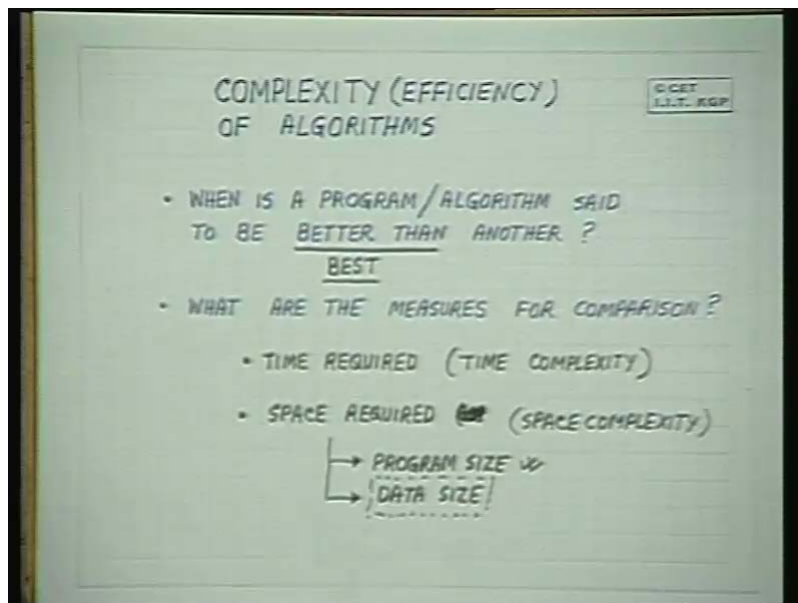


Programming & Data Structures
Dr. P.P. Chakraborty
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 18
Complexity (Efficiency) of Algorithms

Till date we have studied preliminary programming methodology with special reference to recursion and we have also seen some special features of C language which allows for structure declaration and for dynamic allocation using the malloc. And we have also seen using this how we can dynamically form link lists in which elements can be inserted, deleted and other algorithms can be implemented on them. the next phase to all this that is after having formed a background, the next phase is to move on to the design of data structures and good algorithms which effectively means that we must look for the design of efficient algorithms and data structures to finalize what a good program is about.

The foundation of all these things will lie in the concept of efficiency of an algorithm or the complexity of an algorithm and this matter must be understood first before we proceed towards designing good algorithms and data structures.

(Refer Slide Time: 06:56)



So the topic of today's lecture will be an introduction to the complexity or efficiency of algorithm. You understand this, we have to answer two questions. First question is when is a program or algorithms said to be better than another? If I give you the same program to run on a different machine, on two different machines they will take two different amounts of time. And then looking at the programs, one program may be much bigger in size compared to another or one program may take more space than another or one program may be more easy to understand than another. So in terms of all these things,

when do we say that one program is completely better than another. Unless, we understand the concept of what is said to be better than, we will not be able to realize the concept of best and that is what we are trying to find out what is the best algorithm for a particular problem.

To answer this question we must first answer what are the measures of comparison. The first measure for comparing two algorithms is the time required. It is also called the time complexity of an algorithm. This is the time of execution of the algorithm, not the time of compilation, it is the execution time of the algorithm. The second is the space required, these are the two important resources of the computer which is often called the space complexity but this is not the space complexity actually.

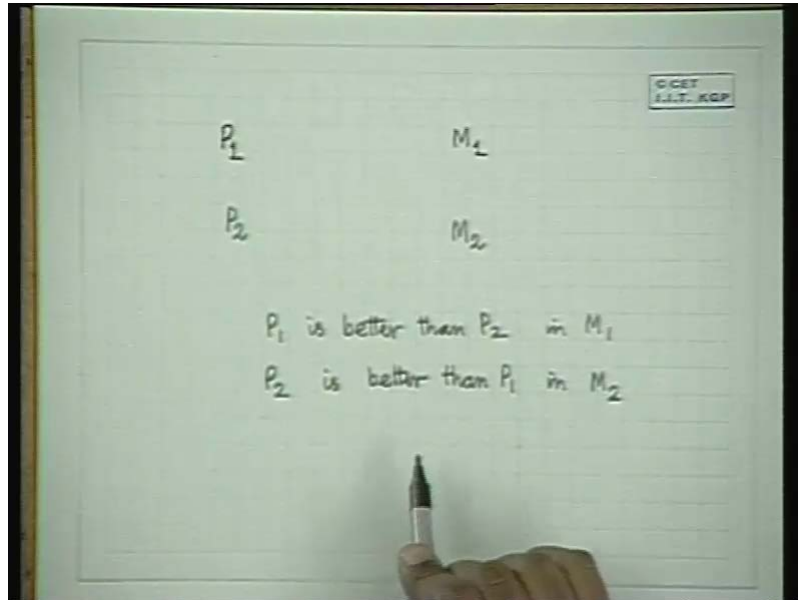
There are two types of space which is required by a program. One is the program size and the other is the data size. For example for the particular program, for a particular problem you may write two programs one of which uses a three dimensional array of n by n by n another which just uses a one dimensional array and does the processing. So the data's declaration size of the two may be different. And obviously the program, one program may be bigger than another. Together as I mentioned they form what is called the space complexity of a program.

Now these are the two important measures of comparison between two program and unless this space required is exponential, the time is the most important criteria. In most cases now that lot of space is available on the computer and unless exorbitant amount of space is required to solve a particular problem, we shall see example of where exorbitant amount of space may be required to solve the problem. Then time is the most important criteria and remember this data size may be dynamic in nature. You may continuously make data, you may acquire dynamically data and then release the data also but because of this dynamic allocation of the data the program size...

On the other hand the program size usually remains constant, it does not change with the size of the, neither with the size of the input, it does not change with the input because if you change the input the program does not change but the data size definitely may change with the input itself because you may allocate dynamically according to the input. So these are the two important criteria which are measured which are used for measuring the complexity of a program. And using these two measures we can compare between two programs.

The other aspects like elegance of a program, easy understandability etc etc come much later after only there is a tie between possibly all these two situations of which the time required as I mentioned previously is the most important criteria.

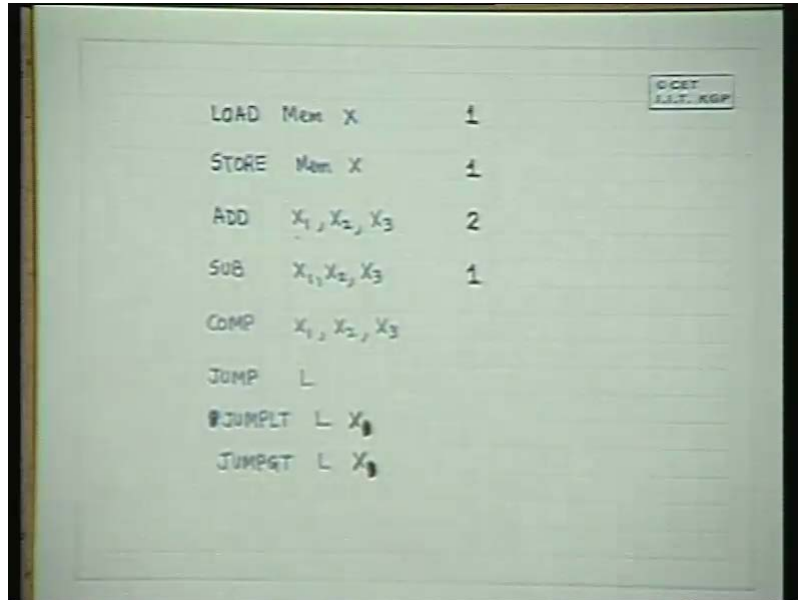
(Refer Slide Time: 08:00)



Now even here there are several issues. The first issue is that given two programs, if I run it on machine a and machine b. Suppose I have got program p_1 and I have got a program p_2 , alright and I have got a computer machine 1 and machine 2. Now in machine one, p_1 is better than p_2 . On the other hand p_2 is better here, better in terms of time. Then what do we say about these two programs? We can't say anything about these two programs and if we are to measure a program with respect to all the time it takes on different machines, it will become impossible to compare two programs. But this obviously, if you want to measure exactly in terms of cpu time then we have to start them running on different machines.

So we cannot measure according to cpu time, if we want a machine independent model of computation or a machine independent model of comparison between two programs. So we have to make a compromise in our analysis at some point of time if we want to say that how we will compare between two programs which may run which or which is the machine independent comparison between them.

(Refer Slide Time: 16:10)

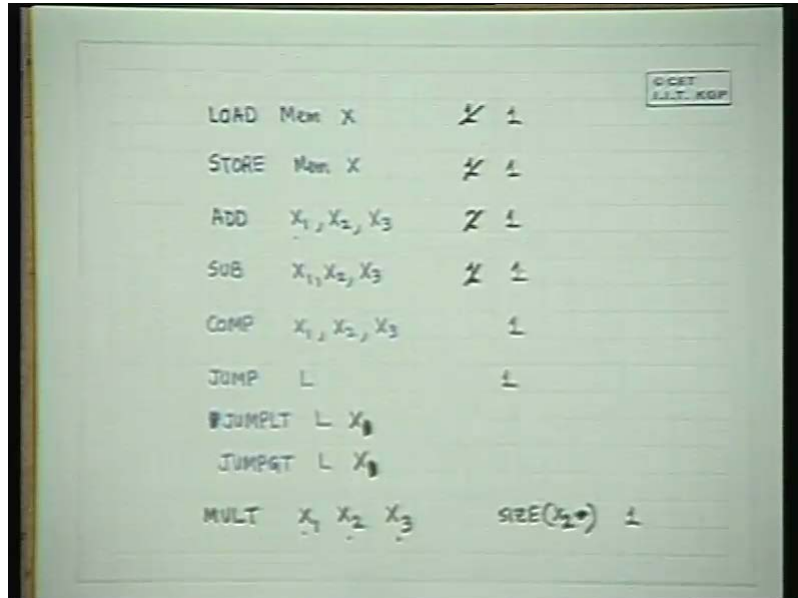


LOAD	Mem X	1
STORE	Mem X	1
ADD	X_1, X_2, X_3	2
SUB	X_1, X_2, X_3	1
COMP	X_1, X_2, X_3	
JUMP	L	
JUMPLT	L X_3	
JUMPGT	L X_3	

So the next issue that we come to is can we have a machine independent model of computation. We can, provided we make certain assumptions and we make certain compromises. Firstly, we have to model the computation of a machine in terms of some abstract machine in the sense that we cannot now measure in terms of exact cpu time. So the first item that we do is we see given a, suppose we have given a machine M_1 and another machine M_2 . The same program p , this machine M_1 has its own machine instructions. This machine M_2 has its own machine instructions. The process of compilation will convert the program p into the machine instructions of m_1 , if you want to execute in m_1 .

It will convert it into the machine instructions of m_2 , if we want to measure in terms of m_2 . And then when it executes on this machine then we get our cpu time computation. So the two machines if they are different in terms of their instruction sets also then also it will become difficult to compare. On the other hand if they have the same instruction set then you could have seen p_1 and p_2 , you could have put them on to that compared their instruction sets and you could have said that for this instruction set, for example I will give you an example of an instruction set. This is say load from the memory a variable and store inside the computer as a variable x say as a register.

(Refer Slide Time: 14:55)



The image shows a handwritten list of assembly instructions on a piece of paper. In the top right corner, there is a rectangular stamp that reads "CGET" above "I.I.T. KGP". The instructions are as follows:

LOAD	Mem X	\neq	1
STORE	Mem X	\neq	1
ADD	x_1, x_2, x_3	\neq	1
SUB	x_1, x_2, x_3	\neq	1
COMP	x_1, x_2, x_3		1
JUMP	L		1
JUMPLT	L x_3		
JUMPGT	L x_3		
MULT	x_1, x_2, x_3	SEE(20)	1

Store this into the memory, add x_1 x_2 and put it into x_3 , subtract x_1 and x_2 and put the result in x_3 , compare x_1 and x_2 and put the result in x_3 . This says that if it is a minus 1, x_1 is less than x_2 . If it is 0, x_1 is equal to x_2 and if it is plus 1 then x_1 is greater than x_2 . jump L, it will jump to the label to the instruction label L, jump less than L that is this will check if x_3 is less than 0, it will jump to L sorry here I will put x_3 , jump greater than L 3 any x , need not be x_3 , if this is less than L 0 then jump to L if this is greater than 0 jump to L this is the unconditional jump, alright.

So a program p_1 or p_2 will be converted into these machine instructions and you can make a relative comparison between them. We can say this takes one time step, this takes one time step, this takes two time steps, this takes one time step etc etc, alright. And given this and if two machines have the same instruction set, you can compute it in terms of instruction steps. And if somebody has got a faster clock than another, obviously it will run faster but then when you make the complete computation they will only differ by a ratio, there will be only a proportional difference between them. Then this is the approximation that we will make.

Now if they have got different machine structures then what do we do? Here again we make another assumption. What we do is we say that we make an assumption that the addition operation in all machine takes a constant amount of time. that is in one machine if it takes unit one time, in another machine it cannot take time which is proportional to the input that is it cannot take a value which is n and in the most simplest condition, what we do is we say that if it takes a constant amount of time, we all make everything 1 1 1 1.

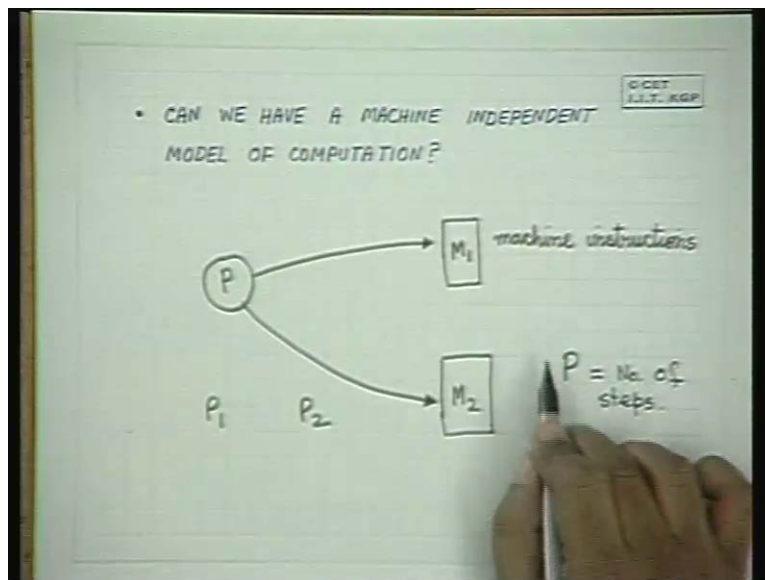
On the other hand if somebody provided you a multiplication operator of x_1 and x_2 and store the result in x_3 then this would possibly not take a constant amount of time. Can

you tell me how much time this will take? This would be proportional to the size of x_1 and x_2 , so possibly this would be size of x_1 plus or into plus or multiplication x_1 plus x_2 . How many times do you have to do addition, to do multiply two elements. You have to do it proportional to that number size of the multiplicand. So it will be size of one of them, smaller of them or larger of them.

If this is unit time then this will be size of x_2 . If addition is unit time then this will be size of x_2 alright or in some computers since these sizes are fixed, you can say this is one unit time because this size is limited and you can define a combinational circuit to design this multiplier which may do it in constant time.

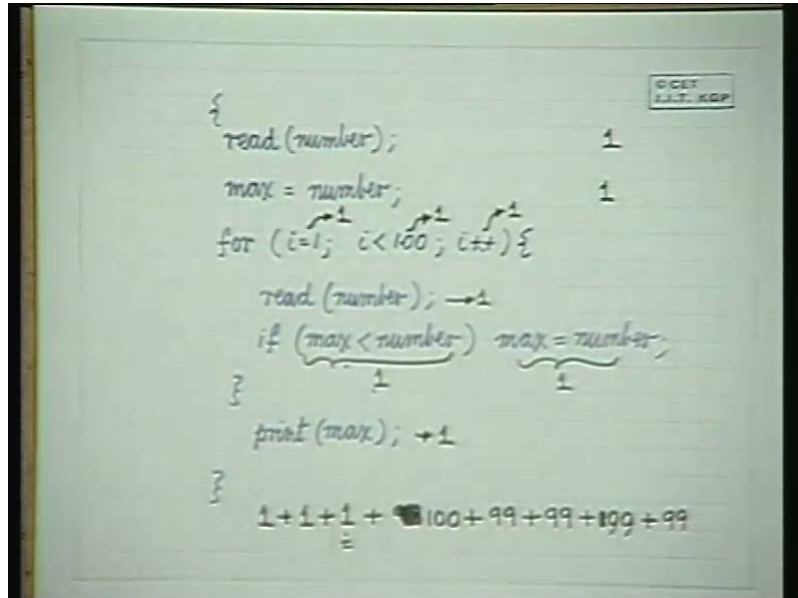
Now for machine independent computation, we define that all the operations take the same amount of time. If somebody has in one machine multiplication is done in the constant time, in the other machine also it must be done in constant time. That is the assumption that we make that all operations take the same unit of time. So now if we, let all the operations take the same unit of time then given a program p , we can compute the number of steps that it takes to execute.

(Refer Slide Time: 15:44)



Now this is an approximation and given two programs p_1 and p_2 , we compare the number of steps that they take to execute. And what is the error that we incur in this comparison? The error is by a ratio in the sense that if this takes say 500 and this takes say 250 then in a faster machine, this may run faster than this one.

(Refer Slide Time: 19:53)



For example how we would compare in the program to find the maximum of n of hundred elements. Take, let us have a look at this program. we read a number, max equal to number for i equal to 1, i less than n, i plus plus read number if max is less than number max equal to number print max. Now this will be converted to machine instruction code but having a look at this code itself, we can make, we can make a reasonable assumption as to how much steps this is going to take.

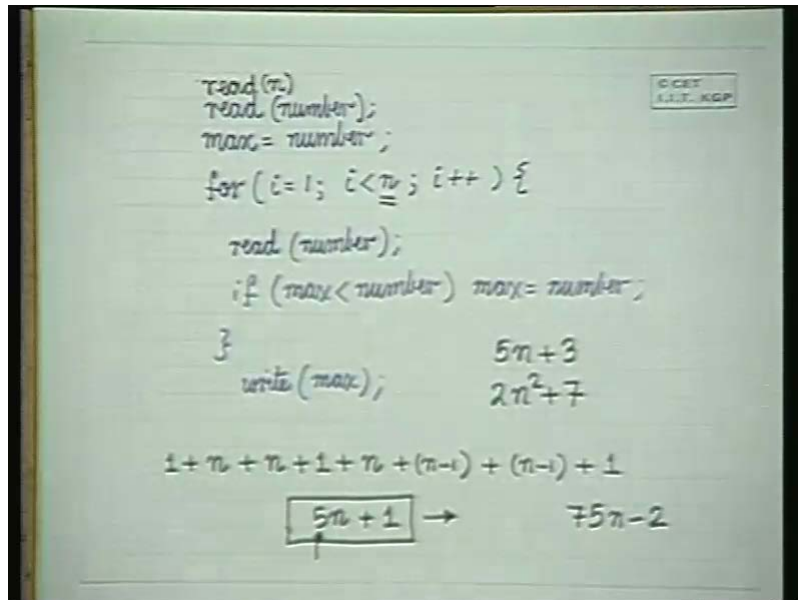
You can make, the assumption that you make for finding out the computation, the number of steps you can make that assumption here also. For example you can say this is a read statement and to read a number which is usually the size, what is the number of steps to read a number. Suppose all numbers are within a fixed size and we will say this it takes one unit of time. This is an assignment statement which again takes one unit of time. This statement will be executed once, so this will take one unit of time. This statement will be executed once in the first when we go in say one unit of time then this will take one unit of time. This will take one unit of time and if this is true in the worst case this will take one unit of time, it will go back and this will take one unit of time. Then again this comparison will come.

So you have got and then you go on doing this hundred times, so this will be done once, this will be done once, this will be done once. This statement will be executed how many times? 99 times, am sorry this will be executed 100 times because the first time it will go and every time it will go. This statement will be executed 99 times, this statement will be executed 100 times. Now this will be executed 100 times, now this will be executed 99 times and this will be executed again 99 times.

So that is this assignment statement will be executed 100 times, the read number will be executed 100 times, print will be executed once and this one and this one sorry this one

and this one will be executed 99 times and we can add up and say this is the time required but there is another problem with this. That is given two programs p_1 and p_2 , how do we generalize this to work for n inputs. This does not, this takes input but the program execution time is independent of the input.

(Refer Slide Time: 22:44)



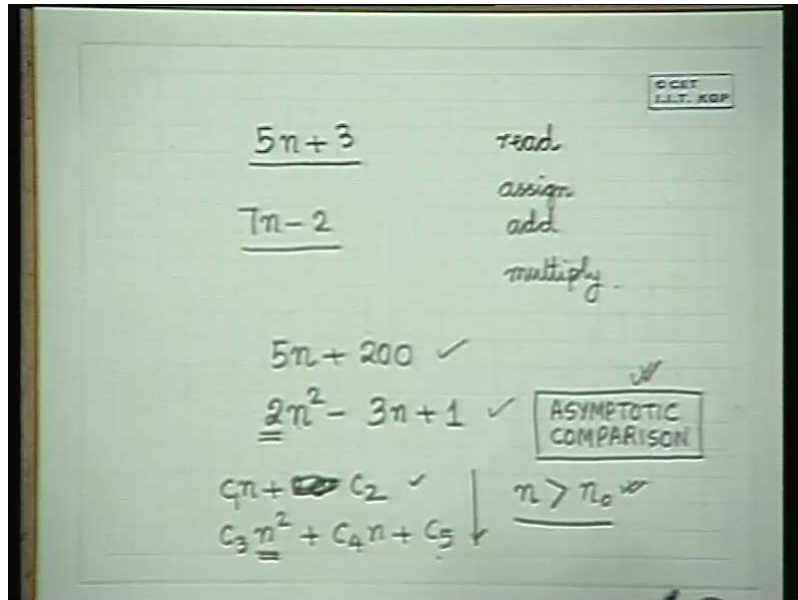
On the other hand if it were a program like this but this n was read. So before this you have read n then what would be the comparison? Read n would be executed once, this read number would be executed, how many times? N , max equal to number would be executed n times, i equal to 1 would be executed once. This comparison would be executed n times this, this one and this one.

So now this time statement is also proportional to the input, alright. So this is proportional to the size of the input. So what do we get? We get 1 2 3 4 5 n minus or plus, plus 1. This is an approximation that we have obtained. The approximation is by making these statements all unit time and we say that this is the time complexity of this algorithm.

Now since we have made it unit time, this will actually has got very little variables. If somebody has got $75n$ minus 2, on a faster computer this may run faster than this. On a two machines, in one may be this assignment goes slower and in another one the read statement goes slower. Then one program may run faster in one computer and the other one may run faster in the other computer. But if one has got n here and suppose somebody has got $5n$ plus 3, another program has got $2n$ square plus 7. Okay am coming, I will repeat a little more here.

Suppose I have got one program which has got $5n + 3$. How did I get this $5n$? I got this by making all statements unit time but this may consist of say some read statement and some assignment statements, some add statements and some multiply statements.

(Refer Slide Time: 29:00)



Another program takes $7n - 2$. Now in one machine I have, can I compare these two and say this is better than this. When I run it on a machine I cannot because it depends on the actual execution times taken for these statements here and the ratio that they have here. For example the read statement may take lot of time in one machine but this one may have, for example the add statement may take lot of time in one machine. This one may have very few add statements and this one may have a large number of add statements and then in that machine this will run faster than this.

On the other hand if you multiply is very few multiplications are here and lot of additions are here and multiply takes lot of time then this will be faster than this in that machine. So when we have got two elements which are, even for different kinds of inputs we cannot compare them even if we have this sort of a statement. So all we can say is that they are proportional to the inputs, they will take a time which is proportional to n . Is this understood, that both of them will take a time which is proportional to n that is all you can say. This is because of the assumptions that we have made but on the other hand if we have somebody which is $5n + 200$ and another one which is $2n^2 - 3n + 1$. Then can you say something about these two? Can you tell me what we can say something about these two is this better than this? yes for n , when n is large that is when n tends to infinity then this will be better than this irrespective of what is the constant here.

If this constant was even, if this constant was any positive number, if this constant is any positive number then when n tends to infinity this will be faster than this. That is you can

say that given $c_1 n + c_2$ and given $c_3 n^2 - c_4 n + c_5$ say plus $c_4 n + c_5$, given two such functions whatever be the values of c_1, c_2, c_3, c_4, c_5 provided they are greater than 0, alright they are. Then when n is greater than some n_0 , beyond n greater than some n_0 this will be first and then this is called asymptotic comparison.

We compare them when the input is large. The question is, is it justified? Is it justified to say that this is better than this. You don't say this is better than this, you say this is better than this only when n is greater n_0 . But that is quite justified because this n_0 usually does not turn out to be very large. If n_0 turns out to be some 20 million and all your data n will always be less than 100 then obviously there is no point in comparing them. But in general when n is less than 100 then both your program will run few milliseconds, there is no point in comparing them because there is no need, you will anyway after pressing the run button you will error. Then why are you comparing them at all, why you doing all this circus of analyzing them to such a great extent.

You are doing them because the time is really expensive and time is really important that is time, it will take a lot of time. If it does take a lot of time then obviously these functions n must be larger. Therefore how will it take a lot of lot of time? These functions must be large. So in most situations where time is important, asymptotic comparison becomes important and is relevant but if your input size is 200 and your sorting numbers then it is irrelevant to compare which sorting algorithm you will use, double sort or quick sort or merge sort because all of them will, in the present technology all of them will give the result at the press of a button. The only time required will be to time to print in on the screen.

This comparison is important when you are trying to spend more and more time to execute. So we will look at asymptotic complexity. an asymptotic complexity, given a function $c_1 n + c_2$ that is if the complexity function $f(n)$ whatever you have computed is a bound that is it will take less than $c_1 n + c_2$ time. We say this order n , I am trying to informally introduce this concept. If $f(n)$ is less than equal to, then it is said to be order n square because this is the term which is going to dictate its complexity when n tends to large. Can you tell me intuitively what this will be? Intuitively which is when n tends to infinity which is bigger of the two, which goes faster. n square. We will formally introduce these functions, I am just trying to informally tell you what this means, alright.

All of them are of any base because they are of any base because $\log_k n$ to the base k will be some constant multiplied by $\log n$ to the base 2. So if I since it's a constant and constants become unimportant here, therefore this log can be consistently taken to be a particular log say log to the base 2.

(Refer Slide Time: 34:40)

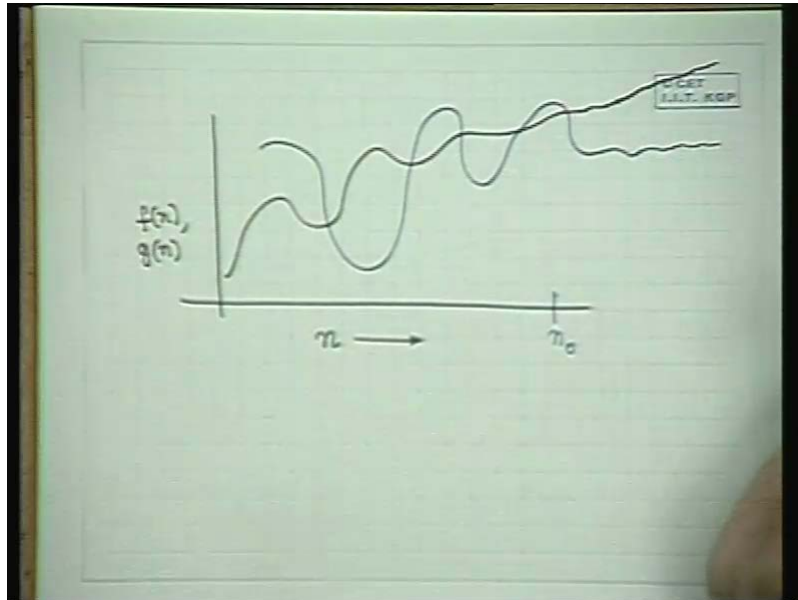
$$f(n) \leq c_1 n + c_2 = O(n)$$
$$f(n) \leq c_1 n^2 + c_2 n + c_3 = O(n^2)$$
$$f(n) \leq c_1 n^2 \log n + 3\sqrt{n} = O(n^2 \log n)$$
$$\log_k n = c \log_2 n$$

$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$	$\rightarrow 0$	$n \rightarrow \text{large}$ n
	$\rightarrow \infty$	
	$\rightarrow c$	

So then asymptotically we compare two of them and we say when limit n tends to infinity, we compare $f(n)$ and $g(n)$ and see which is the smaller of the two. If this tends to 0 then this is better than this. If it tends to infinity then this is better than this, if it tends to infinity then this is better than this and if it tends to constant c then both are of the same complexity. So this is the terminology that we will use to compare the times when n tends to large.

On the other hand when n is small then obviously you cannot make this comparison and you will have to compare, you will have to find out the exact constants. And after finding out the exact constants with n which is the size of the input in this case, you will have to find out this plot and after finding out this plot only then can you say in which range which is better than the other.

(Refer Slide Time: 34:28)



But even this is meaningless because you have made all constants equal to one and on different computers they will take different amounts of time. But asymptotically you can say that beyond a certain point n_0 , this will always be less than this, whatever be the constant c_1 c_2 c_3 this will be better than this or both will be identical that is what we do by this order of complexity.

So, next day what we will do is we will try to have a look at the problems that we have solved before and determine what is the complexity function of these algorithms that we have written and try to compare bubble sort, exchange sort, quick sort, merge sort, tournament sort and other kinds of problems like the Fibonacci number problem etc. so we will tackle these in the next class.