# Design and Analysis of Algorithms

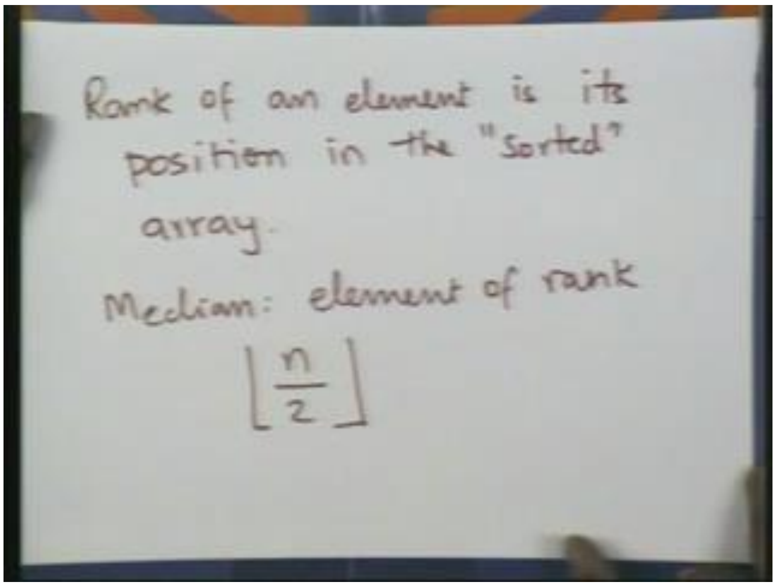
**Prof. Sunder Viswanathan**
**Department of Computer Science Engineering**
**Indian Institute of Technology, Bombay**

## Lecture – 7
## Divide and Conquer – II Median Finding

We had looked at algorithms for sorting, especially quick sort. The way quick sort worked was you picked a pivot, and compared each element in the array with the pivot. And split the array into two parts. Those elements, which were less than the pivot and elements, which were greater than the pivot, and then we recursed down these two parts. And you recursed on these two parts and recursively sorted them. And this then gave you the sorted order of the entire array.

The time taken by quick sort, crucially depends on the position of the pivot on the array. We would like to pick an element, which is the middle of the array. It is called the median. So, if we could always pick the median fast. Then perhaps we can make quick sort work in n log n time. And that is our goal. That is the next goal to see if we can pick we can given an array, we can find the median in time linear in the size of the array. So, let me make a few definitions just to set the ball rolling.

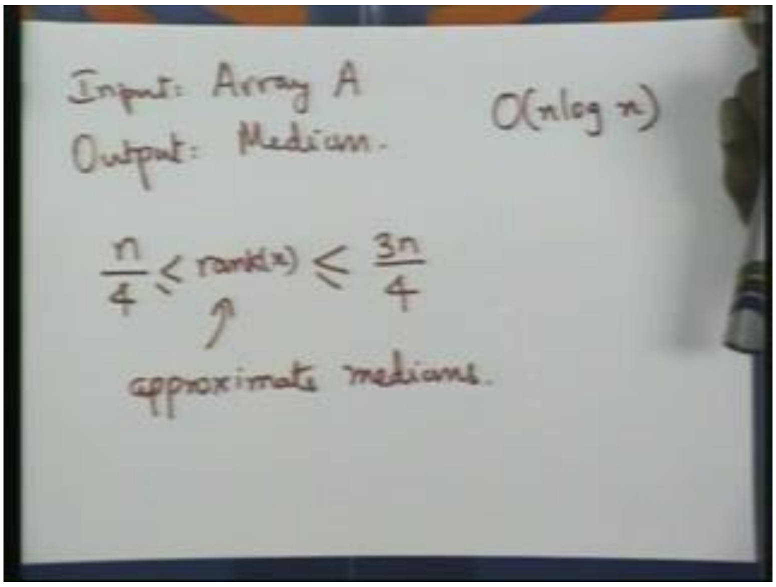
(Refer Slide Time: 02:25)



Rank of an element is its position in the "sorted" array.
Median: element of rank $\left\lfloor \frac{n}{2} \right\rfloor$

The rank of an element is it is position in the sorted array. So, given an array and an element to find the rank of this element sort the array and the position of this element in the array ((Refer Time: 02:57)). We will assume for simplicity that elements in the array are distinct. Every element is distinct. All algorithms that we designed will also work when the elements are not distinct. But, this is just for simplicity. This is the rank and rank of an element and the median is an element of rank n by 2. That is the floor of n by 2, an element of rank floor of n by 2. Our objective is to find the median in an array. How fast can we do this?

(Refer Slide Time: 03:52)



Input: Array A

Output: Median.

$O(n \log n)$

$$\frac{n}{4} \leq \text{rank}(x) \leq \frac{3n}{4}$$

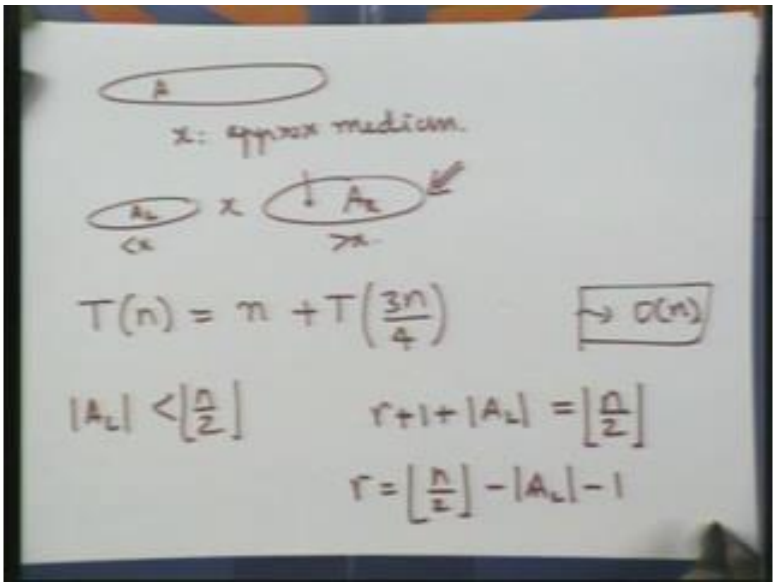
approximate medians.

So, the problem is this. The input is an array A, the output the median. Now, clearly this can be done in order n log n time, by just sorting the array and picking the middle element. Sort the array pick the middle element that gives you the median. And sorting takes n log n time. Our objective of course, is to do this faster. We want to find the median faster. How does one go about doing this? Let can we apply divide and conquer for instance. Supposing, we want to apply divide and conquer.

How do we like to divide the array? How do you divide the array? Again, it seems sort of to think about this and you are up against the wall. So, supposing somehow, you can find a let us say, not the exact median. But, some kind of approximate median. What do I mean by an approximate median? Let us say an element whose rank is greater than n by 4. And rank of x and it is greater than by 4. That is greater than equal to n by 4. Less than equal to 3 n by 4, n is the size of the array.

So, these elements I will call approximate medians. They are not quite the middle element. But, they roughly sit around the middle element. Supposing, you could find supposing somebody gives you an approximate median. Then, what can you do. I mean can does this help. Well, it may help. I mean in the following sense that, so here is my array. I take the array.

(Refer Slide Time: 06:24)



$$T(n) = n + T\left(\frac{3n}{4}\right) \quad \boxed{\to O(n)}$$

$$|A_L| < \left\lfloor \frac{n}{2} \right\rfloor \qquad r + 1 + |A_L| = \left\lfloor \frac{n}{2} \right\rfloor$$

$$r = \left\lfloor \frac{n}{2} \right\rfloor - |A_L| - 1$$

This is my array A. x is an approximate median. Let us, not lose focus of our goal, which is to find the exact median. x is an approximate median and well you just come to know of it somehow. What you could do is this. You split the array as less than x and greater than x. So, here are elements less than x. Here, are elements greater than x. This is less than x. This is greater than x. Once, you do this, you know where the median falls. If this is less than n by 2, if the rank of x is less than n by 2 then the median falls on the right.

It sits in this portion of the array. On the other hand, if the rank of x is greater than n by 2, then the median falls in the left portion of the array. You know where, it which portion of the array it sits. So, the other good thing about this partitioning is that, see both these parts have size at least n by 4, the elements which are less than x, as size at least n by 4. The elements greater than x has size at least n by 4. So, if you can get rid of one part, let us say the smaller part. Then the bigger part is of size utmost 3 n by 4.

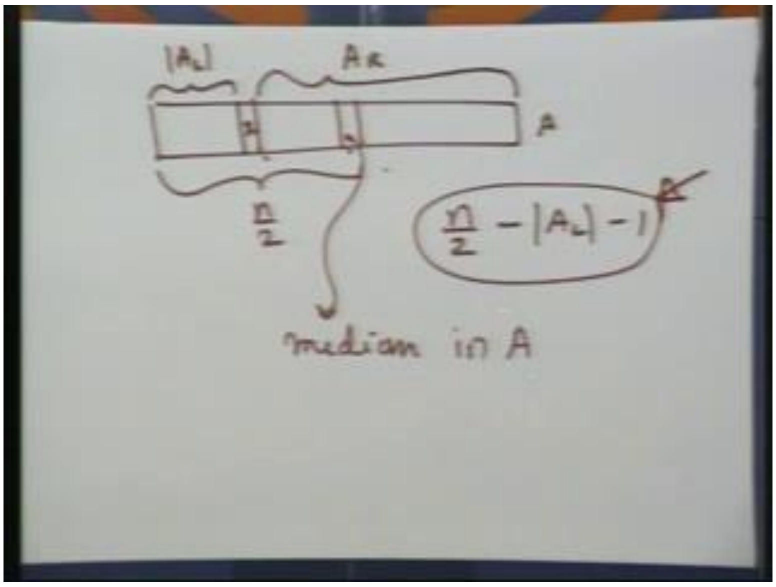
So, you have shrunk the size of the array by a constant factor. Does this is that does this help or actually it does. Supposing I can find this approximate median in let us say, linear time somehow may be you do not know how that is done. So, let us see, how much time for finding the exact median will take. So, what we would like to do is this. We have an array of size n, so let us say the time is T n. We find approximate median, which takes n time. And then, we would like to recurse on a small part.

That part has size 3 n by 4. Well there is still a problem, which we need to address. But, supposing you can do this. Then, you check that this recurrence gives you order n. You can just open out a few terms here. And you can prove that this in fact is order n. There is a problem. The problem is this. I want to find the median of the big array. I somehow get this approximate median in linear time. This we still do not know how this is done. I know that the median is here somewhere. This is the element I want to find.

The element could sit here not exactly in the middle. So, what I want to find in this array is not exactly the median. The element that I want to find depends the rank of the element I want to find depends on the left hand side of the array. So, what I want to find let us say that the numbers. So, let us call these two things let us say, a left and a right. Let us, assume that the size of a left is less than n by 2, floor of n by 2. We will assume this without loss of generality. Now, what I want is the median of the entire array.

What is it is rank in A R. So, it is the element that I want. What is the rank in A R. Supposing that rank is R. What I want is r plus 1 plus size of A L should be equal to n by 2. In other words, I want r to be n by 2 minus size of A L minus 1. So, r is the rank of the median of A in A R. The small r is the rank of the median of A in this array A R. Let me, draw a picture and sort of explain this again.

(Refer Slide Time: 11:32)



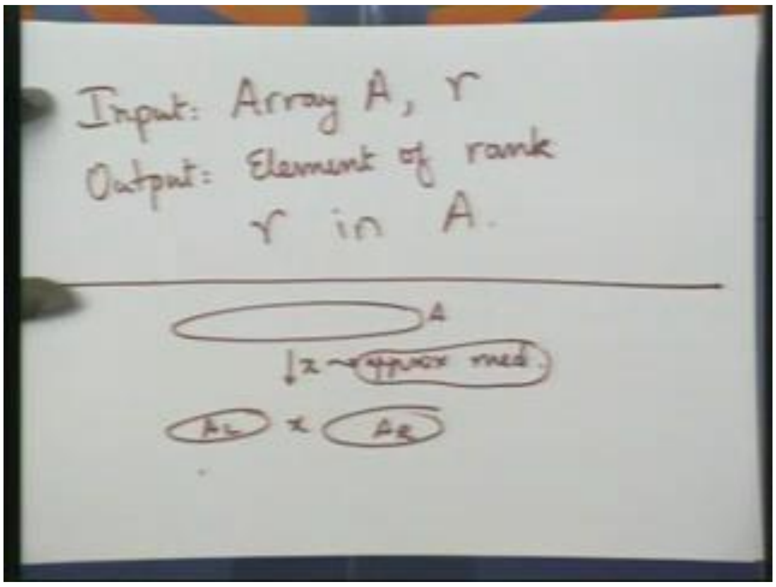
This is A. ((Refer Time: 13:34)) this is sorted here is my middle element. This is n by 2. Now, my element x sits somewhere here. This is this is element x, which is approximate median. And this portion is A L. This portion is A R. When the size is A L this portion is A R. Now, the rank of the median is just the rank, the number of elements in this portion. What is that, n by 2 minus A L plus 1. So, it is nothing but n by 2 minus size of A L minus 1.

So, the rank of this element, which is the median, this is the median that we want to find. So, this is the median A. The rank of this element in A R is this. So, the problem when we recurse we do not really want the median in A. We are not looking for the median. We are looking for an element of rank this. We are looking for an element of rank n by 2 minus A L minus 1. And this need not be the middle element in A L. This is the first problem.

So, when we recurse, we do not really want the i mean it is not just the median. But, we want an element of some particular ((Refer Time: 13:15)). Well, why not look for a procedure to find an element of any rank. So, pay careful attention here. This is a very sort of important design principle. We started off with trying to find the median. We tried to find the recursive procedure. Along the way we have encountered a problem, where we actually want to find an element of some other rank.

Not just rank n by 2, but some other rank. So, this seems to be a more general problem. Finding the median a special case of this problem. Sometimes, it is easier to solve a more general problem, than a specific problem. And in this case, that is what we will do. So, the problem we would like to tackle is not just finding the median. It is finding the element of any rank. So, the input let me write this problem now.

(Refer Slide Time: 14:17)



Input: Array A, r

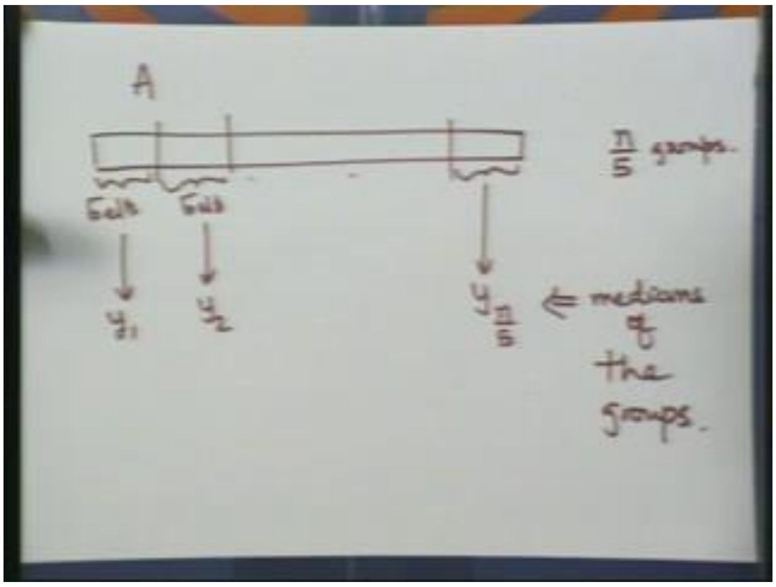
Output: Element of rank r in A.

Input is array A and r. The output is an element of rank r in A. Now, when r is n by 2 we get the median. This is a more general problem, than the one we started out with. And this is what, now we would like to solve. The reason this problem came up is that during the recursion, I mean at least the recursion that we tried we started out with the median. But, when we try and get the recursion going the problem we end up with is this. So, this is the problem we will try and solve.

Now, you see that there is no problem with the recursion at least. So, if I solve, if I find an array A, I start with an array A. Supposing, this is still a step which is left undone. I get x, which is an approx median. I split the array into two parts A L and A r and x. I want to find an element of rank r. If r is less than the size of A L, I recurse on A L. If r is greater than size of A L plus x, I recurse on A R. That is it. That is the algorithm.

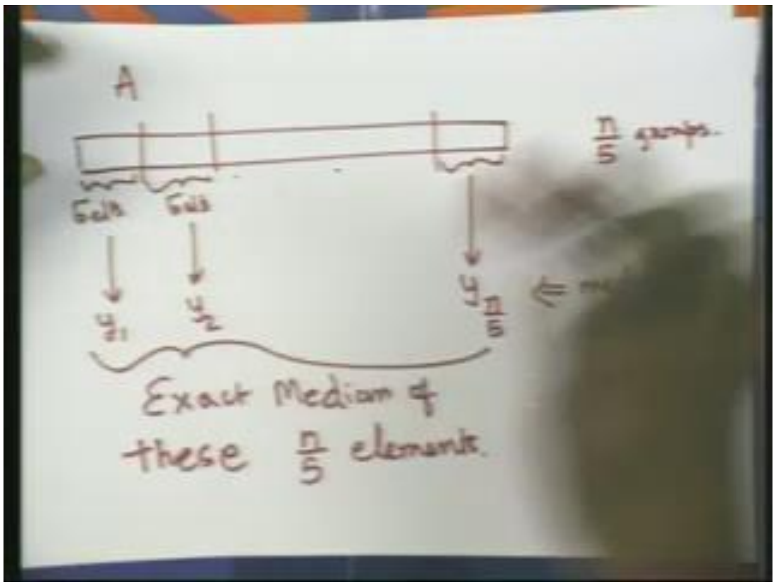
So, as long as I can find a approximate median, I can find the element of any rank, by this divide and conquer scheme. You can write down the recurrence relation. And see that, if we can find this approximate median in linear time. Then, you can find the an element of any rank till linear time. So, how will you find the approximate medium. This is what, this is seems to be as difficult as finding the exact median of any rank. So, it is and the solution that I am going to present. Now, is really smart. So, the way you find the approximate median is this.

(Refer Slide Time: 16:49)



First take the array A. So, unfortunately I cannot tell you exactly how people came up with this. I am just going to present the solution to you. Sometimes, we just have to be really clever and come up the solution. There is no real recipe for finding algorithms all the time. There are the few general recipes that one follows, but often you know these things are so problem dependent. That you just have to think about it think about it and at some point, you know some light bulb goes off somewhere. And you know you come up the algorithm that works. So, here is the algorithm that works. So, take the array A. Split up the elements in the following form.

(Refer Slide Time: 17:32)



So, here is A, I split the elements into groups of 5. So, these are 5 elements. These are the first 5 elements, the next 5 elements and so on. And I have n by 5 groups. I have n by 5 groups in a and each of them has 5 elements. Now, find the median of these 5 elements. This will give me some median y 1. This will be some median y 2. And the last one will give me some median y n by 5 these are medians of the groups. There are n by groups. Each of each consisting of 5 elements and this is the median of these groups.

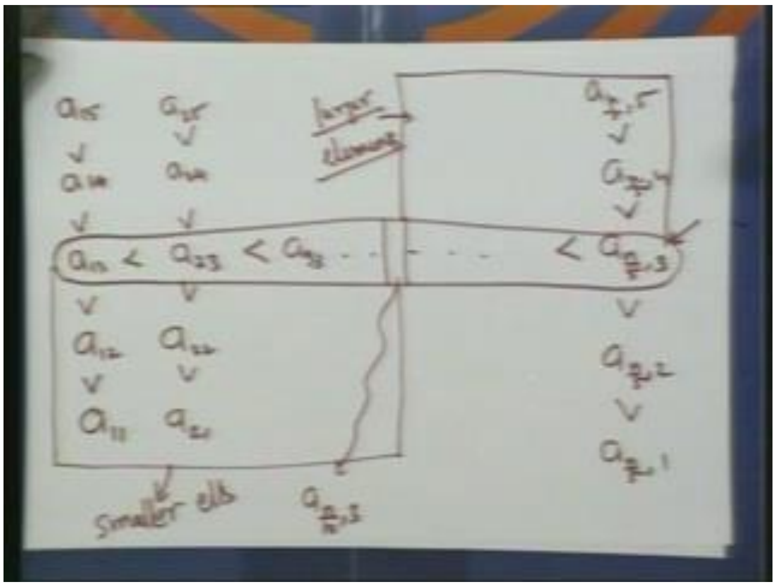
Now, the approximate median, which we will pull out is the exact median, of these n by 5 elements. Approximate median that will output exact median of these n by elements. So, let me repeat this again. You split them into groups of five. There are n by 5 groups. For each group you find the median y 1, y 2 up to y n by 5. Each group, you find the median. Now, you find the exact median of these n by 5 elements. And this is an approximate median. There are number of questions.

Why should this be an approximate median? That is the first question. Second question, we started off trying to find the exact median. We ended with trying to find the approximate median. And we are again finding the exact median. To find the approximate median, we find these y 1 through y n by 5. And again we are finding the exact median. What is going on. Well, the second thing is not so serious. In the sense, that initially we started out trying to find the median of n elements.

We are now finding the exact of median on n by 5 elements. So, the size has reduced. And we can apply the, we can recurse on this smaller size. Remember I said, we can find solve the problem for a smaller size. You know, you can put the solution back up. So, the exact median that we want to find among these n by 5 elements. We just do it by recursion. We just recurse and find the exact median here. That gives you what we want. We said, we want to find an element of any rank.

This is what we want to do. If somehow, we can find the approximate median we have done. To find the approximate median, we have to solve an exact median problem on an input of smaller size. This can also be done by recursion. I have to still convince you, that the exact median of these n by 5 elements will be an approximate median. Let us see why this is. To see this, let me arrange these 5 elements in each group in descending order.

(Refer Slide Time: 21:49)


larger elements
smaller elts
$a_{15} > a_{14} > a_{13} > a_{12} > a_{11}$
$a_{13} < a_{23} < a_{33} \ldots < a_{\frac{n}{5},3}$
$a_{\frac{n}{10},3}$

So, smallest element will be at the bottom say a 1 1, then the next element a 1 2. Then, a 1 3, a 1 4, a 1 5. So, all of these, I will all of these small groups, let me first write them down in increasing order upwards or decreasing order downwards. The next thing I am going to do is look at these middle elements, a you know the medians of these groups. And I will write them in sorted order towards the right. So, a 1 3 will be the smallest element among these medians.

So, a 1 3 is less than a 2 3, less than a 3 3. So on and a n by 5, 3. So, this is also part of a group, which I have written in this order. This is a 2 2. This is a 2 1, a 2 4, a 2 5. Similarly, here a n by 5, 2, a n by 5, 1, a n by 5, 4 and a n by 5, 5. So, the entire array, I have written out this way. Well the last row may not be one of these rows, may not may not have all 5 elements. But if the size of the array is divisible by 5. All of these columns will have sorry all of these columns will have 5 elements.

So, let me again explain this picture. You first, split each of them into 5 parts. There are n by 5 groups. Now, you sort, you sort them in increasing order. Look at the middle elements. Sort them middle elements in increasing order, lay them out. And lay out the entire array in this fashion. So, a 1 3 is this smallest element among the middle elements. It comes in some group. It may be the 20th group or the 15th group or whatever. Whatever it is, I will call it a 1 3.

So, the first group here I get by taking that part of the array, where the middle element was the smallest. When, I look at the middle elements this has to be the smallest. So, then comes the second group and then the third group and so on. And this is the last. I will just arrange the elements of the array this way. Now, what is the median of this, this portion. Remember, that we said that the median of this portion is an approximate median. Median of this middle portion is what we claim to be an approximate medium.

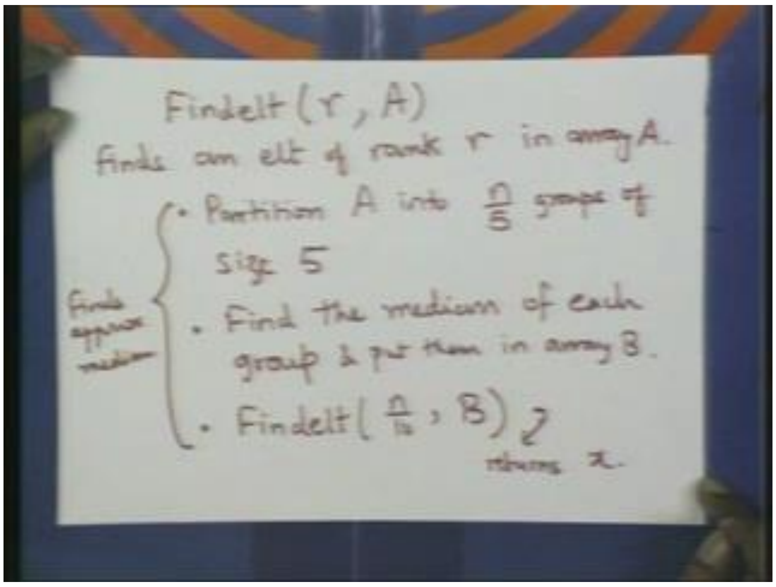
So, what is this middle what is this. So, it sits somewhere in the middle here. So, it is the way we have written this, it is a n by 10, 3. That is the element, which is the median of this, this portion. Now, how many elements are smaller than this medium? Where do you find elements in the array, which is smaller than this medium. Well, if you look at this middle row, everything to the left here is smaller. Also, everything down here is also smaller, which means this entire portion consists of elements, which are smaller. These are smaller elements.

Similarly, if you look to the right these are all larger elements. And also, if I go right and up I also get larger elements. This is also larger than this. So, this portion again consists of larger elements. This portion consists of larger elements this portion consists of smaller elements. And you can see that, each of these this portion is at least 1 4th. In fact, it is greater for our purposes this is at least 1 4th of the entire array. This portion is at least 1 4th of the entire array. That is the reason, this is an approximate median.

We have just shown that, at least n by 4 elements in the array are smaller than this. At least n by 4 elements are larger than this. That is why this is an approximate median. And once, we have the approximate median, well we know what to do next. We take this approximate median and recurse on these two parts. And I have well in sort of vague disjoint way I have given you all the ideas needed to design this algorithm to find a median.

Let us just put all of this together. Let us just write down each step. Put all these steps together stare at it. And then, see if you can analyze it and see what the running time. So, I will not be giving you code. But, I will write down each step carefully. So, here goes.

(Refer Slide Time: 28:10)



Findelt$(r, A)$

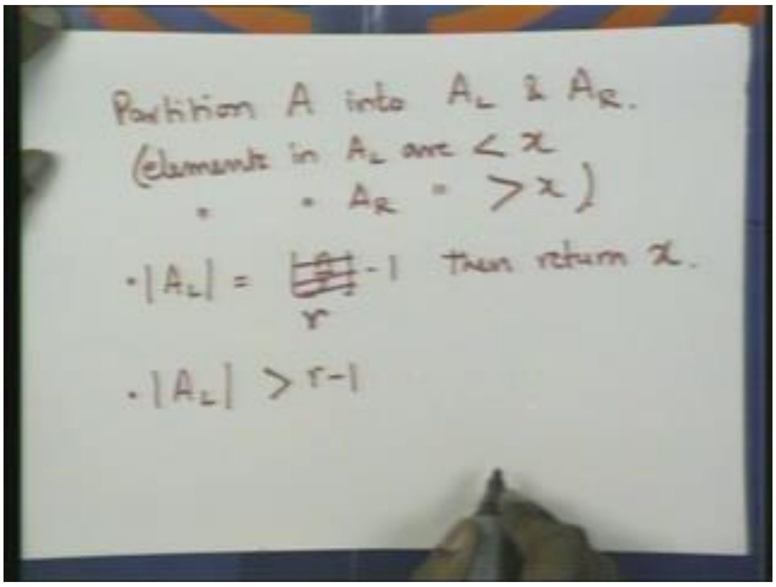
finds an elt of rank $r$ in array $A$.

finds approx median
- Partition $A$ into $\frac{n}{5}$ groups of size 5
- Find the median of each group & put them in array $B$.
- Findelt$(\frac{n}{10}, B)$ returns $x$.

I want to find an element of rank r in an array A. So, let us call this find rank, find element. Let us say, r, A this finds an element of rank r in array A. Now, what do we do. Well the first thing is to somehow gets this approximate median. So, for that well, the first step is this. So, partition array, partition A into n by 5 groups of size 5. Find the median of each group. Now, I want to recurse. I want to find the median of these medians. There are n by 5 groups. So, there are n by 5 elements.

So, each group and put them in an array. Put them in array B say. Array B has size n by 5. Now, I want to find element essentially, I want to find the median in this array So, find element of rank n by 10 in the array B. This is again recursion. So, this portion, find the approximate median. This well this should return an element. So, call this x. So, this returns x. This element x is returned by this call. And this x is the approximate median that we want. Now, what do? Well, we use this is something we have done before. We use x partition array into two parts and then recurse on the right part, that is it.

(Refer Slide Time: 31:14)



Partition A into $A_L$ & $A_R$.
(elements in $A_L$ are $< x$
" " $A_R$ " $> x$)

• $|A_L| = \underset{r}{\cancel{n/2}} - 1$ then return $x$.

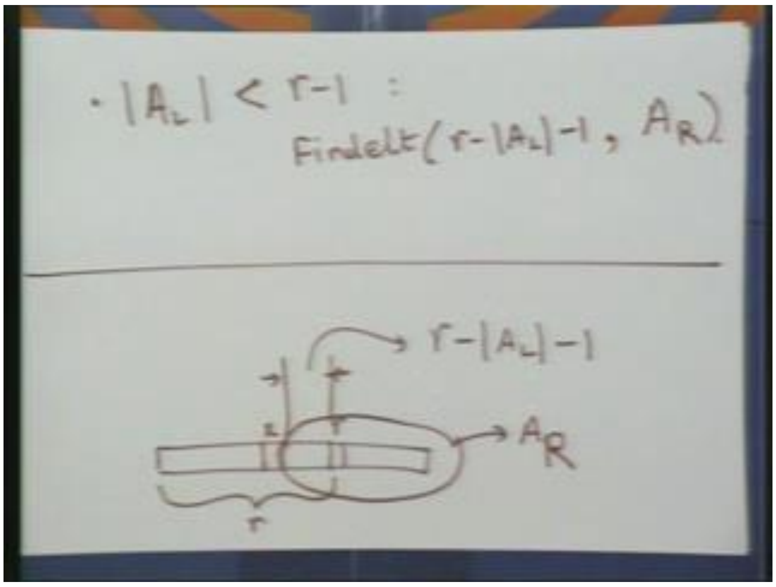
• $|A_L| > r-1$

So, partition A into A L and A R. So, elements in A L are less than x. Elements in A R are greater than x. That is how you get A L and A R. Now, let us see, so now, we are left with cases depending on what the rank of x is we have to either recurse on A L or A R. So, there are 3 cases. The first case is if, x luckily happens to be the element we are looking for. So, when does that happen? So, if size of A L is let us say, n by 2 minus 1. Then, return x, x is the median that we are looking for.

Now, if I am sorry, so size of A L should be r minus 1. We are looking for an element of rank r. So, we go back to here ((Refer Time: 32:40)). What we want to find is an element of rank r in a, not just the median. So, if the size of A L is r minus 1, then rank of x is nothing but r. So, x is the element of rank r and we return x. So, if size of A L is greater than r minus 1. Then, we know that this element of rank r sits in array A L. The element of rank r sits in array A L. So, we recurse on array A L. So, we say find the element.

So, we want still to find an element of rank r. And we now recurse on A L. An element of rank r in A L will also be an element of rank r in A. That is easy to see. And the last case is when size of A L smaller.

(Refer Slide Time: 34:02)



• $|A_L| < r-1$ :

Findelt$(r-|A_L|-1,\ A_R)$

The last case is size of A L is smaller than r minus 1. Then, we need to find an element of rank. So, here is a let us quickly draw picture to find this out. So, here is x. This is A L. And I know that r is somewhere here. This is what I want. Now, this is r, this there are r elements here. There are size of A L elements here, one element here So, the rank of this element in this portion of the array is nothing but this distance. That is nothing but r minus size of A L minus 1. This entire distance is r. This distance is size of A L and x is just one element.

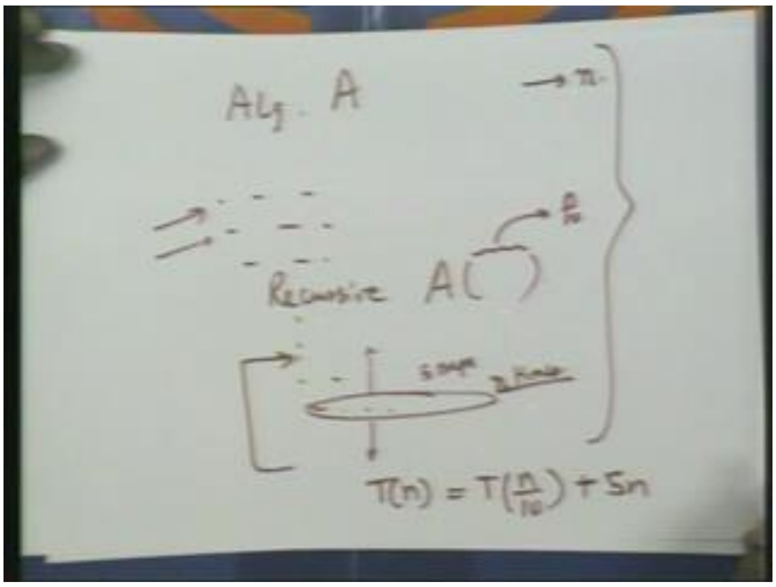
So, we want to find now an element to find element. The rank is r minus size of A L minus 1. And we want to do this in A R. This portion of the array, that portion of the array is A R. That is it. That ends the description of the algorithm. So, let us look at this algorithm again. We want to find an element of rank r in array A. Partition A into ((Refer Time: 36:01)) n 5 n by 5 groups of size 5. Find the median of each group. Now, find the median of these medians.

This returns an element x, which will be an approximate median. And we are now going to partition with respect to x. So, partition A into A L and A R with respect to x ((Refer Time: 36:27)). These are elements smaller than x and those larger than x. And now are going to recurse on one of these two parts. That depends on the rank of this element x. The rank of this element x is nothing but size of A L plus 1. So, if the rank of the element x is r exactly r. Then we return x as the answer.

If the rank of element is x is larger than r that means, the element we are looking for is on the left half of the array. So, we just find an element of rank r in the left half of the array. If this does not work, well the element that we are looking for lies on the right half of the array. And well this is what we do. If the rank is less than r minus 1, then we find an element of rank r minus A L minus 1 in the right half of the array. This is the entire procedure. How do we analyze this? We have to analyze this procedure.

So, we have seen reasons, why this procedure works. Why this we expect that this procedure should run in linear time. Our expectation at this procedure runs in linear time. And we will see why, this procedure actually works in linear time. So, to once so the once you have designed an algorithm. How do you analyze, the easiest way to do it is write down the algorithm. Write down perhaps code for the algorithm. Look at each step. And figure out how many times each step is executed. The step could be a recursive step.

(Refer Slide Time: 38:38)

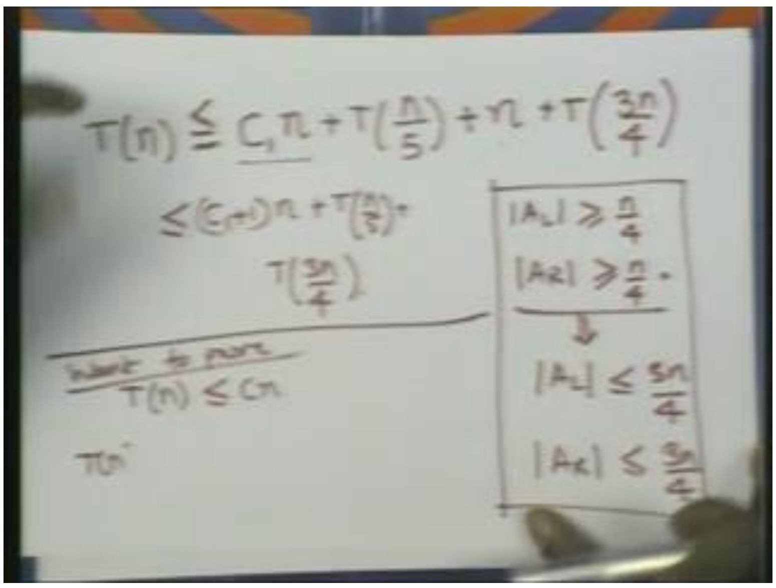


So, here is let us say the, you have written an algorithm, for some problem. There are there are many steps. One of them could be recursive. So, this is an algorithm call it A. So, you recurse on a with smaller input. And then, there is a loop may be and there are steps inside the loop. This is your normal structure of a program. To analyze this, I need to you need to find out, how many times is each step executed. Those outside a loop are executed once. That is fine.

For the recursive call, you need to estimate the size of this input. So, for instance, if initially the input size were n. And this is let us say n by 10. This could be n by 10. And now, you have these things in a loop let us say and you have to find out how many times each of them is executed. Now, supposing there are you know 5 of them, 5 steps here. And each is executed n times.

Then the total time that you will take T n well T n by 10, that comes in the recursive step. And these, totally the total time is 5 times n. There are 5 steps each takes ((Refer Time: 40:14)). Each of them is executed n times. And that gives you 5 n. So, we are going to do something similar to our algorithm for the median. Look at these each of these steps. And we will see how much time each of them takes. So, T of n is the time taken on an array of size n.

(Refer Slide Time: 40:40)



$$T(n) \leq \underline{c_1 n} + T\left(\frac{n}{5}\right) + n + T\left(\frac{3n}{4}\right)$$

$$\leq (c_1+1)n + T\left(\frac{n}{5}\right) + T\left(\frac{3n}{4}\right)$$

want to prove

$$T(n) \leq cn$$

$T(n)$

$|A_L| \geq \frac{n}{4}$

$|A_R| \geq \frac{n}{4}$.

$\Downarrow$

$|A_L| \leq \frac{3n}{4}$

$|A_R| \leq \frac{3n}{4}$

So, we are going to write a recurrence for T of n. So, let us look at the first slide ((Refer Time: 40:45)). Partition a into n by 5 groups of size 5. How much time does this take nothing. This partition is easily done. Find the median of each group. How much time does this take? Well each of those 5 elements, each of those n by 5 groups I could even sort. That will take let us say 25 sort of comparisons for each group. I mean even if I do bubble sort.

And there are n by 5 such groups, so it is some constant times n. So, this step finding the median of each group takes constant times n for the entire array. So, let me write this down. So, this is the first, first step to the cost. C 1 times n. This is to find the median of these groups of each group. There are n by 5 groups. For each of them I spend some constant time. And that gives me a total of C 1 times n. What about this. What about this step. Well, I recurse on something of size n by 10. So, sorry I recurse on B.

This is the rank I want to find. I recurse on B and B had size n by 5. So, I recurse on a problem of size n by 5. So, that is T n by 5 plus. Now, we get here we partition ((Refer Time: 42:41)) this into two parts. So, how much time does this take. This takes n time, because each element is compared with x. So, this takes an additional n. This is to partition this array into two parts. Once we find this x. After, we do this partitioning, now we recurse well, if we are in this case we are done, we return.

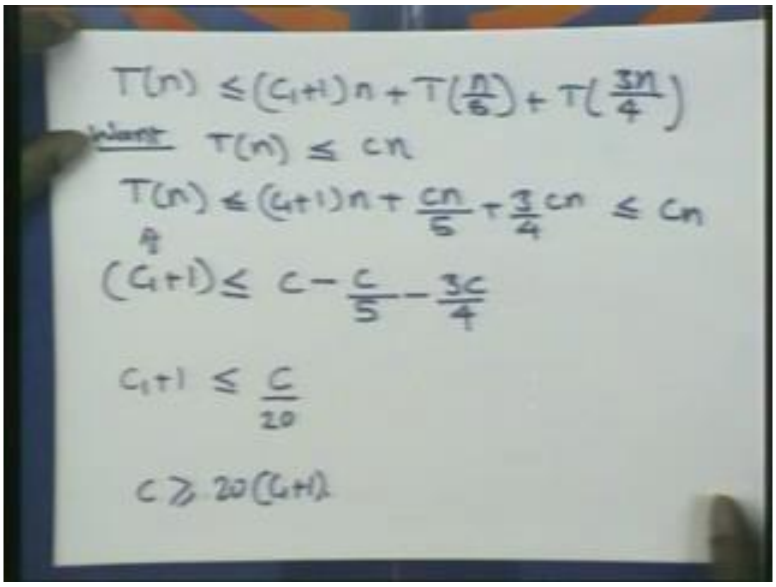
If we are in this case or if we are in the next case, we recurse on a smaller part of the array. So, we recurse on either A L or we recurse on A R depending on where this what was the rank. What was r and what was rank of x. We either recurse on A L or we recurse on A R. Now, we know. So, this is the last step. So, that is what I need to write. So, this T of something this is the recurrent. So, what do I put here. Well, both A L and A R notice, the size of A L I know is at least n by 4. Size of A R I know is at least n by 4.

Or this implies that size of A L is less than equal to 3 n by 4. Why is this. This is because size of A R is at least n by 4. So, size of A L is utmost 3 n by 4. Similarly, size of A R is utmost 3 n by 4. So, whichever side, I recurse the size the maximum size it can be is 3 n by 4. So, I can put 3 n by 4 and I put less than equal to. What I put here the size, can only be less than or equal to 3 n by 4. So, the maximum size an array can have when I do this recursion is 3 n by 4. So, the recurrence equation I get is this.

T n is less than equal to C 1 n. This is to find the median of those 5 of these groups n by 5 groups. Here is a recurrent recursion to find the exact median. This is the time for portioning. And this is the last recursion. We recurse either on A L or A R. And a maximum size of A L and A R is 3 n by 4. So, this is the recurrent. So, T n is less than equal to let us say, C 1 plus 1 n plus T of n by 5 plus T of 3 n by 4. So, what is the solution to this recurrence? That is what we want to see.

So, here is something I want to sort of share with you. So, look at the recurrence. Look at where we recurse. One is a problem of size n by 5. This is a problem of size n by 5. The other is a problem of size 3 n by 4. The sum these two is less than n. n by 5 plus 3 n by 4 is strictly less than n. In such cases usually, you will end up with something which is linear. So, let us try and prove this. So, we know and in such cases instead of writing down this recurrence the way we have been doing there is an easier way to do this. So, I will assume that T n is less than equal to C n. I want to prove this. So, I want to prove this. So, let us just substitute this into the previous equation and see what we get. So, I get T n, so let me do this on a new sheet.

(Refer Slide Time: 46:59)



$$T(n) \le (C_1+1)n + T\left(\frac{n}{5}\right) + T\left(\frac{3n}{4}\right)$$

Want $T(n) \le cn$

$$T(n) \le (C_1+1)n + \frac{cn}{5} + \frac{3}{4}cn \le cn$$

$$(C_1+1) \le c - \frac{c}{5} - \frac{3c}{4}$$

$$C_1+1 \le \frac{c}{20}$$

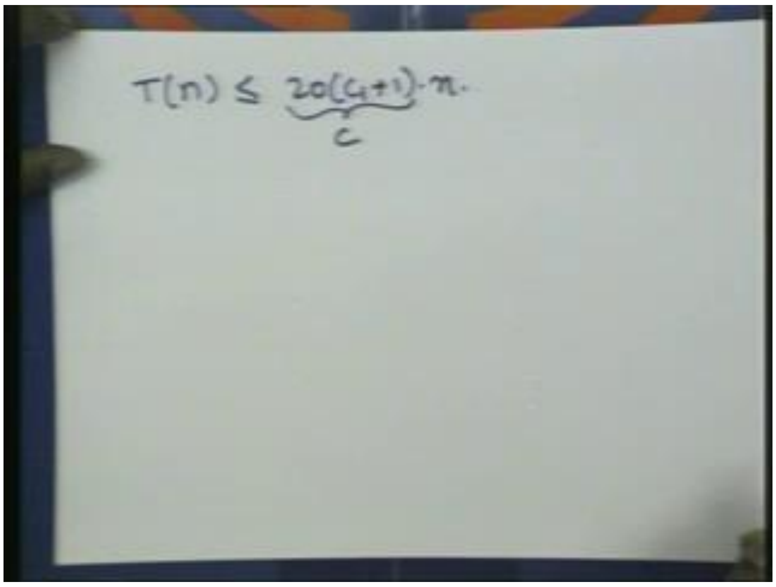
$$c \geq 20(C_1+1)$$

I know this T n is less than equal to C 1 plus 1 n plus T n by 5 plus T 3 n by 4. And I want T n is less than equal to C n. I want to find a right constant here C. So, let us just plug this into the original thing. I get T n is less than equal to C 1 plus 1 n plus well C n by 5 by plugging in C n plus 3 by 4 times C n. Now, if this is less than or equal to C n. If I can find C, so that this is less than equal to C n then we are done. If this is true, we are done.

Because, we can actually, prove this by induction. Even, if you do not understand why we are done at this point. We would like to find a C which satisfies this. So, let us find that. So, we want C 1 plus 1. n plus this plus this less than equal to C n. So, n cancels from these. So, I want C 1 plus 1 less than equal to C minus C by 5 minus 3 C by 4. We want C plus C 1 plus 1 utmost C minus C by 5 minus 3 C by 4. So, this will imply the previous statement.

And just simplifying this, this is nothing but C by 20. C times on by 20. So, as long as C is at least 20 times C 1 plus 1 we are done. So, if C is greater than equal to 20 times C 1 plus 1. We see that, this quantity is less than equal to C n. So, we can choose C to be 20 C 1 plus 1.

(Refer Slide Time: 49:32)



$$T(n) \leq \underbrace{20(c_1+1)}_{c} \cdot n.$$

So now, it looks like T n is less than equal to 20 times C 1 plus 1 times n. So, this is a constant. This is the constant C. This now, we can prove by induction on n, that T n is less than or equal to this C times n. We can prove now by induction of n. For n equals 2 it is true, because you make just one comparison. And for the inductive step for the inductive step we just observe that, T n by the recurrence is utmost C 1 plus 1 n plus T n by 5 plus 3 n by 4. And now I can use induction.

So, ((Refer Time: 50:23)) T n by 5 us utmost C n by 5. And this is utmost this much. And the way we have chosen C, this sum is utmost C n. So, the proof now follows by induction. In fact, initially we guessed that T n is utmost C times n. And the plan was to use induction. Somehow, use induction and prove this. And you just follow your nose in the induction substitute C. And find out ((Refer Time: 50:54)) what is the C that helps you finish the inductive ((Refer Time: 50:58)).

So, this is the other way of solving recurrence relations. If you can guess the answer somehow, then you know that you want to prove this by induction. So, try proving it by induction and come up with the right constants. And then you can once you have found it, once you plug it in and then find out what and figure out what constants to use then you can write the whole thing by induction. So, this proves that the time taken by our algorithm is in fact, linear in n.

So, you can find the median in time, which is linear in the size of the array. If you use this, use this algorithm with quick sort then the worst case time for quick sort would be n log n. But that is if you pick the pivot as a median each time. The normal way you do it of course, it turns out to be order n square.