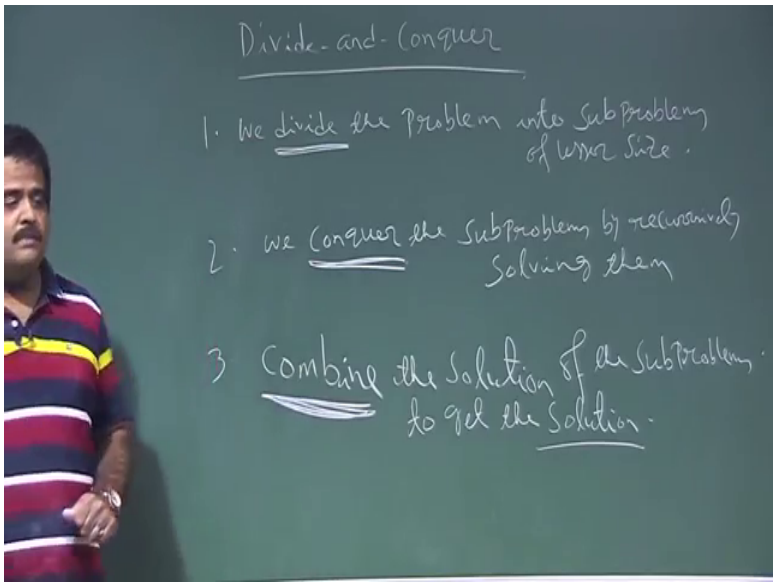
An Introduction to Algorithms
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture – 07
Divide And Conquer

So we talk about different conquer technique for solving certain type of problems. So, so basically it has 3 steps.

(Refer Slide Time: 00:35)



So, it is a design paradigm it has 3 steps. So, first step we divide the problem into sub problems so, that is the divide step. So, basically we have a problem of size n and we divide this problem into sub problems of lesser size, and then we solve this sub problems recursively that is the conquer step and then once we have a solution of this sub problems.

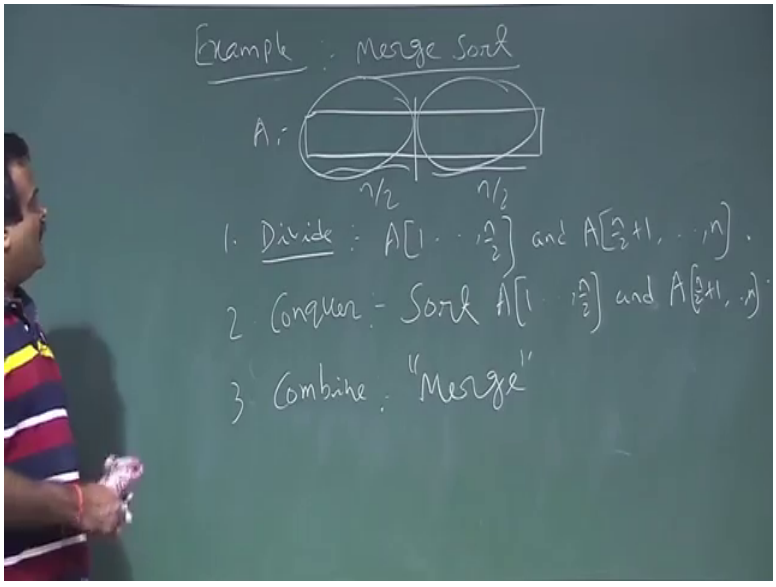
Then we combine this solutions to get a solution of the whole problem let for example, merge sort. So, we have a we have a sorting that is a sorting problem we need to sort a array of size n. So, we divide that into two sub array of equal size and we sort this sub array sort this sub array recursively, that is the conquer step and then we call the mars of routine to combine to get a solution of the whole array. So, basically we divide the problem into sub problems of lesser size that is important.

So, this is basically divide step then we conquer this sub problems by recursively solving them then we conquer the sub problems by recursively solving, them this is the conquer step. So, divide conquer step and then we have a combine step; that means, basically once we have the solution of the sub problems then we combine the we apply a combine step to combine to get the solution whole problem. So, that is called combined step we combine the solution of the sub problems to get the solution of the of the whole problem solution to get the solution of the whole problem.

So, this is the combine step. So, this is basically. So, any divide and conquer technique they has 3 steps basically, we divide step we divide the problem of problem into sub problem suppose we have problem of size n, we divide into the sub problems which is of lesser size then we recursively solve this sub problems by calling it the same function recursively that is the conquer step, then once we have the solution of this sub problems we combine to get the solution of the whole problem. So, this is basically 3 fundamental steps for any different conquer approach.

So, the example is merge sort that is the divide and conquer technique.

(Refer Slide Time: 04:05)



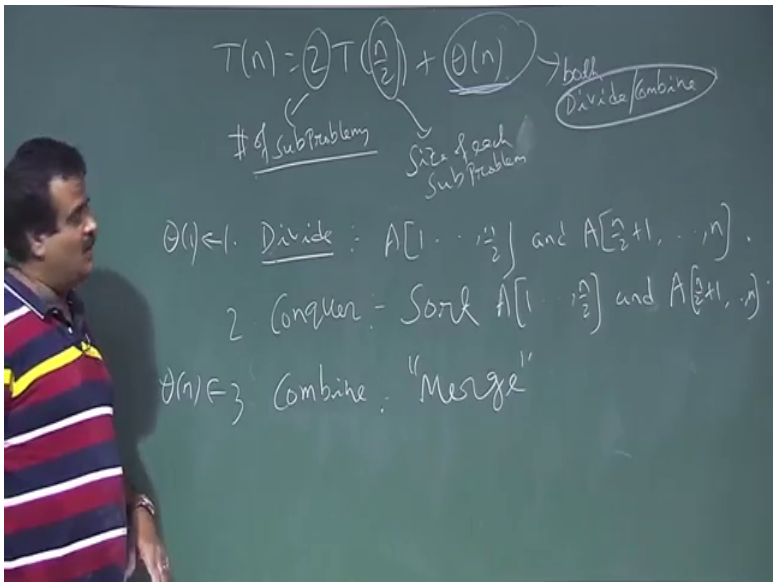
So, merge sort is an example of divide and conquer technique merge sort. So, basically it is a sorting problem. So, we have a. So, in merge sort what we have doing we have a array of size n this is the array of size n we have dividing the array into two sub array n by 2 n by 2 and lower ceiling and upper ceiling, but if n is even though n by 2 n by 2. So,

we that is the divide step we divide. So, this is the divide step we divide the sorry we divide the array into two sub arrays, so then.

So, this is the divide step we divide step is we just divide into two sub arrays 1 to n by 2 and a n by 2 plus 1 to n and the conquer step is we sort this sub array and this sub array recursively by calling the same merge sort. So, there are. So, that is the conquer step. So, we have sorting this sub array 1 to this and we are sorting this wide sub array. Now once we have two sorted array then we call the this is the combine step we call the merge sub routine to get a sorted array.

So, up to sorted array after this conquer step then we call the merge sort to get a sorted array. So, this is the merge sort and in a the recurrence is basically T n is equal to 2 T n by 2 plus theta of n ok.

(Refer Slide Time: 06:13)



So, any divide and conquer technique will have this type of recurrence here two significant is number of sub problems. So, here in merge sort we have two sub problems, and this n by 2 meaning is size of each sub problems.

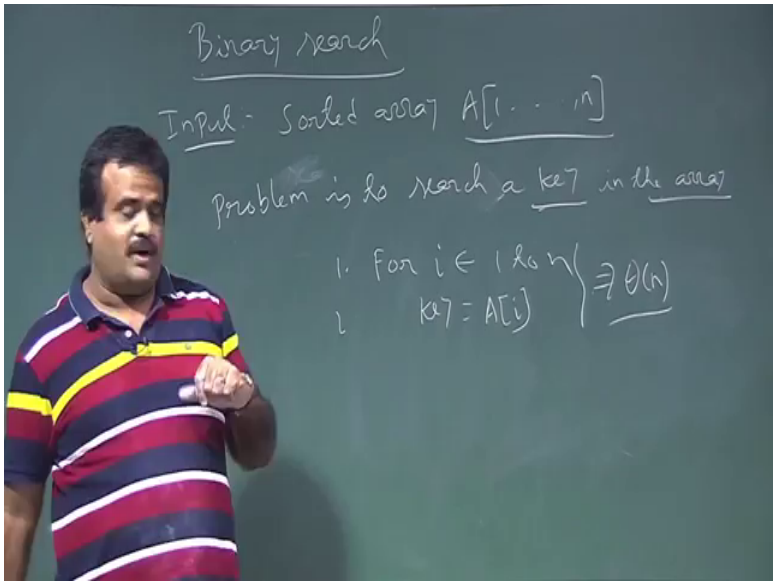
And this is basically what this is basically form merge sort this is the cost for merge because merge sort this divide step is very trivial because we have just going into the middle just 10 by 2 we have just identifying n by 2 that is it is very trivial step for merge sort, but this is crucial this merge, merge will take linear time order of n, but this is cost

for both divide and combine because when we talk about quick sort that is the another example of dividing conquer technique, they are this division this divide step is more expensive than the combine.

Because divide step we need to call what is call a patrician sub routine will patrician this array. So, this will be discuss at the time of quick sort. So, this is the time for both the step both divide and combine this is important. So, this is the any sort any general conquer approach having this type of recurrence where that cost will be denoting by the cost for both the merging both the sorry both the division divide and combine, here division is trivial just say theta 1 here this for merge of this is theta 1 and this is theta n.

So, together will theta n, but when we talk about quick sort this will be theta n and this will be nothing theta 1 may be. So, this is the, this is 1 of the example of divide and conquer technique. So, we will discuss some more problem which can be solve using this technique. So, let us start with very simple one the binary search problem ok.

(Refer Slide Time: 08:59)



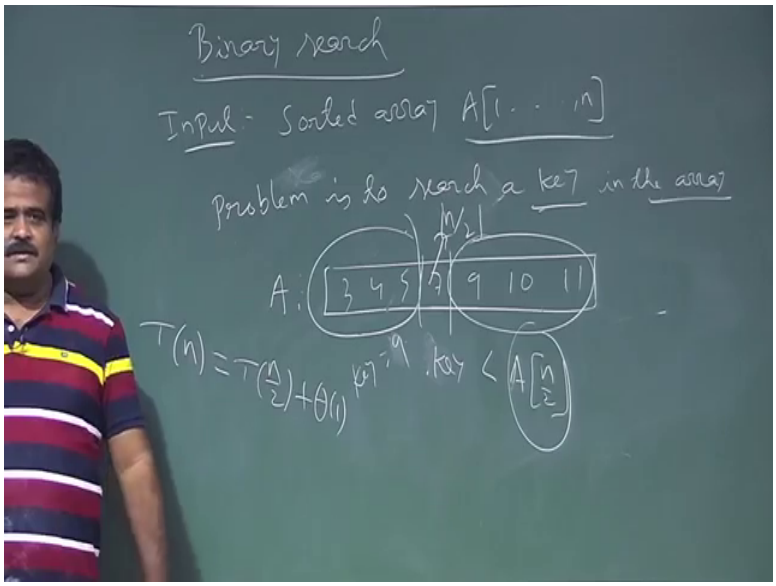
Binary search problem. So, what is the binary search problem? Binary search problem is basically we have a sorted array that is the input and we have another input as a key and we need to find a key whether this is in the array or not.

So, the input is we have a sorted array we have n numbers which, but which is sorted and then another we need to search a and the problem is to is to search a value search a key

in the search key another input, this is also another input this is another integer we need to search this whether this is present in the array or not search key in the array. So, this is the problem. So, how we can do this search?

So, if the. So, what is the naive approach if the say if we have a array. So, we can just do the linear search just for i is equal to 1 to n, we just check whether key is equal to A i or not very simple one this is the search this will take theta of n time because we are not bothering whether the sorted or not, but we have a information that they have sorted. So, that is why we need to take we need to make use of that that they are sorted. So, for that we will do the divide and conquer techniques.

(Refer Slide Time: 10:59)



So, what we do we just this is the array a array which is sorted.

So, this is basically middle one n by 2 for lower ceiling or upper ceiling anyway. So, now, we compare the our key with this value; if key if this value a n by 2 if key is equal n by 2 then we are lucky we just return this we got it first sort best case, but if key is that 3 possibilities equal to we are happy otherwise if key is less than this. Suppose this array is a 3 4 3 4 5 7 9 10 11 and suppose we are looking for say 4 or we are looking for say 5 our suppose our key is 5 we are looking for 5. So, we check with the middle one middle one is 7 this is 7. So, key is not.

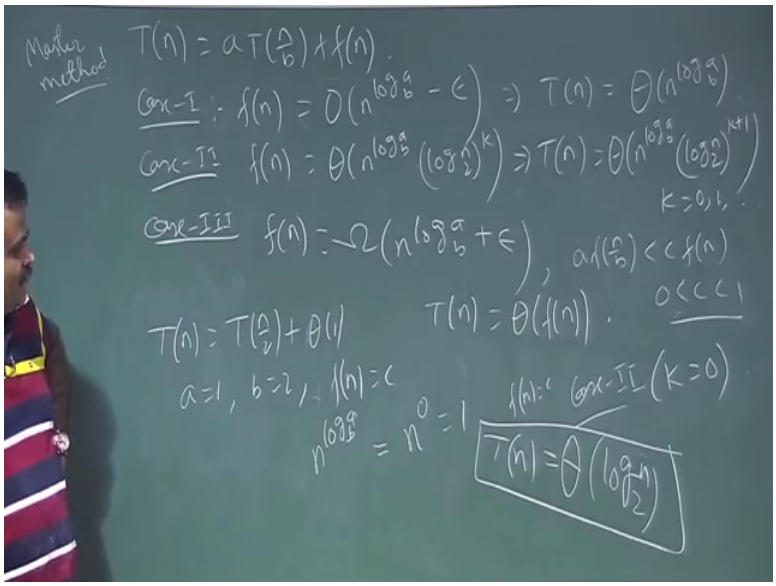
So, 7 is not 5. So, then what we do then we know this is sorted. So, 5 cannot be in the right part of the array. So, 5 has to be in the left part of the array if at all it is there. So, then we must look at the left sub array with the same key. So, that is the conquer step. So, divide step we just compare with the middle element, if the middle element is equal then you return it if the middle element is less than then you look at the right part of left part of the array, and if the middle element is greater than suppose we are looking for say 9 then we must look at the this part of the array because we know 9 cannot be in the left part of the array.

So, we will look at the right part of the array. So, this is what is call binary search algorithm. So, this is a, this is basically a divide and conquer technique. So, in the divide it is very clear in the divide step we just go to the middle of the array element A n by 2 we which compare with the key, if the key is matching we are happy. So, we return the key this is the best case otherwise if the key is less than that value middle one. So, you must have to look at the left part of the array that is the conquer step, then again will search it in the by technical left sub array. So, we reduce the problem into sub problems and then if it is greater than we look at this.

So, this is the divide and conquer technique. So, then what is the recurrence then. So, recurrence will be T n is how many sub problems either we are going for left or we are going for right. So, we are again calling this search either on the left side worst case best case we are getting immediately or in the right side. So, this is basically T n by 2 plus theta 1, because just a one comparison we are doing with the middle one. So, that comparison anyway you do not bother about which computer we are checking this comparison whether Pentium one processor or core two processor.

Because that comparison will take constant time so that is the asymptotic notation we are going to use are good friend good. So, this is our basically recurrence of this binary search now how to get the solution of this recurrence again will take help of the what is call master method.

(Refer Slide Time: 14:51)



So, how to get the solution for this? So, if we compare with this let us recall the master method

So, master method recurrence is a T n by b plus f of n. So, it has 3 cases case 1 case 1 is f n is basically big O of n to the power log a base b minus epsilon.

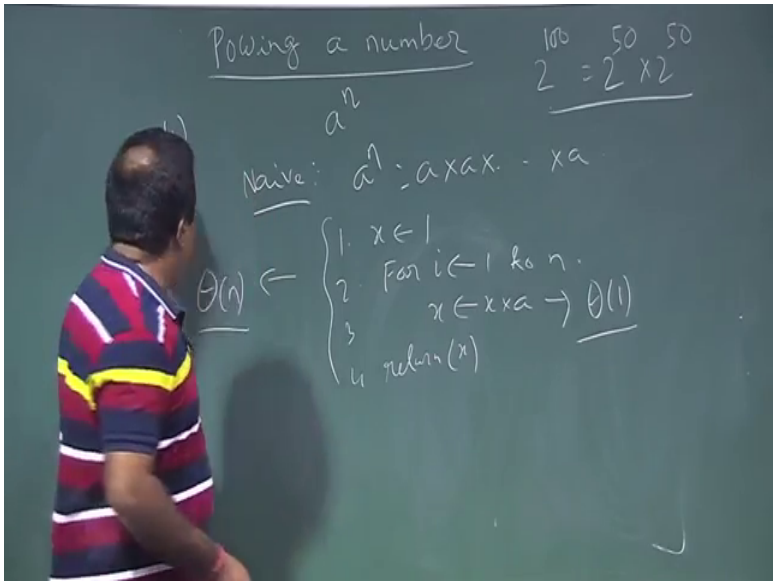
So; that means, this is more than the solution is T n is. So, big theta of n to the power log a base b and case two is basically what case two is their growth is similar or they may vary a log factor. So, this is basically f n is big theta of n to the power log a base b into log n base 2 to the power k then we vary in a log factor or power of log factor. So, in this case solution is basically big theta of n to the power log a base b into log n base 2 to the power k plus 1 and this k is basically 0 1 like this and then we have a third case.

So, will come back to this case 3 which is f n more. So, f n is big omega of n to the power log a base b plus epsilon. So, here epsilon is positive, and we have the regularity condition like a f of n by b must be less the some c of f n had c lies between 0 and 1 this is the regularity condition we have. So, in that case solution is basically big theta of f n. So, this is master method now. So, now, we have a recurrence what? We have a recurrence T n is equal to two T n by b sorry the binary search T n by 2 plus theta 1 that means, c just a constant or we can write theta 1.

Now what is a a is 1, what is b b is 2 and what is f n? F n is basically constant theta 1 now we need to calculate n to the power log a base b, n to the power log a base b this is basically n to the power 0. So, this is basically 1 and f n is also constant. So, which cases we are in we are in? Case 2 with k is equal to case 2 with k is equal to 0. So, case 2 with k is equal to 0then what is the solution? Solution will be T n is equal to big theta of n to the power log a base b that is basically 1 and log n base 2. So, this is the solution for the binary search and this we are getting by the help of master method.

Now let us talk about another problem which can be solved using the divide and conquer technique, which is calls a powering a number. So, this is how to find a to the power n say 2 to the power 1000 or 5 to the power 100.

(Refer Slide Time: 18:56)



So, this is the second problem powering a number. So, basically we want to find a to the power n. So, for example, 2 to the power 100 we want to find or say 5 to the power 50 something like that. So, a to the power n where a or n are both the integer. So, then how what is the a to the power n this is the this is the problem, given a given n we need to find a to the power n. So, this is the problem. So, what is the naive approach?

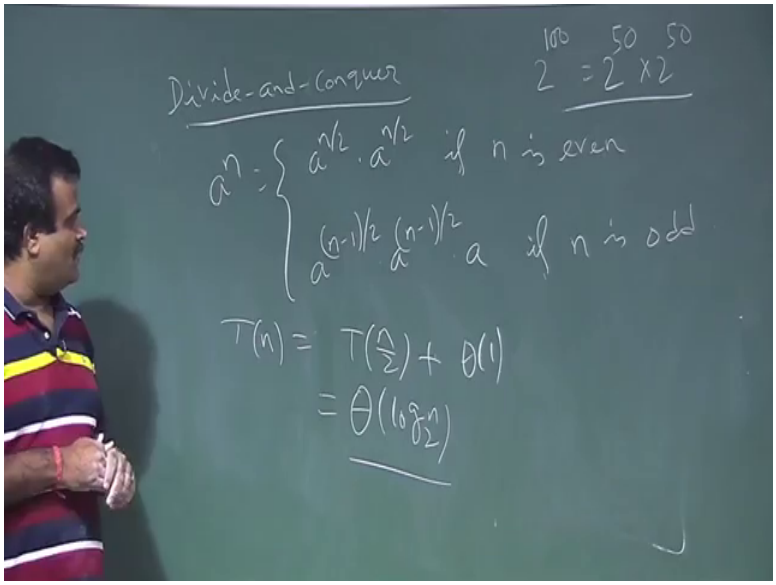
So, a to the power n basically a into a into a into a n times a. So, this is basically a for (Refer Time: 19:48) just simple code for i is equal to. So, before that we need to take a this thing x equal to 1 and then for i is equal to 1 to n x equal to x into a that is it. So, then this is the code then return x. So, x is basically a to the power n. So, then what is the

time complexity of this code how many matrix multiplication sorry not matrix how many multiplication we are happening. So, multiplication we have n multiplication this is the (Refer Time: 20:27) of size n we have n minus 1 because this is a to the power n.

So, we have order of n multiplication and again each multiplication is taking theta 1 time again, we are we are in asymptotic notation we do not care about how much time our hardware is taking to multiply two numbers two integer. So, that time we are not really bothering. So, that time we have treating as theta 1. So, this is time we have treating as theta 1. So, that is the asymptotic sense. So, these algorithm is basically theta of n algorithm because we have a group of order n. Now the question is whether you can do something better than this.

So, now particularly the question is whether we can have a different conqueror approach to solve this problem of powering a number. So, that is basically the idea is if we have to find 2to the power 100. So, this we can find 2 to the power 50 and then you multiply it with it 2 to the power 50 again. So, that way, we have a problem of size high rate. So, you reduce this problem into sub problems say n by 2 like this.

(Refer Slide Time: 21:56)



So, that is the idea. So, let me write the divide and conquer step. So, this is the divide and conquer step for this problem powering a number.

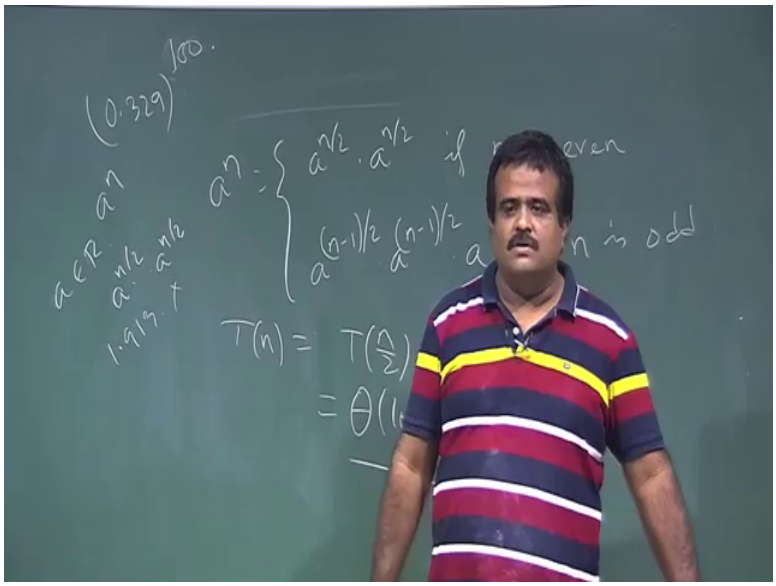
So, what we are doing a to the power n can be written as a to the power n by 2 into a to the power n by 2, if n is even otherwise it is a to the power n minus 1 by 2 into a to the power n minus 1 by 2 into a, if n is odd that is it. So, this is the divide and conquer step. So, basically this formula give us the algorithm, this is the divide conquer step. So, we have a problem of size n we need to find out a to the power n. So, instead of find a to the power n we divide this problem into sub problems, will find a to the power n by 2 then once we get the solution of a to the power n by 2, we multiply with itself then that will give us the.

So, what is the divide step? Divide step is basically we just get a to the power we just get n by 2 and the conquer conquer step means we just try to get a to the power n by 2 that is the recursive call and then once we have the solution then combine step is this multiplication if it is even. If it is odd then what we have if it is odd we have a to the power n minus 1by 2 this is the divide step and then we recursively call this a to the power n minus that is also reduce the problem into sub problems of size n minus 1 by 2 and then combine step here we have if it is what we have 3 matrix 3 multiplication this with itself and with a ok.

So, then what is the recurrence what is the recurrence. So, how many sub problems two sub problems. So, T n is basically 2T n by 2 plus then the divide step and combine step. So, even if it is even we have all multiplication even if it is odd we have 3 multiplication when the divide step is nothing. So, that is basically theta 1 is this recurrence correct now this is not correct because how many sub problems do we need to really calculate twice of this because once we calculate this value a to the power n by 2, then we can store it we do not need to calculate it twice. So, this two on be there.

So, we have basically one sub problems. So, if we calculate a to the power n by 2 is stored somewhere and we multiply it with itself to get the for the combine step that is it. So, this is the recurrence now what is the solution of this recurrence? This recurrence we have seen just now for the, this thing for the binary search method. So, this is basically log of n. So, this is a powering a number now here we are assuming this numbers are integer, but what if numbers are not integer then the problem is suppose we want to find out say suppose we want to find out say 0.329 to the power say 500 or say 100 if we are dealing with the real numbers then the issue is.

(Refer Slide Time: 25:33)



So, then it will be divide. So, this is a to the power n where a is a real number if it is a real number then the problem for (Refer Time: 25:46) number for there is will have the imaginary part of it. So, anyway, then the problem is a to the power n by 2 will be a real number say suppose this is a 1.9123 the say at this is also this. Now if you multiply two real number then we have to fix we have to think of their round of in position. So, these are all the issues will be involved when we deal with the real number. So, that will take more time than this steps, so then only the multiplication.

So, that is why this, this powering a number we restrict our self on a to the power n where a is a integer. So, either we have to deal with the real number position round off all this things will be coming ok.

Thank you.