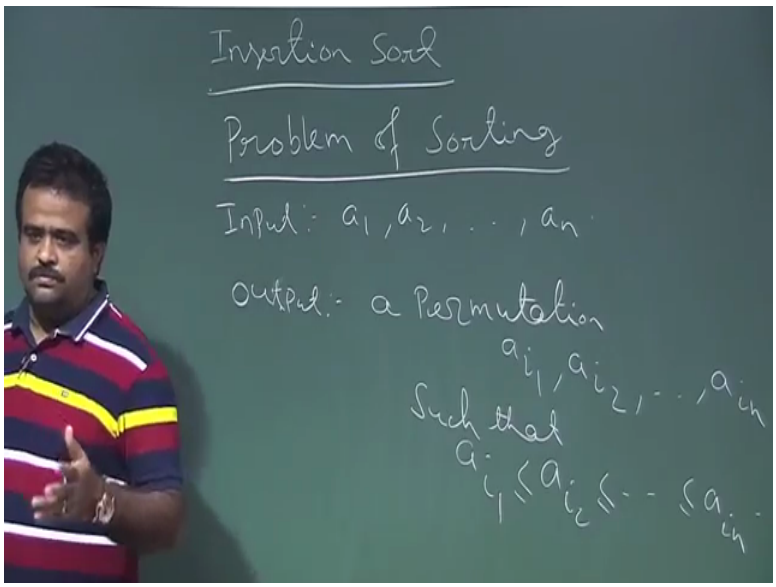
An Introduction to Algorithms
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture – 01
Insertion Sort

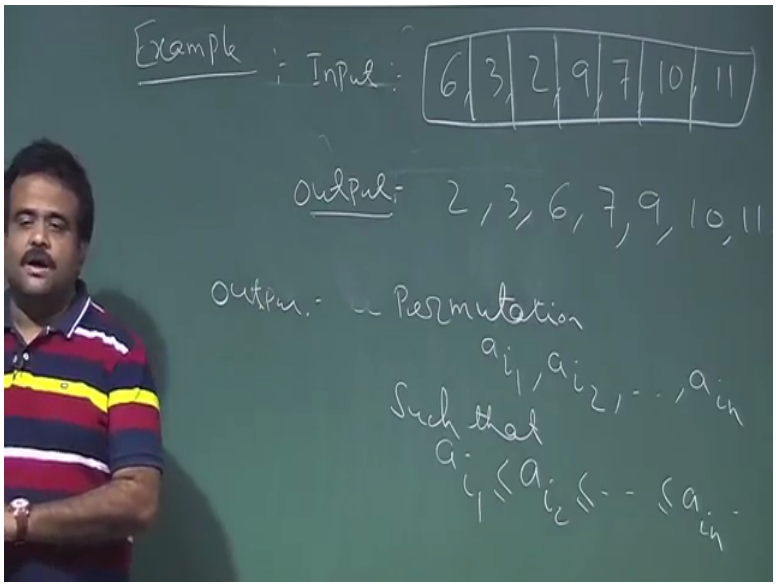
(Refer Slide Time: 00:25)



We talked about. So, we started with the sorting algorithms sorting problems. So, what is the problem of sorting? So, we have given n numbers say input is a number a 1, a 2, a n. So, typically they are integers we can deal with real numbers also in we have to deal with real number operation, but these are typically integer we have a we have given array of n numbers then the output will be the permutation of numbers, the permutation of this number A i 1, A i 2, A i n such that if it is sorting such that it is sort with.

So, there are factorial in permutation. So, among these we have to choose all of one permutation that was which is giving the sorted array. So, this will be the output of any sorting algorithm. So, this is in ascending order. So, now, for example, if we have say input like this.

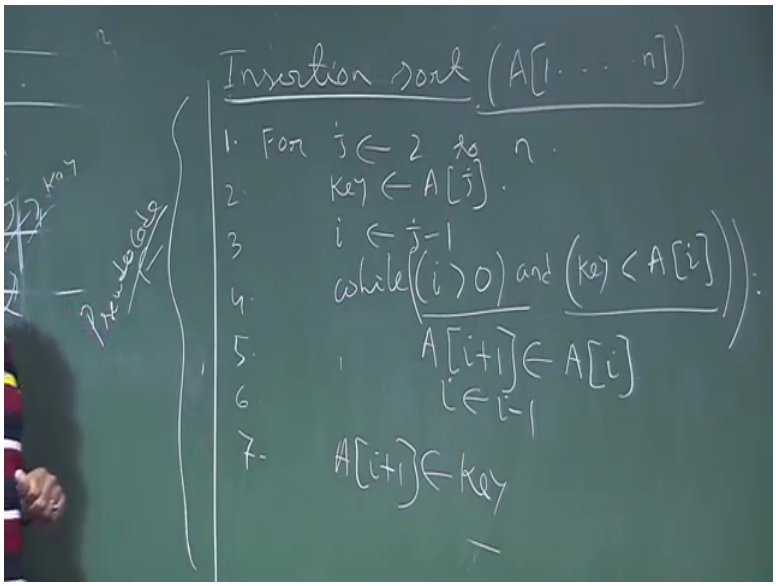
(Refer Slide Time: 02:00)



So, if we have input say 6 3 2 9 7 10 11 suppose this is our input. So, we have 1 2 3 4 5 6 7 we have 7 you have 7 elements. So, you have a array of 7 element. So, you have 1 is to 7. So, then the after sorting output will be the sorted one. So, it is basically a permutation on it. So, 2 comma, 3 comma, 6 comma, 7 comma 9.

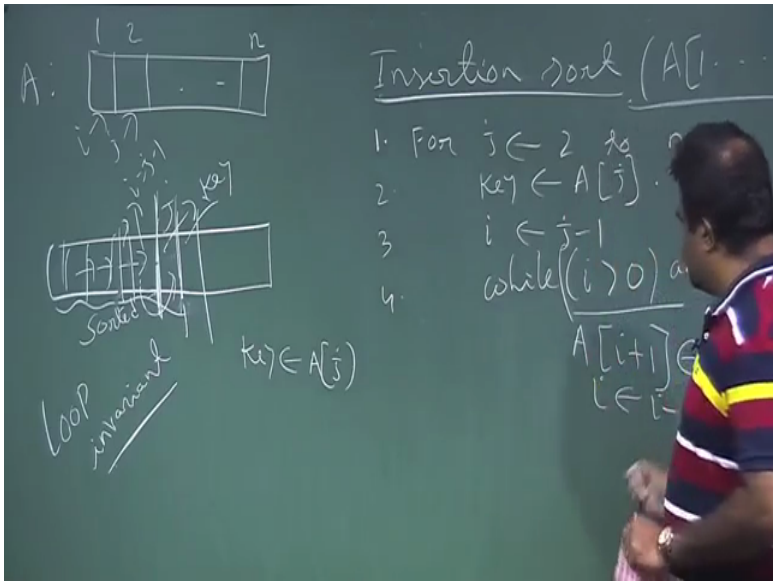
So, we should get this to be a permutation which is sorted. So, this is the output of this should be the output of any sorting algorithm. So, we will talk about, we will discuss some sorting algorithm, which we will which by which we should able to get this output. So, we start with insertion sort. So, we talk about a sorting algorithm which is called insertion sort. So, this is the name of the algorithm.

(Refer Slide Time: 03:22)



Now, what is the input? Input is n numbers. So, suppose you are taking the number is in array. So, this is the input.

(Refer Slide Time: 03:46)



So, we have a array of n numbers. So, 1 2. So, our array is starting from 1 to n, not zero to n minus 1 like in c language, you are not dealing with any specific language. So, that is why we start our array with 1 to n. So, we have a n numbers and we need to sort this numbers. So, we have to write how we will do that that description you have write. So,

that description either we can write in English form that do a we do these do this like this English form or we will write something in a compact way.

So, precise form. So, that form is called pseudo code. So, we will write pseudo code for this. So, what we do. So, we first start with a J loop, J is starting from 2 to n and then we take the key a J into the key. So, this is the A J value. So, or this loop is starting from 2 to n. So, J is starting from here and. So, this is the second step and we choice i to be J minus 1. So, i is pointing. So, i is starting with J minus 1 ok.

Now,. So, what we want, we want this is the we want at some point of time J will be pointing here and this is our key, now we want this part will be sorted this is the loop invariant the outer loop invariant we want this we want this invariant that throughout our loop this property will satisfy wherever J will be pointing, but before that the sub array up to this is sorted ok.

Now, we have to find the this is the new element this is now new j. So, now, if now this is our i. So, i is J minus 1 this is our i. Now if key is greater than A i then we can just increase J by 1 there is no problem, then J will be pointing this 1 and up to this it is sorted because this is greater than this. So, all the problem will occur if this key value if this key is less than this, then we need to find the position of this by shifting this 1 by 1 and each time we must compare with the key value with this values. So, that we are going to write and this will continue until i is equal to 0; if i is equal to 0 you must of because there is nothing to compare. So, that will write here. So, this is basically while i is greater than zero this is 1 condition and so, and if the key is greater than A i we need to do if key is greater than A i then we can just increase J and that will be taking care by this loop.

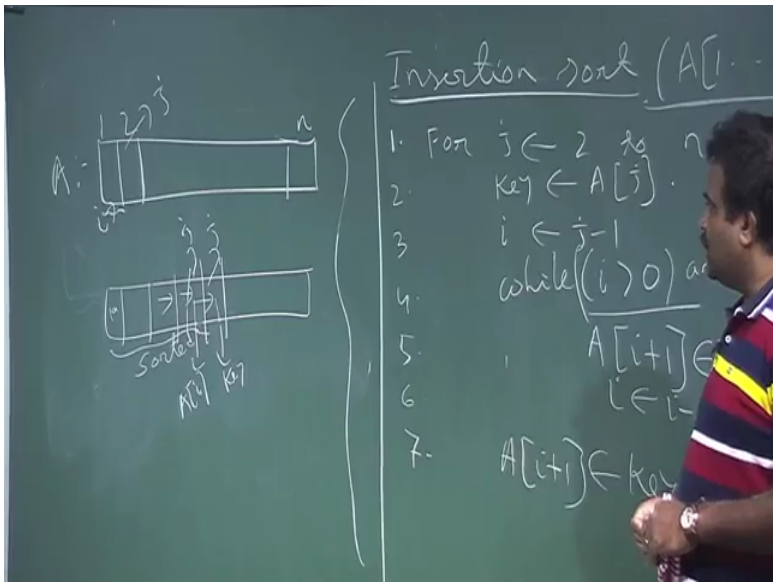
But only thing if k is less than A i then we have a problem, then what we do then we just do like this. So, we shift this by 1. So, i will be shifted this. So, A i plus 1 will be copied to A i and i is decreased by i minus 1.

So, we shift this guy here. So, we already stored this value into the key A J, now we shift this here and now i is pointing the next one. So, now, i is pointing the next one, now we compare this with the now we compare this with the i this value with the key. If the this value is greater than they we will put this key here otherwise we will keep on do like this. So, this is the step and. So, this will continue like this 5 6 and at 7 we just put A i

plus 1 we assign this value the key that is it. So, this is the; this is what is called pseudo code for this. So, the description we have written is called this is called pseudo code this is basically the description what we are doing.

So, this is not the English description, it is very precise way we just write what we want to do in this algorithm. So, in this you, so this is called pseudo code this is if you give this code to any programmer, programmer can easily runs this code in their own computer language like C, C++, Java anything. So, this is the code now as we see we want to take an example how it is running and let us just recall again what we are doing.

(Refer Slide Time: 09:37)

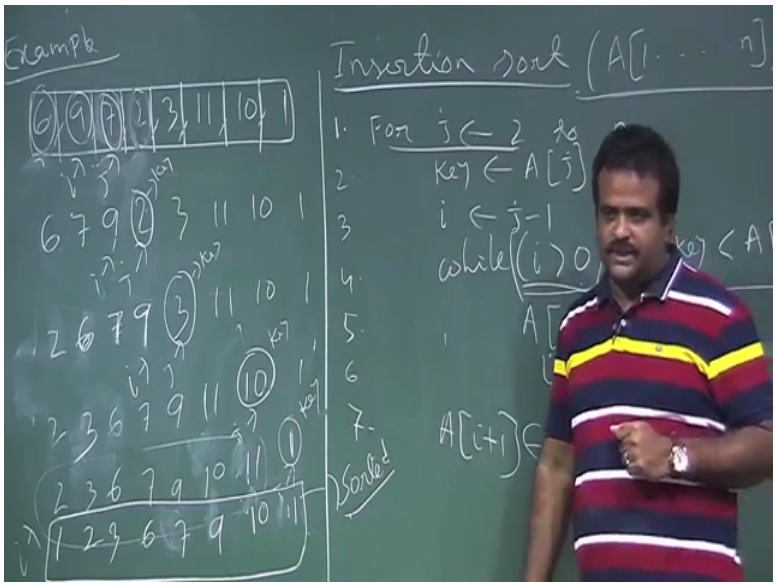


So, this is the i. So, this is 1 2 up to n this is the a array, now we are starting J with this guy this is the J and i is point here i is J minus 1 because if we start like this then this is 1 element this is already sorted.

Now, loop invariant is. So, J will be pointing at somewhere this is the J, now this is sorted this is the loop invariant now this is our key now this is our i A i, this is i J minus 1, i is J minus 1. Now if the key is greater than i then no need to do anything we can just increased J by 1 by this for loop that is it, but the problem is if key is less than A i, if key is less than A i what we do? We shift this one's position then we check the key with this if this is greater than we put key here otherwise we keep on shifting this like this until we reach to the end if we reach to the end we put the keys here otherwise we will the key in a suitable position.

So, that is the loop invariant we are we must loop in our algorithm. So, let us execute this by taking an example. So, let us take an example. So, suppose we have a given from inputs.

(Refer Slide Time: 11:10)



So, 6 9 7 2 3 11 10 and 1 suppose this is our input this is the given array and we need to sort this array, this is the given array there are how many element. So, 1 2 3 4 5 6 7 8 there are 8 elements a 1, a 2, a 8 we need to sort it. So, we need to we are going to execute this piece of code insertion sort.

So, what we do? We start with J from 2 to n. So, J is pointing here. So, this is our key and i will be pointing J minus 1 i is pointing here, now we compare 9 with 6 now 6 is greater 9 is greater than 9. So, you do not need to do anything. So, we just increase J by 1 and that will be taking care by that for loop this loop now again this is out key now we compare this with this now this is again greater than. So, we point this now this is out key now i is this 1 sorry no i will be the next point. So, i is pointing J minus 1. So, every time i is J minus 1. So, i is pointing here sorry. So, this will also change sorry. So, let us just execute from the beginning. So, this is i is pointing here. So, if first of all J is pointing here.

So, i is pointing here. So, this is greater than. So, J will be now pointing here, if J is pointing here now this is our i. Now we have a problem because this is the key and this is less than this a i. So, if it is less than what we do? We shift this 1 position this is 9 and

this is 6 now this is our i 1 now we compare this with the keys 7 this is greater than. So, we stop this inner loop and we put this key to be A i plus 1. So, key will be putted here like this. So, 3 1. So, now, J is pointing here. So, this is by the outer for loop ok.

Now, if J is pointing here now i will be here now here again we have a problem this is the key and this key is less than a i. So, if key is less than A i what we need to do we need to do shift this, and then i is pointing. So, we have to shift this then 7 this now i is pointing here. So, again we compare this this is the key again we compare key with the new A i this is a again less than. So, again we need to shift this by 1. So, again 7 6 7 will come here and i is decreased by 1 i is pointing here.

So, again we have to compare this A i with the key this is key is less than. So, again we need to shift this by 1 and now i is zero now we must stop this if i is 0. So, we stop since i is 0 and we put this key as A i plus 1 i is 0. So, A i plus 1 is 1. So, you put this 2 here. So, now, this is our now J is pointing here now if we observe this loop invariant. So, wherever J is pointing before that this part is sorted this is the loop invariant do we want before that this part is sorted and this is our new element to insert in the sorted array. So, these we have to insert by looking at the position. So, we will shift 1 by 1 until we get the position of this new entry the key ok.

So, this is now our key. So, let us just 2 6 7 9. So, this is our key and this is our i, now we compare this key with A i now still key is less than. So, we have to shift 9 here and 7 6 2 now i is pointing here, now again we compare key with A i again still it is greater A i is greater. So, we shift this 7 here and i is pointing here, now again 6 is greater. So, 6 will be shifted here now i is pointing here now key is now key is greater than 2. So, we must stop this loop if key is greater than 2 if key is greater than A i and then we insert. So, i is this. So, insert the key into A i plus 1. So, i is 1 A i plus 1 is 2. So, you must insert this 3 over here.

Now, this is 11 10 1. So, far now this is our J. So, now, J is pointing here now this is our i, now we compare this two. So, this is our key now here we are lucky because this key is greater than this A i and this part is already sorted. So, you do not need to do anything this i will be automatically increased by 1 by this for loop. So, now,. So, this is this is no change. So, now, this is our i and this is our is this is our J, J will be automatically

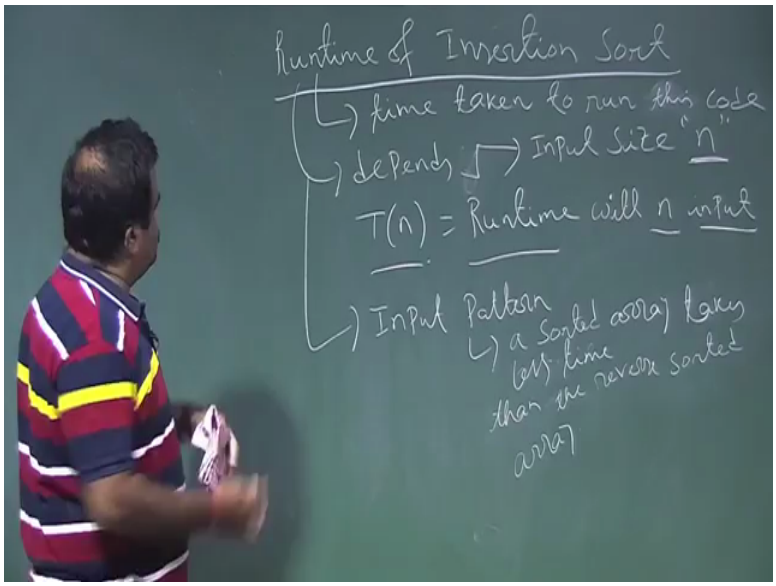
increase by this for loop sorry. So, this is our J now this is our i now because i is always J minus 1 ok.

So, now i is this now this is our key now we compare this now this key is less than a i. So, we need to do something. So, we just shift this by 1 and so 9 7 6 3 2. So, now, i is pointing here. So, if i is pointing here this is the key. So, we compare this key with this 9 and this key is greater than. So, we put this 10 over here. So, this is 1 now J is pointing here this is the now if you see the wherever J is pointing. So, all the element up to that is before that it is sorted, now we in the thing is we need to find the position for this guy this is our now key. So, this is the key now what we do now this is our i, now we compare this 1 is less than this. So, we have to shift this here and then i will be pointing here now we compare this again this is less. So, we have to shift this here.

So, i is pointing here again it is this we have to shift this here like this 7 6 3 2 and now i is pointing 0. So, once i is pointing 0 we stop and we put this 1 to be here. So, this is the final step of our code and. So, this is sorted this is sorted. So, but you see the we will talk about runtime of this algorithm how much time is it is taking by this algorithm that is called running time, how much time it will take to run this code in our computer. So, that is called runtime or the time complexity. So, we will talk about that we will talk about the run time that is the analysis part of any algorithm ok.

So, this is the input this is the output this is sorted. So, if we talk about runtime if you see here for this 10, we have just moving 1 position for this 11 we are not doing anything just 1 comparison, but for this 1 if you see 1 as to come a long way to the beginning. So, 1 has to compare with everybody. So, 1 has to come long way to the beginning.

(Refer Slide Time: 20:37)



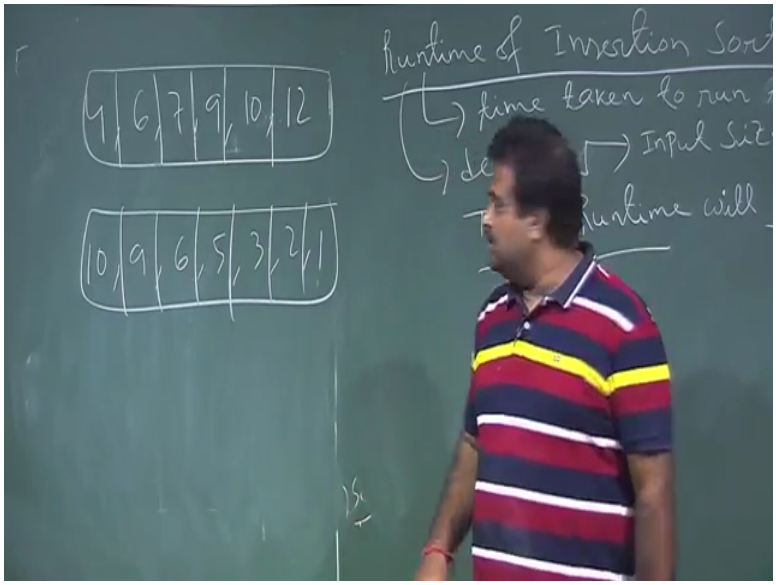
So, from there we can think that the run time will depend on what. So, runtime depend on the input. So, we will talk about runtime, runtime of our algorithm run time of the insertion sort.

So, runtime means time taking by time taking to run this code. So, this runtime will depend on what? It depends on few things first one is size of the input see if you have to sort 10 numbers and if we have to sort 10,000 numbers; obviously, 10,000 number will take more time than 10 numbers. So, runtime; obviously, will depend on the input size that n. So, runtime will depend on the n. So, we want to. So, run time is a function of n. So, we want to parameterize the run time by n. So, you want to fix this n then you want to talk about the run time. So, we want 2 parameters since it is a function of n because; obviously, if number of input is more than the time to sort will be more ok.

So, we will parameterize this by n. So, that is T n is the run time for our code runtime with when the input size is n with n input. So, we fix n then this is a function this is denoted by T n. So, T n is our runtime for n input where n is fixed. So, we fix the size of the input. Now once you fix the size of the input then the for our insertion sort runtime will depend on what here if you see this one for example, if our input is this input say now for this eleven which was not doing anything just 1 comparison, but for this one we have to come long way to the beginning. So, runtime will depend on the input pattern ok.

So, if the array is sorted if the input is already sorted then it will take less time, and if the input is reverse sorted then it will take more time reverse sorted means everybody has to come for to the beginning in each step each for loop. So, it will depend on the input. So, if the input is sorted then the time will take less because if the input is sorted say for example, if our input is.

(Refer Slide Time: 23:50)



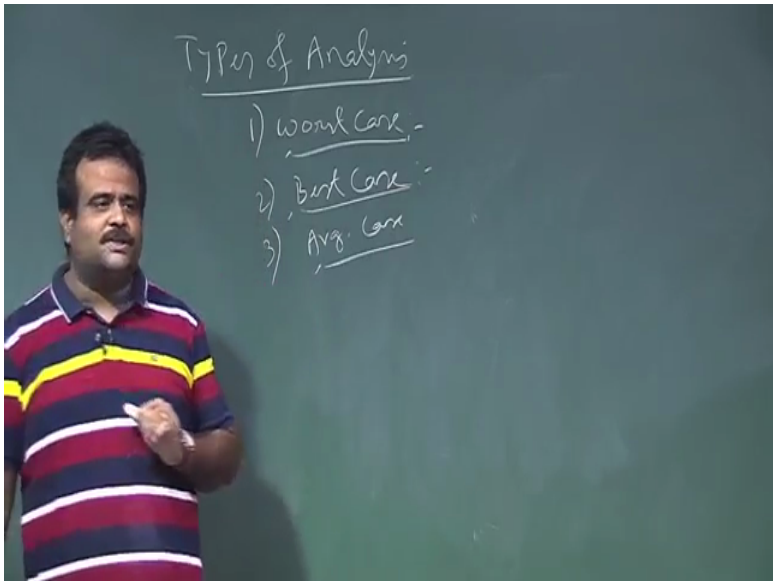
So, if our input is 4 6 7 9 10 12 if this is our input which is already sorted array, then if you run the insertion sort we just compare this with this stop this with this stop like this.

So, this is the this is easier to sort I mean this will take less time than if the input is say reverse sorted say if we have this input 9, 6, 5, 3, 2, 1 if the input is reverse sorted if the input is this, then everybody has to come forward then it is the time the time will take more. So, this runtime will depend on the input pattern, in the sense that a sorted array already sorted array a sorted array is easier to sort will take less time in sorted array less time than then if we if the array is reverse sort it will take more time, then the reverse sorted array.

So, runtime will depend on the pattern of the input. So, if we sort if a array is already sorted then it will take less time if a array is reverse sorted it will take more time because everybody has to come forward in that case. So, based on this, we will talk about 3 types of analysis, worst case, best case and average case analysis. So, this will depend on for

insertion sort it will depend on the pattern of the input. So, we will do the types of analysis.

(Refer Slide Time: 26:05)



So, first one is worst case. So, worst case means it is taking more time maximum time T n is taking. So, we will talk more details in the next class. So, it is taking more time maximum time it is taking by our code and the best case worst case for insertion sort is if the input is reverse sorted then it is taking maximum time and the best cases is if the input for insertion sort if the input is already sorted, then we are we are having the we are spending the less time to sort it and there is another case which is called average case average case means if we assume the on an average how much time it is taking, but it should be some expected expectation. So, for expectation may be we need to take some distribution of the input pattern and then we talk about the expectation. So, this we will discuss in the next class.

Thank you.