelect (A,pos+1,r, k-i)
15 }
```

Using Quicksort Partition algorithm to partition around the k^{th} largest number $O(n)$. Sort the k-1 elements (elements greater than the k^{th} largest element) $O(k \log k)$. This step is needed only if sorted output is required.

Time complexity: $O(n)$ if we don't need the sorted output, otherwise $O(n + k \log k)$.

1.7 Different possibility to design a Random algorithm to determine the k^{th} smallest element in an array.

We can design Randomize quick select algoritm for k^{th} smallest elements and it will have time complexity of O(n)