NPTEL
NPTEL ONLINE CERTIFICATION COURSE

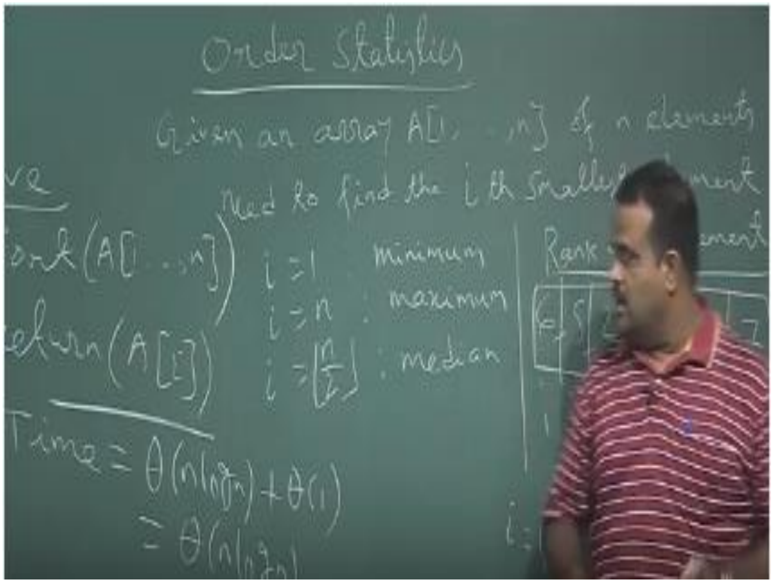
Course Name
Fundamental Algorithms:
Design and Analysis

by
Prof. Sourav Mukhopadhyay
Department of Mathematics
IIT Kharagpur

Lecture 08: Order Statistics

Hi, so we will talk about order statistics.

(Refer Slide Time: 00:26)



So the problem is to find, we have given a array, array of n element and the problem is to find out the ith smallest element need to find the ith smallest element from this array. So if i=1 this is the minimum and for i=n this is maximum and for i=n/2 I mean either lower ceiling or upper ceiling in odd or even number, so this is called media.

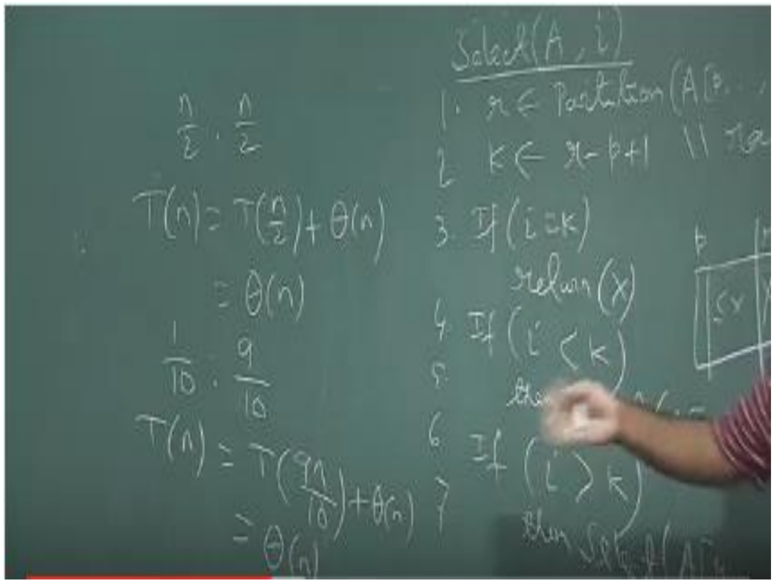
So this is our problem, how to find out the ith smallest element of an given array okay. So we define the rank of an element in array, rank of an element means the position of the element in a sorted array. Suppose we have the number like this 6, 5, 2, 1, 9, 8, 7 suppose this is our input, this is an array. Now if we sort this array we will get 1, 2, 5, 6, 7, 8, 9. So if we have sorted the rank of 6, rank(6)=4 that means the position of that element in a sorted array.

And if we are trying to find out the fourth smallest element, fourth smallest element and that should return as 6. So for the ith smallest finding the ith smallest element means we are trying to find the element whose rank is i. So how we can do that, the naive approach is, we can sort the array first.

And the after sorting we can go to the ith position and we can return the A[i] after sorting. So this is, so what is the time complexity for this, so we are applying sorting algorithm, so if we use the merge sort or heap sort which is nlogn and then this will get constant time, so it is basically nlogn+θ(1) so this is basically order of nlogn okay.

So now we want to do something better than this, actually we want to do a linear time algorithm for finding the ith smallest element. So how we can do that, so we will try to use, we do not make to sort all, because we can just, because we do not, if we sort we will get all the smallest element like this, but we do not need that we are only trying to get the ith smallest element. So what we do, we will apply the partition algorithm of quick sort okay.

(Refer Slide Time: 04:11)



So suppose this is our select we have given array and we are trying to find out the ith smallest element. So now what we do, we just call the partition algorithm of quick sort say our array is here P point to Q okay. So it will return, so it will store the pivot element which is basically A[P] because our original partition algorithm we take the pivot and meant as the first element of the array which is basically A[P].

Now it will store this X and all the elements about here is less than X all the elements about here is greater than X. So we defined this k is equal to p-r-1 sorry +p- sorry this is r-p+1 okay so this is basically nothing but rank of this x okay so it is returning the r position of that array and array is starting from p so the position of this element is r-p+1 that is the rank of the pivot element x now so we are looking for highest smallest element so if I is equal to k then we are done we are lucky return just x that is the highest smallest element we are looking for .

Else if i is less than k that means we are looking for a small element with whose rank is less than rank of this pivot element so that element must be here so that is divided state so now we have to look for that element in the sub array so then we call the select [A[p,…r] ,i] okay otherwise if i is greater than k that means suppose this r is say 5 and we are looking 9th smallest element, so 9th

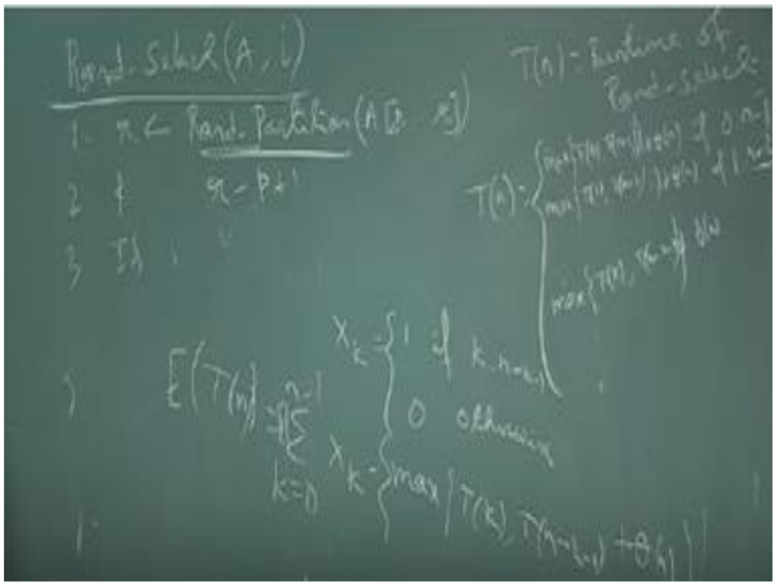
smallest will be this side this sub array so we have to look into this sub array but we i is not 9 because we have already come across with r smallest.

So we look at i-kth small is so then we call this, q[i-k] because we have seen already seen the rank of this element so after that in this sub array we have to look at the smallest element so this is the code for finding the highest smallest element we are using here the partition algorithm of week sort.

Now let us quickly analyses this code now if our pivot is a good pivot so that means if our partition is good partition so what is the time complexity they it will divide the array into this then the time will be because we are looking in one sub array only and this is the cost for partition okay and so this will be O(n) by the master method of case 3 now if it is not that much good but if it is some ratio if our partition is good in the sense that it will divide the array into very small part here and big part here then also that recurrence is like this so we are looking worst case so worst case we have to go for the longest sub array.

And this will also give us linear time complexity but what is worst case because this partition we are using now if we choose the pivot element always happening to be a minimum or maximum.

(Refer Slide Time: 09:14)


Rand-Select(A, i)
1. r ← Rand-Partition(A[p ... r])
2. k ← r − p + 1
3. If i
T(n) = Runtime of Rand-select
0 otherwise

If we then the partition is always this then recurrence is this and this is O(n2) solution which is worst then the sorting and getting the highest smallest so this is the worst case for the also, okay, so for the average case analysis we can choose the partition as a random partition and we have a like randomized quick sort we can have a randomized version of the select which is denoted by rank select okay.

So just our partition is a randomize partition we are choosing any element as a pivot so (A, A) (P, Q) this randomize partition means we are choosing any element as a pivot with probability 1/n that is the randomize version if this and then we will do the same thing k = k is the rank so k = r – p+ 1 and then if k if i= k then the remaining part is same we return x or the yeah otherwise if i<k then we called again this randomize select (A, P, R, I) otherwise.

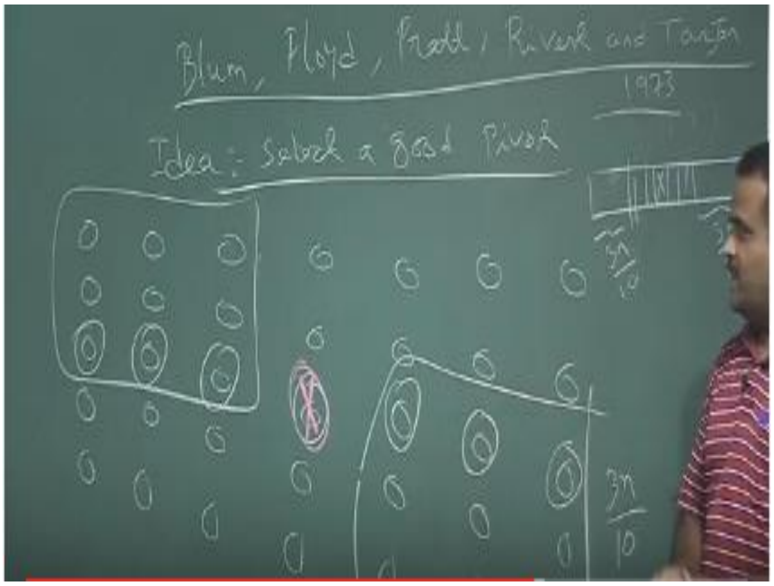
Otherwise means if i>k then again you have to call so then we call, this randomize select on the right of the array but we I = i-k, so this is the code. So this is in this way we cannot come with some input where we are choosing the minimum or maximum as a pivot because we are picking the element pivot as randomly on the run time, so this is the advantage here. Now to analyze this code let T(n) be the run time of this rand select.

Then T(n) will be of the form so if the partition it will depend on the how the pivot is, if the partition is like this if the partition is D^{n-1} if the partition is 1:n-2 like this so if the in generally if the partition k : n-k-1 sorry it will be the maximum of this two because we don't know because we are only looking for only one part so we don't know whichever is the maximum, maximum of T(0), T(n-1) this is also maximum of T(1), T(n-2) like this.

This is also maximum of T(k), T(n-k-1) + $\theta(n)$ this the maximum of this so like this, so to get the expected value of this T(n) we have to take help of the indicator random variable so x_k we defined as 1 if the partition is k:n-k-1, 0 otherwise. So then T(n) will be written as Σ x_k (maximum T(k), T(n-k-1)+ $\theta(n)$, k = 0 to n-1, now if you take the expectation, expectation of whole thing and this can be shown as by some probability calculation which is out of our this course.

This can be shown in $\theta(n)$ so expectable run time is $\theta(n)$ for this randomize portion if the select but worst case is always like this so we can always choose in the vast case the minimum or maximum element. So now we want to make sure that in the last case also we are choosing the good pivot element, okay. So that is the next algorithm which we will see in the worst case it is guaranteed linear time algorithm in the vase case.

(Refer Slide Time: 14:53)


Blum, Floyd, Pratt, Rivest and Tarjan
1973
Idea :- Select a good Pivot
3n/10
3n/10

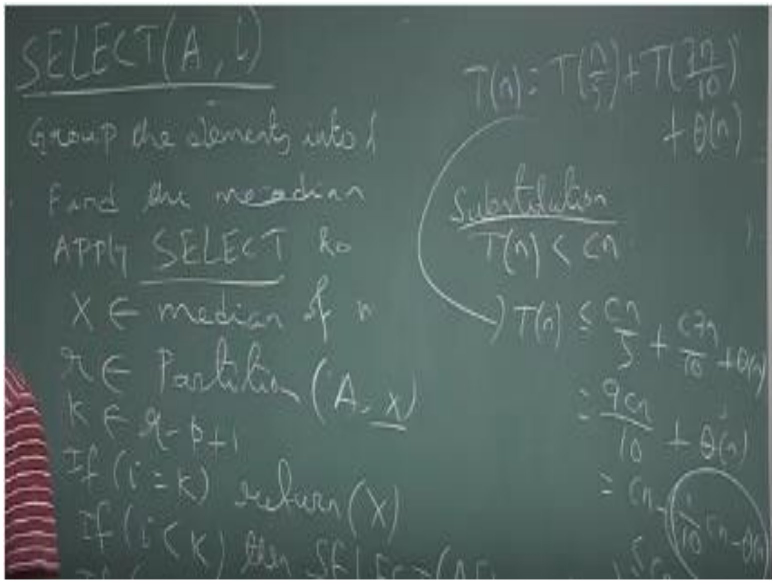
And these algorithm is by this 4%, pride sorry 5% reverse and trojan so they, they gave this idea in 1973, so the idea is to get a good pivot, I mean so that we can have good partition which will partition the array into some part over here, some over here. I mean not the 0: n-1 it should be some little facts on over here get into it, so that is the idea, so idea is to choose a good pivot select a good pivot. Okay, so for that what we do, we just we have array of size n we group that into five elements group like this, so like this five elements group maybe last we have all the three four elements like this,

okay, we have n element so we are grouping them this into five elements so roughly there are n/5 groups. Now we choose the median of each group, and that we can do just your five element we can choose the median that is by constant time we can do. Now after this we choose the median of this media and that is our pivot element X and that media find the medians of the medians we will use the same algorithm recursively, select algorithm we will come to now, we will see the select 0 for recursively and now these we are going to choose as pivot.

Now if we choose these element as a pivot what is the advantage? The advantage is all the elements of over here are less then X, because all the elements they are basically less than these element and they are less then these elements all the elements over here are less than X and all the elements over here they are greater than X, so how many are there so there are n/5 groups so among this, this is half of so n/10. So basically 3n/10, so 3n/10 elements are less than X guaranteed.

And similarly here, 3n/10 elements are get up the next guaranteed, so that means where is X so this is 3n/10 and this is 3n/10 so X will be sitting somewhere here and this is guaranteed so that means we can ensure that we can guarantee that they are some portion here which are less than X they are some portion which are greater than X, so that is the good pivot that means we can ensure that it is not 0: n-1, so we can guarantee that little portion will be here and remaining portion will be here so that is the way we select the pivot element. Okay, so let us write the code for this c2 code for this.

(Refer Slide Time: 18:45)



```
SELECT(A, i)
  Group the elements into 5
  Find the median
  Apply SELECT to
  X ← median of n
  r ← Partition(A, X)
  k ← r - p + 1
  If (i = k) return(X)
  If (i < k) then SELECT(A, [illegible]
```

$T(n) = T(\frac{n}{5}) + T(\frac{7n}{10}) + \theta(n)$

Substitution

$T(n) < cn$

$T(n) \le \frac{cn}{5} + \frac{c7n}{10} + \theta(n)$

$= \frac{9cn}{10} + \theta(n)$

$= cn - (\frac{1}{10}cn - \theta(n))$

$\le cn$

Okay so we call these as SELECT so A is an array and we are trying to get the ith smallest element of this array. So what we are doing, we are grouping group the elements into five elements, five element groups and then finding the, find the median of each group median of each group and so we have some median so n/5 medians so now we have to find the medians of the medians using the same select algorithm. Apply recursively select to find the median of medians, so we have n/5 medians so from that we have to find the medians. Okay, so we are using the same algorithm recursively now we got the medians and that is our pivot.

That medians of medians, these are pivot. Okay, now we call the partition algorithm using this as a pivot maybe this is the pivot element I mean the, we know the position of that. Okay, so now we choose the rank of this element, now the remaining part is very same. So k = r – p + 1 this is the rank of this pivoted and then what we do we do the same thing as or say to us if I = k then we return X else if i is less than k then we call the select on the left part of the array with the same i otherwise if i is better than k this then we up to look at the this is the dividend conquer approach you have to look at the right part of the array you but with so i – k so this is the code.

For so this is now what is the runtime of this code up this so runtime so we are grouping the element into five element group so and then your finding the media so where their roughing your spending consent time and then where applying the select algorithm to find the medias of the media do their size of the element is n/5 and then we are calling a partition ϴn and then this part is same as the select and this part here we are so this is T n/10 so in the washed case we are assuming that x is sitting here and here all the elements are 37n/10 and in that seems well looking at.

Oscs analysis so washed case we approve look at into this so this is basically p(7n/10) so the total runtime is basically what sum of all this so T(n) is basically T(n/5 + T(7n/10 + ϴn that this is the reoccurrence of this code now how to solve this reoccurrence so we can use the recursive tree to get the solution or we can use the substation method so that means we are assuming here that T(n) well looking for a linear time algorithm we are assuming T(n) is less then cn and this is true for all n all k up to n.

So will use this here so we got T(n) is will use this less than cn/5 + c7n/10 + so this basically so this nine 9cn ϴ of n so this is cn – 1/10 cn – ϴn so we can choose c, c is a constant which is in our hand so we can choose c begin up so as that this will re positive so if this is positive then this is less than cn so done so this a this is a linear time algorithm where but this, this can be all can also see by the help of reoccurrence tree where this yeah this we can get this.

Linear time algorithm so now the question is whether this is better lot that randomize version of the select is better so these has so many steps like we have to partition and we have to but this is a granted this is a granted washed case linear time select algorithm okay, but in the randomize portion if the select it is all though it is not granted but we can I mean we it has been absorb that it is giving some better dissolve so we will prefer to use the randomize version of the select all though it is in the washed case it is not good, thank you.