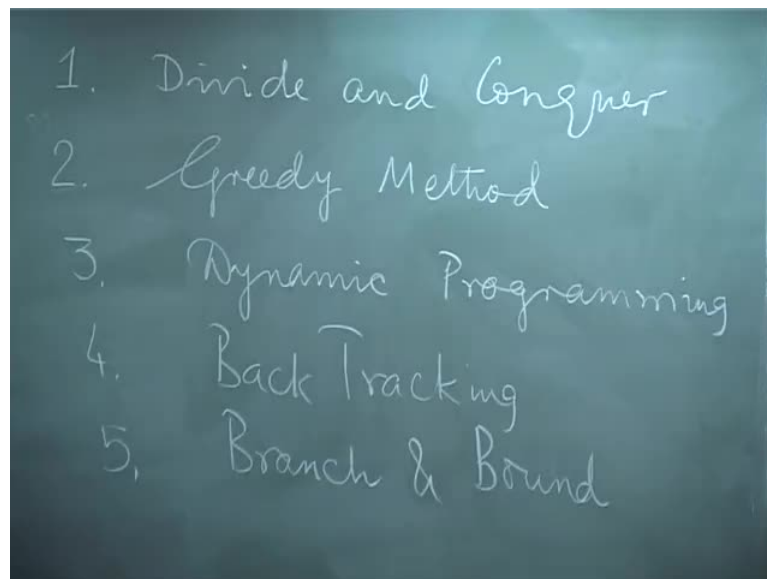


Parallel Algorithms
Prof. Phalguni Gupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 2

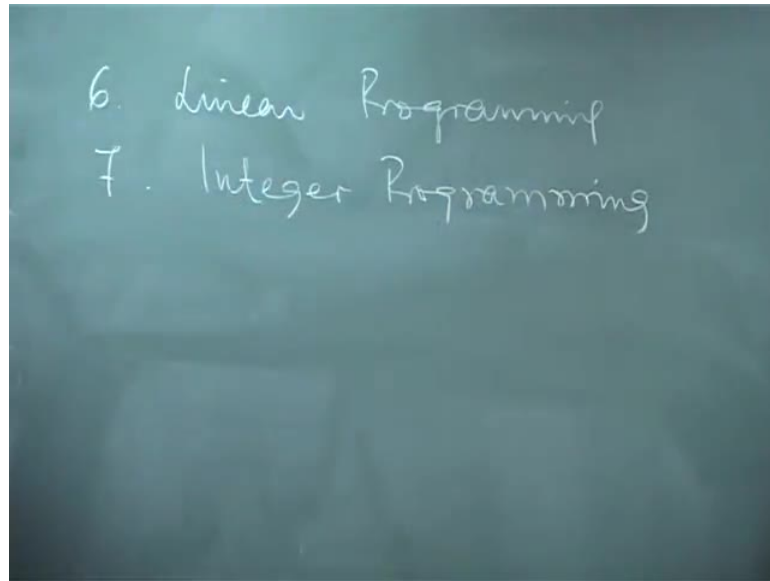
So in the last class, we discuss about that certain sequential data structure and also you have told, what is the algorithm and data structures. So, today I like to consider, the different paradigms on sequential or paradigms we use for sequential are going to consider. I assume that you have the base knowledge, basic knowledge on these paradigms however, for completeness saying I will be covering this, some of this paradigms. And I will be little fast because this is an why, you can consider it is an warm up or class, so that do not take any problem in understanding the parallel algorithms you will be discussing similar type of problems.

(Refer Slide Time: 01:22)



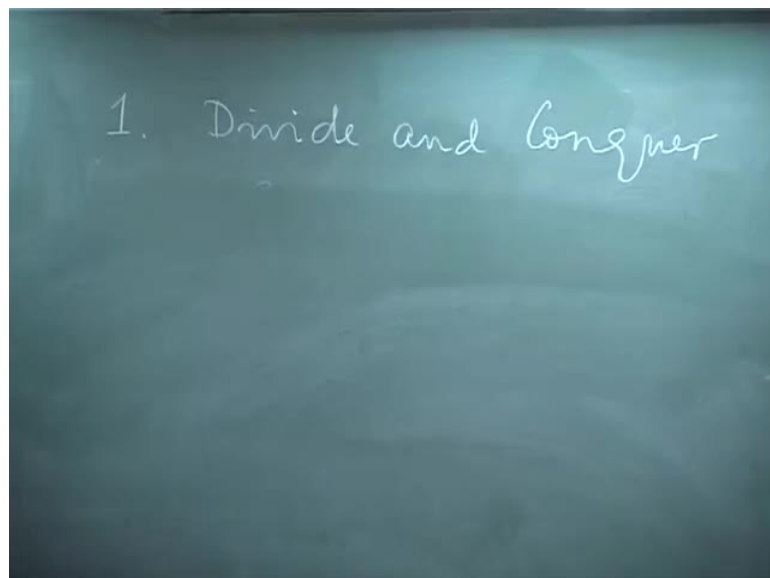
Now, the the different paradigms we use for sequential algorithms are one divide and, divide and conquer, this is one when notepad is, we use it. The next one is the greedy method, third one is dynamic programming, fourth one is back tracking fifth one is branch and bound.

(Refer Slide Time: 02:32)



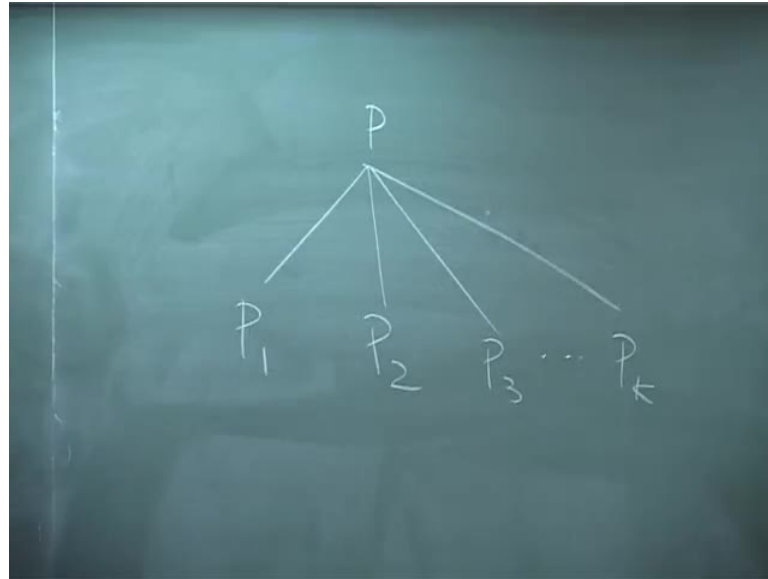
Then, we have linear programming, integer programming and so on, we try to discuss in detail at least the these three paradigms right. And if we can also then discuss about this back, back tracking and, branch and bound paradigms, if time permits.

(Refer Slide Time: 03:22)



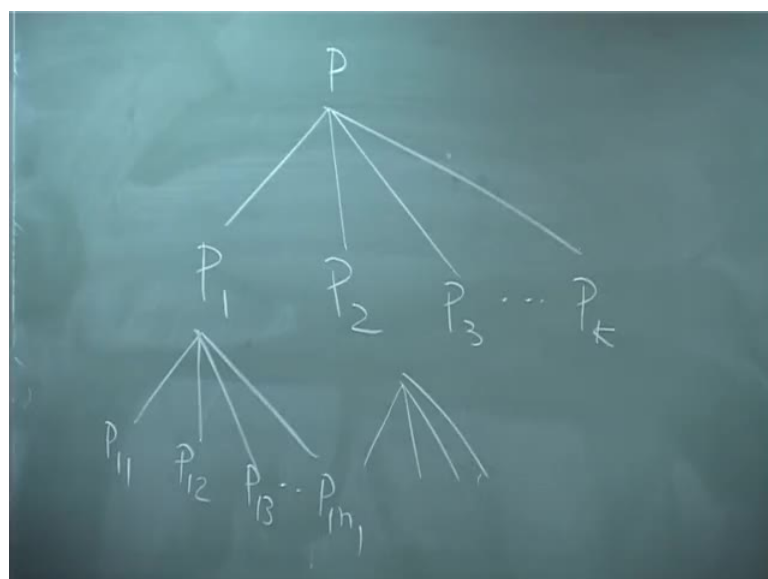
So, now let us consider our first paradigm divide and conquer, what happens here actually suppose, here a problem P and that, you want to solve this problem P .

(Refer Slide Time: 03:36)



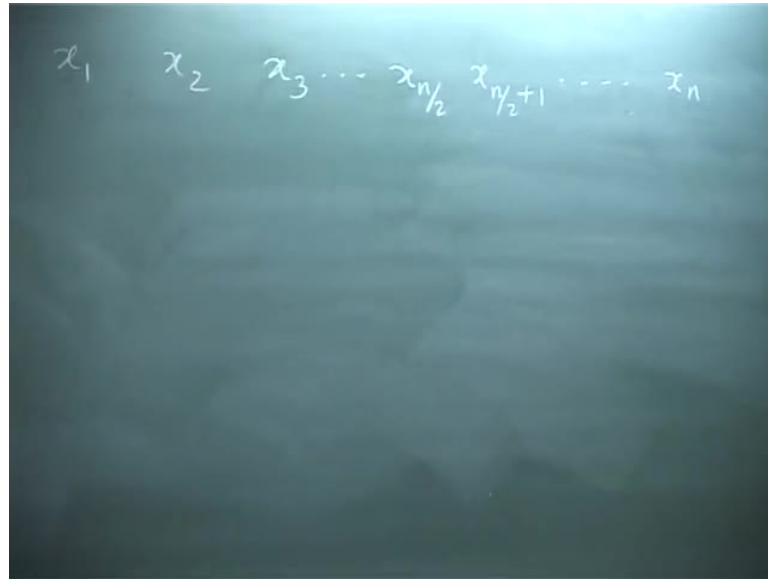
What we will do in divide and conquer, you divide this P, problem P into several sub problems P 1, P 2, P 3, P k. There are k sub problems, you can divide this P problem into several sub problems and these sub problems are having the similar properties, as we have in the case of P. Now, I will solve this problem P 1, problem P 2 and problem P 3 and problem P k, and then I combine the results to get the solution of P, is it ok. Then, what we do, that we have the k sub problems, these sub problems, each sub problem is parallel in nature with reference to P. Now, we try to solve this P 1, P 2, P k and if I can solve this, then you combine result to get the solution of P.

(Refer Slide Time: 04:47)



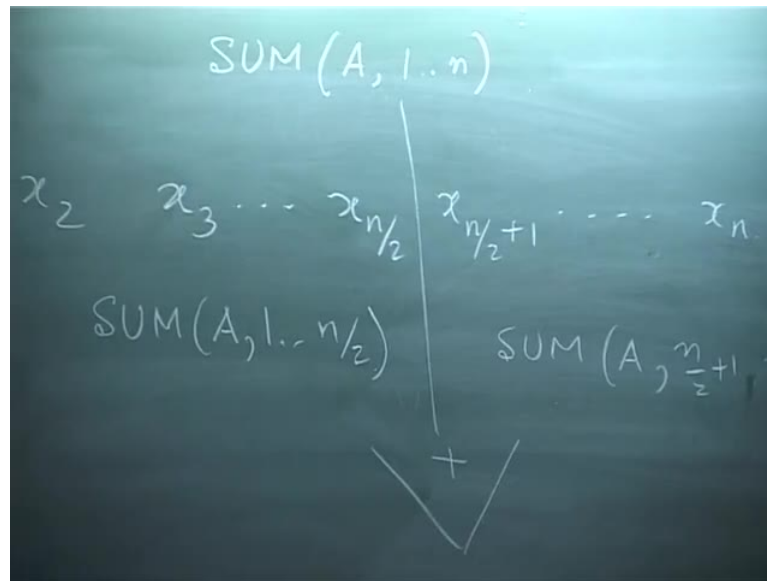
Now, it may so happen that, P 1, P 2 and P k they are also large in size, the problem is not that simple problem. Then, you divide these sub problems into further sub sub problems, where we can have P 1, P 1 1, P 1 2, P 1 3, P 1 suppose, n 1 similarly, the case with P 2 and so on. Now, you solve this problems P 1 1, P 1 2, P 1 3, P 1 n and combine to get the solution of P 1 and so on.

(Refer Slide Time: 05:35)



Now, say for example, we have to find out the sum of n numbers now, by observation you can tell, the finding the sum of n numbers is a sequential process. Now, look at the problem that I have x_1, x_2, x_n this n elements I have, these are the n elements x_1, x_2, x_n .

(Refer Slide Time: 06:02)



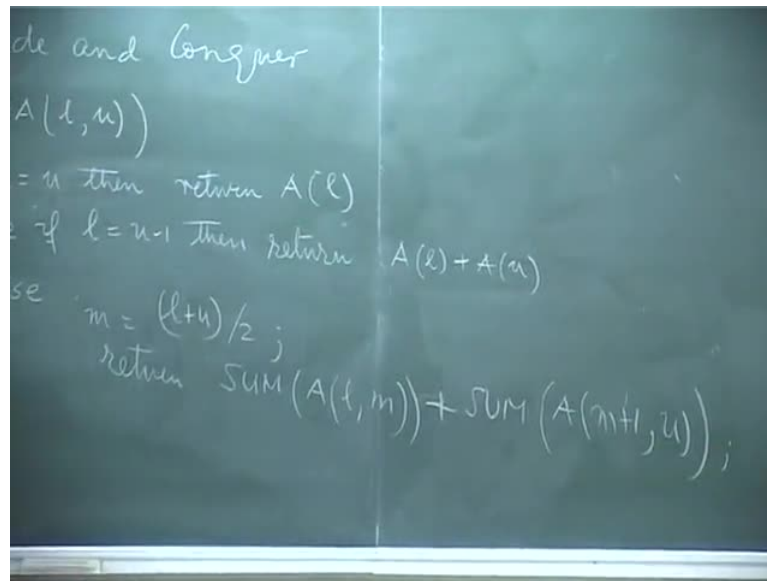
And I want to find out the sum of n numbers, this n numbers so, one we, I can think that I divide this into two parts right say, I want to find out the sum of n numbers. Now, that instead of, finding the sum of n numbers, what I do here now, I will find sum of 1 to n by 2, and this side I find sum of remaining n by 2 elements. Now, I got the result of this sum of this sum of this session and I combined this two, to get the sum of n numbers.

(Refer Slide Time: 06:48)

1. Divide and Conquer
 $SUM(A(l, u))$

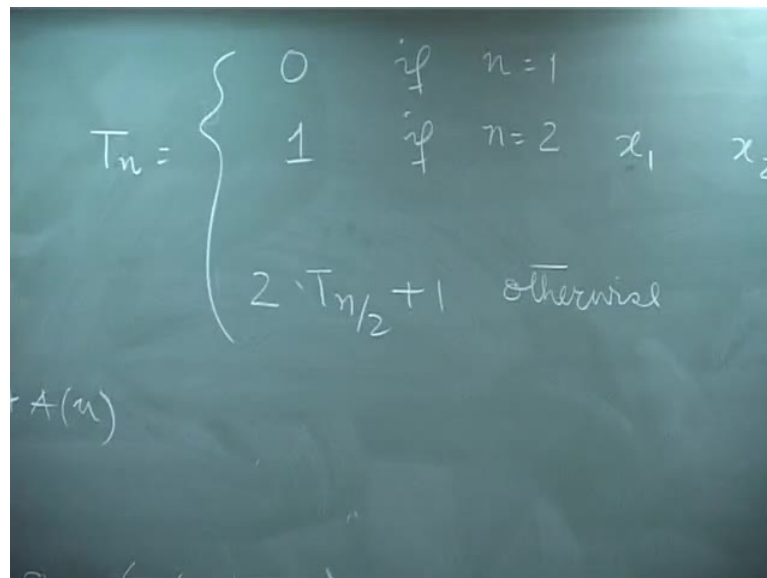
So, if I have to write this in the form of algorithms, I can write sum A say, I want to find out the sum of l to u .

(Refer Slide Time: 06:58)



If l equals to u then, return $A(l)$ else if, l equals to u minus 1 then, return $A(l)$ plus $A(u)$ else, return sum $A(l)$. Return else m equals to l plus u by 2 and then, return sum $A(l)$ m plus sum $A(m + 1)$ u . So, basically you are computing this l plus u and then, you return this sum and then, plus this one.

(Refer Slide Time: 08:58)



So, if you have to compute the time complex in for that then, I have T_n is equal to Zero if n is equal to 1, 1 if n equal to 2 otherwise, $2 T_n$ by 2 plus 1, otherwise.

(Refer Slide Time: 09:39)

$$\begin{aligned}T_n &= 2 T_{n/2} + 1 \\&= 2 \left\{ 2 T_{n/4} + 1 \right\} + 1 \\&= 2^2 T_{n/4} + 2 + 1 \\&= 2^2 \left\{ 2 T_{n/8} + 1 \right\} + 2 + 1 \\&= 2^3 T_{n/8} + 2^2 + 2 + 1\end{aligned}$$

So, you have to solve this when this one is coming under one condition and this is size is half, size is $2 T_n$ by 2. So, T_n becomes $2 T_n$ by 2 plus 1, which I can write 2 times $2 T_n$ by 4 plus 1 plus 1. So, it gives you basically, 2 square T_n by 4 plus 2 plus 1, is it correct? Next time I write, 2 square T_n by 8 2 times plus 1 plus 2 plus 1 so, I get put cube T_n by 8 plus 2 square plus 2 plus 1.

(Refer Slide Time: 10:38)

$$\begin{aligned}&= 2 \left\{ 2 T_{n/4} + 1 \right\} + 1 \\&= 2^2 T_{n/4} + 2 + 1 \\&= 2^2 \left\{ 2 T_{n/8} + 1 \right\} + 2 + 1 \\&= 2^3 T_{n/8} + 2^2 + 2 + 1 \\&\vdots \\&= 2^{k-1} T_2 + 2^{k-2} + \dots + 1\end{aligned}$$

otherwise
 $n = 2^k$

So, if n equal to 2 to the power of k , I get here 2 to the power of k minus 1 T_2 plus 2 to the power of k minus 2.

(Refer Slide Time: 11:06)

Handwritten mathematical derivation on a chalkboard:

$$\begin{aligned}
 &= 2 \left\{ 2 T_{n/4} + 1 \right\} + 1 \\
 &= 2^2 T_{n/4} + 2 + 1 \\
 &= 2^2 \left\{ 2 T_{n/8} + 1 \right\} + 2 + 1 \\
 &= 2^3 T_{n/8} + 2^2 + 2 + 1 \\
 &\vdots \\
 &= 2^{k-1} T_2 + 2^{k-2} + \dots + 1 \\
 &= 2^{k-1} + 2^{k-2} + \dots + 1
 \end{aligned}$$

Additional notes on the left side of the board: $n = 2^k$ and "otherwise".

Now, this is $T_2 = 1$ so, here 2 to the power of k minus 1 so, here 2 to the power of k minus 1 plus 2 to the power of k minus 2 .

(Refer Slide Time: 11:15)

Handwritten mathematical derivation on a chalkboard:

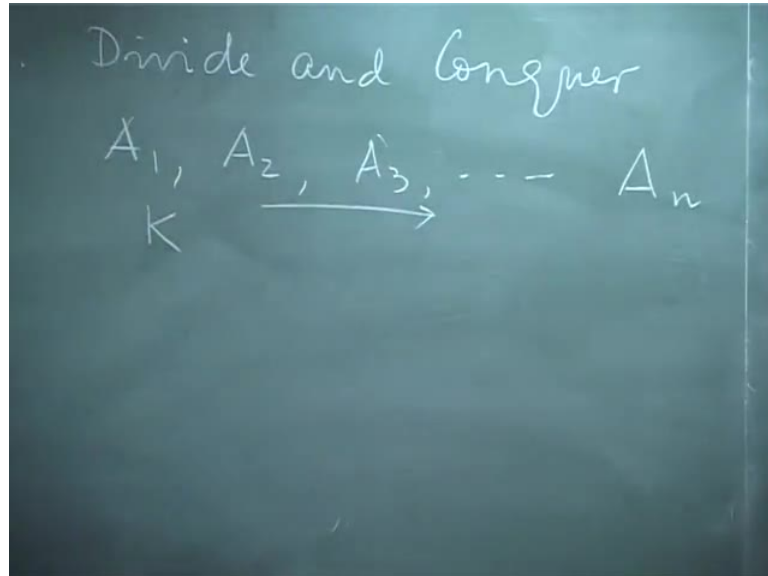
$$\begin{aligned}
 &= 2 \left\{ 2 T_{n/4} + 1 \right\} + 1 \\
 &= 2^2 T_{n/4} + 2 + 1 \\
 &= 2^2 \left\{ 2 T_{n/8} + 1 \right\} + 2 + 1 \\
 &= 2^3 T_{n/8} + 2^2 + 2 + 1 \\
 &\vdots \\
 &= 2^{k-1} T_2 + 2^{k-2} + \dots + 1 \\
 &= 2^{k-1} + 2^{k-2} + \dots + 1
 \end{aligned}$$

Additional notes on the left side of the board: $T_{n/2} + 1$ otherwise, $n = 2^k$, and the formula $\frac{2^k - 1}{2 - 1} = n - 1$.

So, this keeps you 2 to the power k minus 1 , 2 minus 1 , which nothing but, n minus 1 because, 2 to the power of k is n so, you get n minus 1 . And which is the, reality is also true that, to find the sum of n numbers, you need n minus 1 an each other. So, this is our first simple problem now, you know that binary search let us think about the binary

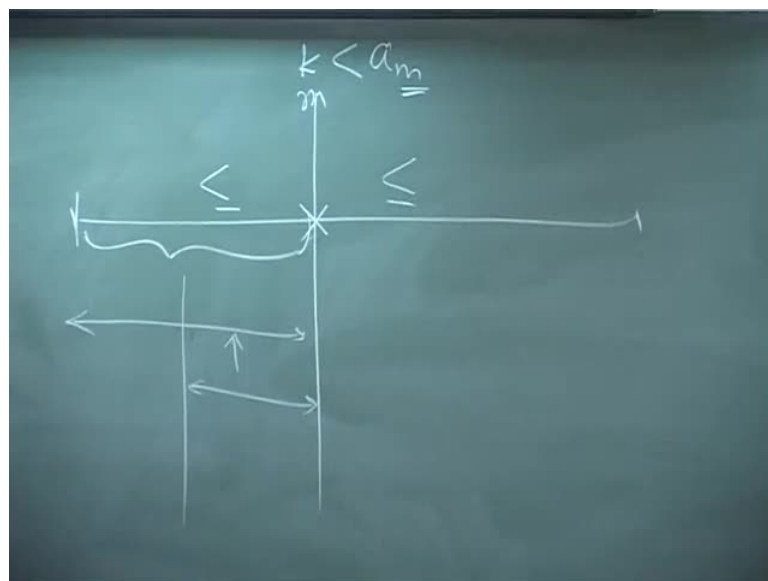
search, what happens. In the case of the binary search we assume that the elements are either (()).

(Refer Slide Time: 12:06)



So, suppose you have n elements, you have $(()) A_1, A_2, A_n$ and n argument k you have. Your interest is to find whether the element k , exists in this sequence of an elements or not now, it is assume the elements are in increasing order.

(Refer Slide Time: 12:33)

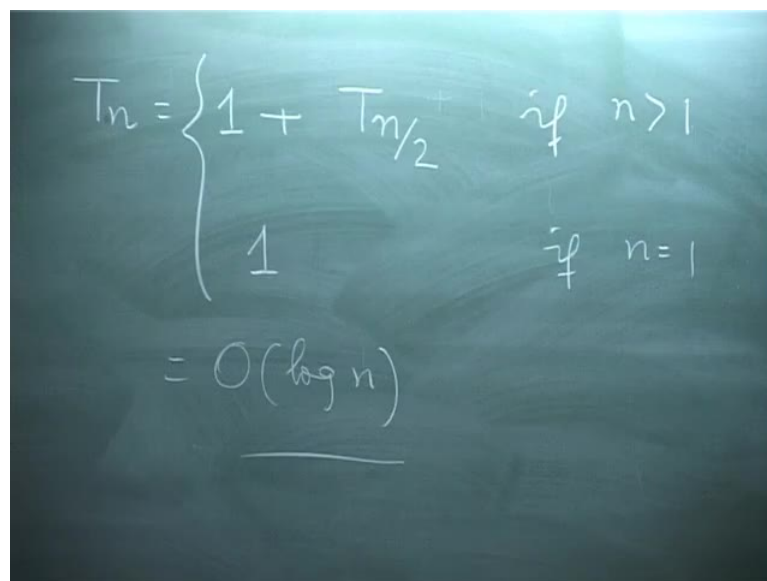


So, in the case of binary search, what we do, we divide these n elements into, you compare the k with the middle elements. It has the property that is, it is in increasing

order then, these elements is less than equal to this, less than equal to this and so on. So, we will be comparing the middle element with the k now, we should find that, k is for find that, k is less than a m where, a m is the middle element of the sequence.

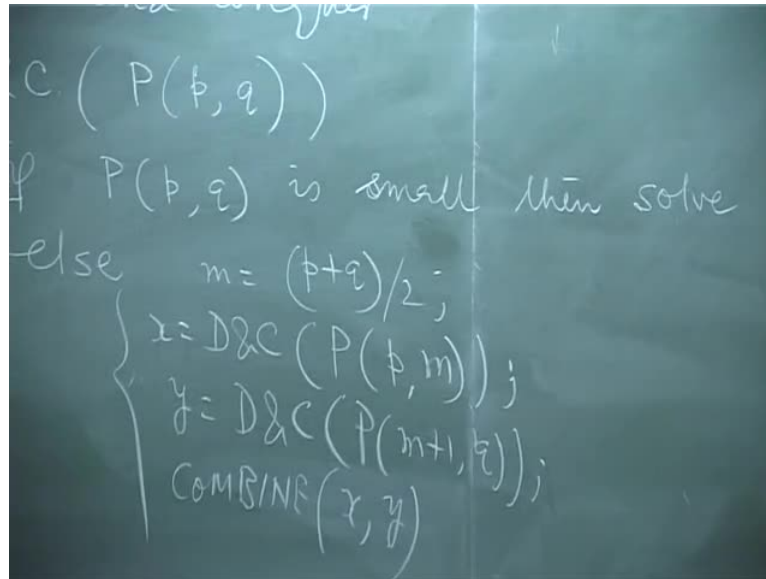
Then, k if k exists at all then, it will lie here so, in that case in that case, my searching zone will be reduce to this part and you can easily discard all the elements of (()) so, my searching area. Now, the searching zone is reduce to this and again, I take the middle element and I compare with that k, is less than equal to, all greater than, if it is equal to m or the middle element, you tell the element exist and you got the below m. Now, if find that k is greater than this side, the k lies in this zone. So, my searching zone will be reduce to this and so on, till you possess this one, till I find that, there exists only one element. And we compare and you have compare and found either exists equal to or not equal to.

(Refer Slide Time: 14:25)


$$T_n = \begin{cases} 1 + T_{n/2} & \text{if } n > 1 \\ 1 & \text{if } n = 1 \end{cases}$$
$$= O(\log n)$$

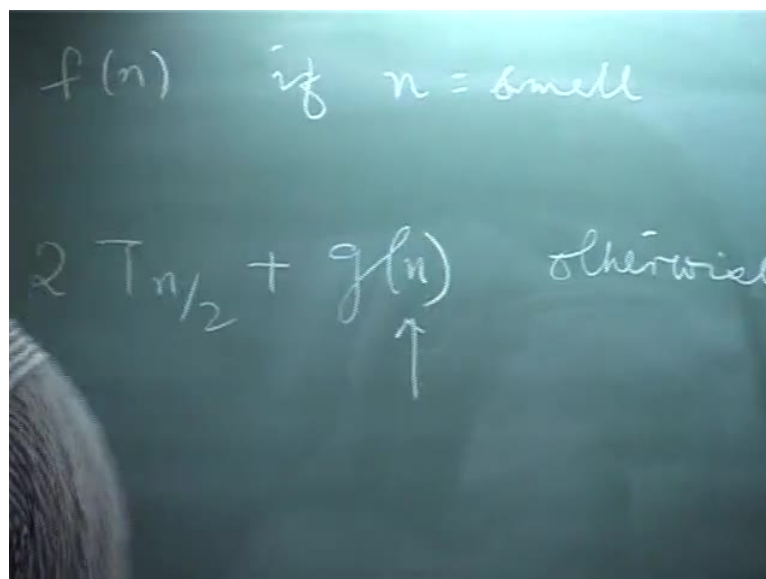
Now, in that case, if I have to estimate the time complexity for that, there it is comes T n is, you will comparing 1 and the size increases to half, if n is greater than 1. And if n equals to 1 then, only one component will be required. Now, the solution of this becomes order log n now, if I have to do or I have to write the generic algorithm for divide and conquer, and with the addition, we will be dividing the problem into the two sub problems.

(Refer Slide Time: 15:08)



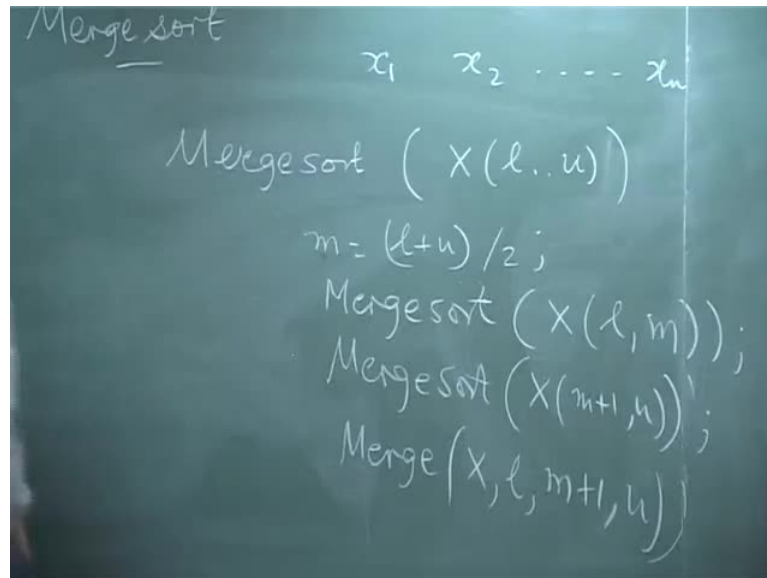
So, I write D and C, the problem P and size is small p and q so, this the problem from the size of p and q. Now, if problem p q is small then, solve else, m equal to say, p plus q divided by 2 and then, you call divide and conquer P p m x equals to, y equals to divide and conquer P m plus one, q. And then, combine the result combine the result x with y that is the required so, this is the generic algorithm for divide and conquer.

(Refer Slide Time: 16:45)



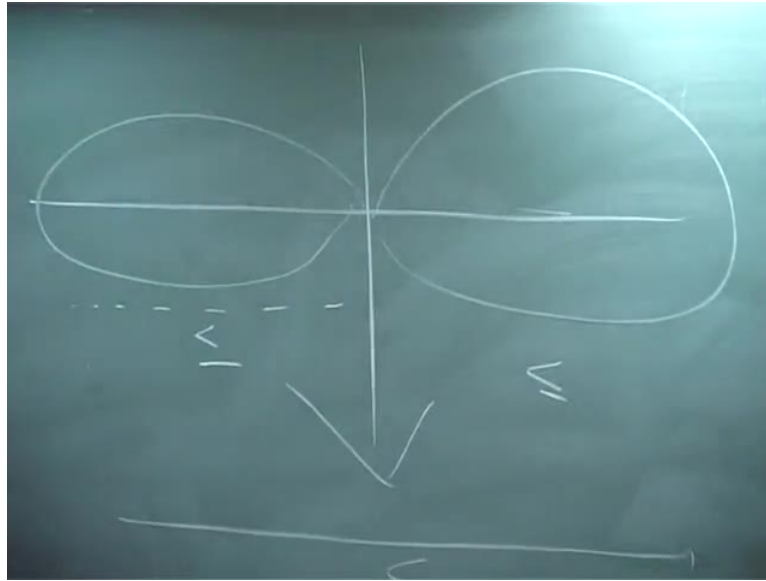
If I have to see the time complexity T_n is equal to say, $f(n)$ if n is small else, this $2T_n$ by 2 plus, some another function $g(n)$, otherwise right. So, this $g(n)$ is for the time you need to combined this together.

(Refer Slide Time: 17:10)



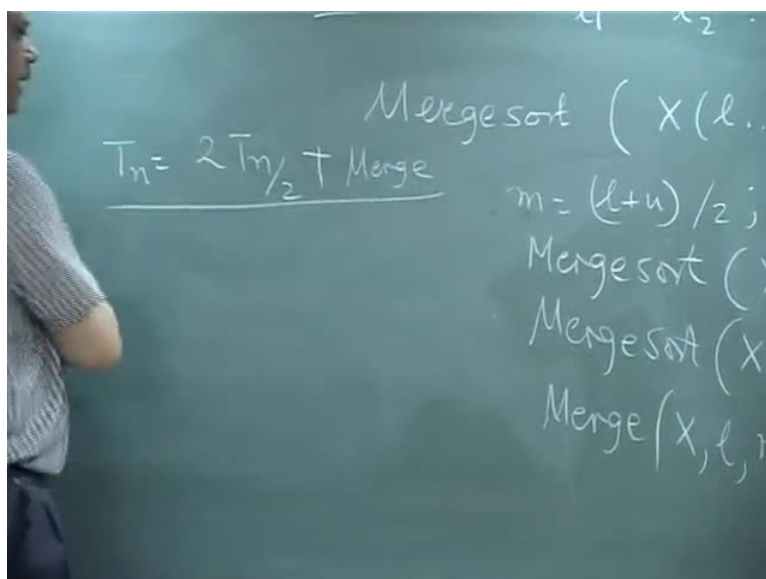
This is an example of divided and conquer paradigms suppose, you have x_1, x_2, \dots, x_n or that n elements and they are not in sorted order, you want to arrange it in ascending order or descending order. Let us assume without the laws are generated, you want to arrange it in ascending order. So, this merge sort is a technique to arrange this n elements either in ascending order or descending order. So, algorithm becomes very simple, if I write merge sort and you have $x_1 \dots x_u$ then, you can write m equals to l plus u divided by 2. Then, you write merge sort $x_1 \dots x_m$ merge sort $x_{m+1} \dots x_u$ then, you have merge x_1, m plus 1, u .

(Refer Slide Time: 19:00)



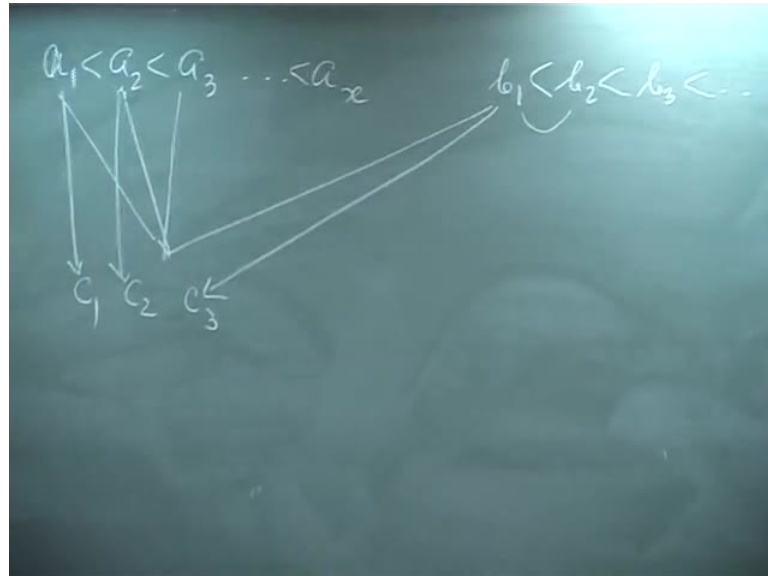
What it means, that basically you have n elements, you have n elements and you are dividing into two parts, you are recursively you bring the merge of this, recursively merge of this. And then, this is in increasing order, this is also increasing order, you are that combining this two results, to get the result is in the increasing order. So, m is the middle of this, you are diving this n elements into the two equal part, you are making the merge of algorithm recursively you are calling this part, merge of algorithm calling this side. Then, you are merging this two to get the results, we will be discussed how to merge this one.

(Refer Slide Time: 19:50)



If it is the case then, my time complexity becomes T_n equal to 2 times $T_{n/2}$ plus the time you need, time you need time you need to merge.

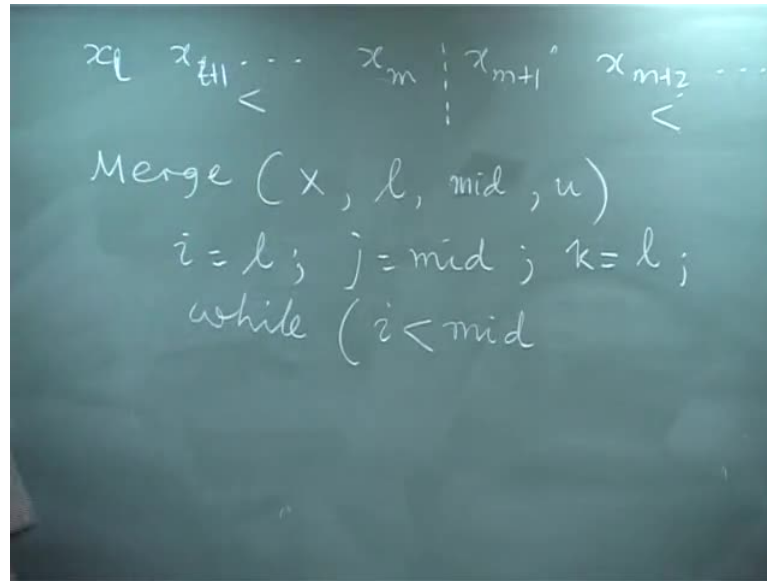
(Refer Slide Time: 20:06)



So, let us think about how to merge, you have a_1, a_2, a_3, a_x and you have b_1, b_2, b_3, b_y . Suppose, you want to merge this two, the sequence with the understanding that, this is satisfying this criteria, this satisfy this criteria and we want to merge these two sorted sequence. So, what I do, I compared a_1 with b_1 , if you find a_1 is smaller than b_1 , then a_1 is the smallest element, which is I write c_1 .

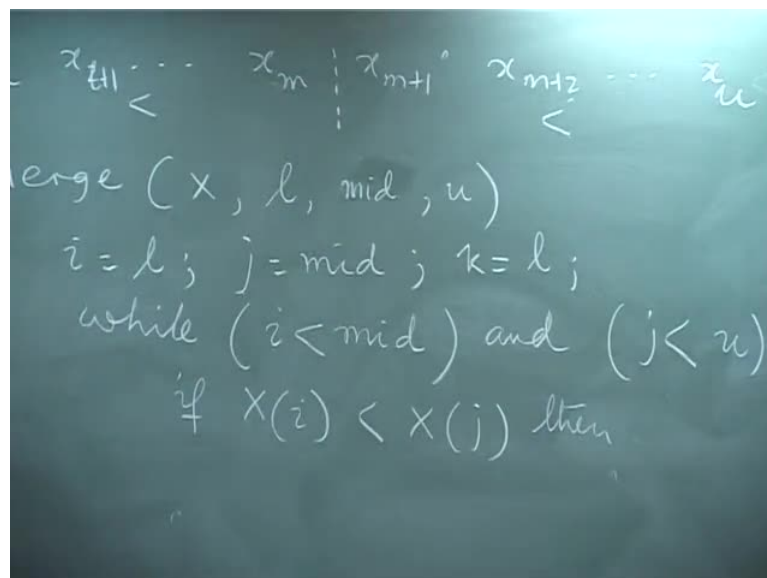
Now, once you have selected a_1 that pointer moves to this area and I know to compare a_2 with b_1 and if you find still a_2 is smaller than b_1 then, you write in the position of c_1 and the pointer moves to a_3 . Now, a_3 is compared with b_1 and if you find that b_1 is smaller than smaller than a_3 then, b_1 is come here and pointer moves to this place. And proceed till till one of the array, already one of the sequence is, already task for to c sequence and that in any elements c_2 is elements is to c to get the final merge sequence.

(Refer Slide Time: 21:52)



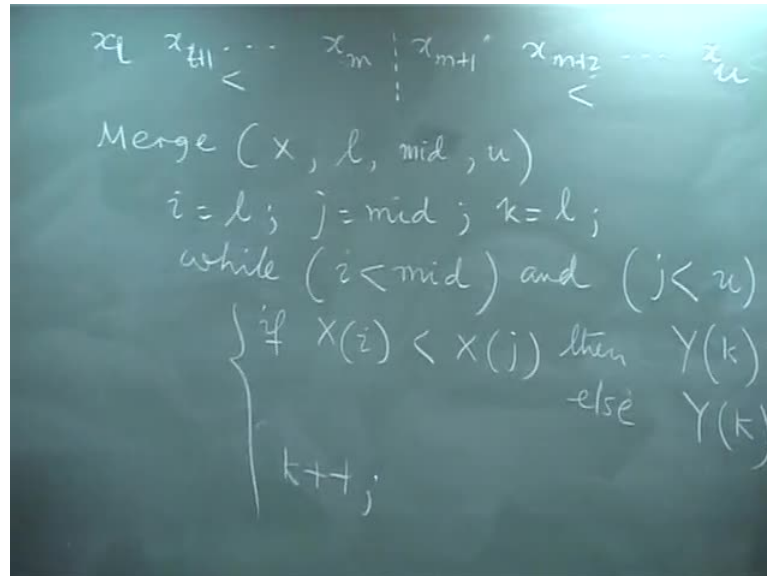
Now, here I have, basically I have one sequence i have one sequence x_1, x_2, x_1, x_1 plus $1, x_m$ then, we have x_{m+1}, x_{m+2} you have, x_u , this is satisfying this property, this is increasing order and this is also increasing order. Now, move to merge sort so, I can write merge $x_1, m+1, u$ or be mid and then, you I write, i equal to l , j equal to mid . And k is the pointer for merge sequence, which is l which is l so, I will be moving this pointer i j and j i and j . And we will see as long as as long as whole sequence is considered, we have to, is not yet considered, we have to do these operations.

(Refer Slide Time: 23:48)



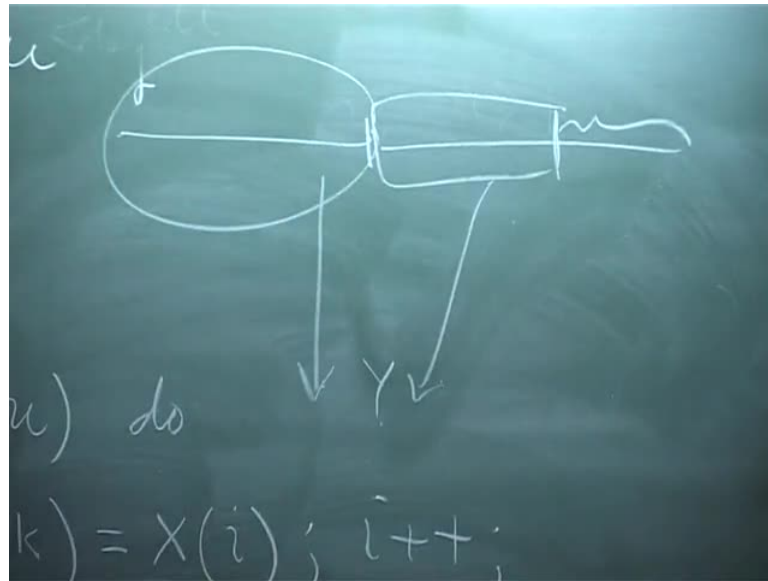
So, while i is less than mid and j is less than u , you have to do the following that is, both the sequences as are not existence. If x of i is found less than x of j then, this sequence contains the smaller element.

(Refer Slide Time: 24:37)



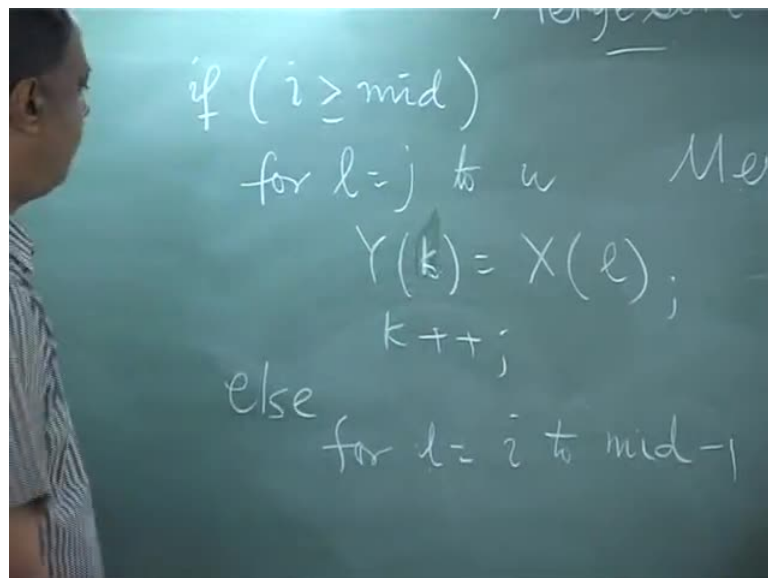
And that has to be moved to sequence say, y of k is your x of i and i is increased by 1. Otherwise is this greater than this. So, will be writing if y of k is your x of j and j is incremented by 1 and in both the case, your k will be moved incremented by 1. This is the things you have to do that, if you find that x of i less than x of j then y of k is updated by x of a x of i and i is increased by 1. Otherwise, k is updated by x of j and j is incremented by 1 anyway, that you have to update the y of k to get the next position of y .

(Refer Slide Time: 25:37)



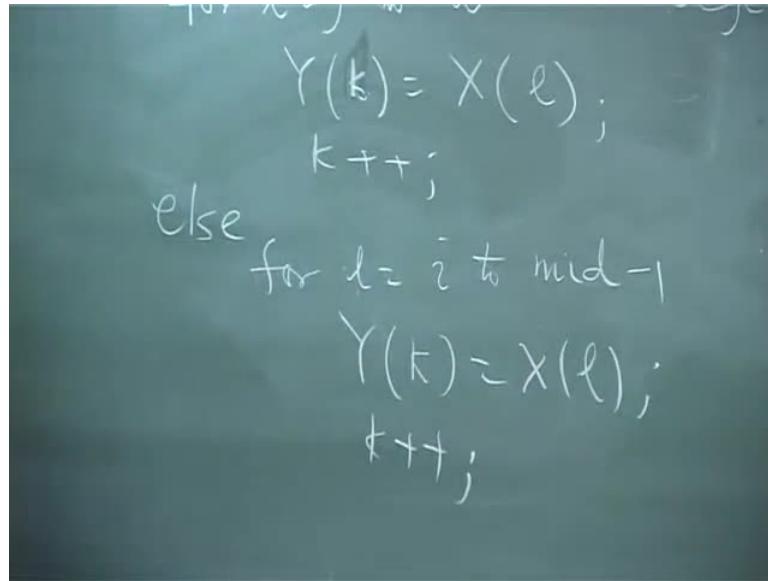
Now, in that process what it may so happen that, one subsequence is already enter into y sequence and some parts of other sequence enter into y sequence. The remaining part is, a 2 shifted to y so, that has to be taken into account.

(Refer Slide Time: 26:03)



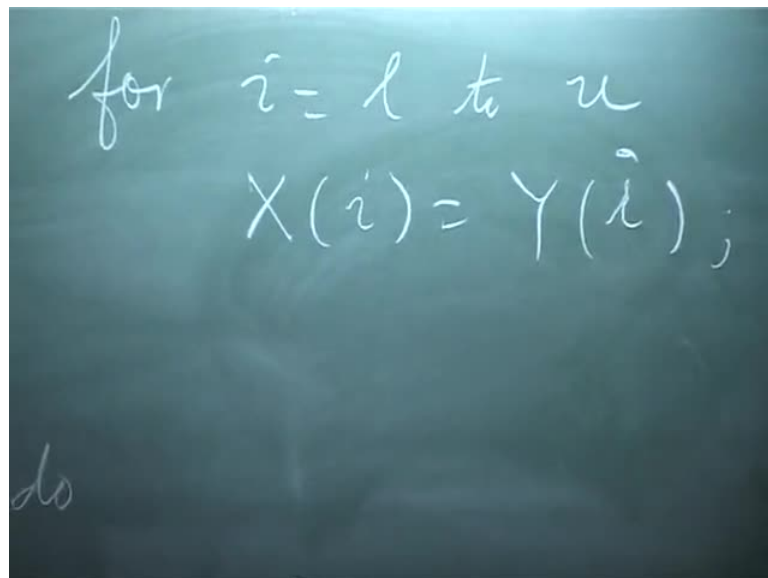
So, this can be done like that, if you find that, i is greater than equal to mid then, there exist some elements of the second sub sequence to be shifted. That is, for l equal to j to u , y of l is your x y of k equal to x of l at k is increased by 1 else, for l equal to i to $mid - 1$ to mid minus one equals i to mid .

(Refer Slide Time: 27:13)



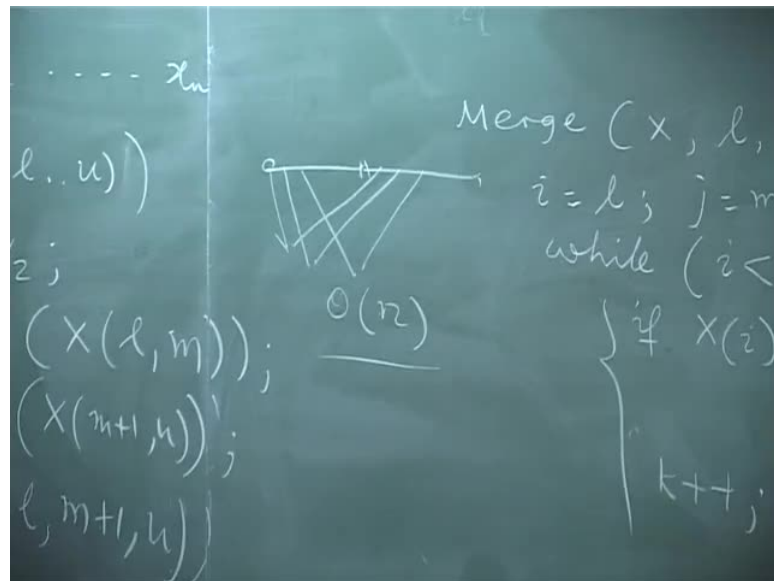
And then, y of k is your x of l and k is increased by 1 so, basically your merging of x sequence, x subsequence and this sub sequence and merge sequence you put it y.

(Refer Slide Time: 27:47)



But, in order to, get the result back to x, you have write a small statement that is, for i equals to one for l to u i equal to l to u, you have x of i equals to y of i so, then result will be stored by to x. When you (()) up to find out the complexity of this algorithm then, you observed that it takes basically, here worst case scenario, what will be the worst case scenario.

(Refer Slide Time: 28:25)



Worst case scenario could be that, one element you have this subsequence and this subsequence. So, you are taking one element here then, next element then next element, next element, next element, next element and so on. So, that my comparisons, this will be compared every time and which, will be taking order l minus u basically, order n , u minus l that basically, order n . And this is also we will take, otherwise also, better moment time also basically will find order n . So, merge routine will take order n time merge routine will take order n time.

(Refer Slide Time: 29:17)

$$\begin{aligned}
 T_n &= 2 T_{n/2} + O(n) \\
 &= 2 \left\{ 2 T_{n/4} + c n \right\} \\
 &= 2^2 T_{n/4} + c n + c n \\
 &= 2^2 \left\{ 2 T_{n/8} + c n/4 \right\} + c n + c n
 \end{aligned}$$

So, in that case, merge sort takes the following time. If T_n is the time complexity to solve then, you have $T_{n/2}$ plus order n , this is for merging. What is the complexity now? It's having 2 times 2 times $T_{n/4}$ so, let me write the T_n is $O(n)$ is $c n$, can write $c n$ here. And here, I write $c n$ by 2 so, this gives you 2 square $T_{n/4}$ plus $c n$ plus $c n$, which gives me 2 square 2 times $T_{n/8}$ plus $c n$ by 4 plus $c n$ plus $c n$.

(Refer Slide Time: 30:23)

$$\begin{aligned}
 &= 2 \left\{ 2 T_{n/4} + c n \right\} + c n \\
 &= 2^2 T_{n/4} + c n + c n \\
 &= 2^2 \left\{ 2 T_{n/8} + c n/4 \right\} + c n + c n \\
 &= 2^3 T_{n/8} + 3 c n
 \end{aligned}$$

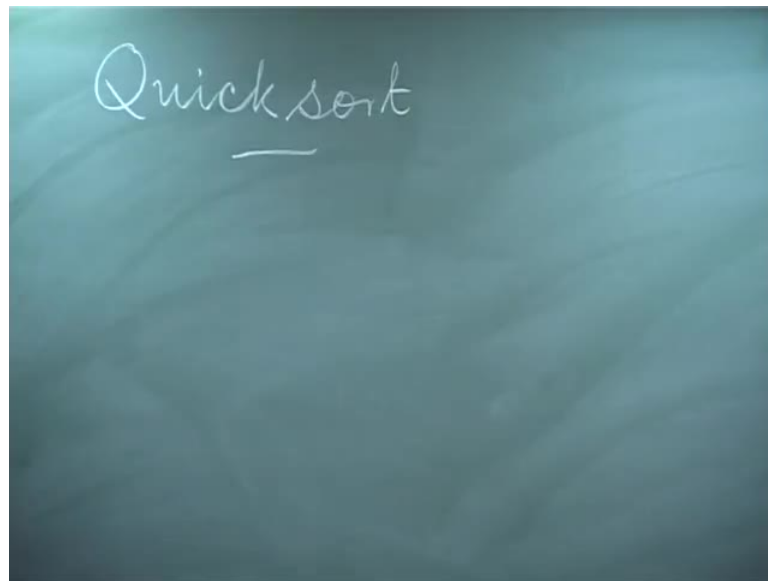
So, this gives me 2 cube $T_{n/8}$ plus 3 times $c n$.

(Refer Slide Time: 30:36)

$$\begin{aligned}
 n &= 2^k & T_1 &= 0 \\
 & & T_2 &= 1 \\
 T_n &= 2^k T_1 + k c n \\
 &= c n \log n = O(n \log n)
 \end{aligned}$$

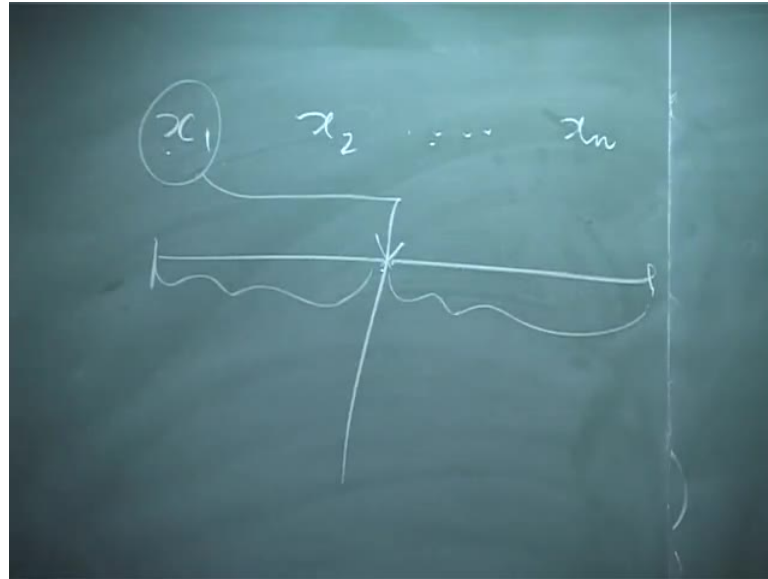
Now, if n equals 2 to the power k and T_1 is 0 , one element nothing to be sorted and T_2 is T_2 is 1 because, only one comparison we require so T_2 is 1 . In that case, T_n becomes 2 to the power k T_1 plus k times $c n$, this is zero so, you get $c n \log n$ times, this is called as order $n \log n$. So, this is matching matches, this matches in the lower bound or lower bound of sorting algorithms, this takes what all, $\log n$ time complexity to solve n element using merge sort technique.

(Refer Slide Time: 32:00)



Now, another example for divide and conquer is quick sort so, you observe that in the case of merge sort, what do you did, we divided the sequence into two equal parts. And the first one we call merge of the first part and the merge of the second part of algorithm then, you merges.

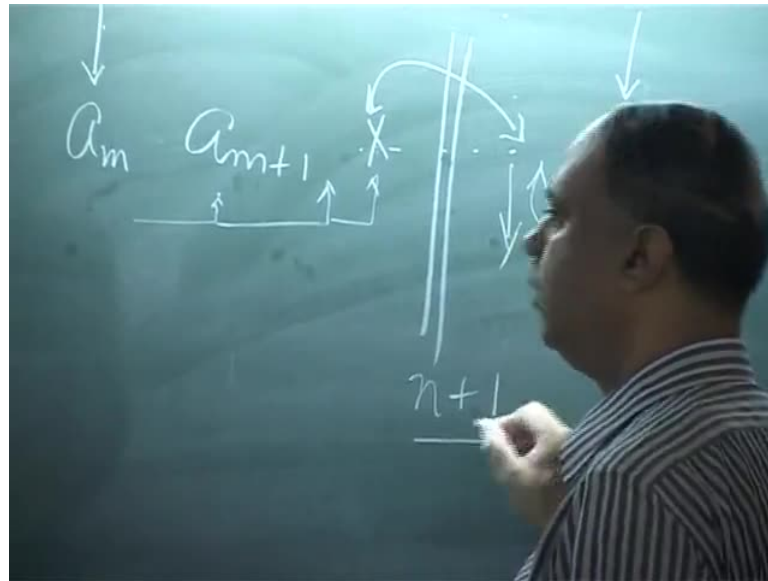
(Refer Slide Time: 32:27)



Now, in the case of quick sort, it is little difference suppose, you have x_1, x_2, x_n now, what we do that we take the first element or one of them, random you can select. But, for simplicity, you select the first element as the partition element. And what it does? This element is huge, to find its position with reference to x-axis; so if we divide or we get the exact position of x_1 , exact position of x_1 , if I sort, if this sequence is sorted, by some, by some technique, we partition this sequence into the two parts in such way that, x_1 gets its position in the sequence, satisfying the property of sorting, agreed. So, I get the position of x_1 here in such a way that, all these elements here are smaller than these elements and all these elements here are larger than these elements.

That means what, we select one element say first element which we termed as a partition element and this element gets not only gets position in the sorting sequence, also all the elements to where of this side or of this side are larger than this elements and none of them is smaller than this elements. Then, you require some equal, quick sort of this part and quick sort of this part finally, you have one element because, know how to partition say, I have a sequence partition say, partition A_m and u .

(Refer Slide Time: 34:31)

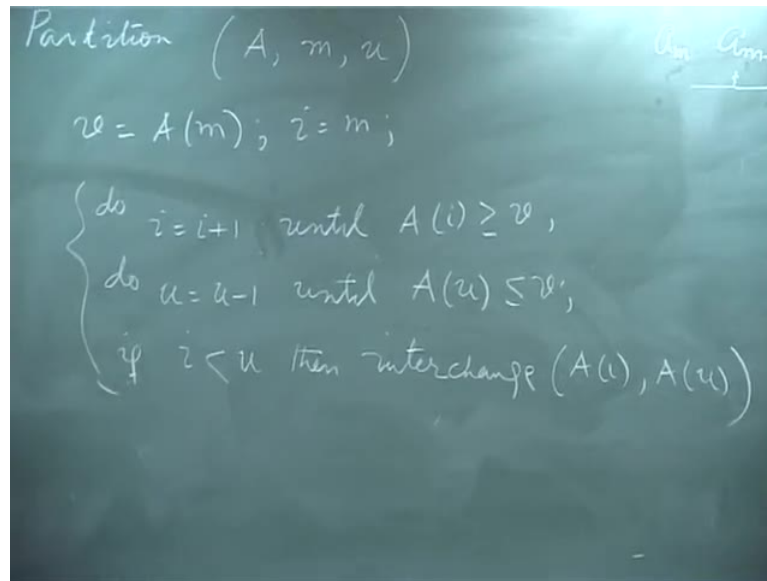


What it means that, you have a m a $m+1$ a u , I put a pointer here and I put pointer here. Now, a_m is compared a_m is compared with this first element, if you find that this is smaller than or equal, you retain as it is. You move the pointer here now, this element is compared with this, if you find that this element smaller than a_m , you go to the next pointer, next element.

You find this element is smaller than this element, you go to the next one now, if you find this element is larger than this element then, you stop here now, you starts searching from this side. Now, if you find it this element is larger than a_m , you go to the next one you find that this element is larger than a_m , go to the next one and so on. Finally, to get into a position where, this element will be, you make at a position where, this element is smaller, then this one.

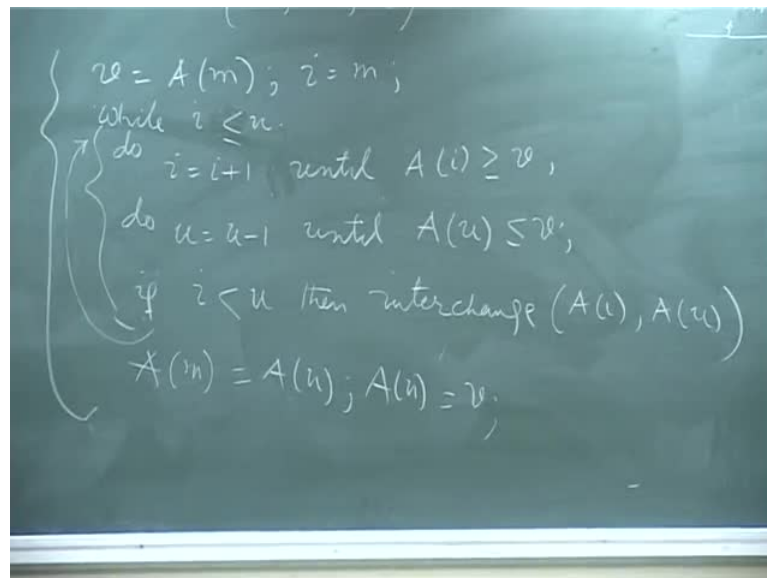
So, you will touch this two you will touch these two and then, you proceed again so, finally, you will get a position here where, it is crossed over. So, you will touch this element to this one and so, you will find that a m (()) position. And all this elements, the small of these elements, all this elements should be smaller than larger than this element. Now, since there if I consider the size is n then, I need $n+1$ comparisons so, this is the idea of partitioning element then, the idea of partition algorithm, which takes $n+1$ you need of time.

(Refer Slide Time: 36:34)



If I have to write the algorithm then, I can write this way say, v equals to A of m and I equal to m then, do i is increase by one until A of i is greater than equal to v, do u is decrease by 1 until A of u is less than equal to v. If i is less than u then interchange interchange a of i be a of u.

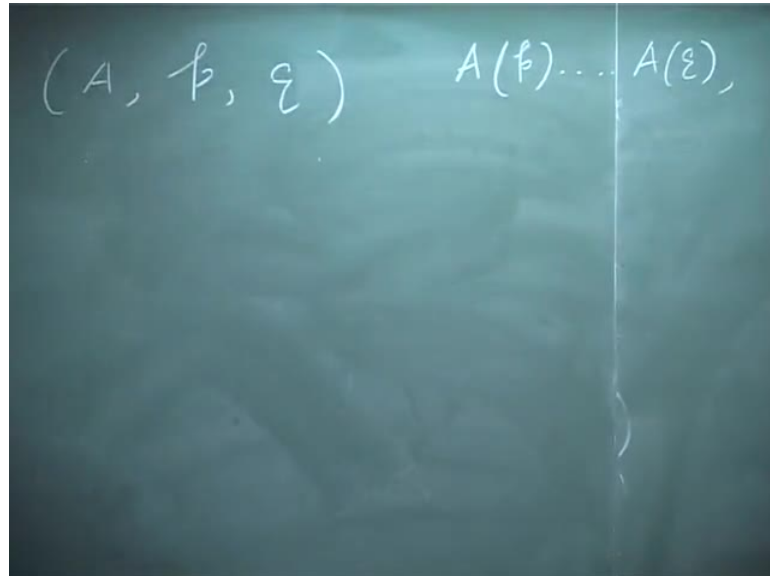
(Refer Slide Time: 38:00)



After doing this, you repeat till there is a cross over that is, i is while while i greater than i less than n, less than equal to n, you do all these things. And then, you have A of m is equals to A of u and A of u is equal to your v. So, this is your partitioning elements,

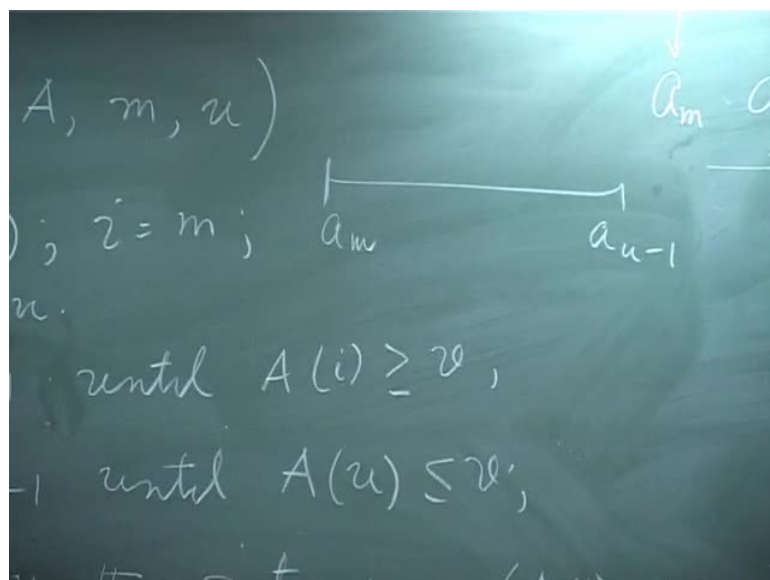
partitioning algorithms and obviously than this takes order n plus 1 unit time to do the to get the partitioning position to get the position of x , it is sub sequence of n element.

(Refer Slide Time: 39:05)



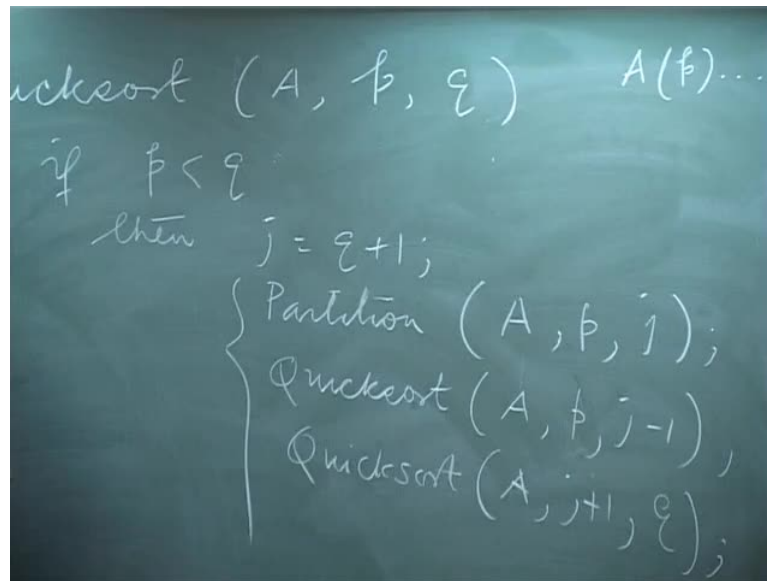
Now, if you have to write the quick sort algorithm so, you have quick sort A p q that is, we looking part to sort A of p to A of q .

(Refer Slide Time: 39:30)



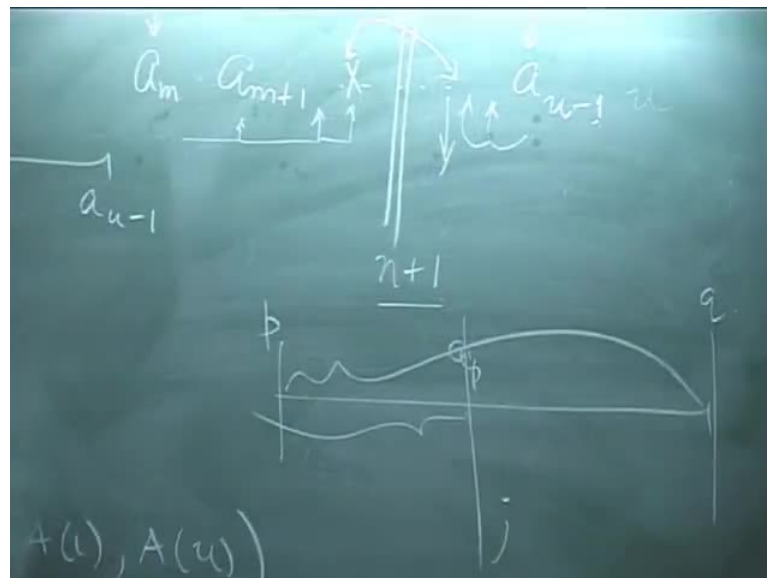
Here, one to remember, I forgot to mention that that instead of, u it is u minus 1 instead of u , it is u minus 1 that is, we are partitioning the sequence from a m to a u minus 1.

(Refer Slide Time: 39:53)



If p is less than q if p is less than q then, j equals to q plus 1 q plus 1 and then, i call partition partition A p and j , then you call quick sort A , p , j minus 1. Because that means, the position, first element first position that is, p at position has got its own position, which is j . And then, you get quick sort A , j plus 1, q .

(Refer Slide Time: 41:00)



So, idea is very simple that, what happens that, this is p and this is q so, i get a j , this A p moves here all this elements fall of the p of the $(())$ and as long as p is less than q , due to the some partitions. Now, this quick sort algorithm is worst, can you tell me when, you

observe that this complexity is depended on that partition, take this. So, we are assuming that assuming that the j is here but, is so happens that, all the elements all the elements of this part are larger than this edge then this quick sort will not have any meaning, because because that does not exist any elements in this side and you have remaining n minus 1 elements to be sorted by this. So, again next intention, we may have if all this elements are larger than these elements and so on.

(Refer Slide Time: 42:28)

The image shows a chalkboard with the following handwritten equations:

$$\begin{aligned}
 T_n &= cn + T_{n-1} + T_0 \\
 &= T_{n-2} + c(n-1) + cn \\
 &= T_{n-3} + c(n-2) + c(n-1) + cn \\
 &= T_{n-n+1} + c.2 + c.3 + \dots + cn
 \end{aligned}$$

So, in that case what happens, if T_n is the time complexity that, you need order in n times to partition and size is not reused, only one element is one side and the other side there is nothing, no elements right. This side no element, this side you have n minus 1 elements so, we have T_n plus T_{n-1} plus T_0 , T_0 is 0 so, you get next time T_{n-2} plus $c(n-1)$ plus cn right. Next time again no elements in one side so, you get so, we have n minus 1 c 2 c 3 c n , this is one; one means T_1 T_1 is 0.

(Refer Slide Time: 44:02)

A chalkboard showing the derivation of the recurrence relation for the worst-case time complexity of a partitioning routine. The equations are written in white chalk on a dark green background. The derivation starts with $T_n = cn + T_{n-1} + T_0$ and proceeds through several steps to show that the complexity is $O(n^2)$.

$$\begin{aligned} T_n &= cn + T_{n-1} + T_0 \\ &= T_{n-2} + c(n-1) + cn \\ &= T_{n-3} + c(n-2) + c(n-1) + cn \\ &= T_{n-n} + c \cdot 2 + c \cdot 3 + \dots + cn \\ &= c \{ 2 + 3 + \dots + n \} = O(n^2); \end{aligned}$$

So, you get $c \cdot 2$ plus 3 plus n , which is order n square so, in the worst case, it comes order n square. Now, invalid what it means, that when the elements are in increasing order or decreasing order, you will find that partition routine will give you, not give you that good partitions, it will find that one side you have n minus 1 elements, the other side you would not have any other elements so, the complexity in the worst becomes order n square. Now, what is the best case, can you tell me, the best possible case is when the partitioning elements divides the two sequence divides the whole sequence into two equal parts.

(Refer Slide Time: 45:12)

A chalkboard showing the derivation of the recurrence relation for the best-case time complexity of a partitioning routine. The equations are written in white chalk on a dark green background. The derivation starts with $T_n = cn + 2 T_{n/2}$ and proceeds through several steps to show that the complexity is $O(n \log n)$.

$$\begin{aligned} T_n &= cn + 2 T_{n/2} \\ &= cn + 2 \left\{ cn/2 + 2 T_{n/4} \right\} \\ &= cn + cn + 2^2 T_{n/4} \\ &\vdots \\ &= \underline{cn \log n = O(n \log n)} \end{aligned}$$

In that case in that case, the time complexity becomes becomes $c n$ into 2 times $T n$ by 2, if it is $c n$ plus 2 $T n$ by 2 so, I can write $c n$ plus 2 times, $c n$ by 2 plus 2 times $T n$ by four. And this becomes $c n$ plus $c n$ plus 2 square $T n$ by 4 right and you can show, this is nothing but, $c n \log n$, which is order $n \log n$. So, in the best case, it is order $n \log n$ in the worst case, it is order n square.

(Refer Slide Time: 46:16)

The chalkboard contains the following mathematical expression and diagram:

$$T(n) = n + 1 + \frac{1}{n} \sum_{k=1}^n (T_A(k) + T_A(n-k))$$

The diagram below the equation shows a horizontal line representing an array of length n . A vertical line divides the array into two parts of length $k-1$ and $n-k$. A small 'p' is written to the right of the array, and an asterisk '*' is written below it.

Now, let us try to find out the average time complexity or average number of comparisons you need in the quick sort. Let us assume that, $T_A n$ is the average time complexity this is the nothing but, $c n$ is a time for the partitioning, plus 1 by n . Actually to be in exact, it is n plus 1 and it is 1 plus n summation say, T_A of k it should be k minus 1, $T_A n$ minus k , this is k minus 1.

k is 1 to n , what it means, that you are partitioning it, this is k so, if it is k , your value remains so so, this side it is n minus k , this side you have k minus one and k can be 1, k can be 1 2 n k can be 1 2 n . So, that is the time complexity, average time why 1 by n you are assuming, they are equally likely so, that is why, it is 1 by n .

(Refer Slide Time: 47:48)

$$T_A(n) = n+1 + \frac{1}{n} \sum_{k=1}^n (T_A(k) + T_A(n-k))$$

$$n T_A(n) = n(n+1) + T_A(1) + T_A(2) + \dots + T_A(n-1)$$

$$= n(n+1) + 2 \sum_{i=1}^{n-1} T_A(i)$$

So, this gives you basically n times T A n equal to n into, n plus 1 plus 1 by no plus T 0 T 1 T n minus 1 plus T n minus 1 T 0 this is nothing but n into n plus 1, 2 times summation t, this is T A 1 T A 1 plus T A 2 T A n minus 1, T A i, i is 1 to n minus 1.

(Refer Slide Time: 49:27)

$$T_A(n-1) + T_A(n-1) + \dots + T_A(0)$$

$$T_A(0) = T_A(1) = 0$$

Because T A 0 does not have any meaning, not only T A 0 does not have any meaning, T A 2 also does not have any, T A 1 does not have any meaning. (()) T A 0 is equal to T A 1 is equal to 0 so, i take, i equal to 2 to n minus 1.

(Refer Slide Time: 50:06)

$$T_A(n) = n+1 + \frac{1}{n} \sum_{k=1}^n (T_A(k) + T_A(n-k))$$

$$nT_A(n) = n(n+1) + T_A(1) + T_A(2) + \dots + T_A(n-1) + T_A(n)$$

$$= n(n+1) + 2 \sum_{i=2}^{n-1} T_A(i)$$

$$(n-1)T_A(n-1) = n(n-1) + 2 \sum_{i=2}^{n-2} T_A(i)$$

Now, I want to solve this, we have to solve this, let us replace n by n minus 1, what i get n minus 1, T A, n minus 1 equal to n, n minus 1 plus 2 times, i equal to 2 to n minus 2, T A i.

(Refer Slide Time: 50:33)

$$nT_A(n) = n(n+1) + T_A(1) + T_A(2) + \dots + T_A(n-1) + T_A(n)$$

$$= n(n+1) + 2 \sum_{i=2}^{n-1} T_A(i)$$

$$(n-1)T_A(n-1) = n(n-1) + 2 \sum_{i=2}^{n-2} T_A(i)$$

$$nT_A(n) - (n-1)T_A(n-1) = 2n + 2T_A(n-1)$$

When you if subtract is it, i get n T A n minus, n minus 1 T A, n minus 1 here, we get n square n square get cancel, 2 n 2 n and here, i get plus 2 times T A n minus 1.

(Refer Slide Time: 51:19)

$$\begin{aligned}
 n T_A(n) &= 2n + (n+1) T_A(n-1) \\
 \frac{T_A(n)}{n+1} &= \frac{2}{n+1} + \frac{T_A(n-1)}{n} \\
 &= \frac{2}{n+1} + \frac{2}{n} + \frac{T_A(n-2)}{n-1} \\
 &= 2 \sum_{k=3}^{n+1} \frac{1}{k} + \frac{T_A(1)}{2}
 \end{aligned}$$

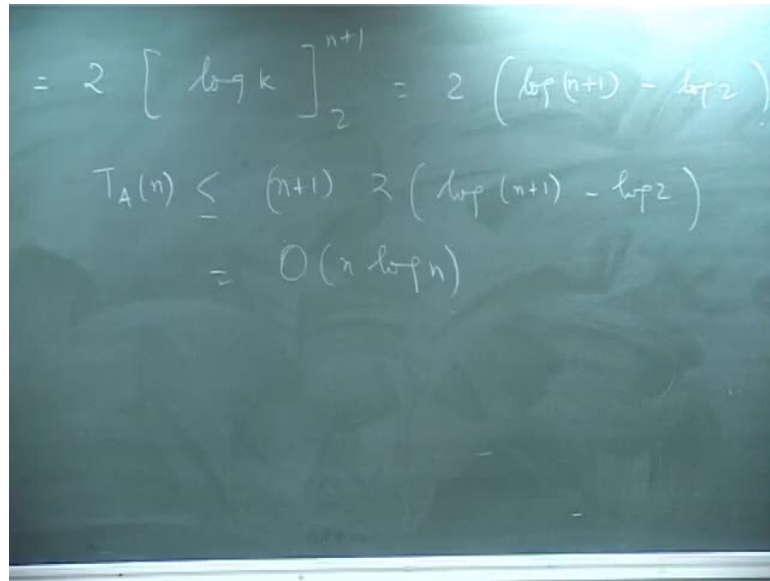
So, if I bring this element this side, I get $T_A(n)$ equal to $2n$ plus $n+1$ $T_A(n-1)$. Now, dividing both sides by $n+1$ into $n+1$ that is, I get $T_A(n)$ by $n+1$ plus 2 by n plus $T_A(n-2)$ by $n-1$.

(Refer Slide Time: 52:48)

$$\begin{aligned}
 \frac{T_A(n)}{n+1} &= \frac{2}{n+1} + \frac{T_A(n-1)}{n} \\
 &= \frac{2}{n+1} + \frac{2}{n} + \frac{T_A(n-2)}{n-1} \\
 &= 2 \sum_{k=3}^{n+1} \frac{1}{k} + \frac{T_A(1)}{2} \\
 &= 2 \sum_{k=3}^{n+1} \frac{1}{k} \leq 2 \int_2^{n+1} \frac{1}{k} dk
 \end{aligned}$$

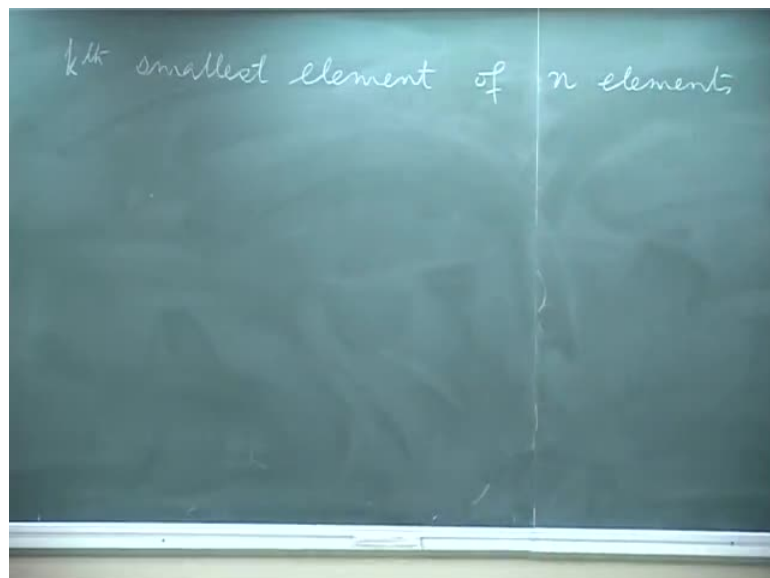
This gives the 2 summation over 1 by k , k equal to 3 times so, plus T_A so, if I write here 3 if I write here 3 , this becomes 1 , this becomes 2 so, $T_A(1)$ is 0 . So, this is equal to 2 times summation k equal to 3 to $n+1$, 1 by k this is less than equal to 2 times integration 2 to $n+1$, 1 by k dk .

(Refer Slide Time: 53:14)


$$= 2 \left[\log k \right]_2^{n+1} = 2 (\log(n+1) - \log 2)$$
$$T_A(n) \leq (n+1) 2 (\log(n+1) - \log 2)$$
$$= O(n \log n)$$

Then, that equal to 2 times log k and n plus 1, 2 that is, 2 times log of n plus 1, minus log two. So, that is $T_A n$ is less than equal to n plus 1, into 2 times log of n plus 1 minus log 2, which is order n log n. So, average time complexity to solve n elements using the quick sort technique is order n log n. See this am could have above in it, just to just for the completely say, I felt that I should tell you, how to compute the average time complexity of n algorithm.

(Refer Slide Time: 54:30)

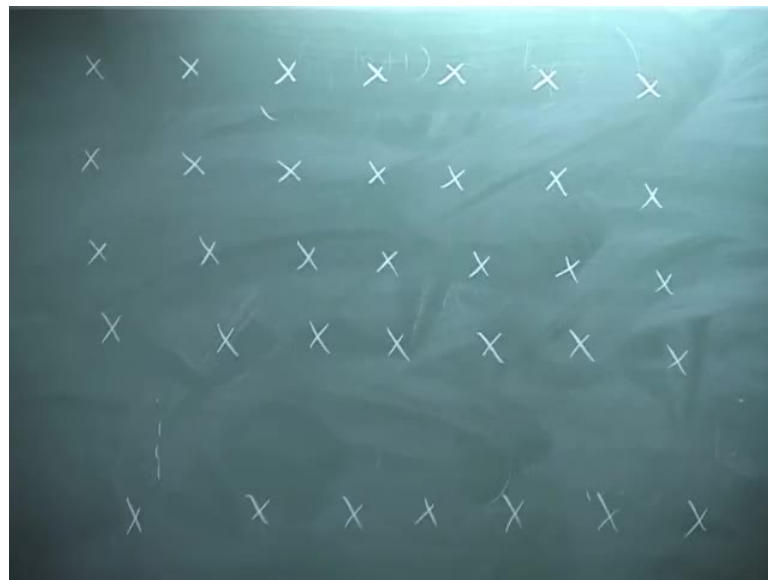


k^{th} smallest element of n elements

Now, let us consider another example, which will be the last example today and actually you have been taught algorithm, which is finding the k th element, k th smallest element of n element. This is disagree to blum's technique and I think I do rub it, do you know any algorithm search? No right. So, let us consider this algorithm and let us try to understand of it is been designed right.

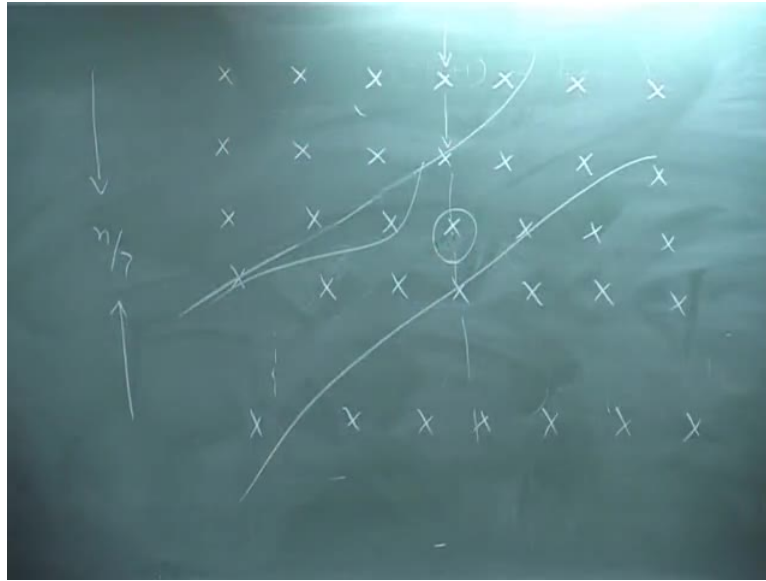
See one way is that, you first find the minimum elements then, you find the second minimum element then, you find the third minimum element and so on. Instead of doing that, let us change from this point of view that, you have the n element, these n elements are divided into $n/7$ groups. And each group is having the 7 elements except first and the last group, which may have one or two sorting may be (()) may not, if divisible by seven and you can padding it by a large number.

(Refer Slide Time: 56:00)



So, you divided is given by seven groups and each group is having 7 elements.

(Refer Slide Time: 57:05)



So, this side we have n by 7 groups and the 7 elements is those now, in the constant, we will use the constant amount of time, you can solve these 7 elements of each groups right. So, you solve each group in constant time so, there n by 7 so, your type of order becomes order n by 7 . Now, this is your median element, these are all median elements of this groups now, find that we say, find the k th element, you find the median of this n by 7 elements.

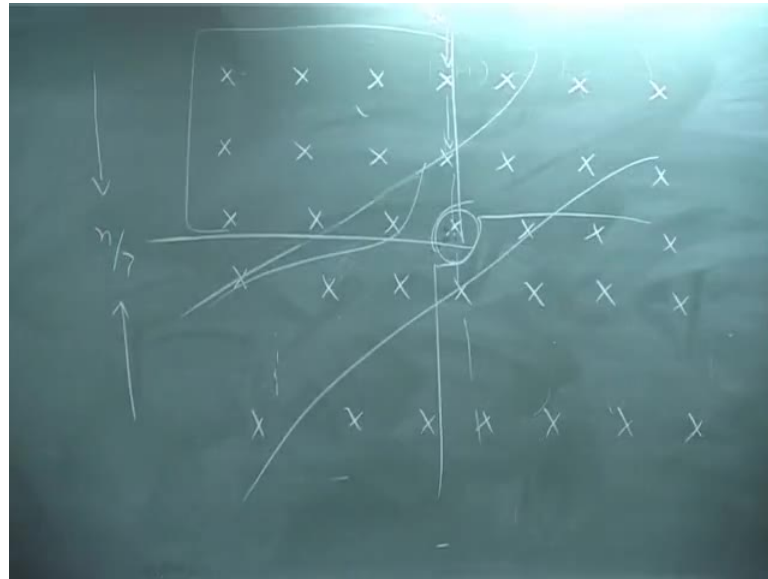
Because, you can call that, one finding the median of the n by 7 medians right, as you are surely finding the k th element now, these median let us quality n , this is the median of median. Now, this median of used to partition to partition these whole sequence into 3 groups, one all the elements smaller than these, another group is all the elements equal to these, and another is all the elements larger than these.

So basically, you will partition 3 groups, one is all the elements all the elements smaller than these, of the all the elements equal to this and all the elements equal to this. When you observe that, if the number of elements smaller than these, smaller than these median element, if your find this more than k then, k eth element is lying here. And if you find no, are the element this size of, the size of this number of elements of number of smaller than this is smaller than k , the smaller than k .

But, in but, if i find the number of elements equal to median number of elements such that, this element is equal to, k is equal to the need no of element. And which are equal to

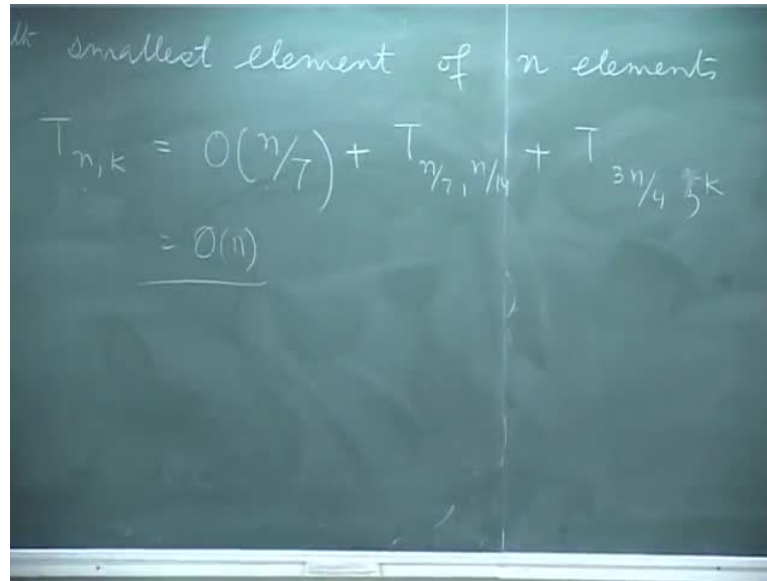
the median elements plus number of elements less than equal to the median elements. If you find that, is greater than k greater than k then, the median element is the k eth element otherwise, k th element is lying in this zone. So, you reduce the searching zone accordingly now, one thing again assured that, at least 25 percent of the, size of the n may be discarded at any instant of time or at any iterations.

(Refer Slide Time: 59:39)



This is because of the fact that because of the fact that, all these elements will be smaller than this element and all these elements will be larger than this element. Now, here since this is the median so, this is you have the half of the, there is n by 4 elements smaller than these and this side also n by 4 larger than these. Now, this side again that half of this side will be larger than the is smaller than this, half of these smaller than this, half of these smaller than this. So, this is giving the 25 percent elements smaller, atleast smaller than this and the 25 percent will be atleast larger than this and you can draw a conclusion on this part. So, it can giving you the assurance that, atleast 25 percent of the half n elements, you will be discarded at any instant of (()).

(Refer Slide Time: 60:43)



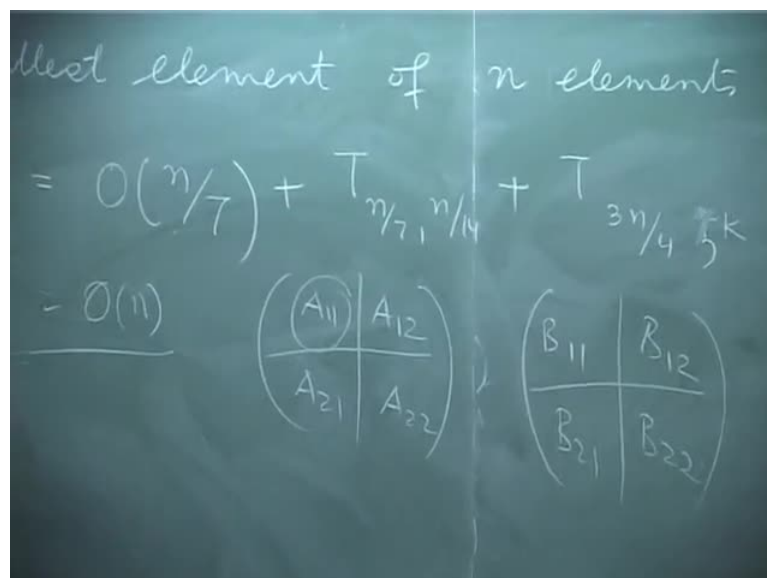
Handwritten text on a chalkboard:

k^{th} smallest element of n elements

$$T_{n,k} = O\left(\frac{n}{7}\right) + T_{\frac{n}{7}, \frac{n}{14}} + T_{\frac{3n}{4}, k}$$
$$= O(n)$$

You know the point of complexity, if I had the $T_{n,k}$ is the time complexity, to find the k^{th} element from the n elements. So, first you may bring the sorting of each group there are there are n by 7 groups so, time is order n by 7 and after that, you will be calling the median of median elements. So, basically, you have n by 7 and k is n by 14 and after finding that, you will be discarding 25 percent, your size becomes $T, 3n$ by 4 and then, k and the solution of this is order n .

(Refer Slide Time: 61:39)



Handwritten text on a chalkboard:

Handwritten text on a chalkboard:

Handwritten text on a chalkboard:

k^{th} smallest element of n elements

$$= O\left(\frac{n}{7}\right) + T_{\frac{n}{7}, \frac{n}{14}} + T_{\frac{3n}{4}, k}$$
$$= O(n)$$

A_{11}	A_{12}
A_{21}	A_{22}

B_{11}	B_{12}
B_{21}	B_{22}

I am not discussing how to solve it because, it is two dimensional recurrence relation, which I do not want discuss in this class. Another example for the divided and conquer (()), which is matrix multiplication here, I will not discussion in detail. Suppose, here the two matrices and you partition it into two parts so, this is suppose, you have A_{11} , this is sub matrix A_{12} , A_{21} and A_{22} .

You have B_{11} , B_{12} , B_{21} and B_{22} then, C_{11} is compute can be computed, by computing that A_{11} dot B_{11} and plus A_{12} and B_{21} right. Similarly, you can computing C_{12} and C_{21} and C_{22} so, this can be done recursively using the divide and conquer strategy that you can try at home. In the next class, we will be discussing about the other strategies, if possible in the next class, we will be talking about greedy method, dynamic programming and if time permits, little about the back back tracking.