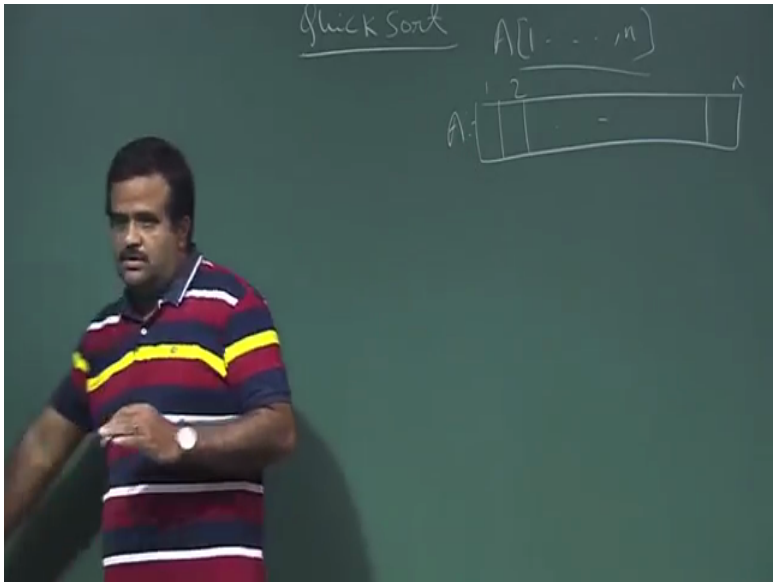
An Introduction to Algorithms
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture –10
Quick sort

So will talk about example of another divide and conquer technique which is basically quick sort.

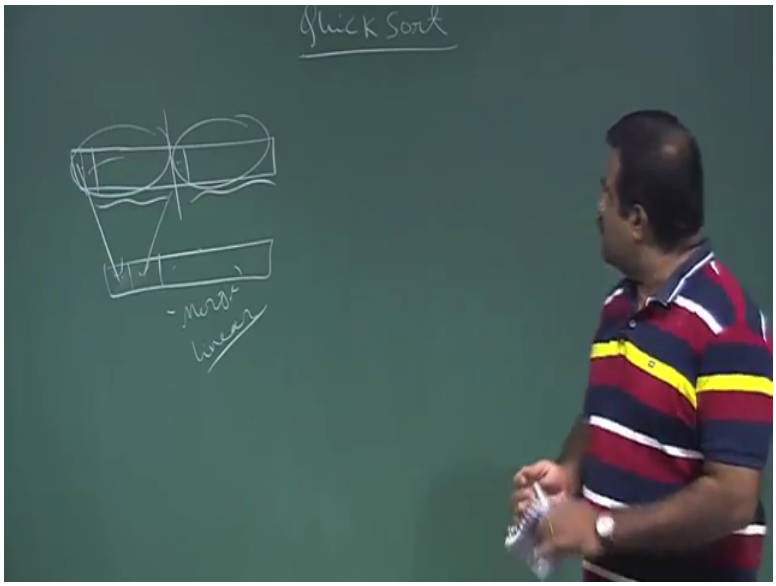
(Refer Slide Time: 00:33)



So, it is basically sorting algorithm where we have a we need to sort a array of yes this is the input we have given n numbers, the numbers have in array what could be any other form now we need to sort this number. So, this is the sorting algorithm and we have seen. So, far 2 sorting algorithm like insertion sort merge sort.

So, merge sort is one of the one of the example of divide and conquer technique now quick sort is another divide and another example of divide and conquer technique. So, it has basically 3 steps. So, every divide and conquer technique has 3 step, we divide we have given a problem of size n we divide the problem into sub problems of lesser size that is the divide step and in the conquer step we conquered the sub problem by recursively solving them, and then once we have the we got the solution of this sub problems then we combine the solution to get the solution of the whole problem so, that is the divide and conquer technique.

(Refer Slide Time: 01:43)

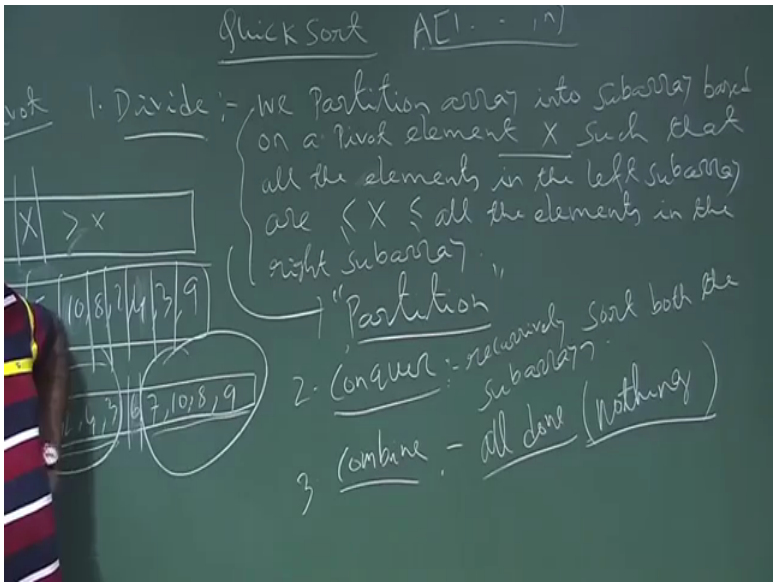


So, in merge sort what we have we did in merge sort we have a array of size n we need to sort we divide into 2 part half half, we done that is the divide step. So, merge sort divide step is very simple and the conquer step we sort this sub array we sort this sub array by recursively calling the same merge sort, then once we have the solution of these 2 sub array then this is sorted and this is sorted then we call the merge sub routine. So, this will take a array extra observably array. So, it compare the minimum with minimum.

Whichever is the minimum it will output that next minimum like this. So, this is the merge sub routine. So, that is the combine step in the merge sort. So, merge sort combine step is crucial than the divide step. Divide step is very straight forward we are going to the middle of the array we divide the array into 2 parts that is it that is the divide step, but combine step we have a merge sub routine and that is of linear time and for that we need to have a extra storage. So, that is why merge sort is not a inplace sort for merge sort we need a extra storage.

So, now we talk about quick sort which is a inplace sorting algorithm, and which is also divide and conquer technique.

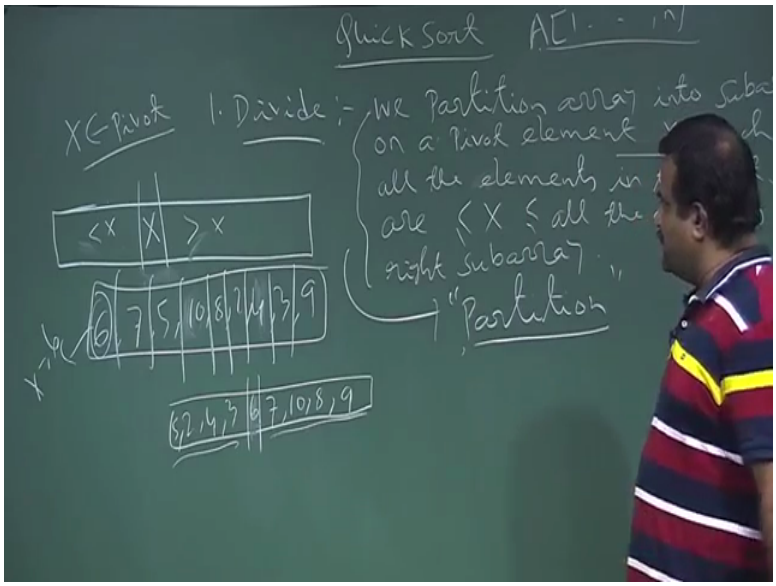
(Refer Slide Time: 03:01)



So, in the divide first step is divide step in the divide step what we are doing? We divide this is the divide step basically we partition the array, we have given a array of size n initially we partition the array into 2 sub array based on a pivot element x, x we called an element among this anyone could be a pivot, but for our patrician this is done by a sub routine which is called partition.

So, based on a pivot element x which is basically an element from this array such that all the elements in the left sub array are less than x or less than equal to x and then all the elements are in the right sub array are greater than x, all the elements in the right sub array ok.

(Refer Slide Time: 04:48)



So, basically we have this is our given array. So, what we are doing and this is done this this is done by a sub routine which is called partition.

So basically, we choose among this element we choose one element as a pivot say x is pivot element; for our partition algorithm we take first element as a pivot, but pivot could be any of these any of the element from the given array. So, we choose this pivot now in this divide step what we are doing this partition algorithm, partition sub routine it is partitioning the array into 2 part it is putting the pivot in its correct position correct position in the sense. So, if we sort this array the position of x it should be in the sorted array that is called correct position.

So, x will be sitting this correct position and all the elements in the left sub array will be less than x, if there are repetition will less than equal to x solve the element and get if there are all the elements are distinct and this will be strictly less than x this will be greater than x. So, this is we want in our this divide step. So, they did not be sorted suppose for example, if we have a array like this 6 7 5 8 I say 10 8 2 2 4 3 9 suppose this is our given array this is the input.

This is our given array say now suppose we call the partition algorithm on this array by choosing this 6 as the pivot. So, our x is 6 6 as a pivot now we want this divide step should give us. So, it should put the 6 over some where here. So, this is 6 and it should partition this array into 2 part all the elements less than 6 would be in the left sub array.

So, who are the less than 6. So, 2 4 3 should be in the left sub array and the all the element greater than 6 will be in the right sub array.

Who are the greater sorry 2 5 is also there. So, 5 2 4 6 these are the element less than 6 and 7 10 8 9 how many there 1 2 3 4 5 6 7 8 9, 1 2 3 4 5 6 7 8 9 yeah. So, this is the output after the partition call on this array. So, you have only this if you see this is this is not sorted this is not sorted, but only thing 6 is in correct position. So, you put the 6 in the correct position; that means, if you sort this array where this pivot should be x would be x is in correct position. So, this is the sub routine partition will do.

So, it will just divide the array into 2 two part the left sub array all the elements in the left sub array will be less than x all the elements will be left sub array will be greater than x. So, this is the job of the partition algorithm and this is the divide step. Now what is the combine conquer step then. So, now, we have 2 sub arrays now conquer step is what now we have to sort this sub array this sub array by recursively calling the same quick sort. So, we recursively sort both the array.

Both the sub arrays this is the conquer step; that means, now again we need to call this quick sort here quick sort here then in that quick sort call again we need to choose this as a pivot if we if we choose first element as a pivot then again it will partition into sub array like this. So, you continue and then what is the combine step. So, in conquer step once we call the conquer step, then this after the after the completion of the conquer step this is sorted array this is sorted sub array and 6 is in correct position.

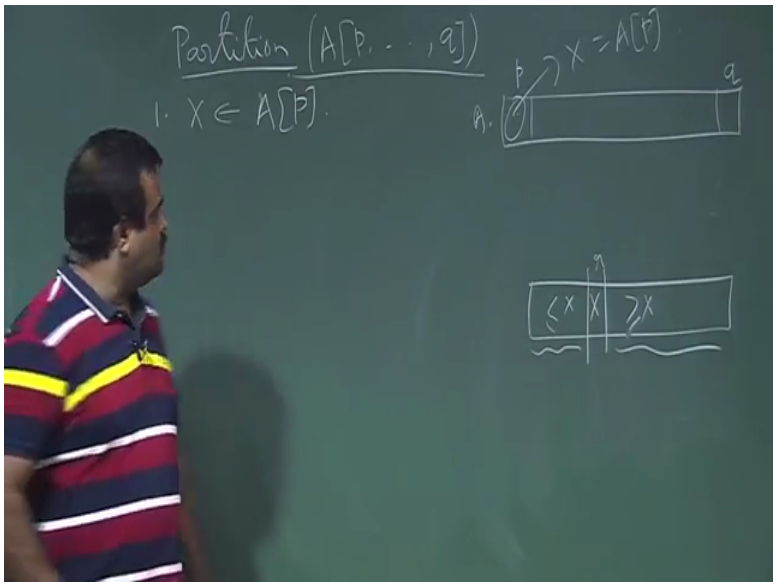
So, what is the job remaining what is the combine step what need to combined? Nothing has left all done all done. So, combine step is nothing we are done all done it is nothing has left. So, that is the combine step. So, we have we have this combine step is trivial nothing has left. Now the idea is to have a partitions of routine in linear time that is the key idea of the quick sort we need to have a partition algorithm in linear time. So, we will we will discuss that in the in a moment that what is the partition code what is the pseudo code for partition which will give us the run time linear.

So, that is the key point of the quick sort, and if we just see the quick sort this if you compare the quick sort with merge sort, merge sort the divide step was trivial we just go into the middle of the array and we divide into 2 part and conquer step we sort this part we sort this part, and the combine step was the merge sub routine. So, combine step was

more than the divide step in the merge sort, but here divide step is more than the combine, x combine is nothing in the quick sort. So, this is the divide and conquer technique for quick sort now we talk about the partition algorithm.

So, this partition sub routine how this array will be divided into 2 sub arrays such that all the elements which are less than x will go to the left sub array, all the element which are greater than x will go to the right sub array. So, that is the and that should be done in linear time. So, that is the partition code. So, let us just write the partition code pseudo code for partition.

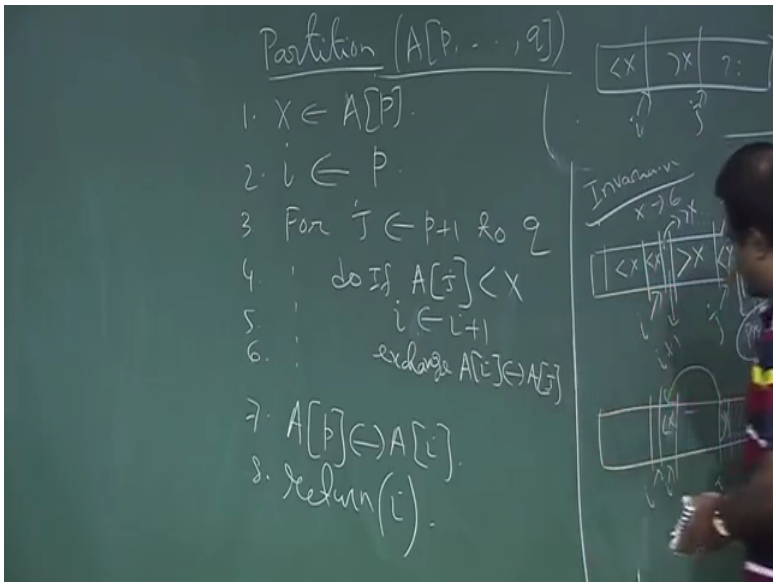
(Refer Slide Time: 11:44)



So, it is basically. So, partition which is taking an array which is p to q. So, initially it will take 1 to 8, but again all the sub array we are calling the partition.

So, it will be index would not be one and throughout the subsequent call. So, index will change. So, that is why we taking the general index. So, initially p is 1 q is n. So, we have a array which is A which is starting from p to q this is a this is a input of this partition ok.

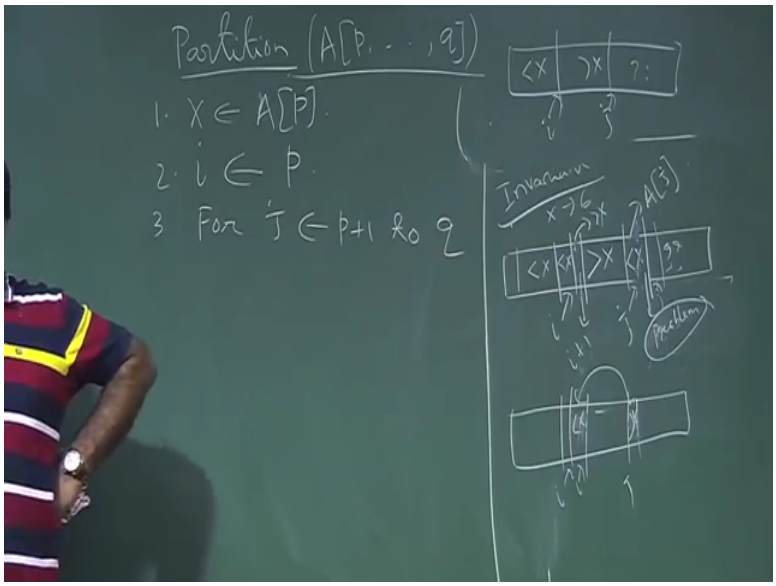
(Refer Slide Time: 12:24)



So, for our version of the partition we are taking first element as A pivot. So, A p. So, whatever the value over here this we are taking as pivot and what we want from this? We want at the end of this code it should partition this array into 2 parts such that x will be putting somewhere here and this index is r, such that all the elements over here will be less than x all the elements here will be greater than x if they are distinct otherwise if there are some equal you can use this.

So, this is what we want for our partition algorithm. So, let us write the code. So, we start with 2 index i is p and we have a for loop for J is equal to p plus 1 to q. So, J is equal to p plus 1 to q and so, this is the loop now if we to write a code it will help us if we know what is the loop invariant we are looking for in this, then it will be it will help us to write a code every algorithm every pseudo code if we know what we are looking for then it will be it will help us to write the code.

(Refer Slide Time: 13:40)



So, what is the loop invariant here we are looking for? So, we are looking for this this is the loop invariant. So, i is pointing somewhere here and J is pointing somewhere here now what do you want the loop invariant is all the elements here would be less than x less than x, less than equal to or less than if there are identical element.

And all the elements here will be greater than x, this we want all the elements here should be greater than x and this is yet to decide. So, that is the invariant. So, what is the initial step initial step is initially i is pointing here and J is pointing here and initially everything is yet to decide. So, that is the initial condition nothing is decided yet. So, that has to decide that is the initial step, but at some point of time this will be the situation. So, we take this. So, now, if this one J is varying, now if this is our A J now if A J is greater than x suppose A J is greater than x then you can increase J by 1 and that will be taking care by this for loop this is the for loop in j.

So, it automatically increase if you do not do anything. So, if A J if this is greater than x then cool, no need to do anything here. So, it will be just automatically increase J by 1 now the problem is if this guy if this is less than x if this is less than x, say suppose x is 6 and suppose this is 4 for example,. So, this is less than x now we cannot just increase J by 1. So, you need to do something. So, that what we need to do that we have to think. So, we cannot just increase J if this is say this is less than. So, this is our a j.

So, if this then we have a problem then we have a problem then that problem how we can tackle. So, then we cannot just increase J by 1 we cannot just increase J to this point by this for loop in that case we are in we are not have we are in we are not valuating our invariant. So, for that what we want we know we want to exchange this with somebody which is greater than x. So, what we know, we know the element which is sitting here that i plus 1 this is i plus 1 we know this element which is sitting over here here is greater than x, now if we just in exchange this and this.

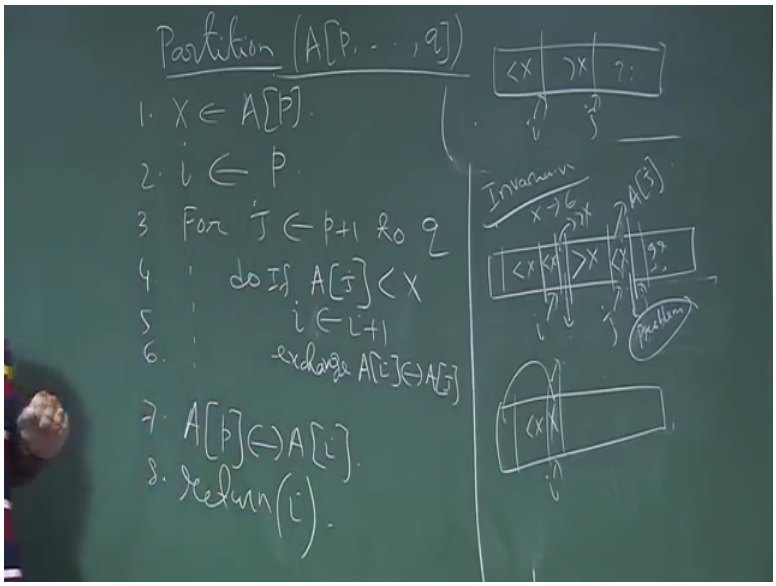
And if we increase i by i plus 1 then we have through then also we have this this was i now the which now this is J, now this is less than x and this is greater than x we know now if we increase i by 1 and if we exchange this guy with this now this x is coming here and now this x is greater than x and now this guy is greater than x and this is less than x then we have through then still the loop invariant is there loop invariant is the point where i up to i. So, this is the loop invariant we want so.

So, if this is the i then all the elements here is less than x if this is J all the elements here will be greater than x this is the way we will handle this situation. If the A J is less than x if A J is greater than x no need to do anything. So, that we have to write do if A J is less than x, if the A J is less than x then we have to do this what we do we increase i by 1. So, i will be i plus 1 we increase i by 1 now the A i is some element which is greater than x now we exchange J i with A J now we exchange A i with A J that is it now we exchange J i with A J exchange.

So, if we exchange J i with A J then peacefully we can increase J by this that will be taking care by this for loop. So, this is 5 this is 6 and then. So, this is the loop and this will do for until the J exhaust and then finally, we have 7. So, after exhausting this loop what we do we will put the pivot element in its correct position. So, that will be done. So, J exhaust now i will be pointing somewhere. So, now, where i is pointing that guy is less than equal to x. So, now, we exchange A p with A I; A p with A i and we return A i. So, that is the, that will give us the position where we have put it x.

So, this will exhaust the J will exhaust. So, J will exhaust. So, J is gone and now i is pointing somewhere here and all the elements over here is less than equal to x. So, now, J is gone J loop is exhaust.

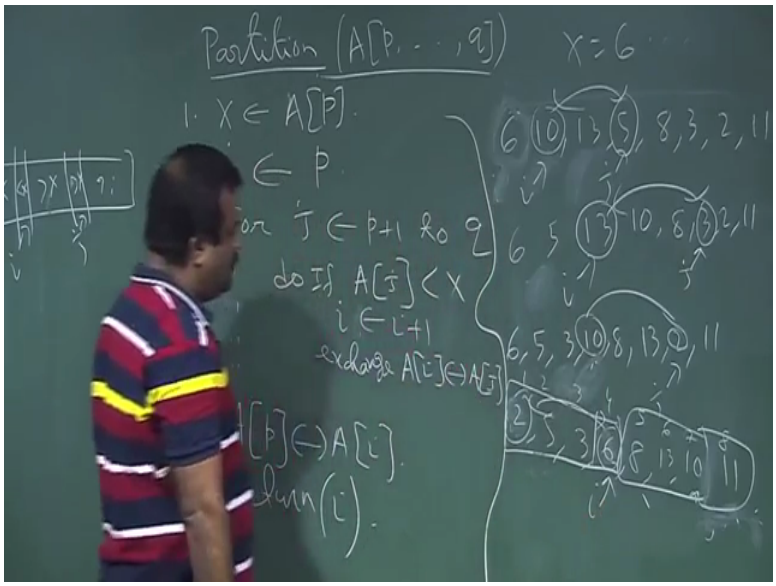
(Refer Slide Time: 19:21)



So, now we put it here. So, x will put in here. So, now, we return this this is our i we return this value of this element to indicate that where we kept the pivot element. So, that it will identify the left sub array right sub array, because again in conquer step this is only the divide step. Again we in conquer step we have to call the quick sort on this sub array on this sub array.

To if we can identify to identify the left sub array we need to have the index where have we kept the x. So, that is the point that is why we need to return i, unless we return i we do not know where we can kept x then again we call we have to call this quick sort on this part we have to call this quick sort on this part. So, this is the pseudo code for partition algorithm now let us have a example then it will be more clear.

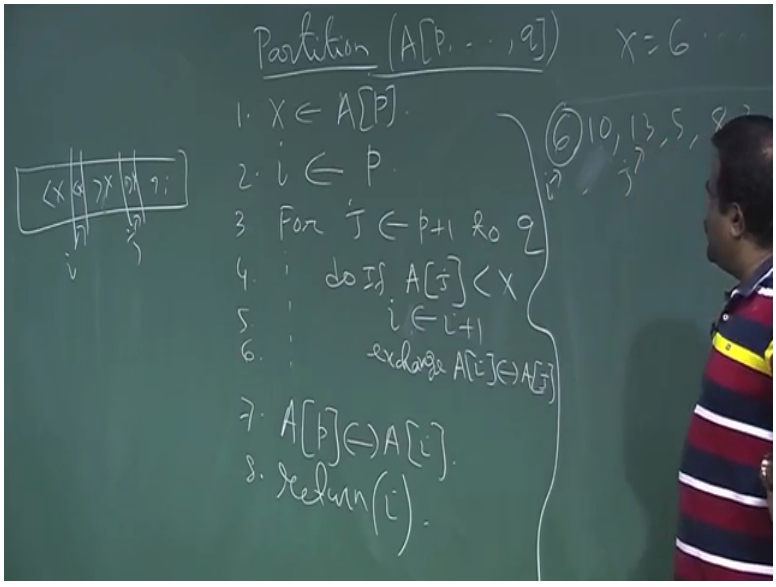
(Refer Slide Time: 20:49)



Let us execute this code one an example on an input. So, say let us take an input say example of partition.

Suppose we have given this input 6 10 13 5 8 3 2 11 suppose this is our input and we want to execute this partition algorithm on this input. So, this is our array. So, this is a p to q, now we choose this as a pivot. So, our x is our pivot is x is 6. So, this is our pivot now we point this to be i this is p plus p and we start our J loop from here, and every time we increase J by 1 by this for loop, now if we now compare and what do we want.

(Refer Slide Time: 21:53)



We want that this is the loop invariant you want we want; however, this i is pointing and wherever is J is pointing we want all the elements should be less than x all the elements should be less than x all the elements should be greater than x and this is yet to decide. So, this is the loop invariant you want. So, now, we check this this is our now A i now if A i is greater than x then they need to do anything we do will not enter into this loop then the J will be increase by 1 just by this for loop, now J is pointing here now again J is 13 A J is thirteen which is greater than x, x is 6.

So, again we increase this by 1. So, this is basically J is now pointing here now if J is pointing here now this is our A J this is our A J now we compare this with 6 now this is less than now we cannot just increase J by 1 then that will highlight our invariant. Now what to do? Now we know that the person is sitting here the element is sitting here is greater than. So, we just increase i by 1 and we exchange this with this. So, this will be like this. So, 5 will come here 13 10 then 8 3 2 11 and now i is pointing here and J is just this exchange now J will be increase by this now J is pointing 8 ok.

So, now what to do now again we compare A J with 6 now A J is greater than. So, we can peacefully increase A J by 1 and that will be taking care by this for loop. So, now, J has increased by this now A J is 3 now we got a problem now A J is say A J is 3 which is less than pivot element 6, now we need to do something what is that we increase i by 1 and we know that person the element sitting over here is less than. So, we can exchange this and this and this will give us 5 6. So, this is 3 and 10 8 say 13 will be gain there 1 11 ok now i is pointing this 3, and now J is pointing this. So, 3 and 13 exchange J is pointing this 2 ok.

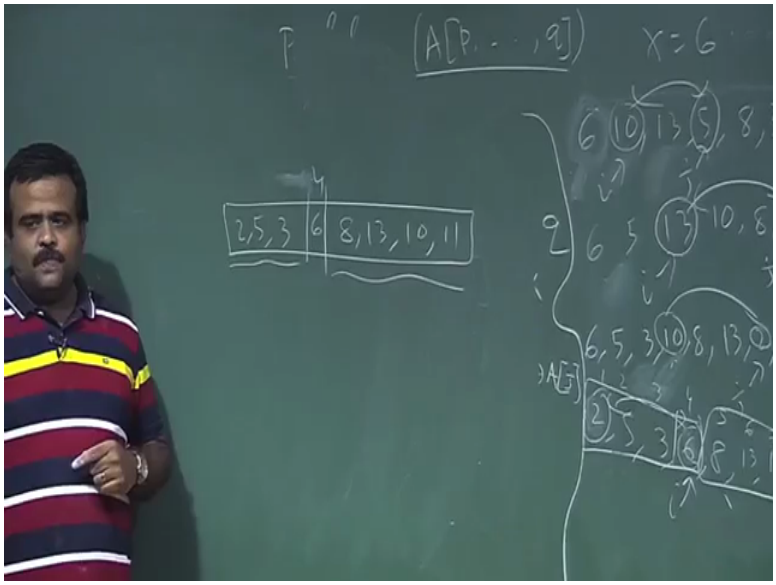
So, this is the situation now again A J is less than pivot now again we have to exchange this. So, involve the elements sitting over here is greater than. So, i is point i is increased by this and we exchange this and this. So, now, this is the 5 3. So, 2 will come here 8 13 10 will go there and 11. So, now, i is pointing this 2 and J is pointing here, now if you see all the elements up to i these are less than equal to x and all the elements up to J these are greater than equal to x.

So, this is the loop invariant you want for our this partition code, this is the exactly this is the loop invariant you want for our partition code and then we increase this J by 1. So, J will be pointing this now this is 11 so, nothing to be done. So, this is one. So, J exhausts.

So, J loop is gone. So, we cover everything. So, now, this is our i. So, i is basically this this is index if we write 1 2 3 4 5 6 7 8. So, this is basically 4, now i is 4 now what we do we have to put this pivot in its correct position.

So, we exchange this A p with A i. So, we exchange this to 6 and 2 we exchange this to then this will be 2 and this will be 6 and we return i. So, we return 4. So, this will be return as like this. So, this will return this is 6 will be sitting here and this is index 6 4 and all the element here is 2 5 3 and 8 13 10 11.

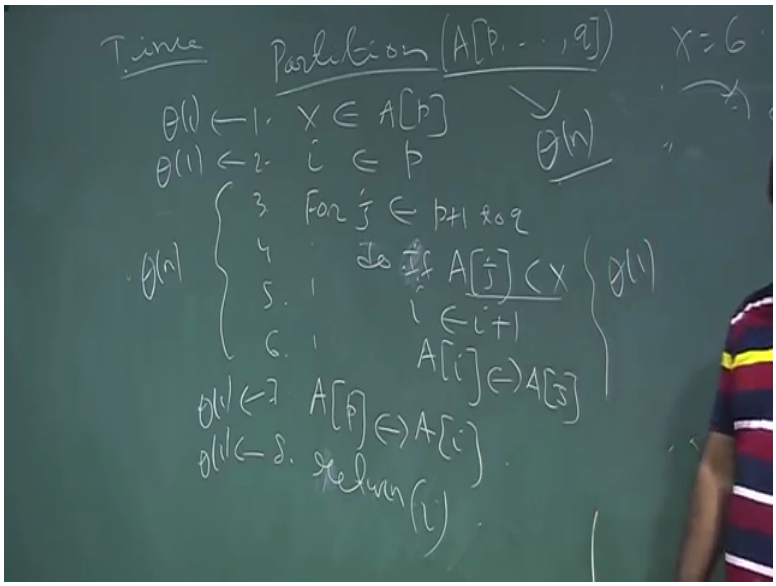
(Refer Slide Time: 26:17)



And this is the index we have to written to identify the left sub array and this is the right sub array. So, this i is the index this is say r. So, this is the i. So, this 4 is return. So, the partition will return the element 4 ok.

So, this is the partition algorithm now what is the time complexity of this algorithm, because it is it is just have a for loop for loop in j.

(Refer Slide Time: 27:07).



So, just if we just erase that. So, just if you write the partition. So, this is basically just quickly I will write this. So, this is the, for J is equal to p plus 1 to q. So, so we want to find the time complexity for this. So, if do if A J is less than x then what we have doing we are doing i is equal to i plus 1 and we exchange A i with A J that is it.

So, that will be 5 4 5 6 and then after this we just exchange A p with A i and we return we return A i. Now we want to find the time complexity for this what is the time complexity for this. So, this will take constant time this will take also constant time now this is a loop in this is a loop in size n because we are assuming order of size of this array is n q minus p is n if we have a input of size n then what is the time complexity.

So, now this is n now whatever we are doing here these are all constant, because we do not care about how much time will be taking by this checking less than then this assignment because we want this analysis to be mess in independent analysis. So, that is why we need to bring the asymptotic notation may be in our computer in our (Refer Time: 29:07) machine this will take may be microsecond, but if you have old computer it is taking more time. So, we do not bother about that; that is why these are all constant time we are taking theta 1 time for this. So, this is again theta 1 theta 1.

So, this is this is basically theta of n this is a linear time algorithm. So, this is a linear time algorithm for this partition if we have a given array of size n. Now, will talk about

quick sort how we will use this partition in the quick sort algorithm, so that will discuss in the next class.

Thank you.