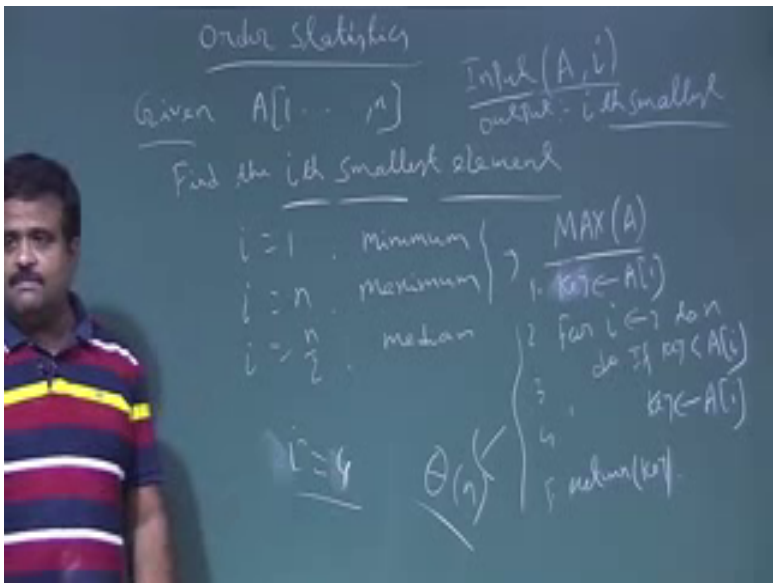
An Introduction to Algorithms
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture – 18
Order Statistices

(Refer Slide Time: 00:26)



So, we talk about order statistics. So, this problem is basically we have given some numbers given an array of an element and the problem is to find the i th smallest element that is the called order statistics finding the i th smallest element we have given n numbers and we need to find the i th smallest element. So, this problem is called order statistics problem or this is called order statistics now if i is equal to 1; this is called minimum. So, for i is equal to 1 means first smallest element.

Which is basically minimum i is equal to n this is called maximum and for i is equal to n by 2 it is called median provided n by 2 is A n is a even number otherwise we can put a ceiling or n plus n minus 1 by n plus 1 by 2 anyway. So, if i is so it is basically the median. So, now, the question is how we can solve this problem in general. So, for any i say suppose for these 2 problem it is quite simple. So, how we can find the minimum or maximum? So, you have given an array

So, what we can do we can just scan the array to get the minimum or maximum; so, very simple code. So, suppose this is a maximum finding max of A. So, just fill line code. So,

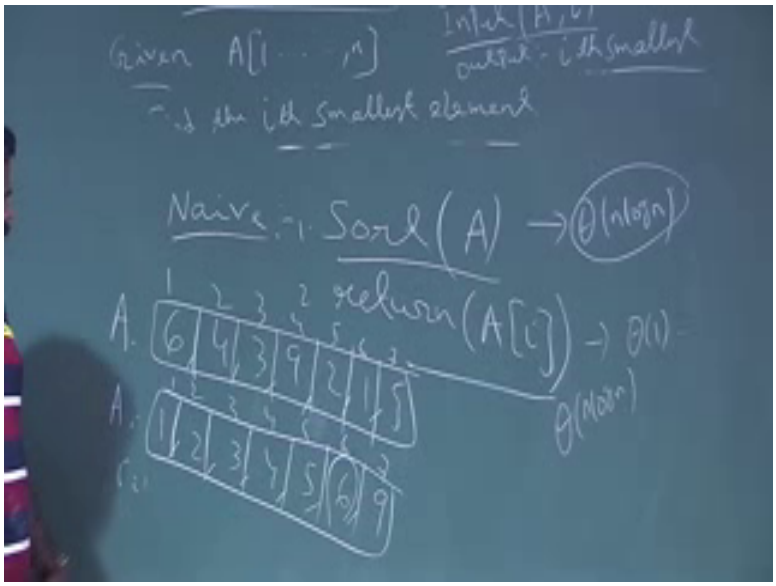
we start with for i. So, we have we are assigning this key; key value is basically A 1 and then we are having a for loop for i starting from 2 to n we compare with the key do if key is. So, we are finding maximum do if key is less than A i then we replace key by the new A i; that is it and finally, after this loop we just return the key as a maximum very simple code.

So, you basically we need to scan the array basically need to scan the array and every time if the element is greater than that what we have candidate of the maximum then we replace that. So, this code will take how much time this is a for loop of size n this is a linear time order of n, now this is for maximum or minimum now suppose we want to find out the second minimum second smallest element say i is equal to 2 then can we extend this code or say it could be difficult to extend this code even for i is equal to 3 or 4.

Because we may need to store these and then again compare this, this is this simple code will not extended i mean for this for any value of i in general. So, i is also part of the input. So, basically the order statistics problem is to we have A array and input is basically A array input is an array and A i and output is basically i th smallest element i th smallest element. So, now, what is; how we can solve this problem in general like for any i. So, the naive approach is we can sort this numbers.

And then we can go to the i th position and we return that value that is it. So, that is the naive approach what is the naive approach. So, we can just sort this array using any like sorting algorithm if we restrict our self as a; I mean if there is no if we have to use the comparison based sort because it is using the we do not need to use the auxiliary storage for that auxiliary storage means that we do not need to bound our input in a some range. So, yeah, what is the naive approach?

(Refer Slide Time: 05:13)



So, we sort it we sort A and then we return A i after sorting. So, that is our i th smallest element we sort the elements we sort the array and then after sorting. So, suppose this is yeah. So, suppose this is the input 6 4 3 9 2 1 5 suppose this is the input this is the array A array 1 2 3 4 5 6 7; there are 7 numbers. So, now, suppose we want to find out the say fifth smallest element. So, what is our naive approach we sort this. So, once we sort it the array will be like this 1 2 3 4 5 6 9 are we missing any numbers 1 2 3 4 5 6 9 fine.

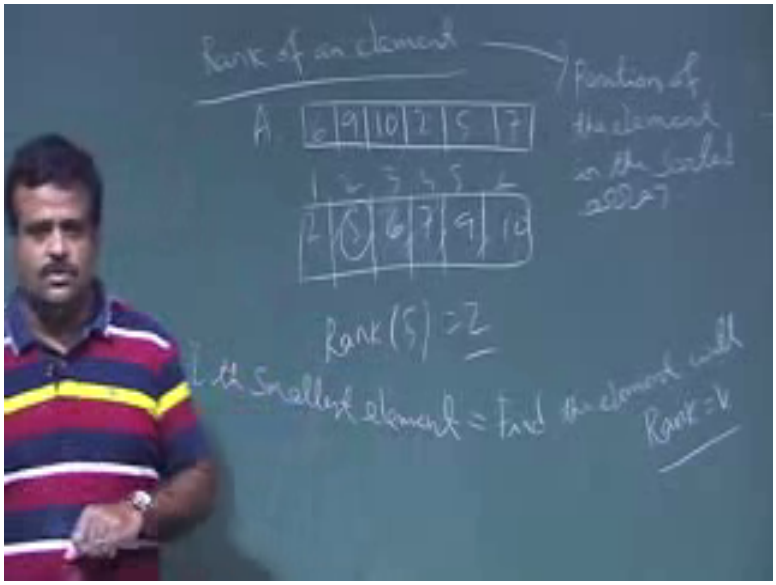
So, this is the sorted one after sorting with this 1 2 3 4 5 6 7. So, if we want to find out say 6 smallest element i is equal to 6 then we just return this one A 6 after sorting. So, what is the time complexity? So, this will take linear time sorry this will take order of n log n because to make it linear we need to restrict our self in A range of the input and also we need to take that size of the storage and this is the basically just of return order of one. So, this will take basically order of n log n provided we use a sorting algorithm which is n log n I mean heap sort we can use even merge sort.

But merge sort we need a same size array for the margin now we want to do something better than n log n. So, that is the that is the that is the algorithm we will discuss today how we can do better than n log n because we do not need to sort I mean sorting is something extra we are doing because we just ask for the i th smallest element i also is an input we do not need to really. So, if we sort then we can find all the smallest element

i is i for 1 to n. So, that that we do not need really we just need the i th smallest element for a particular i.

So, how to do that? So, let us define rank of an element first let us define rank of a element rank of. So, we have A array given a array of some element n element say 6 9 10 2 5 say 7 we have this element.

(Refer Slide Time: 08:23)



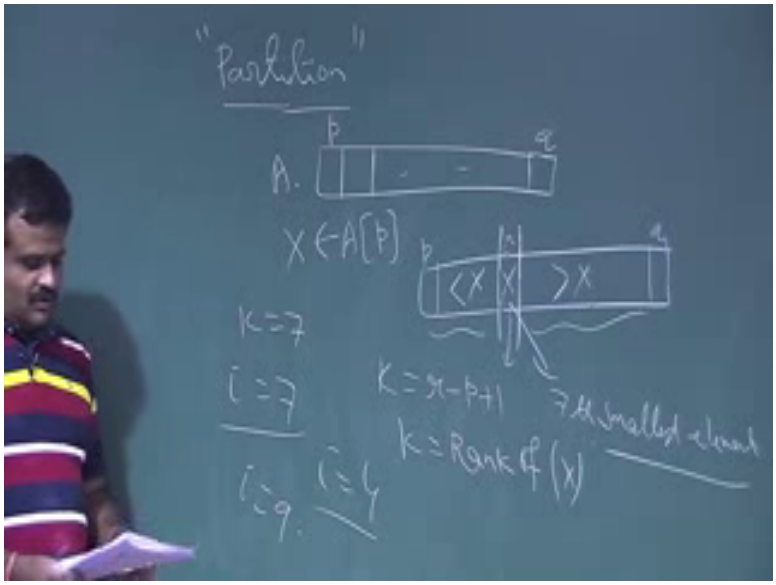
So, this is our input now what is the rank of say if I ask rank of say 10 rank of 5 rank of a element is basically position of the element in the sorted array rank of an element is position of that element of the element in the sorted array sorted array. So, if we sort this. So, what is the rank of? So, if we sort this then it is basically 2 5 7 sorry 2 5 6 7 9 10.

So, 1 2 3 4 5 6 1 2 3 4 5 6, so, if I ask rank of 5 rank of 5 means basically 2 because position of this 5 in a sorted array is basically it is in second position two. So, rank of 5 is basically 2 rank of 7 is 4 rank of 9 is 5. So, basically order statistics problem is basically finding i th smallest element means we are trying to find the element whose rank is i. So, the i th smallest element means to find an element this problem is same as to find an element find the element whose rank is i the element with rank i rank equal to i. So, finding the i th smallest element is same as we are looking for a element whose rank is i.

So, how we can do this other than sorting and going to the i th position and get it. So, that we have to discuss. So, any anything; anything; any sub routine you can think for

which can help us for this I mean we do not need to sort the complete array, but anything any quick sort sub any algorithm or any sub routine we use in quick sort that could help us yes partition. So, let us talk about that how we can use the partition sub routine which we use in quick sort to find the i th smallest element, so, our partition; partition sub routine in quick sort.

(Refer Slide Time: 11:41)



So, what we had what is the partition sub routine basically just to recall suppose we have A array; A array say p to q. So, what we are doing in partition. So, we choose this A p as a pivot element first element we choose as a pivot element and after partition what we are doing we are dividing the array into 2 sub array we are putting the pivot element in this position and all the elements over here is less than x all element over there is greater than x and this is this is the position we are returning we are returning the position of this r if this is p to q and in the partition.

So, this is basically partition call we are taking the first element as a pivot our original partition algorithm we can take any element as a pivot, but for; we are taking first element as a pivot and then after partitioning what it is doing it is dividing the array into 2 sub array and here it is putting x in such a way such that the left sub array all the element must be less than x and the right sub array all the elements must be greater than x and x is in correct position.

So, x is basically p minus x is basically in the r th position now; now if we define say r k is equal to r minus p plus 1 then can you tell me what is the meaning of this k we just defined r minus p plus 1 this is basically the our array starting from p to q. So, this is the position of this is the rank of k rank of x. So, k is basically rank of x because if it is sorted in the sorted array x is in right position x is in correct position. So, what is the position of x if this is starting from 1 1 it is starting from p.

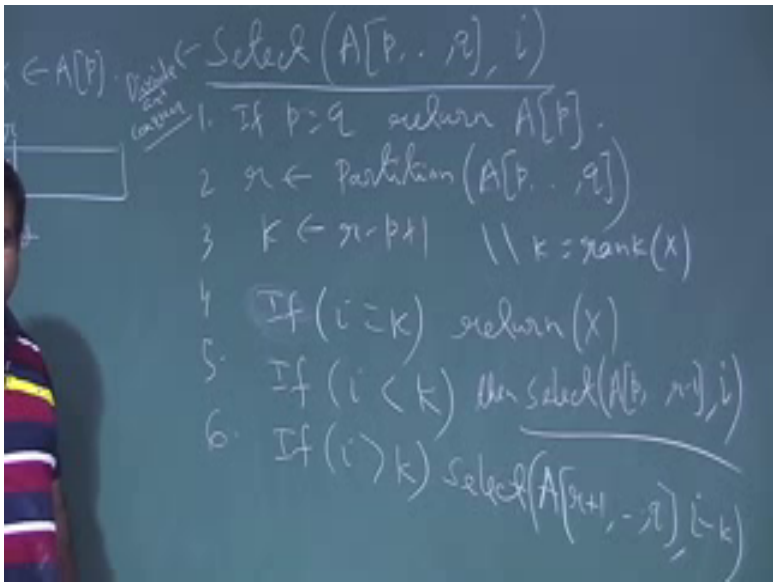
So; that means, r minus p plus 1 if it is starting from 1 if p is equal to 1. So, this is basically r th position, but our array is general array our index is start from say p to q. So, if p is 1 if our array is starting from 1 to n p is 1 then this is cancel out it is r. So, r is basically the rank of the x because x is in correct position when you sort it x position is not changing in the quick sort we sort this we sort this x is in correct position. So, that is the divide and conquer we use, but not we are going to sort this. So, what we do now, now if we are looking for the k th smallest element then we got that is our x.

Now, suppose if k is say 7 suppose our pivot element this is 7 rank is 7. So, this is our seventh smallest element suppose this is our seventh smallest element k is 7. Now suppose we are looking for i is equal to 7; seventh smallest element then we got i straightaway, we stop and happy now suppose we are not that much happy lucky suppose we are looking for i is equal to say 4 we are looking for fourth smallest element we know this is seventh smallest element so; that means, fourth smallest element must be here because this is the seventh smallest.

So, now we call the same function, we look at the same this is also divide and conquer technique we look at the fourth smallest element in this sub array with i is equal to 4. So, this is also divide and conquer technique we will we will write the code now suppose we are looking for say ninth smallest element now we know this is the seventh smallest element. So, ninth smallest element cannot be this side this sub array. So, we need to look at the ninth smallest element here.

But we have already reach the seventh smallest element. So, we will look at the second 7 9 minus 7 there is a second smallest element in this sub array. So, this is the divide and conquer step. So, we do not need to sort completely, but we will take use of this partition sub routine.

(Refer Slide Time: 16:53)



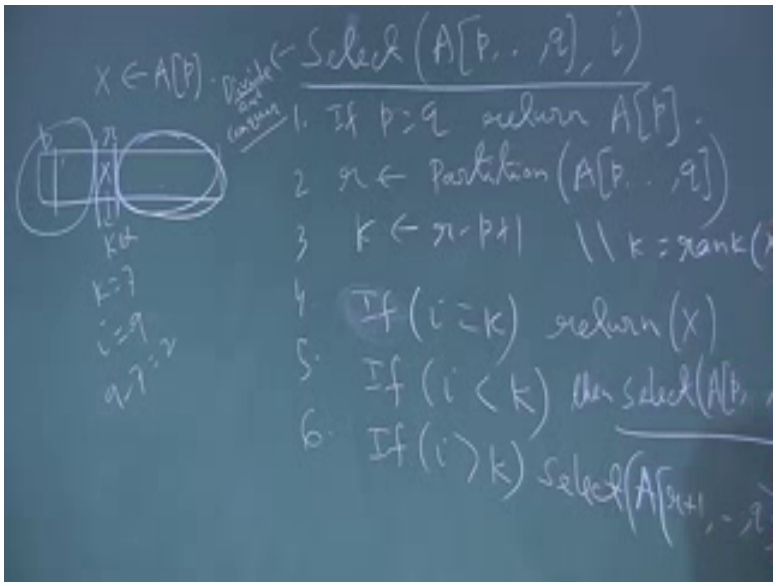
So, let us write the code for this. So, this is we call select algorithm. So, this we call select. So, what is the input; A; which is starting from p to q and i.

So, now, if p is equal to r q; we return, we stop, we return A p otherwise what we do we call the partition we call our partition algorithm p to q. So, it will choose the first element as a pivot and then it will just divide the array into 2 sub array 1 is. So, it will choose. So, x is basically A p and now this is the p th 1 and it will return this r now we take k is equal to r minus p plus 1 and this k is basically rank of x.

Now, if we are looking for k th smallest I mean if i is equal to k i is equal to k then we just return x p turn x that A p I mean that pivot element basically happy lucky case I mean otherwise if i is less than k then we again call this select on this sub array. So, suppose this is the k th smallest and our i is less than k sub. So, we have to look at this sub array we call this is A divide and conquer technique. So, this select is A divide and conquer approach. So, divide and conquer divide and conquer approach. So, if i is less than k.

Then again we call this select on the left sub array p to r minus 1 with this i again we call this; this left sub array with p to r minus 1 and with i else if i is greater than k then again we have to call select in the right sub array r plus 1 to q with A new i and that i will be i minus k because we have already seen the k th smallest and then suppose if k is equal to say 7 and if you are looking for ninth smallest; that means, if if this is 7 say.

(Refer Slide Time: 20:01)

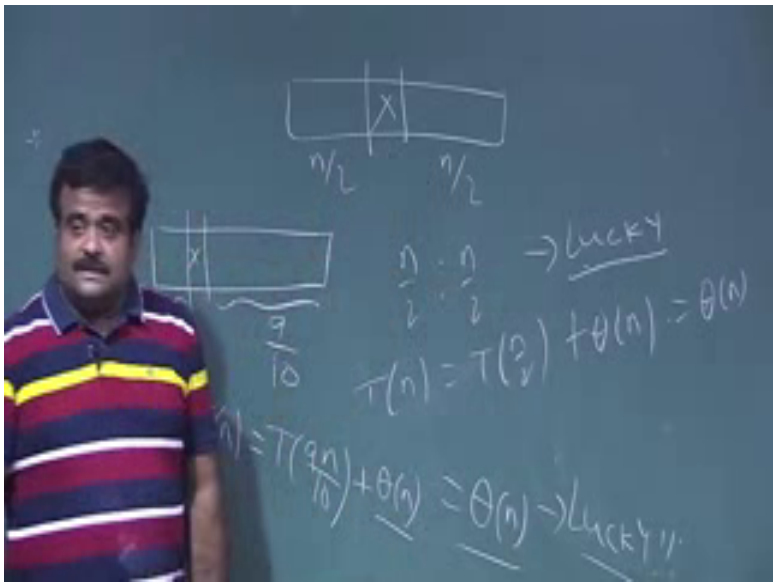


And if we are looking for 9 smallest; that means, we have already reached to the seventh smallest here.

Then we have to look first 9 minus 7 second smallest in this sub array. So, we call this; this is the conquer step. So, this is the code this is the select code sorry select; s e l e c T select this is the select code. So, here basically we are using the partition algorithm of quick sort. So, what is the time complexity of this? So, when we will be lucky we will be lucky if we get this in this sort in that case we are calling the partition it is order of n that is it because partition algorithm will take order of n. So, this is a linear time now even if we are not lucky here. So, we are calling here.

So, it will depend on this our pivot element I mean it will depend on the partition. So, if our partition is say. So, if our partition is say half half then we are. So, it will depend on the partition or the pivot element.

(Refer Slide Time: 21:19)



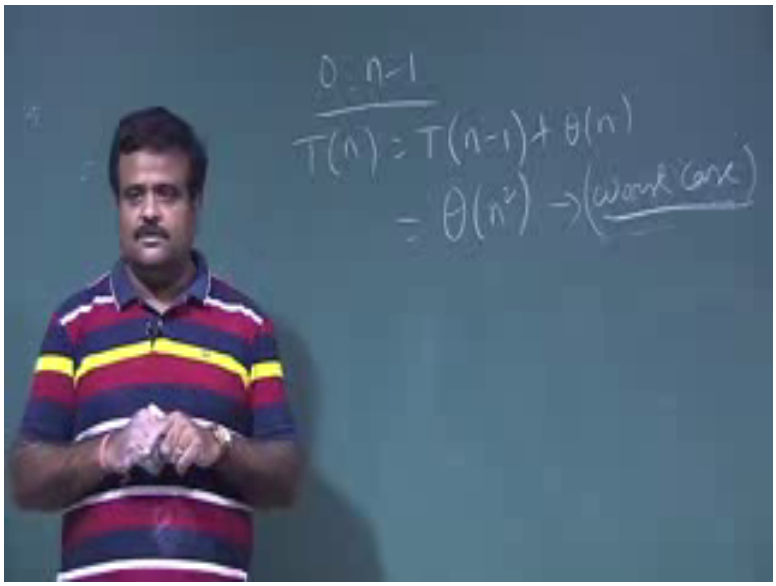
Now suppose our partition is say half half suppose x is sitting here n by 2 n by two. So, if the partition is n by 2 is to n by 2 then; what is the recurrence? So, we are having the recurrence T n is equal to T of n by 2 plus theta of n. So, this is the recurrence for this now this will give us what this will give us theta of n by again master method.

So, this is the lucky case if the partition is this one so this is the best case now we also have almost best case now suppose partition is not that much good, but suppose the partition is say some little fraction over here 1 by 10 and this is a 9 by 10 suppose almost best case this one if the partition is this. This type of analysis we did in the quick sort if the partition is this then also what is the run time complexity then the time complexity is basically in the worst case maybe we have to look for the bigger sub array it is partitioning into 2 sub array.

This is the smaller 1; this is the bigger 1. So, we are looking for the worst case runtime. So, in the worst case we have to go for the bigger part of the array. So, this will be T of n by 10 plus this is the cost for partition sub routine. So, this will again give us linear time by the master method, but only this if it is 1 by 100 by 99 by 100. So, if we can ensure that little fraction in one side then also we are lucky. So, this is almost best case we have seen this type of analysis in the; this is the lucky case, but when is the unlucky case what is the worst case of this algorithm select.

So, worst case is basically if we choose our pivot is minimum or maximum then what is the partition then the partition will be 0 is to n minus 1 that is the worst case and we are choosing our pivot minimum or maximum then what is the recurrence; recurrence is T n is basically T n minus 1 plus this is the cost for partition.

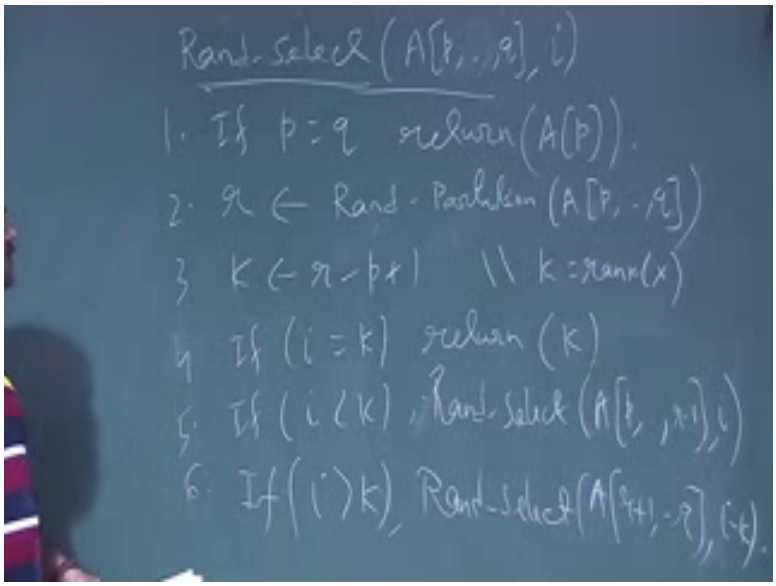
(Refer Slide Time: 23:31)



So, this is again basically theta of n square which is worst than sorting in n log n and going for the i th smallest element. So, this is the worst case worst case run time for this select algorithm.

So, now, how we can, but this is the worst case when the partition is 0 is to n minus 1 now the k now how we can ensure that like we did this type of analysis in quick sort randomized version now how we can ensure that on an average we will be lucky. So, if the partition is sometimes lucky sometimes unlucky we have seen also the idea is to choose the pivot randomly. So, this is the randomized version of this select. So, that is basically rand select.

(Refer Slide Time: 24:44)



So, what we are doing here we are basically the same code p is equal to q we return a p and then we are just basically using a randomized version of the quick sort sorry randomized version of the partition where we are choosing the pivot element randomly. So, that may give us with a hope that it may we may be lucky sometimes it may choose the good people basically. So, let us talk about this rand partition a comma. So, this is say a is starting from p to q and then i. So, randomized partition a comma p to q and then the remaining part is sent r minus p plus 1.

This is the k is the rank of x and then we just do the same thing if i is equal to k then we return k otherwise if i is less than k then we have to call again this randomized select rand select from p to r minus 1 comma i. So, same code, but every time we have to call the randomized version of this if i is greater than k then you have to call rand select a to r plus 1 to q, but now the index is i minus k. So, this is the randomized version of that. So, in the next class we will analyze this code and we will see the expected runtime of this code is order of n linear time.

But in the original version of the select we have seen the in the worst case the partition will be 0 is to n minus 1 in that case it is the order of n square algorithm, but this we will do in the next class the analysis.

Thank you.