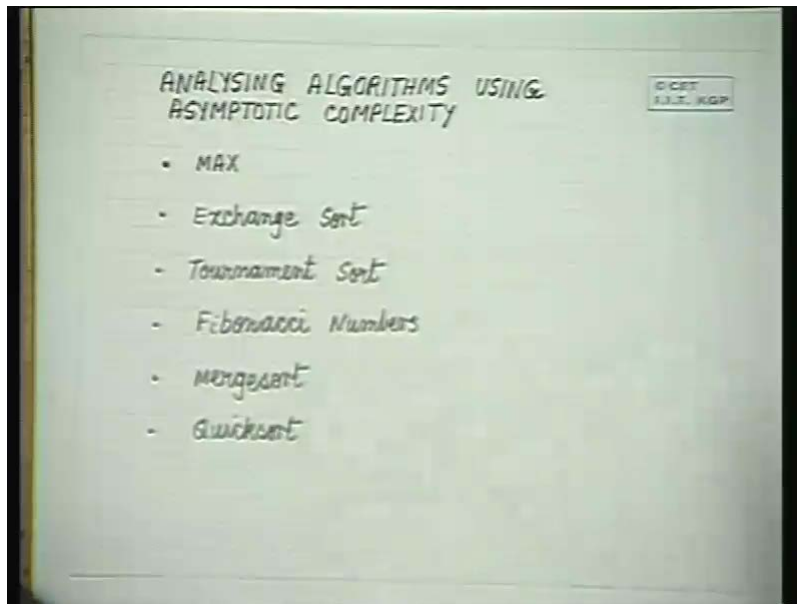


Programming & Data Structures
Dr. P.P. Chakraborty
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 20
Asymptotic Analysis of Algorithms

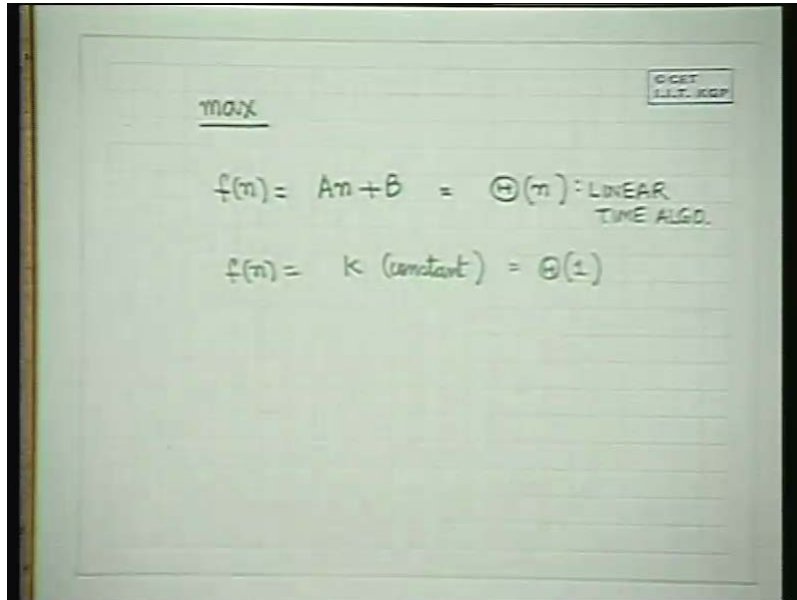
We shall continue our study of asymptotic analysis of algorithms. And today after having seen growth functions the order notation, the upper bound, the lower bound notations we will now try and analyze some of the algorithms or programs that we have written till now and see what this analysis means and also study some of the techniques of analyzing or trying to find out these asymptotic growth functions. So we will pick up quickly these problems, the maximum of n numbers, the exchange sort, the tournament sort, Fibonacci numbers, the merge sort and the quick sort, these are the one that we have done.

(Refer Slide Time: 01:46)



We have also done towers of Hanoi, permutations, the standard problems which we shall come to later on when the time permits. The maximum of n numbers was something which we did in the pervious class and without writing the routine again we found out that $f(n)$ is of the form $A n$ plus b and we can say this is order n .

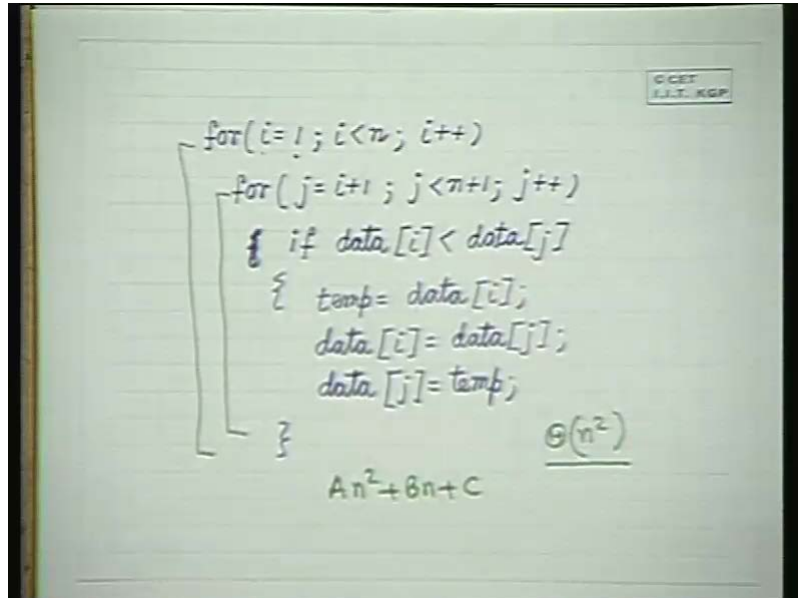
(Refer Slide Time: 03:08)



Just before we continue with this, if $f(n)$ is equal to k which is a constant then it is said to be order one, this constant is usually called order one. It is the same as order zero, order three, order four, order five, all of them mean the same thing but the language which we use is constant complexities order n , when we say order one. When we say order n , we say it is a linear time algorithm. Now if a function is varying with n and you have to read all the inputs, anyway it cannot be better than linear time. But the constant minimum is something we can analyze a bit later on but presently we are only bothered about asymptotic analysis.

So finding the maximum, finding the minimum, all of them are linear time. And you really cannot find the maximum of n numbers in less than linear time. So that part is over if it is unsorted, if it is sorted then obviously you can find it in constant time. If it's already sorted and you know finding maximum of n numbers is obviously constant but an unsorted arbitrary list of n numbers, finding out the maximum is the best possible order that you can get asymptotically, the order is linear.

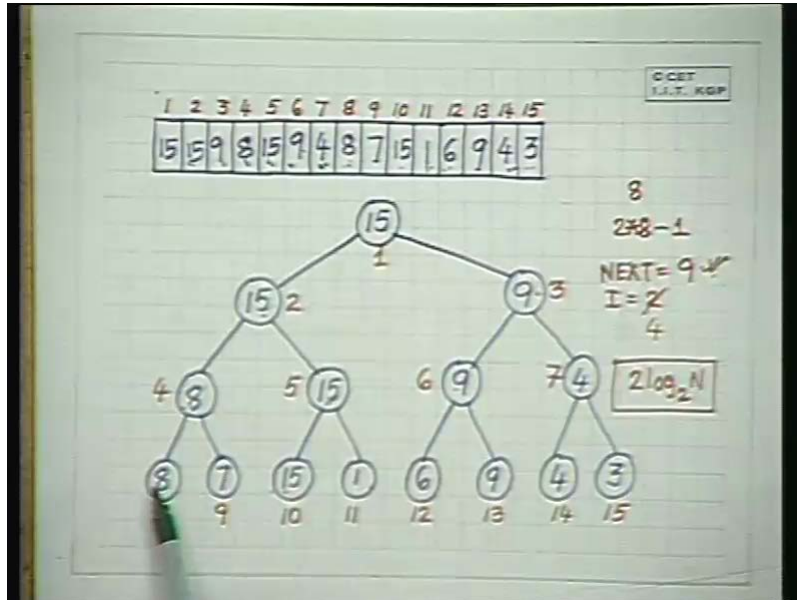
(Refer Slide Time: 05:52)



So, the next one which we will come to is the exchange sort. The exchange sort, the core part of it there are two loops, for i equal to 1 i less than n or if you wrote 0 then you would have written n minus 1. For j equal to i plus 1 to j equal to n that is less than n plus plus. If $data[i]$ is less than $data[j]$ then this is what you do. So if you make a quick analysis then this part will be executed once, this will be executed n times, this will be executed n times, this will be executed n times. In a loop, this will be executed n into n minus 1 whatever this similarly. And the inner loop will be executed means, what order of complexity you can find the exact, you can put 1 1 1 and find the exact value but if you have got the hang of order of analysis, you can say it will be of the type of a n square plus b n plus c .

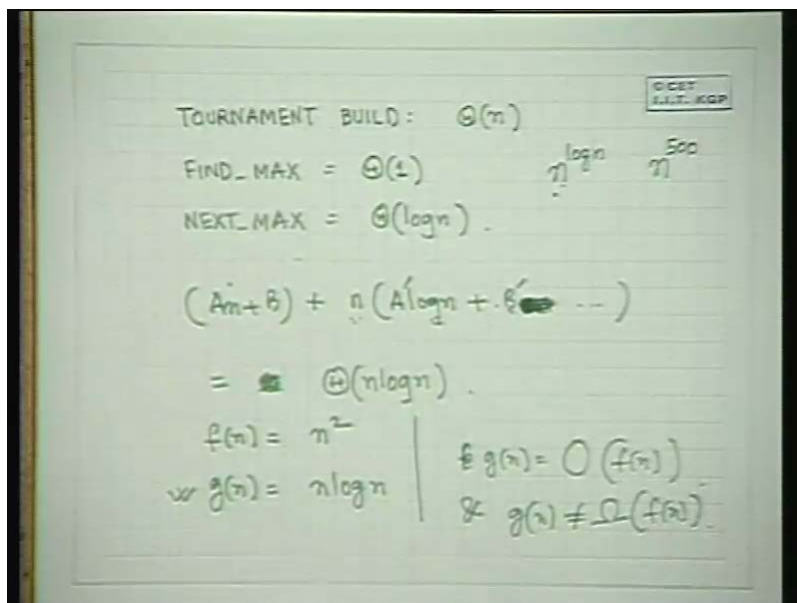
Very quickly you can say it is going to be of this type, though the constants will be dependent on what you do and you can say that exchange sort is order n square algorithm. So this is an order n square algorithm. Is that okay?

(Refer Slide Time: 06:08)



Let us come to this tournament. You remember, this diagram of a tournament in which we found out we had these n numbers and we used an array of size $2n - 1$ and we compare 2 input in the index of the half place. So building up of this tournament we did in a loop from back, compare these two 14 and 15 and put it in 7. 6 and 9 and put it in 9, 15 and 1 and put it in 5, 8 and 9 and put it in 4 here and this is what we did. So this loop is of order n . If this is $2n - 1$, this will be approximately proportional to $2n - 1$. So building up the tournament is of order n time. So tournament build, if you do not understand please stop me.

(Refer Slide Time: 11:21)



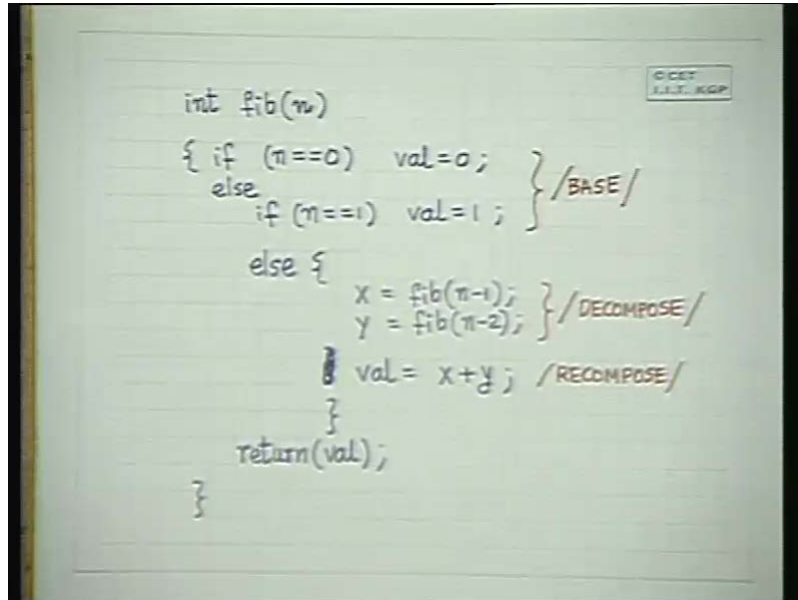
Tournament build is order n . find out the maximum element is constant time after tournament build. Get next max, what did we do? We went along the maximum and if we assume the, we know the minimum then you make it less than the minimum and then you go up. Now how many comparisons do you make when you go down? $\log n$ and when you go up, $\log n$. So to get the next max, you make two $\log n$. two $\log n$, so next max and if you repeat this next max continuously then you will require some, this is $f(n)$ isn't it, which is $An + b$. Find_max, Next_max this is what you will continuously do plus n into this was of the form $a \log n$ plus any terms which are less linear time terms etc etc Bn plus whatever it may be happen, it may continue.

So this term will be of a , **this is a one** say this is a one say this is a dash, this is b dash. So this whole term very quickly you can see that the highest order term will be $a \log n$. No, now **b dash, b dash n , b dash n is not here sorry** b dash may be some constant time may be required, there is not n sorry then this would have been n order n because $\log n$ is not there. So this, the whole sorting of n numbers will be done in order $n \log n$ time which is much better.

So now let us see, one we have got $f(n)$ is equal to n square. The other we have got $g(n)$ which is equal to $n \log n$. these are the two orders, order function. Now clearly $g(n)$ is upper bounded by $f(n)$ and $g(n)$ is not lower bounded by $f(n)$ but these two are not of the same order. Therefore this is much better than this. there are various other issues relating to order specially when you consider complexities of the type n to the power say $\log n$, how does it compare with n to the power 500? How will you compare these two? Take the logs on both sides and compare and you will get a comparison between both of them. If you cannot directly compare these two, take log on both sides and compare. This will be better than this.

See this one, its growth is a polynomial where this growth, this $\log n$ will increase with n . So, this one will grow faster than this one. So between these two algorithms, this will be better than this. In fact for any constant n to the power k , for any fixed constant may be even 2 milli ampere, it will be better than this in order terms. So we have got this tournament sort is order $n \log n$ algorithm, therefore its more efficient in asymptotic worst case complexity than a exchange sort but if you take 100 elements may be you just don't know because the constants may be very big here.

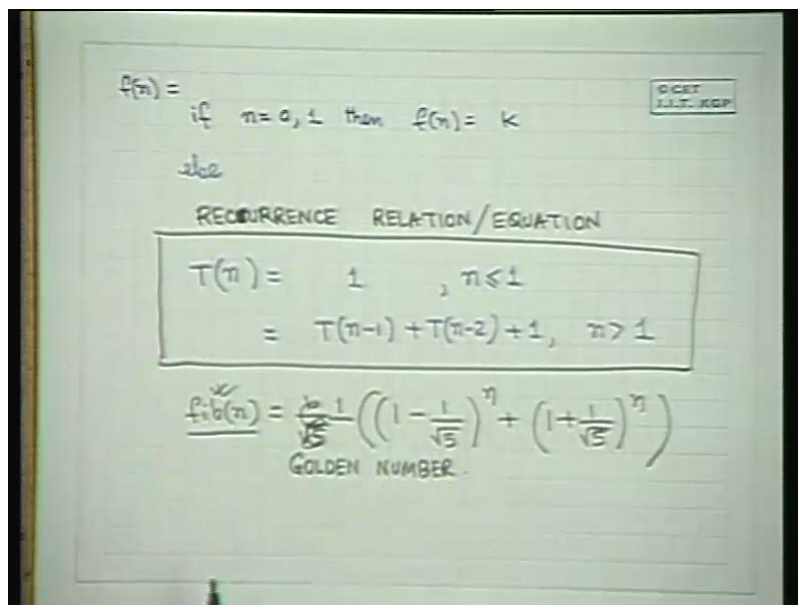
(Refer Slide Time: 12:24)



```
int fib(n)
{
  if (n==0) val=0;
  else if (n==1) val=1; } /BASE/
  else {
    x = fib(n-1);
    y = fib(n-2); } /DECOMPOSE/
    val = x+y; /RECOMPOSE/
  }
  return(val);
}
```

Let us come to the Fibonacci number and here we come, in the previous two cases we had very simple loops which we could analyze. In the Fibonacci number problem, how will we find out, what is the complexity of this? Can anybody help, any suggestions. What is the complexity of this? If n is equal to 0 then it is one check constant time, it is constant time. If n is equal to 1 it is constant time, otherwise **otherwise** what?

(Refer Slide Time: 15:59)



$f(n) =$
if $n=0, 1$ then $f(n) = k$
else

RECURRENCE RELATION/EQUATION

$$T(n) = 1, n \leq 1$$
$$= T(n-1) + T(n-2) + 1, n > 1$$
$$\underline{fib(n)} = \frac{1}{\sqrt{5}} \left(\left(1 - \frac{1}{\sqrt{5}}\right)^n + \left(1 + \frac{1}{\sqrt{5}}\right)^n \right)$$

GOLDEN NUMBER.

So we have been able to say for Fibonacci numbers $f(n)$ is equal to let us write it out like this. If n is equal to 0 or 1 then $f(n)$ is equal to constant k is order 1 else what? It is the

time to do this plus the time to do this plus another constant which is say one. So now the complexity equation again becomes a recursion and the complexity equation, the time n for Fibonacci is equal to let us say constant we can make it one, we are doing asymptotic analysis for n less than equal to 1 and is equal to T of n minus 1 plus t of n minus 2 plus 1 for n greater than 1. We have not found the solution, we don't know the order yet, this is what we have. No, can you tell me what order this is? This is called a recurrence relation or a recurrence equation. This one can be solved. Do you know the answer to this one? Can you solve it in terms of Fibonacci n ?

See it looks very similar to Fibonacci numbers itself. by the way do you know what is the value of $\text{fib}(n)$, $\frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2} \right)^n$ or something like that. It is like this something like this, it's called a golden number but this $T(n)$ can be written in term of $\text{fib}(n)$ you know. Can you make a guess? Is $T(n)$ equal to $\text{fib}(n)$?

(Refer Slide Time: 18:50)

The image shows handwritten notes on a slide. At the top right, there is a small box containing the text '© CECI I.I.T. KGP'. The main content consists of several lines of mathematical equations:

$$T(n) = 1, \quad n \leq 1$$

$$= T(n-1) + T(n-2) + 1, \quad n > 1$$

$$T(n) = \text{fib}(n+1) - 1?$$

$$T(n) = \text{fib}(n) + \text{fib}(n-1) + 1$$

$$= \text{fib}(n+1) + 1$$

$$T(n) = \text{fib}(n+1) - 1?$$

$$T(n) = \frac{\text{fib}(n) - 1}{\text{fib}(n+1) - 2} + \frac{\text{fib}(n-1) - 1}{+1} + 1$$

Just, what is $T(n)$? $T(n)$ is equal to 1, $\text{fib } n$ plus 1, is it equal to $\text{fib } n$ plus 1. How will you prove this yes or no, how will you prove it? Substitute here by induction, $\text{fib } n$ plus 1, so this will be $T(n)$ if $T(n)$ is $\text{fib } n$ plus 1 then this by induction must be fib of n plus fib of n minus 1 plus 1 which will be fib of n plus 1 plus 1 which is not true. Now looking at this, can you make an adjustment? Put $\text{fib } n$ plus 1 minus 1, then you will get 1 minus 1 here, you will get 1 minus 1 here and you will get 1 minus here.

$T(n)$ is equal to fib of n plus 1 or not? This is not true right, if it were true by induction this would have happened. $T(n)$ minus 1. So by induction this should have been fib of n plus fib of n minus 1 plus 1. Now fib of n plus fib of n minus 1 is fib of n plus 1, so you will get fib of n plus 1 plus 1 which is not true. But if you put it just 1 minus 1 here, you will get a minus 1 here, will get a minus 1 here, you will get a minus 2 here and this will

give you fib of n minus 1. So even if you don't know this, you can say $T(n)$, this much you can say. Isn't it? Now fib of n plus 1 is an exponential number, just see.

(Refer Slide Time: 20:13)

Handwritten notes on a slide:

$f(n) =$
 if $n=0, 1$ then $f(n) = k$
 else

RECURRENCE RELATION/EQUATION

$$T(n) = 1, \quad n \leq 1$$

$$= T(n-1) + T(n-2) + 1, \quad n > 1$$

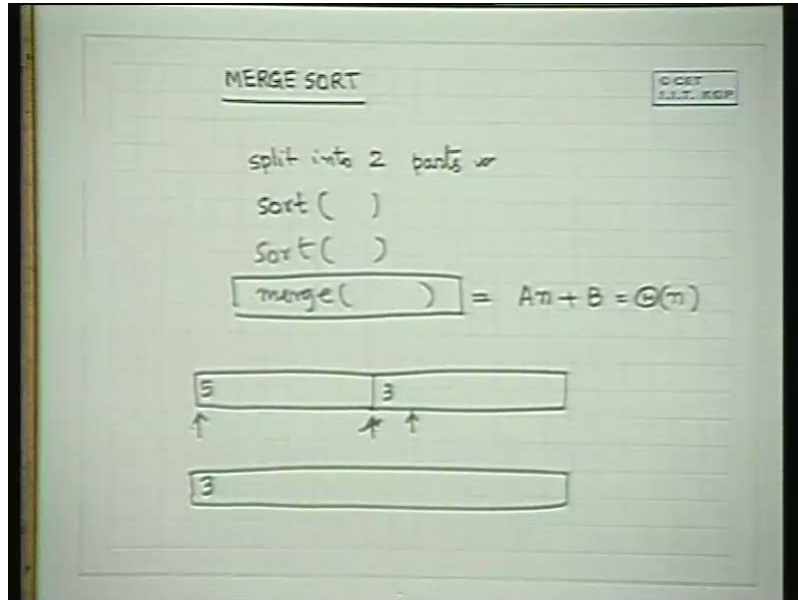
$$\text{fib}(n) = \frac{1}{\sqrt{5}} \left(\left(1 + \frac{1}{\sqrt{5}}\right)^n + \left(1 - \frac{1}{\sqrt{5}}\right)^n \right)$$

GOLDEN NUMBER.

$$T(n) = \Theta(\text{fib}(n+1)) = \Theta(2^n)$$

So $T(n)$ will be you can always write, atleast it will be upper bounded by 2 to the power n. It will also be lower bounded by 2 to the power n, you can show that. I just leave it to you to try out because these are both numbers which would be exponential, the growth of this will be exponential. So this algorithm takes is of order exponential time 2 to the power n, it grows very rapidly compared to an algorithm like n square or n cube which grows as a polynomial function of n. You have to see what we can do about this. But the one point that we get from analyzing recursive equation is that these recursive definitions, the complexities of these recursive definitions can be formed by solving certain recurrence equations. Is that idea clear? That the solution to recursive definition are the complexity, time complexity, asymptotic time complexity of recursive functions or inductive definitions can be obtained by solving certain recurrence relations.

(Refer Slide Time: 23:15)



And we have already got several such problems that are in hand and we will see one of them. Let us see the merge sort problem. If you remember the merge sort problem was if we split into two parts, equal parts or unequal parts or whatever. So split into two parts and then sort individually and then merge the two sorted list. And we did not decide on where to split, right. We did not decide on where to split, we said we will decide later on. Now is the time we will see, if we split in different ways what we will get.

Now first let us understand what this merge routine does. This merge routine it takes in our final implementation in the same array a set of numbers one pointer here, one pointer here and in another array it compares these two, the smaller of the two if you are sorting in that order is written. Suppose this is 5 and this is 3 then 3 will be written and this pointer will move here, this index would move here.

So for every number which is written here at most one comparison is done, at most one comparison is done. So for finding out merging of m plus say m_1 plus m_2 numbers so total of n numbers if I have to merge, I have to do some... My complexity will be of the form like this no more than this because I do not have to do more than n comparison and more than n comparison and more than n such assignments here. That's all I have to do. So merging of n elements is order n time, so we know merging of n elements is order n time.

So we come to merge sort, we are splitting into two parts. Now we could split one n minus 1 2 n minus 2 3 n minus 3 or we could split up half half three fourth one fourth one tenth nine tenth. So suppose we split up into 1 and n minus 1. Then what will be our equation? $T(n)$ is equal to constant for n equal to 1 that is known and there is one element, it is constant time.

(Refer Slide Time: 26:46)

Handwritten derivation of a recurrence relation:

$$\begin{aligned}
 T(n) &= 1, \quad n=1 \\
 &= T(i) + T(n-i) + n, \quad n > 1 \\
 &= T(n-1) + n + 1, \quad n > 1 \\
 &= (T(n-2) + (n-1) + 1) + n + 1 \\
 &= T(n-2) + n + n + 1 \\
 &= T(n-3) + (n-1) + n + (n+1) \\
 &\vdots \\
 n - (n-1) &= T(1) + n - (n-3) + 3 + 4 + 5 + \dots + (n+1)
 \end{aligned}$$

The final result is identified as $\Theta(n^2)$.

Otherwise it is $T(1)$ plus $T(n)$ minus 1 plus merge time, this is order n . Order n you can write as n also because it will not affect the order of the complexity which is the same as $T(n)$ minus 1 plus n plus 1. So $T(n)$ is $T(n)$ minus 1 plus n plus 1 which is, which starts expanding it out. How will you solve this? Can you solve this? Try it, expand it, $T(n)$ minus 2 plus n minus 1 plus 1 plus n plus 1. This $T(n)$ minus 1 will break up as $T(n)$ minus 2 plus n minus 1 plus 1 like $T(n)$ broke up as n minus 1 plus n plus 1, this will break up as n minus 2 plus n minus 1 plus 1. So it will be $T(n)$ minus 2 plus n plus n plus 1.

The next step you can write in one short $T(n)$ minus 3 plus n minus 1 plus n plus n plus 1. And this will go right up to $T(1)$ plus when n , when this is n minus three this starts from n minus 1. So when this is 1 which is equivalent to n minus n minus 1, 1 is equivalent to n minus n minus 1, it will start from n minus n minus 3. If this is 3 this is 1, so if this is n minus 1 this is n minus 3 which is the same as 3, 3 plus 4 plus 5 plus plus n plus 1.

So this will be $T(1)$ plus 3 plus 4 plus 5 or something like that. Isn't it? Now this is a series summation which will sum up to order, straight write can you write the order term. n square. So if you split up 1 n minus 1 2 n minus 2, we are going to get an order n square algorithm, alright.

(Refer Slide Time: 30:04)

OJET
I.I.T. KGP

$$\begin{aligned}
 T(n) &= 1, \quad n=1 \\
 &= 2T(n/2) + n \\
 &= 2(2T(n/4) + n/2) + n \\
 &= 2^2 T(n/2^2) + \cancel{n} + n \\
 &= 2^2 (2T(n/2^3) + n/2^2) + n + n \quad (+n \log n) \\
 &= 2^3 T(n/2^3) + n + n + n \\
 &\vdots \\
 &2^k T(n/2^k) + kn = \boxed{n + n \log n}
 \end{aligned}$$

Let's see the, if you split up n minus 1 1 also you are going to get the same thing but suppose we split half half then our equation is $T(n)$ is equal to 1, n is equal to 1 and is equal to 2 of $T(n)$ by 2 for the 2 half's n by 2 plus n by 2 plus n . So we have to solve this one, lets start expanding, two of expand $T(n)$ by 2, $T(n)$ by 4 plus n by 2 plus n which is the same as 2 square $T(n)$ by 2 square plus n by 2 to the power 1 plus n by 2 the power 0.

This will come in sorry, this will n n n , 2 into n by 2 this will be n , am sorry this is not this. This part is okay. What is this part? n plus n , right. This is it. This will be again broken up 2 square 2 $T(n)$ by 2 cube plus n by 2 square plus n plus n . This is 2 cube t , if you stop me if you cannot follow.

Now I can write the general term. Can anybody tell me the general term? $2^k t$ n by 2 k plus kn , alright. So now what happens? When does this become one? When n by 2 k is equal to 1 which implies n is equal to 2 k which implies k is equal to $\log n$ to the base 2. At that point of time, this is 1, 2 k is n so this becomes n into 1 plus $n \log n$ which makes it order $n \log n$. So now if we split half half, we have got $n \log n$. If we split 1 n minus 1, we get n square. So you remember what we were talking about in the initial class when we said that you leave that split alone. These are all the possible ways of designing, we will analyze it and then we will choose. Now you can check if I split three fourth, one fourth what will happen or you can write out an equation like this.

(Refer Slide Time: 35:18)

The image shows a piece of lined paper with handwritten mathematical equations. At the top, it states $T(n) = 1, n = 1$. Below this, it shows a recurrence relation: $= \min_i \{ T(i) + T(n-i) + n \}$. A horizontal line is drawn below this equation, and the word "QUICKSORT" is written and underlined. Below the line, the recurrence relation is simplified to $T(n) = 1, n = 1$ and $= 2T(n/2) + n$. In the top right corner of the paper, there is a small rectangular stamp that reads "© CRET I.I.T. KGP".

I want, what is my choice of algorithm? $T(1)$ is this, otherwise minimize this function for whatever I minimum. Find out the minimum value of i , find out that value of i for which this is minimized and solve this equation. And if you can solve this equation, you will get the optimal choice. This I may depend on n , it may say if n is 5 then you choose two. If n is 7 you choose three, if n is 9 you choose whatever it is or you may get a very simple solution like always choose i is equal to n by 2. You may not, this may not lead to a very simple equation, simple solution but this is what we have to do if we want to optimize your choice. This is what is meant by the choice of that split. Is that clear? So we have now got an idea that in merge sort, if I split it into two equal half I get an order $n \log n$.

So I will use, you will see all books of merge sort split into half because there is another proof that this minimizes i equal to n by 2 minimizes it. But there are other proof, if you write n by 4 and 3 n by 4 you will still get $n \log n$. If you split it up in any fraction, you will get $n \log n$, half you will get $n \log n$. There are more details you can prove that if you split in half you spend, you do less comparisons than you do when split in three fourth also that you can choose but in terms of asymptotic complexity, we have reached this point in merge sort saying that merge sort is best, we can prove which I am not doing here. Merge sort is best when you split into half.

Let us have a look at quick sort, very similar. Quick sort equation is also very similar. $T(n)$ is equal to 1 for n is equal to one and it depends on where your pivot element splits you. Now the pivot element you have chosen randomly, alright. The pivot element is chosen randomly, so it can split you in one n minus one.

So in the worst case quick sort can become ordered n square because choice of pivot element is not in your hands but if you could choose the pivot element in such a way that it broke up the list into half half then you would have got order $n \log n$ algorithm. But

then how will you find out how to split into half half? That is if you could split in linear time, look you cannot take n^2 time to split into half half that will spoil your whole complexity equation. So, given a set of elements which is that element which splits it into half half, the median. So you have to find the median in linear time. If you could find the median elements of n elements in linear time then you would choose the median and do a quick sort based on that median.

Then you would have got worst case order $n \log n$ time but then how will you find the median in linear time. All of us find median by sorting. It's not a chicken and egg problem, there is a method to find the median in linear time but that algorithm is slightly more complex. I will not go into it today but you can find the median in linear time. So in the worst case quick sort is order n^2 , in the best case it is $n \log n$ provided you can get it. Then you can start proving in the average case what happens and other things, we will discuss it later on. but our tournament sort and merge sort in the worst case they are order $n \log n$ algorithms and therefore they are better in the worst case scenario than quick sort or exchange sort or bubble sort but the merge sort has to be done by splitting half half, not arbitrarily. So asymptotic analysis helps you a lot, not only to find out how good your algorithm is, it helps you to decide on how to do recursive decomposition and where to choose some of your choice points which you may have left for the analysis part.

It will also help you to understand which data structure is better and how to analyze the efficiency of your data structure which we shall come into the, which we shall discuss later on. But this phase of understanding what is meant by complexity, what is meant by orders, how do we analyze complexity, how do we use recurrence equations etc will form the basis of the second phase, the last two parts of this course mainly data structures, complete design of algorithms as well as data structures which we shall pick in the final phase of this course.