Programming and Data Structure

Dr.P.P.Chakraborty
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture 26

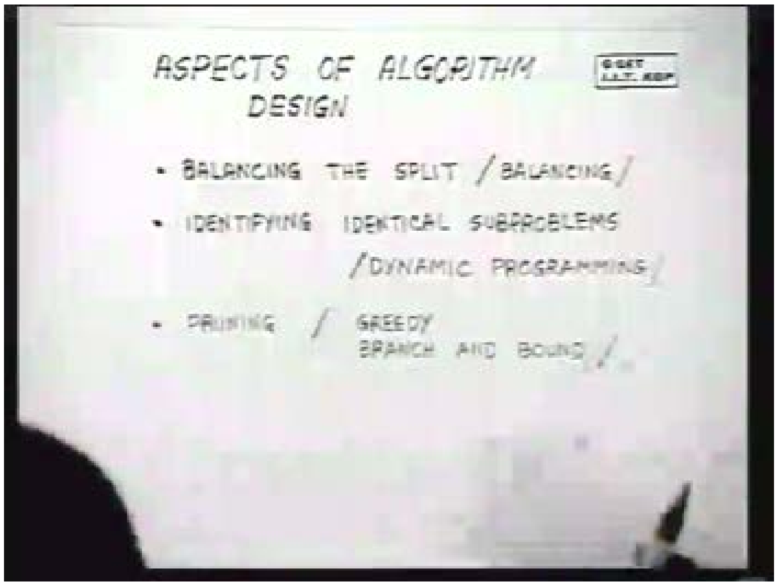
Algorithm Design -1

Having made a preliminary study of data structure design, we shall now go back and study some aspects of the design of algorithms and try to make out how to really what are the basic steps which are involved in algorithm design aspects which are not purely data structure design concepts. We will just revisit some examples which we did before and we will see some of the very important issues which are related to the design of the algorithms and related to problem decomposition and how to generate a good algorithm from an initial problem or from an initial solution. We have seen before that we will use recursive definitions to define a problem and now we will see how given a recursive definition, how we could analyze it further to obtain an algorithm which does not have some of the defects of the original recursive algorithm. And later on this algorithm design and data structuring will go hand in hand to make a complete design of an algorithm, to constitute a complete design of a solution.

So we will have a quick look at these aspects of algorithm design because these two aspects will be integrated for program development. Among the three, we will study three major concepts, there are three major aspects one of them we could term, we have seen this idea before balancing the split when we did finding the minimum of n numbers or sorting of n numbers. And we discussed the issue of where to split the list into two parts and how it leads to recurrence equations which are to be solved to find out the optimal split. So this is one aspect. The second aspect this is usually called balancing in classical algorithm design terminology. The second is identifying identical sub problems, you will see that when you give a recursive solution, several sub problems which are identical and you would like to solve them only once rather than solving them again and again. And the technique for identifying identical sub problems and solving them only once has got a name which you will see in textbooks, that book is called dynamic programming.

It has a different meaning or different style in other aspects where you come to state machines and there you talk of dynamic programming in a slightly different sense but inherently both of them have got very very similar meanings and third which we call pruning.

(Refer Slide Time 04:53)



ASPECTS OF ALGORITHM DESIGN

- BALANCING THE SPLIT /BALANCING/
- IDENTIFYING IDENTICAL SUBPROBLEMS /DYNAMIC PROGRAMMING/
- PRUNING / GREEDY BRANCH AND BOUND /

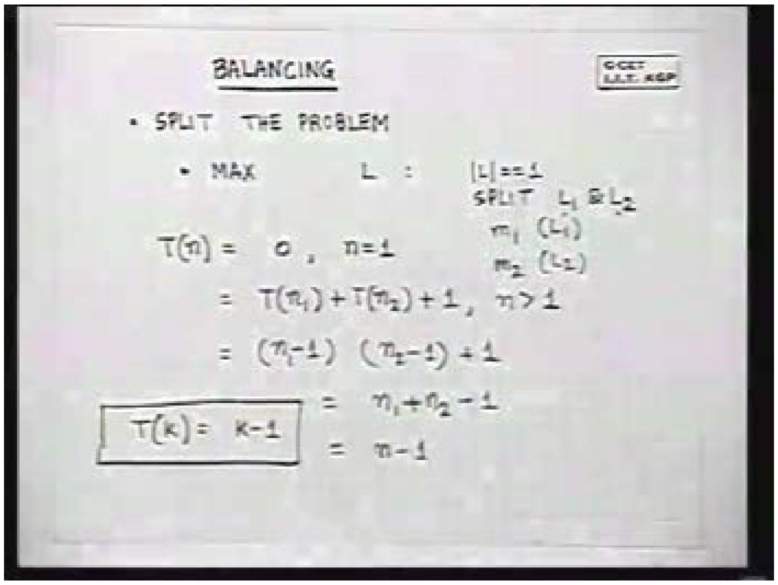
In word you try to see which of, you can make choice between sub problems and choose only one of them or decide that this sub problem need not be solved to solve the original problem and here you read two types of strategies we will see, one is called the greedy approach the other is called the branch and bound. So we will see what these things mean and try and get a flavor of how algorithm design involves these three core aspects. That is now formal methodology for this but these are some intuitive ideas which can be formalized by trying out various solutions and solving then you will get very good ideas. I would not go into actual formal definition of all this because that would need more terminology.

So, the first aspect we will take is balancing because we have seen this before. And to quickly recapitulate, in balancing we have given a problem and where we had split the problem into two or more parts. And in the solution, we did not choose where to split in the initial solution we will only analyze and then say where to choose. Let us see some example. Suppose we have to find the maximum of n numbers. Now to find the maximum of n numbers, given a list L what was our recursive solution? If the size of L was 1 then obviously the problem is the base condition otherwise you split into two non-empty sets L_1 and L_2 all right and then solve these two recursively and after you solve these two recursively, you get m_1 and m_2, m_1 from L_1 and m_2 from L_2 and then you compare these two and get the final solution.

So where will we split? We will be splitting in the middle, we will split left, we will split right, so to answer that question we saw that you have to solve recurrence equations. For example here the recurrence equation is related to the number of comparisons that have to be made because here the number of comparisons for n numbers is equal to 0 when n is equal to 1 and is equal to the number of comparisons in the first split list plus the number of comparisons in the second split list plus 1 when n is greater than 1, all right.

So this was how we solved the problem and if you open up this you will see that this will come to, you replace T (i) T (k) is equal to k minus 1 and you will see that the recurrence is solved. T (k) is equal to k minus 1, the solution to this what ever be n_1 n_2 will be k minus 1 because we replace this, this will be n minus 1 n_1 minus 1, this will be (n_2 minus 1) plus 1, this will come to n_1 plus n_2 minus 1 and we know n_1 plus n_2 is equal to n because you have split n into n_1 and n_2, so n_1 this will be n minus 1. So what does this reveal to us?

(Refer Slide Time 09:03)



BALANCING

- SPLIT THE PROBLEM
  - MAX L : $|L| == 1$

SPLIT L_1 & L_2

n_1 (L_1)

n_2 (L_2)

$$T(n) = 0, \quad n=1$$
$$= T(n_1)+T(n_2)+1, \quad n>1$$
$$= (n_1-1) \; (n_2-1) + 1$$
$$= n_1+n_2-1$$
$$= n-1$$

$$\boxed{T(k) = k-1}$$

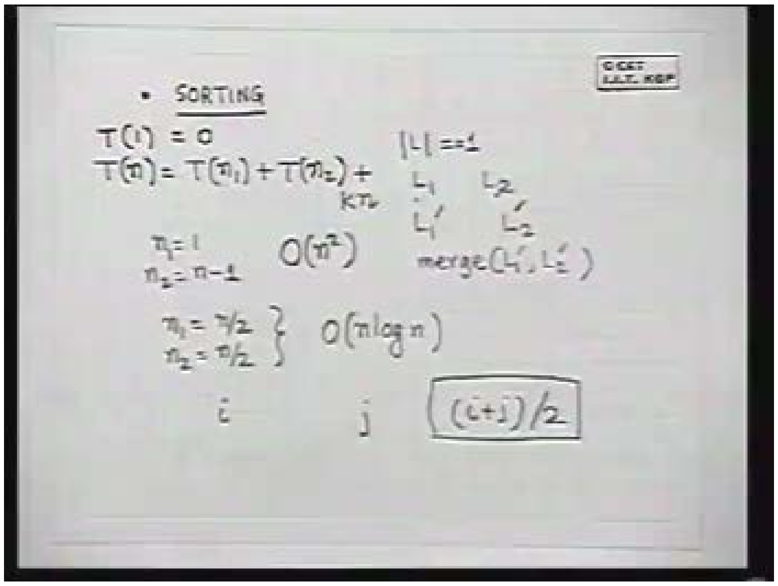
This reveals to us that whichever way we split, whatever of the values of n_1 and n_2, you are going to require n minus 1 comparison. So you can split it in anyway you like and the algorithm will be correct. So the choice is now clear you can choose anything you want, you split 1, n minus 1, 2 n minus 2 or what ever and therefore the problem can be easily solved and it will be optimally solved all of them are optimal. So, now we are left, suppose we make a choice and we make 1 n minus 1, now the next step is what have we chosen for representing L. So we have balanced the problem optimally and we have chosen 1 n minus 1. So now we are left to choose the data structure for L.

Till now L is a set. So what are the operations on L now? If it is 1 n minus 1, it is removed the front of the list, so head and the rest of the list. So if you remove the front of the list, you will get two parts. So now suppose we are given L and you read in the element, first you read in the number of elements and then so you can use an array to solve this problem without any difficulty or even if you use a linked list to solve the problem also there is no problem, there is no difficulty. So using an array or a linked list for this set L and choosing the first and rest, the problem can be solved optimally. So this is using a very very simple problem, I have shown you how you split the problem, how you have chosen the data structures for l and that’s all. You can now finalized the complete… all right. Any questions?

Let’s take another problem sorting. And here in the merge sort style, we saw that when L is equal to 1 the problem is solved, otherwise again you split into L_1 and L_2 which are non-empty. You recursively solve these two to get L_1 dash and L_2 dash and then you merge these two by a merge algorithm. And we saw the recurrence equation which crops up in this situation is T (n) is equal to $T(n_1)$ plus $T(n_2)$ plus some k n_1 because the merge algorithm is linear time, we know that merge algorithm can be done in n comparisons when there are n elements. So you can choose any constant into n, this is what it will take and T(1) is equal to 0 and we saw that if you take one n_1 equal to 1 and n_2 is equal to n minus 1 then this will lead to order n square.

On the other hand if you take n_1 is equal to n by 2 and n_2 is equal to n by 2, you will get order n log n. So you have decided to split in the middle. Now once you decide to split in the middle, the choice of split is now made, so we are left to design the data structure for L.

(Refer Slide Time 12:49)



- SORTING

$T(1) = 0$ $\quad |L| == 1$

$T(n) = T(n_1) + T(n_2) + kn$ $\quad L_1 \quad L_2$

$L_1' \quad L_2'$

$n_1 = 1$, $n_2 = n-1$ $\quad O(n^2)$ $\quad$ merge(L_1', L_2')

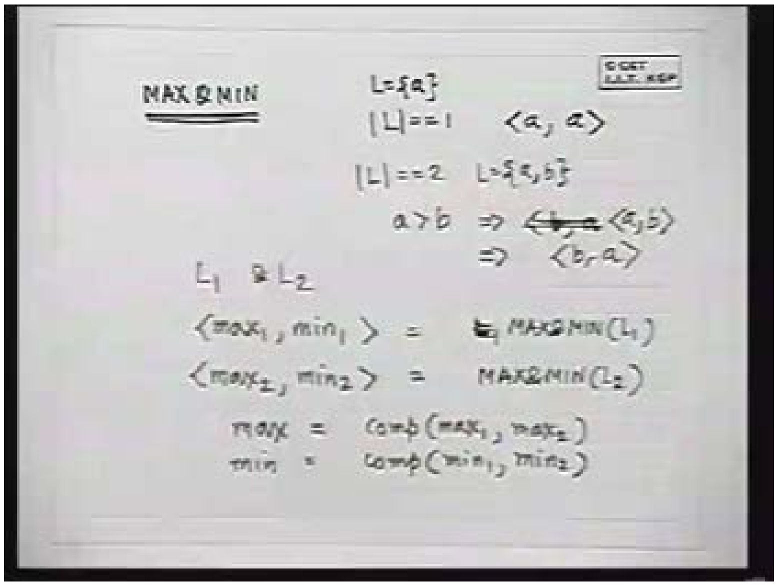
$\left.\begin{array}{l} n_1 = n/2 \\ n_2 = n/2 \end{array}\right\} O(n \log n)$

$i \quad j \quad \boxed{(i+j)/2}$

Now if we choose a linked list to make a data structure for L, splitting will again be expensive because you will start from the beginning and split. You will have to scan from the beginning and find out the middle element. On the other hand if you could work it out on an array and if you knew the i th and the j th, the start and the end index then the middle element can be simply chosen by this computation. So if you could know the dimensions, the maximum limits on the array then obviously it is best to use an array.

On the other hand even if you don’t know after you read all the elements, you just put them into an array, malloc an array of that size, you know these are the elements, you malloc an array of that size and work on that, that will be first split because this split otherwise even for splitting you will take linear time that will not affect the complexity. Overall order n log n will remain because it will only add some more constants to this part. Splitting will take some k_1 n and merging will take some k_2 n and they will add up to form some k_3 n. So you will get T n by 2 plus n by 2 plus some k_3 n which is still the order n log n but you can array the access and the split may be at least constant but n log n is ensured even if you use a linked list. Is that understood? This is how you proceed to do a data structure design, subsequent to the design of the algorithm. Well, let's choose another problem which we have done in a slightly different way. Max and min, the maximum and the minimum of n numbers, similar we will use a similar approach. If L is equal to 1, size of L is equal to 1 then recursively the max and min returns 2 values. Isn't it? It will return a maximum and a minimum, so it returns the structure. So if L is 1 it returns a a, that is which is because L was equal to a in this case, only one element. If L is equal to 2 then it returns a comparison of, suppose L was a b then it compares a greater than b.

(Refer Slide Time 16:16)



MAX & MIN

$L = \{a\}$

$|L| == 1 \quad \langle a, a \rangle$

$|L| == 2 \quad L = \{a, b\}$

$a > b \Rightarrow ~~\langle b, a \rangle~~ \langle a, b \rangle$

$\Rightarrow \langle b, a \rangle$

L_1 & L_2

$\langle max_1, min_1 \rangle = MAX\&MIN(L_1)$

$\langle max_2, min_2 \rangle = MAX\&MIN(L_2)$

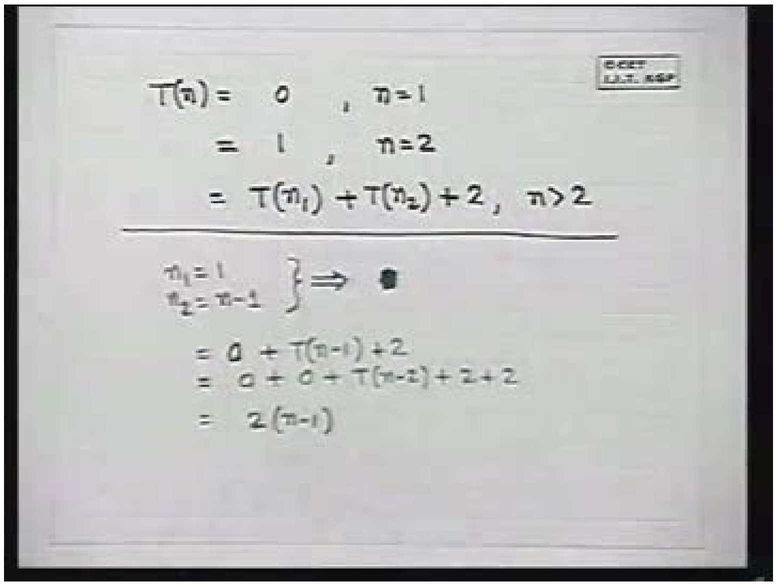
$max = comp(max_1, max_2)$

$min = comp(min_1, min_2)$

If a is greater than b then it returns b a sorry a, b else it returns b, a, all right. Now two elements, when you come to more than two elements, you split into L_1 and L_2 okay and recursively you will get max 1 and min 1 from solving L_1, all right. You recursively solve max and min L_1 that is you will write max and min L_1 to get max 1, min 1. You recursively solve max and min L_2, to get this. Now you compare max 1 and max 2 to get max is equal to compare max 1 and max 2 and min is equal to compare min 1 and min 2. Isn't it?

Suppose this base condition is clear, when L is of size 2 you make one comparison and you will get the maximum and minimum. When L is more than 2 then you split into L_1 and L_2 and you recursively solve L_1, L_1 will return some max_1, min_1, L_2 will return max_2 min_2. The maximum of L will be the larger of max_1 and max_2 and the minimum of L will be the smaller of min_1 and min_2, all right. Any questions? So this is one approach to solve the problem. So let us see what recurrence relation this gives us. This gives us T (n) is equal to 0 if n is equal to 1 is equal to 1 if n is equal to 2 and is equal to T (n_1) plus T (n_2) plus 2 for n greater than 2, all right. Now where will you split this one, where will you split this? If you choose n_1 equal to 1 and n_2 equal to n minus 1 then can you guess what this will lead to? This will lead to order, work it out 1 plus T (n minus 1) plus 2 sorry 0 (Refer Slide Time: 18:30) that will lead to 0 plus 0 plus T (n minus 2) plus 2 plus 2, 2 (n minus 1). Isn't it? Order n, order n but 2 (n minus 1).

(Refer Slide Time 18:56)



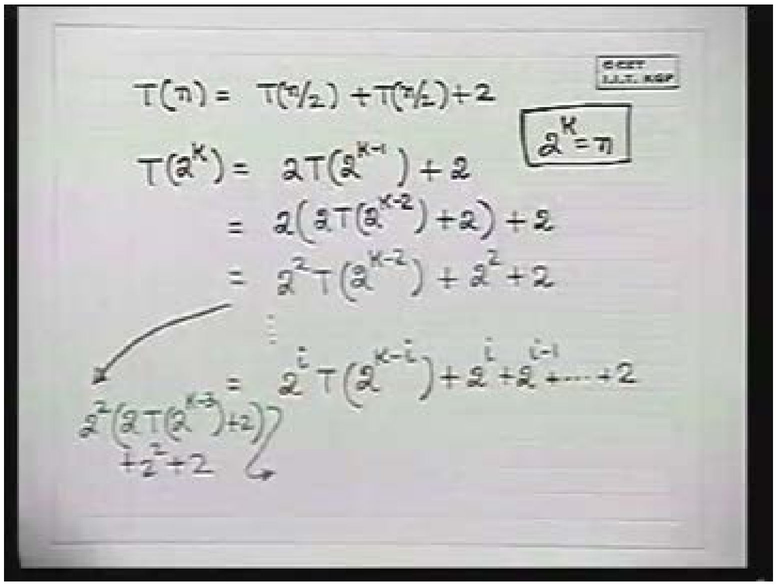
$$
\begin{aligned}
T(n) &= 0, && n=1\\
&= 1, && n=2\\
&= T(n_1)+T(n_2)+2, && n>2
\end{aligned}
$$

$$\left.\begin{array}{l} n_1=1\\ n_2=n-1\end{array}\right\} \Rightarrow$$

$$
\begin{aligned}
&= 0 + T(n-1) + 2\\
&= 0 + 0 + T(n-2) + 2 + 2\\
&= 2(n-1)
\end{aligned}
$$

Now suppose you split in the middle and let us take for example that there are only 2 to the power n elements and you split in the middle then what would you get? T (n) is equal to T (n by 2) plus T (n by 2) plus 2, right. So, suppose this is some 2 to the power k, it will make it easier to analyze, so T (2 to the power k) is twice T (2 to the power k minus 1) plus 2. We have chosen 2 to the power k equal to n, that is suppose n is 16, then 2 to the power 4. So this will, can you solve this? This is 2 into 2 T to the power 2 k minus 2 plus 2 plus 2, right. So that is equal to 2 square T to the power 2 to the power k minus 2 plus 2 square plus 2, 2 to the power i T 2 to the power k minus i, right plus 2 to the power i. Will this term come? Try another term and see if this term comes, 2 square and open up, 2 T to the power k minus 3 plus 2 plus 2 square plus 2. So you will get 2 cube k minus 3 plus 2 cube plus 2 square plus 2, right if you open up.

(Refer Slide Time 21:19)



$$T(n) = T(n/2) + T(n/2) + 2$$

$$\boxed{2^k = n}$$

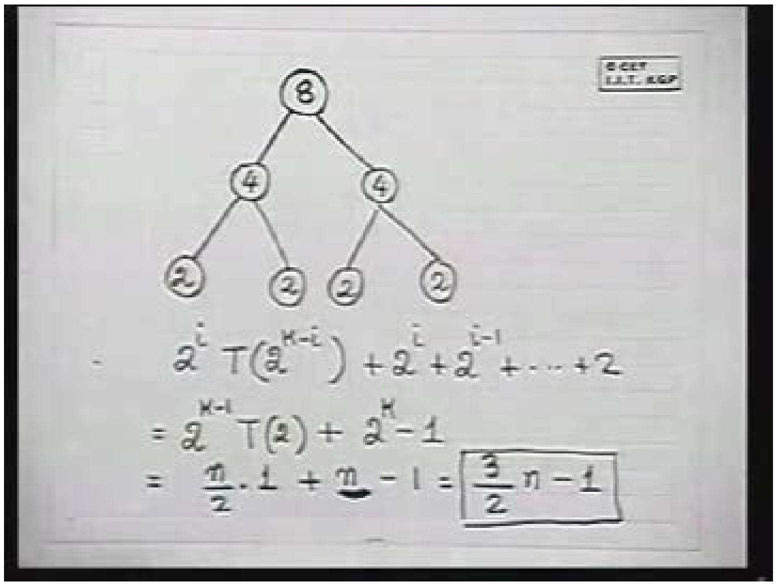
$$T(2^k) = 2T(2^{k-1}) + 2$$
$$= 2(2T(2^{k-2}) + 2) + 2$$
$$= 2^2 T(2^{k-2}) + 2^2 + 2$$
$$\vdots$$
$$= 2^i T(2^{k-i}) + 2^i + 2^{i-1} + \cdots + 2$$

$$2^2(2T(2^{k-3}) + 2) + 2^2 + 2$$

So what does this lead to? This 2 to the power 0 will make it up to 0, when this will become 0 when i is k, so 2 to the power k then T (1) plus what? Have I made a mistake somewhere? One minute. If this sums up, it will come to 2 power k plus 1 minus 1. Isn't it? I think I have made a mistake somewhere. No, T (1) is equal to 0, you can cut it out that's alright but this equation was T n_1 plus T n_2 that's fine. I think I have made a mistake somewhere, this is 2 to the power k minus 1, n by 2 is correct plus 2, it's okay. So this comes to twice n minus 1, this is not the correct solution. No, no, no this is not the correct solution, let us work it out for 8 and see how many comparisons come. Let us work it out for n is equal to 8. For 8 you need to make, you split it into 4 and 4. No, no, I have understood where I made my mistake, 2 2 2 2 right, you will not go beyond 2 because your recurrence stops at 2.

So we have actually stopped it at 1 but we should have stopped it at 2. Got it? So this part is correct. This whole thing is correct but it could have stopped it at T_2, it need not have gone till T_1, so let us proceed from there. So we were at 2 i T (2 to the power k minus i), this was, this is correct, the expansion is correct but what the mistake we did was we broke the recurrence, we made it 1, 2 to the power 0 that is 1 but the recurrence will stop at 2. So i will be k minus 1, so we will stop at 2 to the power k minus 1 T (2) plus this will be 2 to the power k minus 1, right. Is that understood now? That is the advantage of that T (2) because here we stopped at 2 not 1 and this would give us, how much is this, n by 2 into, T (2) is what? 1 plus n minus 1, so this gives me 3 by 2 n minus 1.

(Refer Slide Time 25:19)



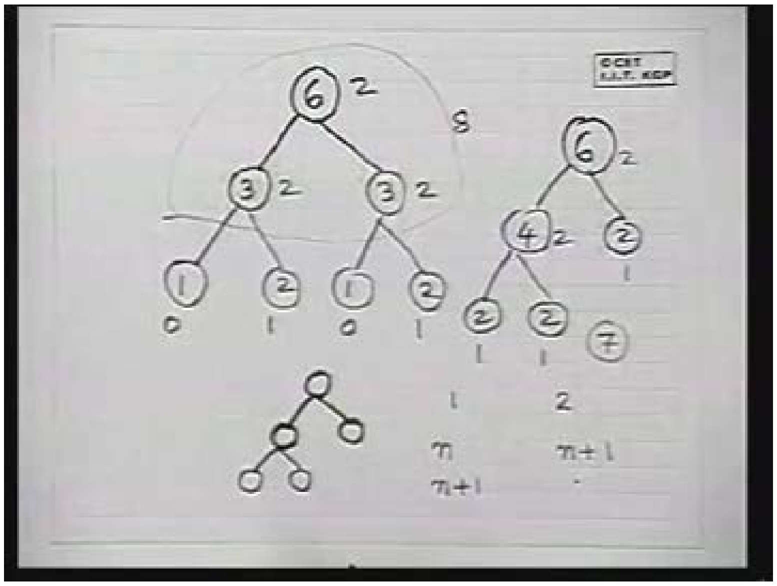
$$2^i T(2^{k-i}) + 2^i + 2^{i-1} + \cdots + 2$$
$$= 2^{k-1} T(2) + 2^k - 1$$
$$= \frac{n}{2} \cdot 1 + \underbrace{n} - 1 = \boxed{\frac{3}{2} n - 1}$$

So now see, while both were order n, splitting in the middle give me slight reduction in the number of comparisons. So I would still split in the middle. But is this the only option? Let us see. If you analyze it a bit more carefully, you will see that in any tree developed like this, see this will split up into things like this given say n it will split up into two parts n_1 and n_2 say n_3 n_4 like this. The leaf nodes will be of two types. If the value, if the n value here that is the number of elements is 1 or 2 and the internal nodes will have more than 1 or 2. Clear? Internal nodes will have more than 1 or 2. Now internal nodes are those nodes were the split occurs and the number of comparison at every internal node is 2. The number of comparisons at every internal node is 2.

At the leaf node it is 0 for this and 1 for this. So to solve the problem optimally, what would you do? Suppose you had got 4, the choice here is 4 can be split as 2 2 and 4 can be split as 3 1 and then 3 has to be split as 2 1. Now when ever you split it into 2 here, this is optimal compared to this, just have a look. Now let us see 6. I am not going formally purposely just to 6 could be split as 3 3, this is not 2 to the power n, 1 2 1 2. What is the cost? 0 1 0 1 2 2 2. What is the total cost? 8 and if I split it up, what is the total cost? 7. See even splitting in the middle is not optimal, to get an optimal split you have to analyze the tree. If you minimize the number of internal nodes, you have some external nodes. Now all internal nodes are of maximum cost, the external nodes cannot be both 1 1 unless the internal node is 2. So if you minimize the number of internal nodes, you will get the maximum benefit.

I repeat, every internal node is of cost 2, all right. Now if you minimize the number of internal nodes for if there are 2 internal nodes, there have to be only 3. If there are 2 internal nodes then how many external nodes can there be in a tree? If for two internal nodes can there be 4, for 1 internal node there can be 2. So let us take the base condition, for 1 internal node there will be 2 leaf nodes, maximum. Now if there are n internal nodes and there are n plus 1 leaf nodes, if you make 1 internal node for every internal node if you make 2 more leaf node then this will increase by 1, this will increase by 2 because 1 internal node has been reduced and become, one leaf node has become internal node.

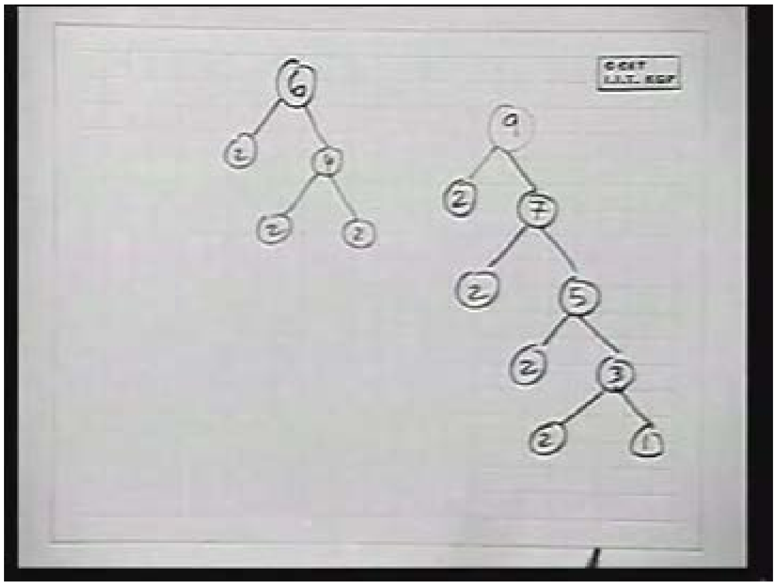
(Refer Slide Time 30:40)



So suppose see take 3, take this case there is 1 internal node and 2 leaf nodes. Now if you increase the number of leaf nodes, you will choose one of them. So one leaf node will go vanish and 2 nodes will get added. So, effectively one leaf node will get added. Isn't it? So you can show that for n internal node, there will be n plus 1 leaf node, maximum in this structure. And though I do not wish to go into more details, you can just analyze it a little and see that if you minimize the number of internal nodes here then you will get the optimal solution not even half half does not optimally split it. That worked because we chose 2 to the power of k and in 2 to the power k, everything will land up in, 2 is at the end. So the choice, a better choice would be if n is even split even even, if n is odd split odd even. If n is odd, one internal node of cost one has to be done.

So, if you choose a split this way that if n is even split even even, if n is odd split odd even. You will get the optimal solution. You can work it out in your rooms and check, it is even better than half half because half half will not split in that way. So, one choice is take out 2 at a time, suppose you have got 6 take two at a time out. Suppose you have 9 take 2 at a time 2 7 2 5 2 3 2 1. It will be optimal because you have split, if it is odd split odd even, it will work and if you split it this way you will always get the optimal solution.

(Refer Slide Time 32:32)



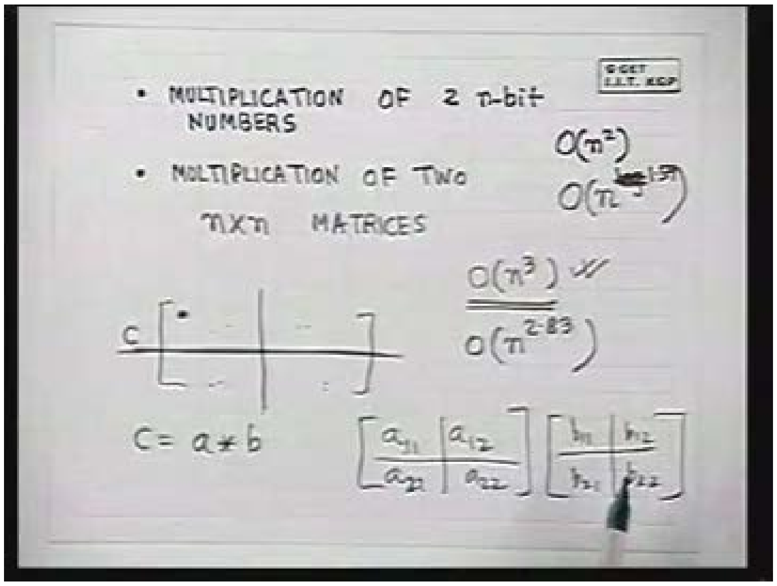
So once you decide on your split, so this is one problem which you can decide on your split and you will get the optimal solution in terms of number of comparisons and one such split is just take out 2 2 2 2. Now once you have decided this, the data structure design is left. So, now you can use it as an array or a linked list, it will not matter because you have to take two elements at a time. In constant time, you can split them. On the other hand if you split in the middle using a linked list, you would have problem. Suppose you chose middle you cannot split in the middle using a linked list, you would not get a constant time split.

So if you split in the middle you have to use an array. On the other hand if you split like this you can use a linked list or an array. So this way you can balance the split. Balancing has got some other, there are several interesting problems related to balancing and you can see in the book 1 or 2 of them, one is multiplication of 2 n bit numbers. Normally we could multiply 2 n bit numbers in order n square time, assuming that addition of 2 bits is constant time and addition of 2 bits and addition of 1 bit and 1 bit is constant time then addition of n bits and n bits is order n time, all right then you can do multiplication in order n square time.

You have a look at the book, any book on this one, specially how (Refer Slide Time: 34:34) you see that if you split it up in a very interesting way, you will get order of n to the power 1.59. How 1.59 comes is an interesting feature. It is related to how to balance this, secondly, in a very similar way multiplication of 2 n by n matrices. Normally we could do in how much time, how many individual multiplications you would require in addition? n square or n cube. Isn't it? n cube, n cube, our standard loop algorithm will take n cube multiplication, every row with every column, so to produce every element here we require n square, order n comparison multiplication. In c, c is equal to a into b, so for every element in c, we need to make n additions and multiplication. Isn't? a 1 1 into b

1 1 a 1 2 into b 2 1 a 1 3 into b 3 1 that way. So to produce every element, we require n such operations, so to produce n square elements we will require n cube operations.

(Refer Slide Time 37:04)



- MULTIPLICATION OF 2 n-bit NUMBERS

$O(n^2)$

$O(n^{1.59})$

- MULTIPLICATION OF TWO nxn MATRICES

$O(n^3)$

$O(n^{2.83})$

$C = a * b$

$$\left[\begin{array}{c|c} a_{11} & a_{12} \\ \hline a_{21} & a_{22} \end{array}\right] \left[\begin{array}{c|c} b_{11} & b_{12} \\ \hline b_{21} & b_{22} \end{array}\right]$$

Now if you split the problem and multiply it 4 by 4, split it into one fourth size. So you split a into a 1 1 a 1 2 into 4 parts into 4 sub matrices and similarly b and then do a recursive multiplication of them in a very intelligent way then you will be able to deduce this to something order of 2.83, these come from log values. So this algorithm is called the Strassen's algorithm, you must have heard of the name may be. These are all related to how to optimally balance the split. So, this is one aspect of algorithm design. Next day we will have a look at dynamic programming.