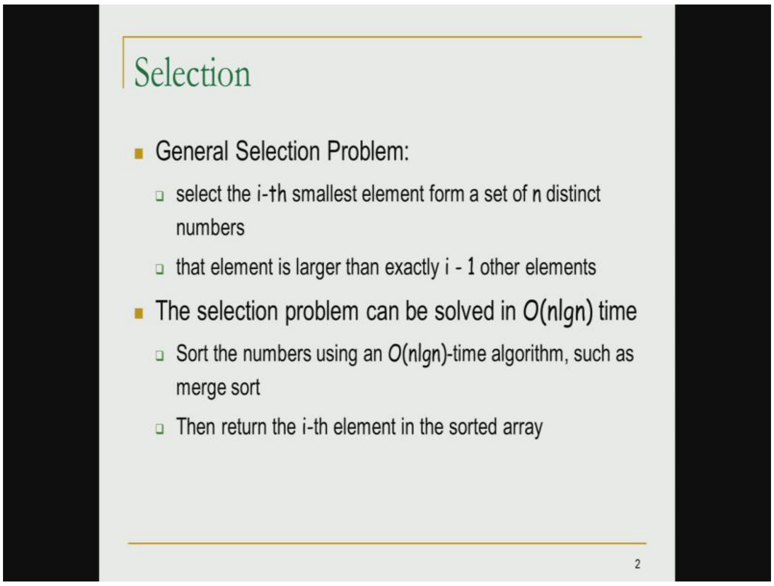
Programming, Data structures and Algorithms
Prof. N. S. Narayanaswamy
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

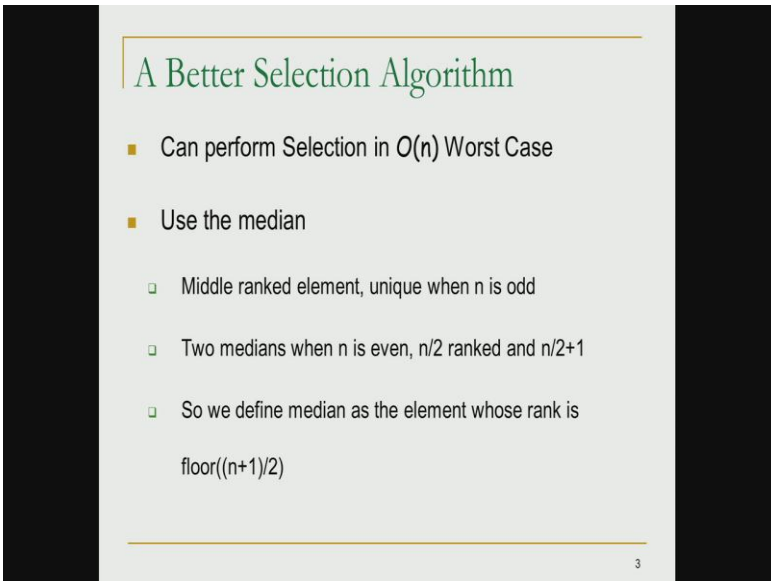
Lecture – 34
Algorithms
i-th Smallest
Order Statistics

(Refer Slide Time: 00:15)



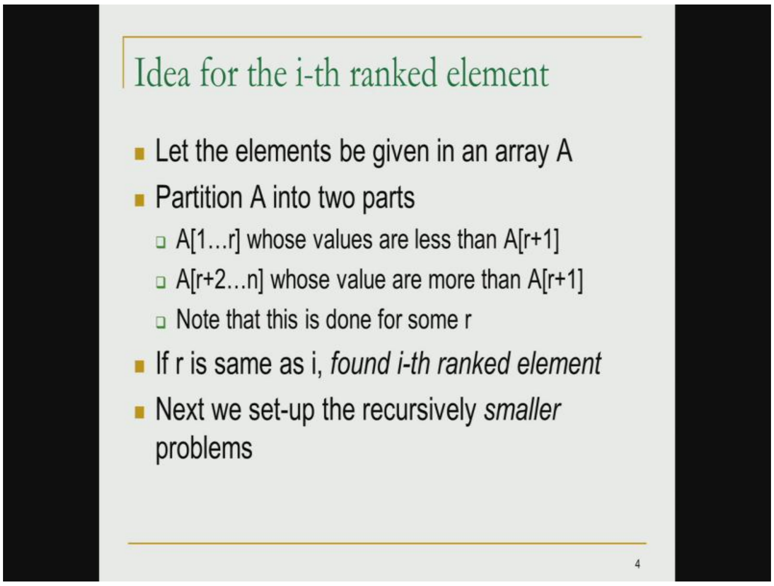
Today's lecture we are going to look at algorithms for finding the ith smallest number in a given data set. This speciality about the problem that we are going to look at is that is a recursive algorithm, and it is very efficient compare to the most simplest algorithm to answer this question. The problem and question is given n distinct numbers an imagine to be presented in an array, the goal of the problem is to find the ith smallest element which is also given us what are the input? The value i is also given us part of the input. In other words we want to find an array element, which is larger than exactly i minus 1 elements in the given datas. One natural approach is solve the question is to sort the given n distinct elements in ascending order, and 1 can use an algorithm like merge sort which is known to run in order of n log n time, and then we return the ith element in the sorted array.

(Refer Slide Time: 01:23)



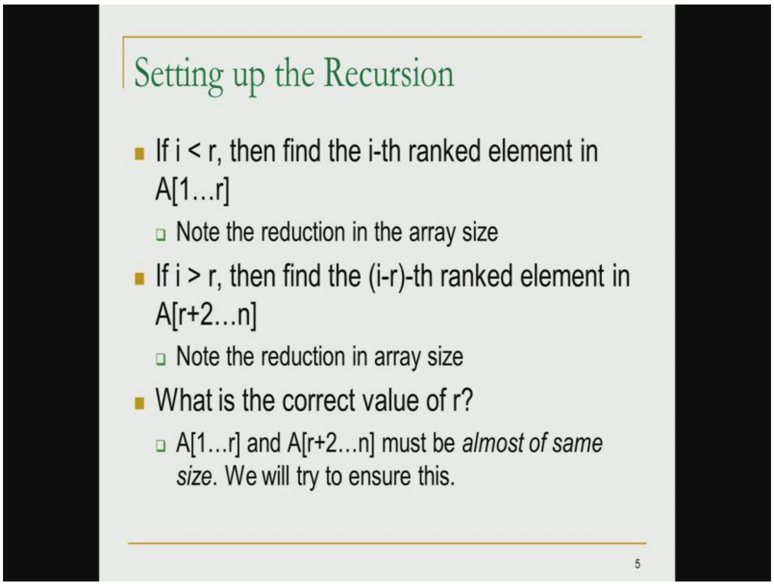
This definitely does solve the problem the focus of this lecture is to see if we can design better selection algorithms, in other words our aim is to design selection algorithms which run in time order of n in the worst case. The main idea behind the algorithm that we are going to look at is the concept of median. Let us just recall the definition of the median. Median is the middle rank element in a given set of numbers. If there are n distinct odd numbers in the dataset then the middle rank elements is the unique element. If n is indeed even then there are 2 medians; the elements ranked n by 2 and the elements ranked n 1 by 2. If n is odd - this is the single element, and if n is even - this is the n by 2 th ranked element.

(Refer Slide Time: 02:32)



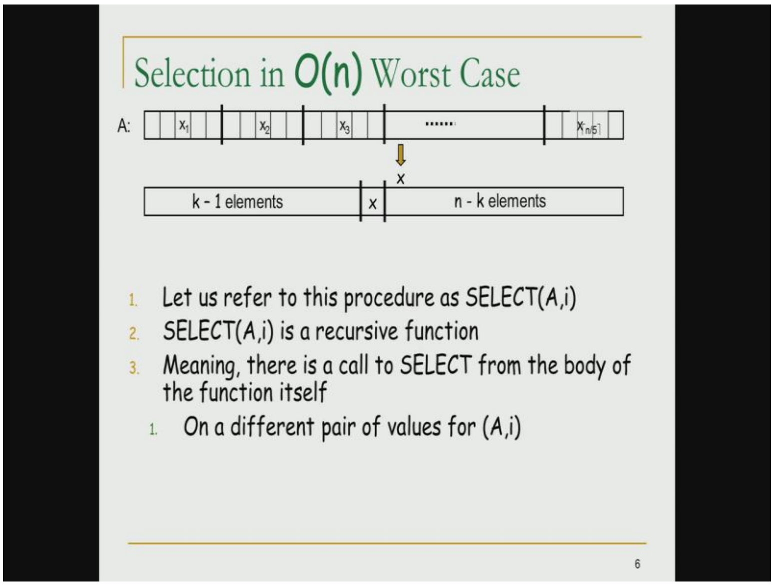
The idea for identifying the ith rank element is that we use the power of recursion, and given the elements in an array, we aim to partition the elements of the array into 2 parts, based on an index r, the elements a of 1 to a of r; that is the first r elements in the array r of values smaller than the r plus 1th element in the array. And the remaining elements that is the elements with the indices r plus 2 to n or of value more than a of r plus 1. Indeed if r has the value i then we have indeed found the ith ranked element. This is very clear, because the first i minus 1 elements are smaller than the ith element, and therefore a of r is the ith ranked element for r is equal to i. We are going to do this for some carefully chosen r, so that we can get our desired efficient algorithm. The way to do is this if r is not equal to i then we ensure that we have recursively smaller sub problems to solve. So, that we can get to the ith ranked element as quickly as possible.

(Refer Slide Time: 03:59)



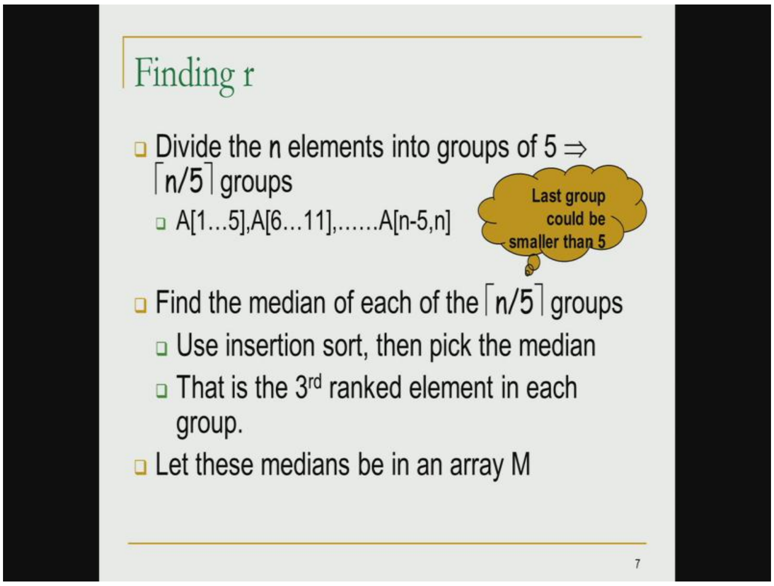
So, now we setup the recursion. If indeed i is smaller than r then we find the ith ranked element in the set a of 1 to a of r. This is indeed the most natural thing r is larger than i and all the elements below r, all the elements whose indices the range 1 to r in the array a or indeed smaller than r. And therefore, the ith ranked element would also have an index smaller than r, therefore its natural to search for the elements in the range a of 1 to a of r, the ith ranked element in the range a of 1 to a of r, i is more than r clear that we look for the i minus rth ranked element in the range a of r plus 2 to n. In either case it is clear that we have smaller recursive sub problems, but to get are efficient running time we will see that it is desirable to have the 2 parts in the partition to be of almost same size, and we will try to ensure this.

(Refer Slide Time: 05:17)



So, let us go to the design of the whole algorithm, we refer to this algorithm as a select procedure; the procedure has 2 arguments; ((Refer Time: 05:29)) array a, and a rank i. The output of this function is to return the value of the ith ranked element in the array a. Select A of i is the a of recursive function. Let us recall recursive function it is a function which makes calls to itself with of course, different parameters.

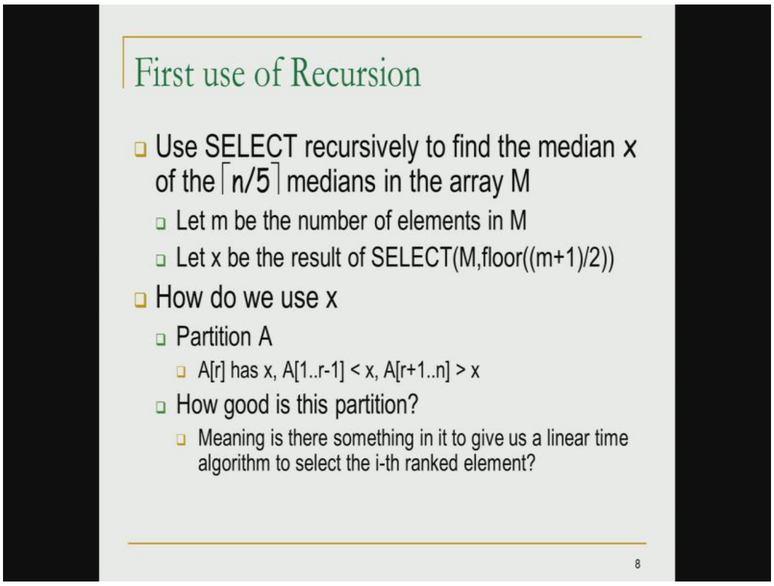
(Refer Slide Time: 05:56)



So, what we now do i, we try to partition, we come up with a procedure to partition the array based on an index r. To do this what we do is to take the n elements and divided into n by 5 groups, the n by 5 groups are the elements with index 1 to 5, the elements a of 6 to a of 11 and so on upto a of m minus 5 to a of n. It is important to notice here that the last group could be of values smaller than 5. For example, if n is not a multiple of 5, then clearly the last group will be smaller than 5. In each of these groups we find the median element, we call that an each group there are 5 elements except for the last one, for the purpose of this discussion, let us assumed with the last one also has 5 elements right.

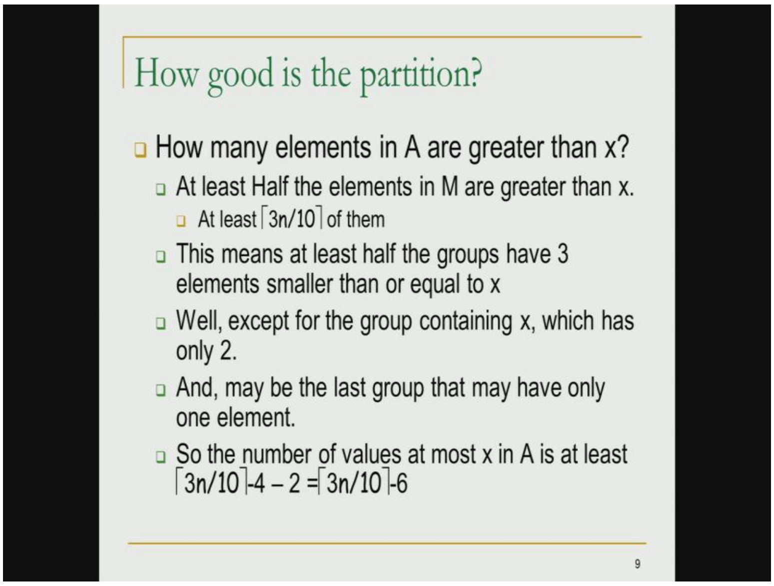
And we pick the median element each of these n by 5 groups, and for this we can use insertion sort and sort the 5 elements and pick the median. In other words this will be the third ranked element in each group. What we do is, we visualize these median elements from each of these groups in an array m.

(Refer Slide Time: 07:23)



Now, we here is the first use of recursion. We look at the array m and ask for the median of the n by 5 elements in the array M. So, the recursive call is described here, if there are m elements in the array, then the recursive call is select on the array m, the median element which we know is floor of m plus 1 by 2, let the returned value be x. Now what we do is the partition the array A into 2 parts around the element x. We will see how to do this, but before that let us also analyse how good this partition is. What do we mean by a good partition we analyse, what properties of this partition are there which could give us a linear time algorithm to find the ith ranked element.

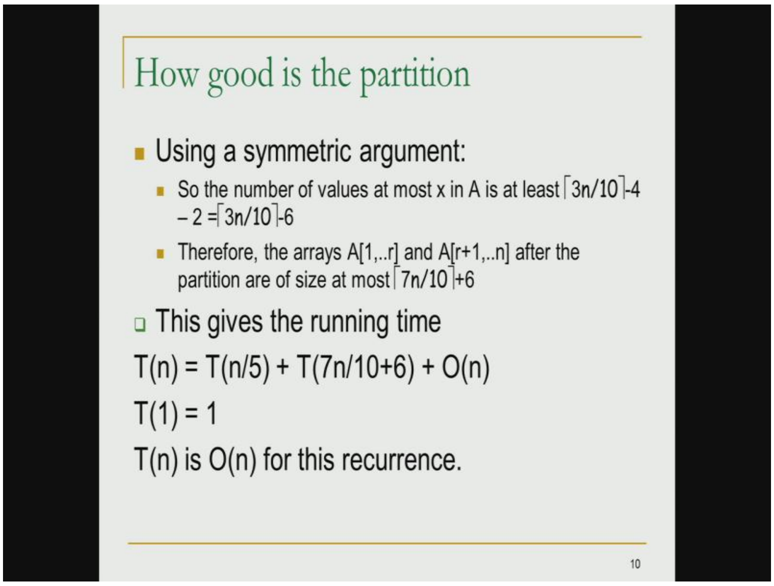
(Refer Slide Time: 08:18)



So, let us see how good this partition is, let us make some observation. Let us ask in the array A how many elements are greater than x. Recall that x comes from the array M and indeed it is a median element of the array M, M has n by 5 elements in it. Therefore, x has n by 10 elements larger than it; that is half the elements in the array m or larger than x, x being the median element in the array M. Let us also recall that ((Refer Time: 09:05)) each element in M is a median element in those n by 5 groups, and there are 3 elements which are at least as larger each element of M in the array A. I repeat this, for each element in the array M, there are definitely 3 elements in the array A, which are at least as larger as that particular element. Therefore, there will be at least 3 n by 10 elements, which are at least larger than x in A.

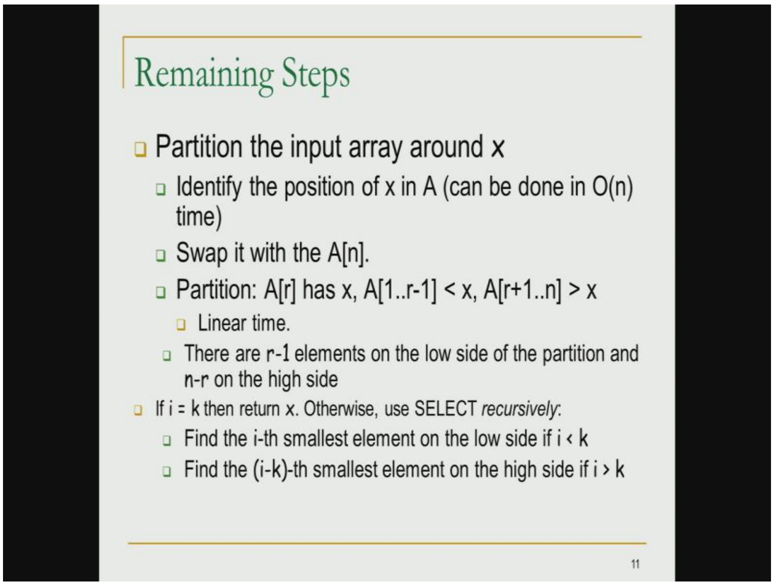
Therefore, at least half the groups have 3 elements smaller than or equal to x. This is except for the group containing x which has 2 elements, and the last group which may contain only 1 element. Therefore, the number of elements which are smaller than x and A is at least 3 n by 10 minus 6.

(Refer Slide Time: 09:57)



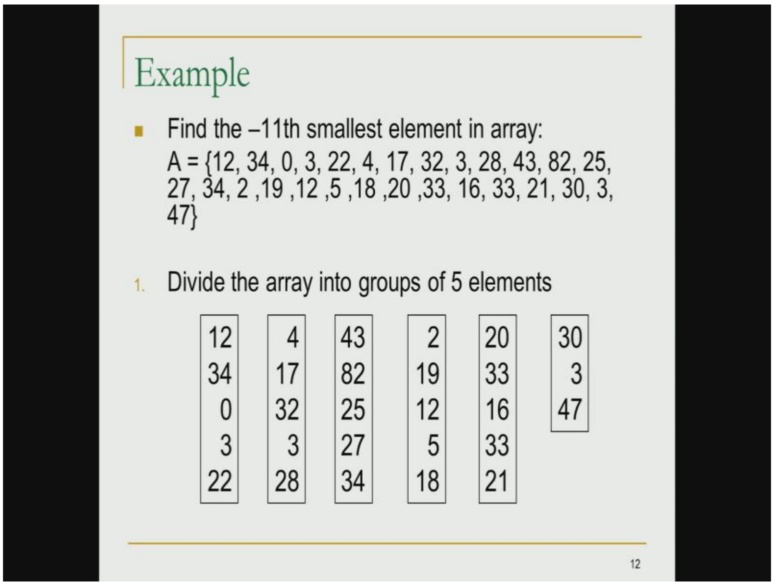
So, let us completes the discussion of how good this partition is using a symmetric argument, the number of values atmost x in A is also atleast 3 n by 10 minus 6, as a consequence of this argument it follows that the 2 arrays - A of 1 to A of r and A of r plus 1 to A of n, after the partition step have size at most 7 by 7 n by 10 plus 6, this is because the array has a total of n elements, and therefore the number of elements in the 2 partition 2 arrays A of 1 to A of r and A of r plus n to n is a tmost 7 n by 10 plus 6. Therefore, by making a recursive select call on one of these two arrays, we get a running time which is given here which is T of n the time taken to find the ith ranked element in the given array A is equal to T of n by 5 which is used to find a median element in the array M plus T of 7 n by 10 plus 6, which is the time taken to solve the recursive sub problems plus the time taken to create the partition, which is also a linear time procedure. Finally, if the array has a single element to find the ith ranked element is extremely easy right. So, it just is. So, T of 1 is taken to be i. It is easy to see the T of n is order of n for the recurrence, we do not evaluate the recurrence here, but this is a recurrence which evaluates to order of n.

(Refer Slide Time: 11:38)



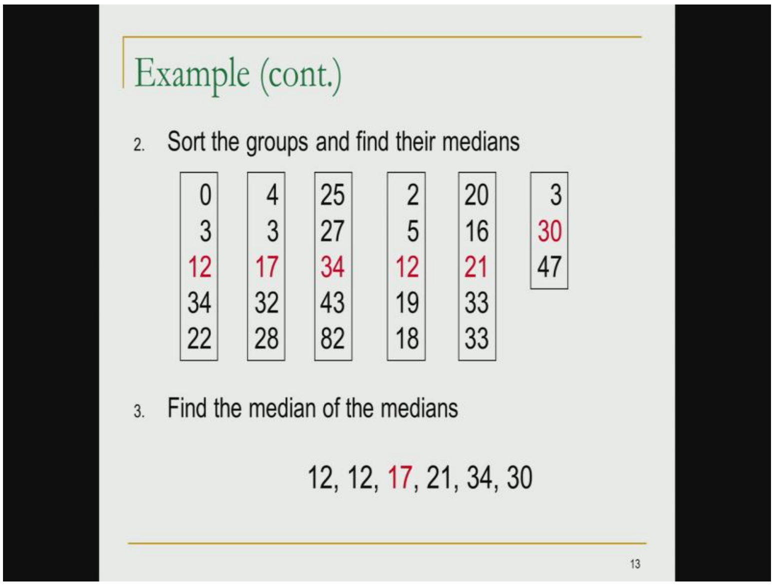
What are the remaining steps of the algorithm? We partition the input array around the element x to do this the following steps have to be done, we identify the position of x in the array A. This can be done an order of n time by scanning the element, scanning the array A for the element x. And then we perform a partition procedure linear time partition procedure which identifies an index r, such that A of r as x and the elements 1 to r minus 1, the elements in the indices 1 to r minus 1 smaller than x, and the element in the indices 1 to r plus 1 smaller than x, and the elements in the indices r plus 1 to n r more than x. This is exactly what we want? There are r minus 1 elements on the low side of the partition, and n minus r elements in the high side of the partition. So, the algorithm is almost complete now, if i is equal to k then we return the value x, otherwise we use select recursively and the recursive calls are made as follows, we find the ith smallest element in the low side, if i is smaller than r and we find the i minus r smallest element in the i side, if i is more than r.

(Refer Slide Time: 13:00)



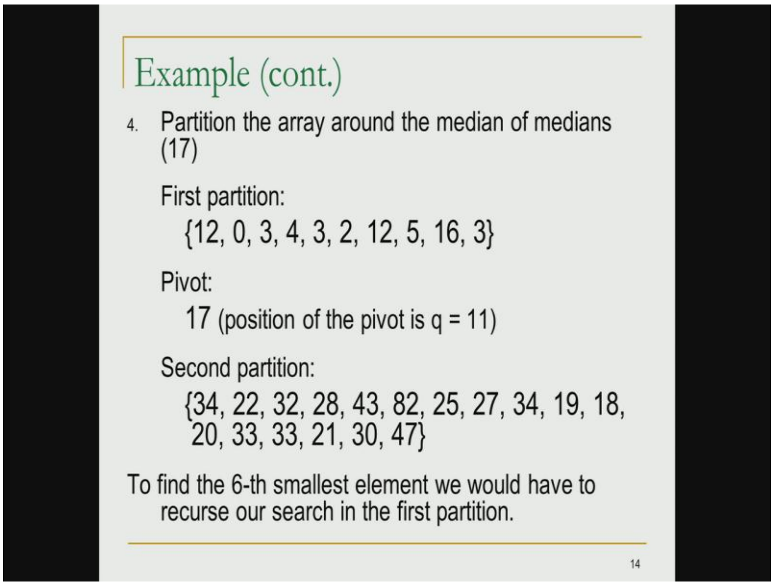
Let us run through an example of this algorithm. In the given array of 28 elements, we want to find the eleventh ranked element. In the first step we divide the array into 5 groups of 5 elements each which may counts for 25 elements, and 1 group the last group of 3 elements.

(Refer Slide Time: 13:22)



We sort that groups and then find the median element in the 6 groups, and the median elements are marked in red. Observe that in the group of size 5, the median element is the third element, and the last group the median element is the second element. In the third step we find the median of the medians, and this is our array A, this is our array M and one can see that there are 6 elements and the median element is the elements 17.

(Refer Slide Time: 14:02)



We partition the array around 17. And in the first part we have all the elements which are smaller than 17, and 17 is the eleventh ranked element, and the algorithm terminates and returns the value 17 as the eleventh ranked element. If for example, we want to find the 6 th smallest element, then we would after recurse our search in the first part which is the value which are smaller than 17. And similarly if we want to find an element of rank 15, then we would have to find the element of the rank 4, in the second partition. This completes the description of the algorithm to find the highest smallest element in a given array A.