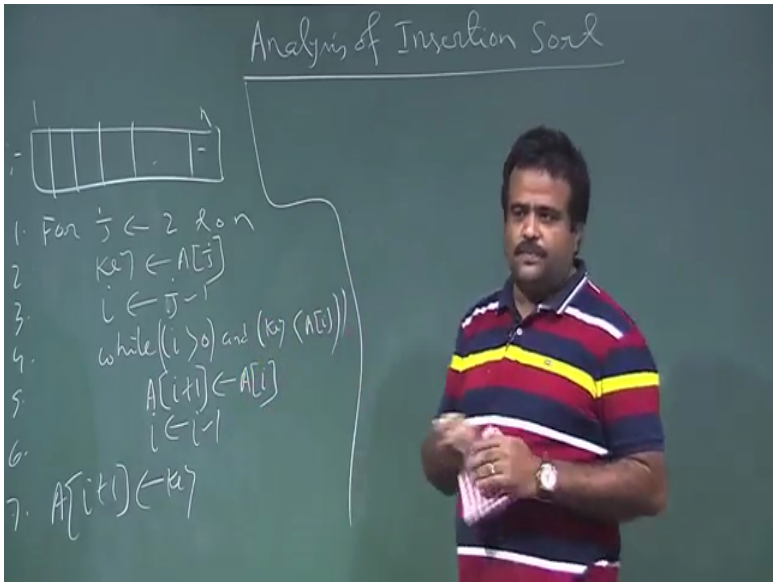
**An Introduction to Algorithms**
**Prof. Sourav Mukhopadhyay**
**Department of Mathematics**
**Indian Institute of Technology, Kharagpur**

**Lecture – 02**
**Analysis Of Insertion Sort**

So we now we want to analyze the insertion sort. So, basically we want to analyze the time complexity of insertion sort how much is the run time for insertion sort.

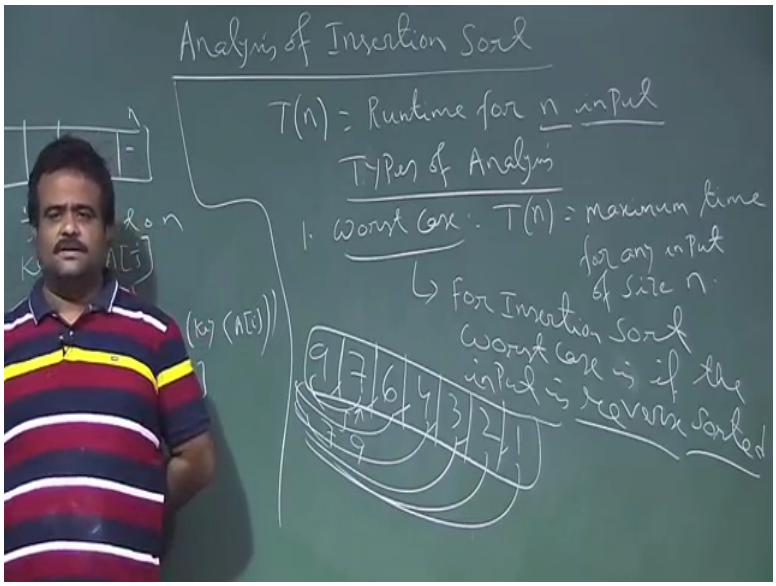
(Refer Slide Time: 00:30)




So, just for recap for insertion sort what we are doing we are we have a sorted we have a array which is the input a 1 to a n this is the a array. So, we have this code for J is equal to 2 to n. So, you are putting this key as a J and i we are pointing to J minus 1 and now we have a while loop while is greater than 0 and this key is if the key is less than A i then you have to do something and that is basically A i plus 1 is greater than A i and then we are doing i is equal to i minus 1. So, this is 1 2 3 4 5 6 and 7 we are just putting A i plus 1 is equal to c.

So, this is the code for insertion sort and we have seen the run time will depend on the size of the input it will depend on n.

(Refer Slide Time: 01:46)




So, that is why we parameterize by n. So, this is the runtime for n input my size of in n and then we have seen the input will depend on the sorry runtime will depend on the pattern of the inputs. So, if we have a already sorted array then it will take less time, than if we have a reverse sorted array it will take more time. So, this is the type of. So, this is the types of analysis 3 types of analysis we will do.

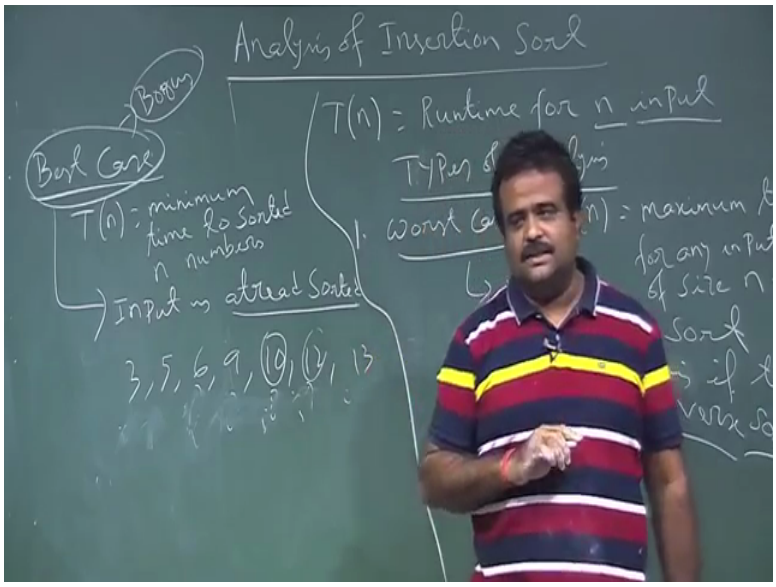
First one is worst case; that means, T n is the maximum it is guaranteed, maximum time for maximum time our code is taking for any input for any input of size n. So, we fix n that is why it is T n, we fix n if we fix n then this worst case is the T n is the maximum time. So, what is the maximum time our code is taking? So, we. So, so that is worst case. So, for insertion sort one is the worst case insertion sort worst case is if the input is reverse sorted, then it is taking maximum time because in that case. So, for insertion sort worst case is case is if the input is reverse sorted.

So, that means, if we have say number say 9 7 6 4 3 2 1 if this is the input, suppose this is the input we want to run this insertion sort on this input. So, what it will. So, everybody as to come forward. So, the i is J is pointing here. So, then this is greater. So, 9 will come here 7 will come here. So, everybody will come forward. So, that is the maximum time it will take there is no other input which can give the more runtime than this. So, this is sort of guarantee. So, we are having maximum time. So, this is sort of

guarantee and this is good because if we say I am sorting algorithm and maximum my worst case runtime is this much.

So, that means, nobody that is the worst case means that is the maximum time my algo is taking so that means, nobody will can nobody cannot come with some input where my code is performing bad than that what I am claiming as a worst case. So, that is the that is the result worst case is the most usual analysis one must do for any algorithm, and the best case is then the next one is best case which is called bogus also because it is a cheating.

(Refer Slide Time: 05:28)




So, this is the best case. So, this is best case means, the T n is minimum time it is taking minimum time to sort I mean this is the sorting algorithm insertion sort we are taking about, minimum time our algo is taking for input of size n minimum time to sort n numbers this is the best case.

So, this is the minimum time. So, for insertion sort when is the best case will occur if the input is already sorted? So, if the input is already sorted; that means, what; that means, if we have a input like this say 3, 5, 6, 9 10 12 13 this is a already sorted input. So, for this input if we run the insertion sort, then how much time will spend we just start with J is this, now this is greater than this we do not. So, all the one comparison then J is here all the one comparison. So, J, i will be here J is here. So, all the one then J is pointing here now this will be i then this is our key and we compare this. So, all the one comparison

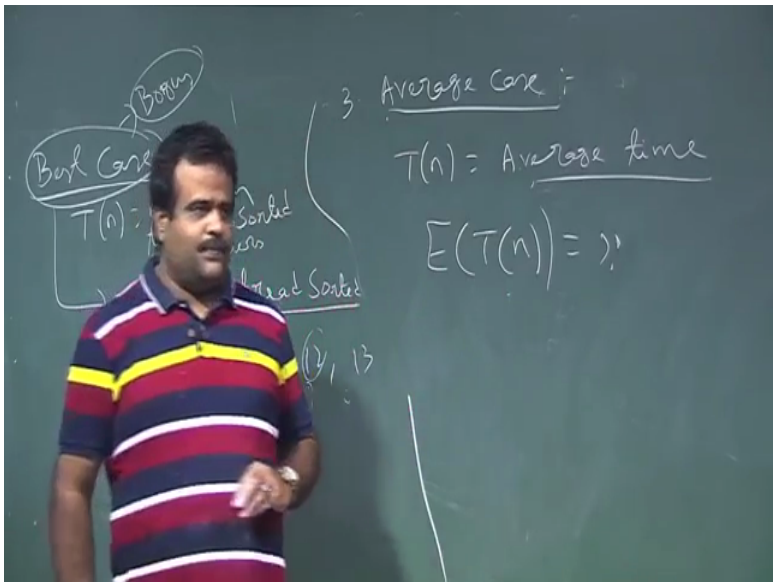
again. So, J is pointing here, i is pointing here all the one. So, this is the best case because just every for we just carrying the, we are just reading the array and only comparison is occurring. So, this is the best case scenario.

So, this is minimum time our algorithm is taking here is no other input which can be lesser runtime than this so, but the thing is, but this is only for I mean this best case is happening for insertion sort if the arrays already sorted. Now which case you want to be usually do best case or worst case. So, that is very crucial point. So, best case means it is not guaranteed, I mean see suppose Microsoft open competition and ask this group to have a to build sorting algorithm ask this group to build a sorting algorithm. Now you build a sorting algorithm and you build a sorting algorithm you submit to the Microsoft and Microsoft give your code to this group and you are code to other group, and ask to find out the input while it is performing that.

Now, you are telling that my worst case runtime is this, worst case it is taking this much time and you are telling a Microsoft my best case runtime is this, but you have to run this for this type of input; for this type of input I am getting best case you are not telling the guarantee one you are guarantying that you your runtime cannot go beyond this you are telling your best case; that means, for this type of input your code is good, but their telling my code is worst case runtime is this; that means, whatever input you can have you cannot go beyond this runtime your runtime cannot be more than this. So, that is sort of guarantee.

So, but if you tell only best case then anybody can adversely can come with some input where your code is performing bad because you are only highlighting your best case. So, that is why this is sort of this is bogus this is sort of cheating. So, we will not go for. So, it will no prefer to the do this do this analysis, because this is for some certain input where as this worst case is the usual analysis we will do because this is the guarantee one it is guaranteed.

(Refer Slide Time: 09:51)

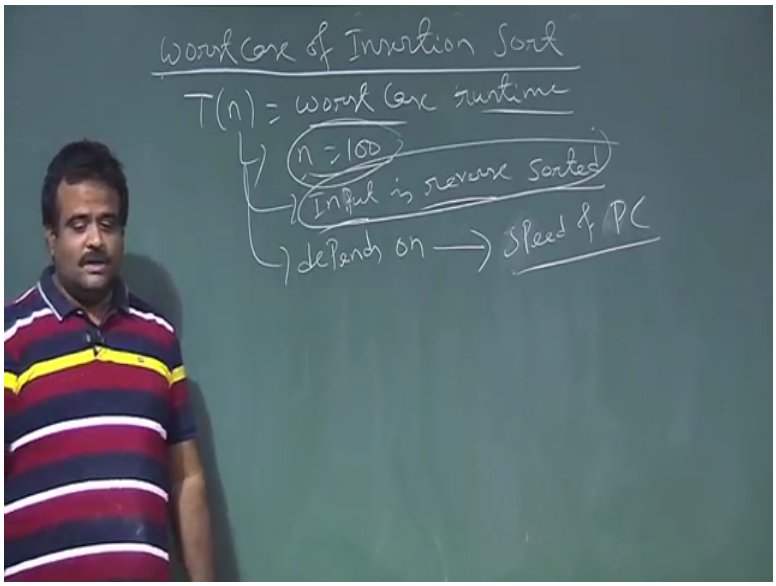



So, we can guarantee that our code is our algorithm is taking this much time and there is third one is average case analysis. So, this is another one average case analysis average case.

So, average case means. So, on an average how much time it is taking we know the worst case we know the best case. Worst case is for certain input best case is for certain input now if we. So, this is the T n is the average runtime average time; that means, for average to be very honest we need to take the expectation. So, maybe we need to deal with the expected value of T n this how much. So, there we need to talk out we need to assume some distribution on the input, input is following equal in likely or something. So, maybe we will do some type of analysis in our course, but this is basically average case expected runtime ok.

So, this is the 3 types of analysis one can think of any algorithm worst case is the maximum time for any input, best case is the minimum time for certain input, minimum time which is occurring for certain input and the average case is the on an average or expected runtime. Now suppose we want to do the. So, we know the worst case this is the usual analysis one must go for.

(Refer Slide Time: 11:36)




But suppose we want to do worst case analysis of insertion sort. So, we want to do the worst case of insertion sort.

Suppose T n is the worst case runtime worst case runtime for insertion sort. So, worst case means. So, what do we fix we fix n, n is fix suppose we are talking about we have say 100 input we are sorting 100 input and since it is worst case for insertion sort the input is reverse sorted. So, you fix this to information input is reverse sorted. So, this 2 is fixed we fix the size of the input that is say 100 we have sorting 100 elements, and the inputs are in the reverse sorted way. So, that is that is why it is giving us the worst case.

Now, tell me if we fix this 2 situation will it run time depend will the runtime depend on any other thing any other factor will be interfering in the runtime. So, that is a crucial question any other factor will depend will be. So, this then the runtime will depend on what any other factor we already fix n. So, T n is we already fix the worst case. So, the input is reverse sorted then after that will it depend on any other thing. Suppose you are running this on a your band new laptop code do processor and you are running this same code in a reverse sorted input, input size is fix on a computer old computer you are gang gang gan further is or gang gang gang further is having. So, which will take faster you are your p c you are laptop will take fast at time.

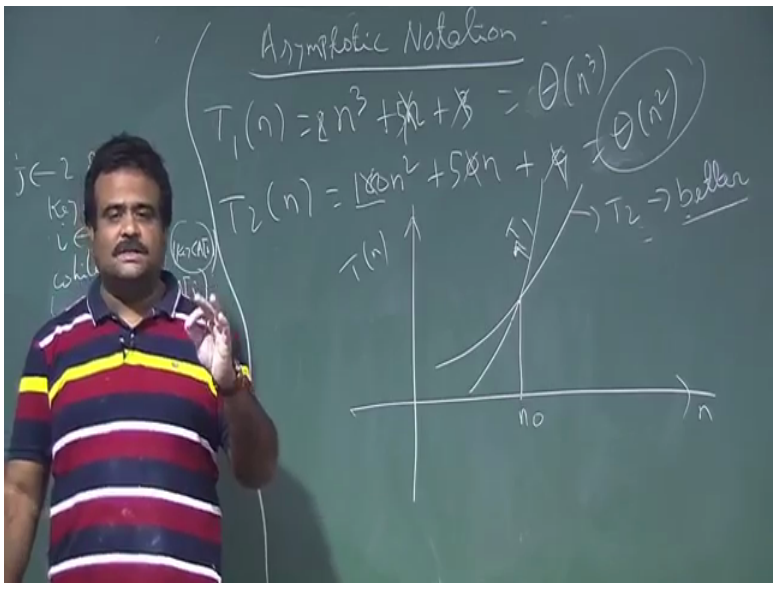
So, your laptop can give the output in a second, but out grandfather computer can take on hour to give the output. So, it will depend on the speed of the computer where you are

running speed of our p c where we are running the code. Now it is depending on the computer where you are running the code now then how will judge whether 2 code is good which one is good suppose you have a sorting code they have a sorting code. So, now, how to judge whichever is good.

Now, we have to run this code in a same computer not only that we have to run this in a same computer in a same situation in a same time that is not possible. So, if I am running your code in day time where this sun is from this is summer time, and it is very hot time and this we do not have a c in the room and our p c is taking some time to adjust the fan all this thing. So, you are computer will I mean and there code you are running in the night time.

So, it is very difficult to run to piece of code in a same environment in a same situation. So, that is a headache I mean it is not possible every time if our computer is multi user every time some processer is running something. So, it is very difficult to run it is very difficult to get 2 similar situation of a walk station. So, that is a botheration. So, then how we can judge 2 two piece of code whichever is faster than whichever is slower that is very difficult to judge in this way, because we cannot give the same identical situation of this p c where we are running this code and so that is a really a botheration. So, to avoid this botheration what we need to bring which is asymptotic notation.

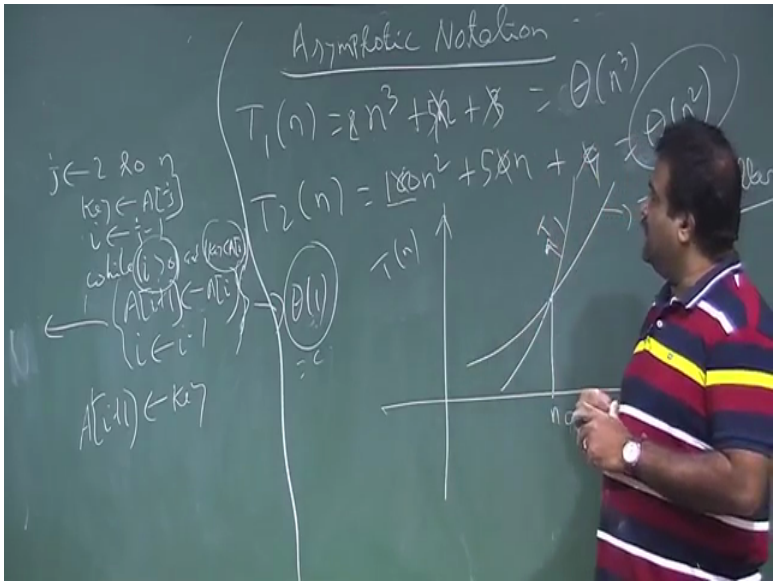
(Refer Slide Time: 16:07)



This is a very big idea I means it is a great idea where we can get rid off from the computer from the machine dependency ok.

Now, where from this idea is coming suppose we have 2 code one is having runtime this say n cube plus n plus 3 or 2 n cube plus 5 n plus 3, and you have another piece of code which is say 100 n square plus 50 n plus 9 these many seconds. So, how we get this we just get a we have a say for example, insertion sort, we know the we have how many loops we know.

(Refer Slide Time: 17:11).




So, insertion sort we have a outer loop say J is equal to 2 to n and then key we are assigning to this and i is equal to J minus 1, and then we have a for loop i is greater than 0, and the key is less than a i, then we have A i plus 1 is greater than A i and then we have decreasing i by 1 and final here we are assigning A i plus 1 is equal to key.

So, this is the a piece of code for insertion sort now these are basically coming how much time we are spending for this assigning, how much do you add for this compare how much time we are comparing. So, it depends on the machine. So, how much our machine is. So, our architecture our odor is taking for this operation. So, our add or our multiplexer all this things. So, these are all basic dependence terms, and this and these are coming from this loops ok.

So, suppose we have a sorting algorithm with this, this is the runtime for our machine for a machine and this is the runtime for another machine. Now can you tell me which code is better because these factors are coming how much time we are spending in doing this assignment doing this comparison, how much our computer is taking for that. So, this may be this may independence terms.

Now, if you draw this curve you should draw this true curve in terms of n, suppose this is n and this is the runtime T n. So, one curve will be like this and another curve will be like this. So, after certain point of time n 0 so this will be our what this will be out T 2 and this will be our T 1. So, you see as we tend in large then T 2 looks better, but if you take n small then because of this factor it is looking T 1 is better, but if we make n large then it is it is clear that T 2 is better T 2 is better than ten right because the growth. So, we want to see the growth. So, that is why we need to bring this asymptotic notation.

So, asymptotic notation in the sense what we do in the engineering sense, we ignore all the lower order term then we ignore the leading coefficient this will give us the what is called big theta we will formally define this big theta and then again we do this same thing we will ignore the lower order term this is. So, this is better than this so; that means, we want to see the growth of this runtime as n tending to infinite.

So, as n tending to infinite as n is large then we will get rid of from this missing dependent term, because these are basic dependent terms these are the terms how much time we are spending to compare 2 number in our machine. So, I add or multiplexer all this term we do not care about that because it is very difficult to get a identical situation for a machine. So, that is why we do not bother how much time we are spending here. So, this is we are just following this is a constant term, this is basically we are following this is theta one term, because we do not we do not care this is how many second 5 second 4 second, but this is some constant work we are doing in terms of n, because this we just assigning this one assignment and one assignment here. So, this is basically theta one some constant amount of time that constant is nothing not related to n that constant is the time our computer will take.

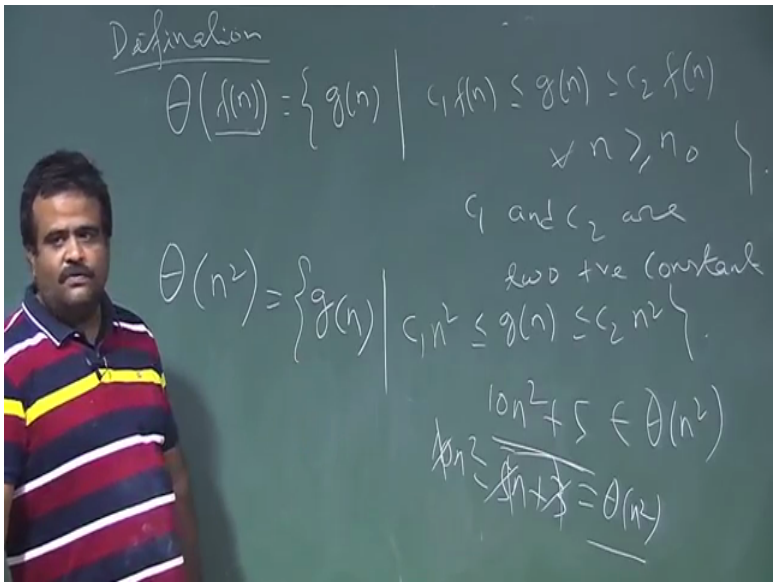
Now, if it is our banding machine it will take less time if it is our grandfather machine it will take more time, but why we should be bother about that machine we are running the code we want to get rid of from this machine dependent things. So, that is why we just

talk about this is the constant time theta bond time it may be 10 second, it may be 5 second it may be millisecond for a band new computer, but we do not care about that we will just thing that this is the this is the constant time theta bond time this is some c some constant theta bond time.

So, this is the big idea this asymptotic notion. So, now, we have a method to compare 2 piece of code without depending on the machine, otherwise we have to run this code into 2 machine in a identical situation then we check the exact time how much they are spending, exact time how much this code is spending then if it is spending 10 second if it is spending 10.5 second we said this code is better, but that also difficult because we need to have the identical situation identical condition of the this 2 machine. So, that is why this is the best way, this is the very useful way big this is the very useful way to compare 2 code in a asymptotic sense; that means, we want to see the growth of the runtime we want to see when n is large when the input size is large tending to infinite where the runtime is going.

So, that is the observation we will do for asymptotic notation and so, in the asymptotic sense we will see that this time. So, this is the loop for this is the loop in n and this is the loop depending on how much we are coming there, but there this is the time decimal we are spending constant time some constant time. So, this will write in theta of one way. So, now, we formally define the asymptotic notation we have 3 asymptotic notation, one is big theta just now we have said another one is big o another one is big omega mainly 3 notation we use. So, they are mathematical definition are there.

(Refer Slide Time: 23:58)




So, this is the definition. So, we define the big theta. So, big theta of f of n, it is basically set of all function g of n such that it is bounded by c of f of n less than equal to g of n less than equal to c 2 of f of n and for all n after (Refer Time: 24:38) we do not care about the low low n because we want to see the growth of this runtime. So, we want to take this is to be greater some as n tending to infinite where it is going and where c 1 and c 2 are 2 positive function 2 positive constant.

So, if, for example, then big theta of n square if f n is n square then it is basically set of all function g n such that we must have 2 constant such that c 1 n square less than equal to g of n less than equal to c 2 of n square for some n after some n 0 and c 1 c 2 are 2 positive constant. So, for example, if we have say 10 n square plus 5. So, this is belongs to order of n square because for this we can always chose c 2 for some n. So, that it will be bounded by both side.

Similarly, say 15 square minus f n plus 3 this is also belongs to theta of n square, but we do not we will floppy we do not write belongs to we will just write this is theta of n square. So, as you said in engineering sense what we do we just ignore all the order term we ignore the leading coefficient and this is n square this is theta of n square. So, this is the asymptotic notation of theta of big theta of n square. So, in the next class we will talk about another asymptotic notation what is Big O and Big Omega.

Thank you.