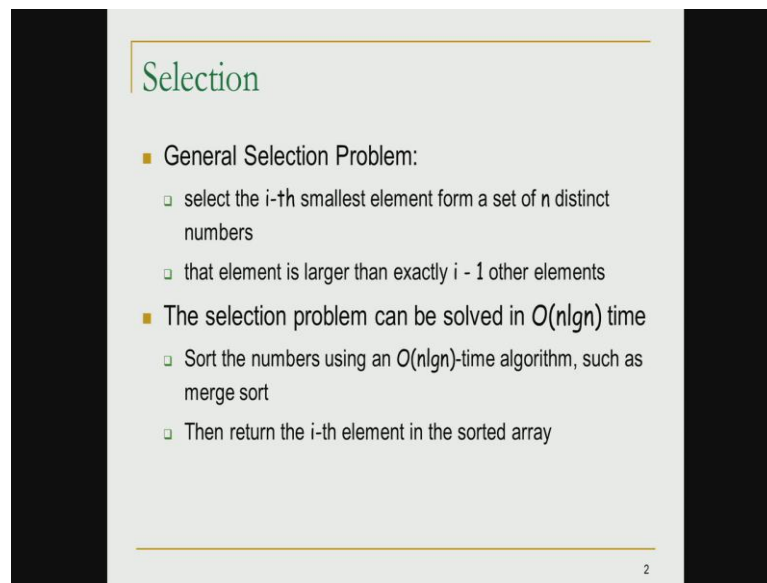


Programming, Data structures and Algorithms
Prof. N. S. Narayanaswamy
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 34
Algorithms
i-th Smallest
Order Statistics

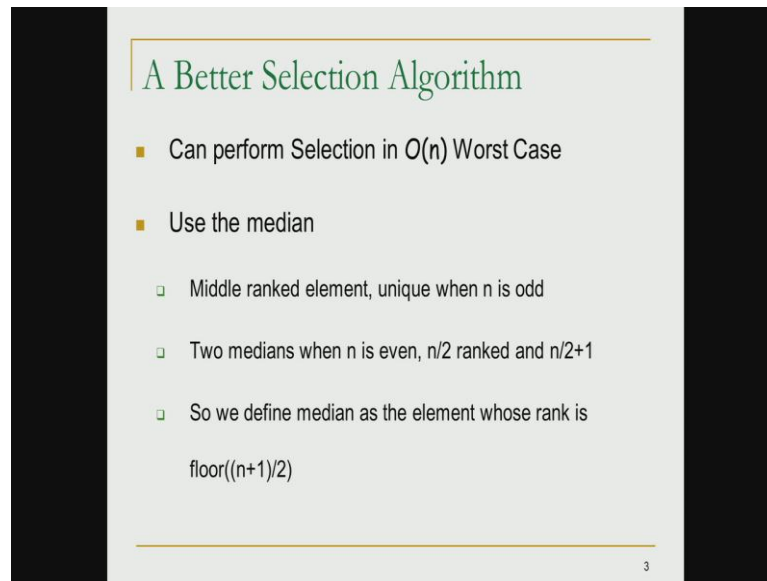
(Refer Slide Time: 00:15)



The slide is titled "Selection" in green text. It contains two main bullet points, each with sub-bullets. The first bullet point is "General Selection Problem:" followed by two sub-bullets: "select the i-th smallest element from a set of n distinct numbers" and "that element is larger than exactly i - 1 other elements". The second bullet point is "The selection problem can be solved in $O(n \lg n)$ time" followed by two sub-bullets: "Sort the numbers using an $O(n \lg n)$ -time algorithm, such as merge sort" and "Then return the i-th element in the sorted array". A small number "2" is visible in the bottom right corner of the slide.

Today's lecture we are going to look at algorithms for finding the i th 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 i th 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 i th element in the sorted array.

(Refer Slide Time: 01:23)



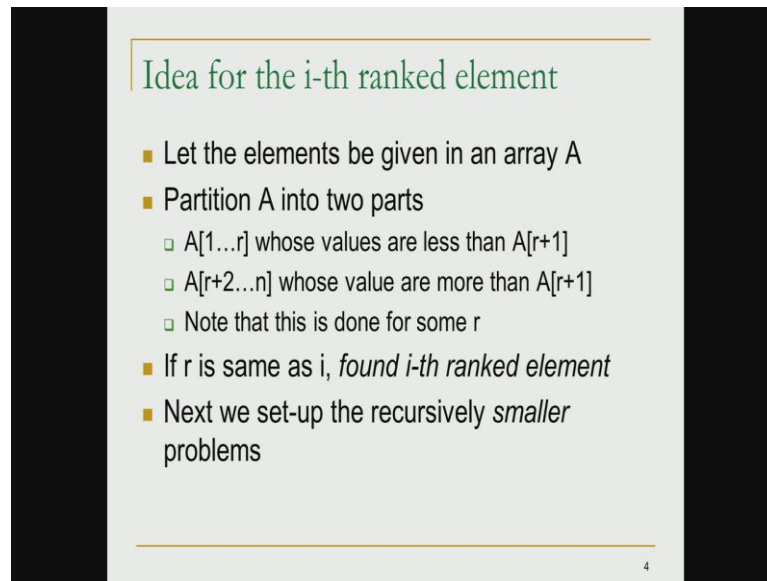
A Better Selection Algorithm

- Can perform Selection in $O(n)$ Worst Case
- Use the median
 - Middle ranked element, unique when n is odd
 - Two medians when n is even, $n/2$ ranked and $n/2+1$
 - So we define median as the element whose rank is $\text{floor}((n+1)/2)$

3

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 2th ranked element.

(Refer Slide Time: 02:32)



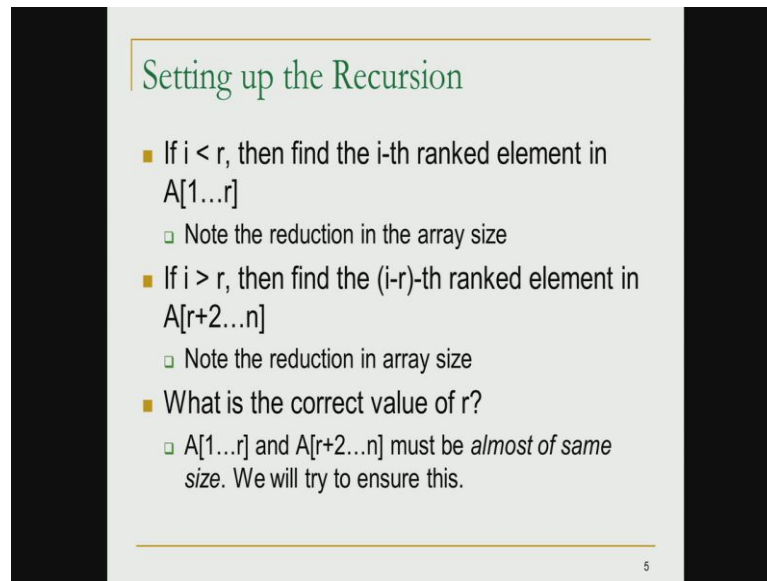
Idea for the i -th ranked element

- Let the elements be given in an array A
- Partition A into two parts
 - $A[1\dots r]$ whose values are less than $A[r+1]$
 - $A[r+2\dots n]$ whose value are more than $A[r+1]$
 - Note that this is done for some r
- If r is same as i , *found i -th ranked element*
- Next we set-up the recursively *smaller* problems

4

The idea for identifying the i th 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 i th ranked element. This is very clear, because the first i minus 1 elements are smaller than the i th element, and therefore a of r is the i th 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 i th ranked element as quickly as possible.

(Refer Slide Time: 03:59)



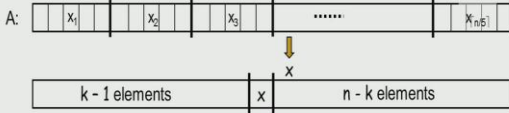
The slide is titled "Setting up the Recursion" in green text. It contains three main bullet points, each with a sub-bullet point. The first bullet point is "If $i < r$, then find the i -th ranked element in $A[1\dots r]$ ", with a sub-bullet "Note the reduction in the array size". The second bullet point is "If $i > r$, then find the $(i-r)$ -th ranked element in $A[r+2\dots n]$ ", with a sub-bullet "Note the reduction in array size". The third bullet point is "What is the correct value of r ?", with a sub-bullet "A[1...r] and A[r+2...n] must be *almost of same size*. We will try to ensure this." A small number "5" is in the bottom right corner of the slide.

- If $i < r$, then find the i -th ranked element in $A[1\dots r]$
 - Note the reduction in the array size
- If $i > r$, then find the $(i-r)$ -th ranked element in $A[r+2\dots n]$
 - Note the reduction in array size
- What is the correct value of r ?
 - A[1...r] and A[r+2...n] must be *almost of same size*. We will try to ensure this.

So, now we setup the recursion. If indeed i is smaller than r then we find the i th 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 i th 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 i th 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 r th 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)

Selection in $O(n)$ Worst Case

A: 

1. Let us refer to this procedure as $SELECT(A,i)$
2. $SELECT(A,i)$ is a recursive function
3. Meaning, there is a call to $SELECT$ from the body of the function itself
 1. On a different pair of values for (A,i)

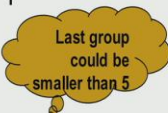
6

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 i th ranked element in the array a . $SELECT(A,i)$ 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)

Finding r

- Divide the n elements into groups of 5 \Rightarrow $\lceil n/5 \rceil$ groups
 - $A[1...5], A[6...11], \dots, A[n-5, n]$
- Find the median of each of the $\lceil n/5 \rceil$ groups
 - Use insertion sort, then pick the median
 - That is the 3rd ranked element in each group.
- Let these medians be in an array M

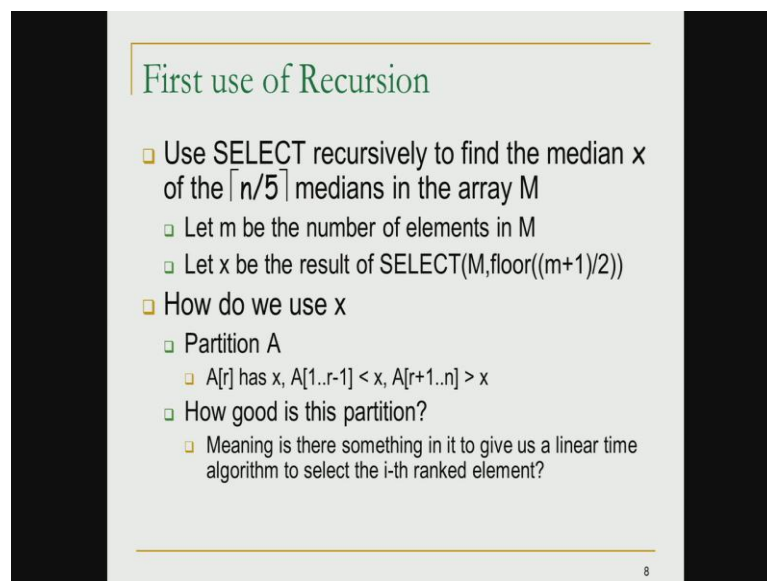


7

So, what we now do is, 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)



First use of Recursion

- Use SELECT recursively to find the median x of the $\lceil n/5 \rceil$ medians in the array M
 - Let m be the number of elements in M
 - Let x be the result of $\text{SELECT}(M, \text{floor}((m+1)/2))$
- How do we use x
 - Partition A
 - $A[r]$ has x , $A[1..r-1] < x$, $A[r+1..n] > x$
 - How good is this partition?
 - Meaning is there something in it to give us a linear time algorithm to select the i -th ranked element?

8

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 i th ranked element.

(Refer Slide Time: 08:18)

How good is the partition?

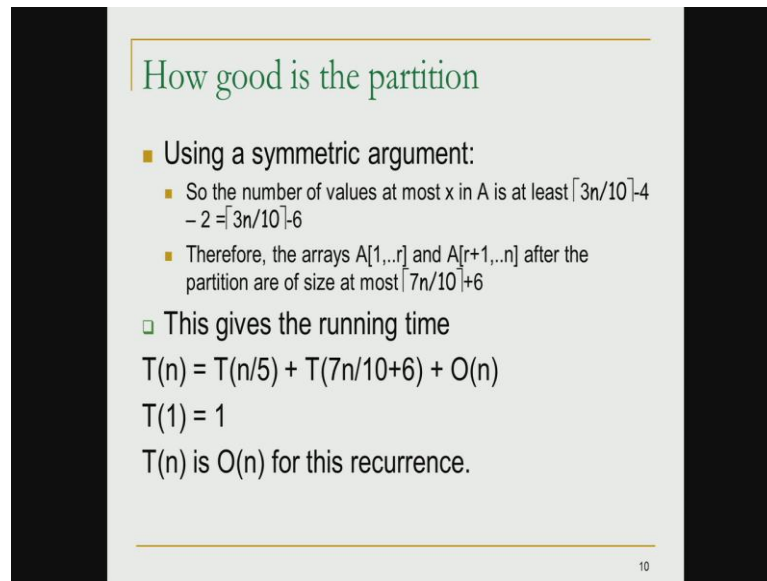
- How many elements in A are greater than x?
 - At least Half the elements in M are greater than x.
 - At least $\lceil 3n/10 \rceil$ of them
 - This means at least half the groups have 3 elements smaller than or equal to x
 - Well, except for the group containing x, which has only 2.
 - And, may be the last group that may have only one element.
 - So the number of values at most x in A is at least $\lceil 3n/10 \rceil - 4 - 2 = \lceil 3n/10 \rceil - 6$

9

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)



How good is the partition

- Using a symmetric argument:
 - So the number of values at most x in A is at least $\lceil 3n/10 \rceil - 4 - 2 = \lceil 3n/10 \rceil - 6$
 - Therefore, the arrays $A[1..r]$ and $A[r+1..n]$ after the partition are of size at most $\lceil 7n/10 \rceil + 6$
- This gives the running time

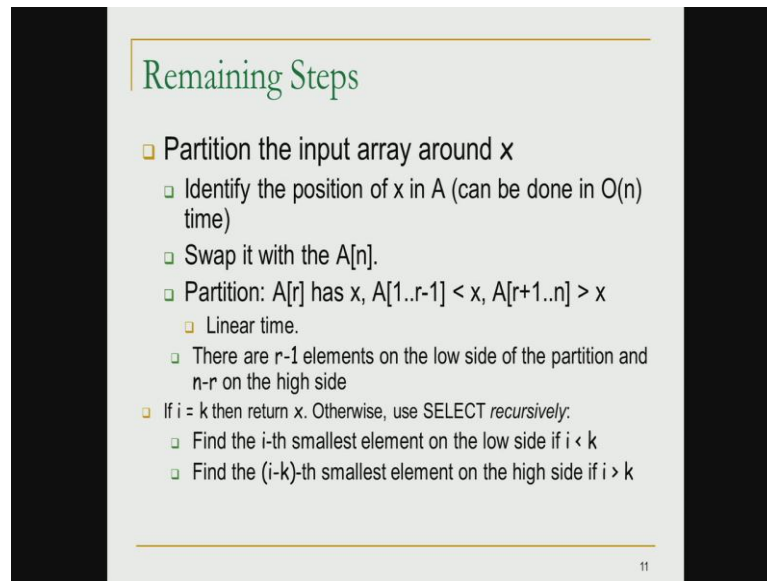
$$T(n) = T(n/5) + T(7n/10+6) + O(n)$$
$$T(1) = 1$$

$T(n)$ is $O(n)$ for this recurrence.

10

So, let us complete the discussion of how good this partition is using a symmetric argument, the number of values at most x in A is also at least $3n/10$ minus 6, as a consequence of this argument it follows that the 2 arrays - A of 1 to A of r and A of $r+1$ to A of n , after the partition step have size at most $7n/10 + 6$, this is because the array has a total of n elements, and therefore the number of elements in the 2 partitioned arrays A of 1 to A of r and A of $r+1$ to n is at most $7n/10 + 6$. 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 i th ranked element in the given array A is equal to T of $n/5$ which is used to find a median element in the array M plus T of $7n/10 + 6$, 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 i th ranked element is extremely easy right. So, it just is. So, T of 1 is taken to be 1. 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)



The slide is titled "Remaining Steps" in green text. It contains a list of steps for partitioning an array around an element x. The steps are as follows:

- Partition the input array around x
 - Identify the position of x in A (can be done in $O(n)$ time)
 - Swap it with the $A[r]$.
 - Partition: $A[r]$ has x, $A[1..r-1] < x$, $A[r+1..n] > x$
 - Linear time.
 - There are $r-1$ elements on the low side of the partition and $n-r$ on the high side
 - If $i = k$ then return x. Otherwise, use SELECT *recursively*:
 - Find the i -th smallest element on the low side if $i < k$
 - Find the $(i-k)$ -th smallest element on the high side if $i > k$

At the bottom right of the slide, there is a small number "11".

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 i th 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)

Example

- Find the -11th smallest element in array:
 $A = \{12, 34, 0, 3, 22, 4, 17, 32, 3, 28, 43, 82, 25, 27, 34, 2, 19, 12, 5, 18, 20, 33, 16, 33, 21, 30, 3, 47\}$

1. Divide the array into groups of 5 elements

| | | | | | |
|----|----|----|----|----|----|
| 12 | 4 | 43 | 2 | 20 | 30 |
| 34 | 17 | 82 | 19 | 33 | 3 |
| 0 | 32 | 25 | 12 | 16 | 47 |
| 3 | 3 | 27 | 5 | 33 | |
| 22 | 28 | 34 | 18 | 21 | |

12

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)

Example (cont.)

2. Sort the groups and find their medians

| | | | | | |
|----|----|----|----|----|----|
| 0 | 4 | 25 | 2 | 20 | 3 |
| 3 | 3 | 27 | 5 | 16 | 30 |
| 12 | 17 | 34 | 12 | 21 | 47 |
| 34 | 32 | 43 | 19 | 33 | |
| 22 | 28 | 82 | 18 | 33 | |

3. Find the median of the medians

12, 12, 17, 21, 34, 30

13

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)

Example (cont.)

- Partition the array around the median of medians (17)

First partition:
{12, 0, 3, 4, 3, 2, 12, 5, 16, 3}

Pivot:
17 (position of the pivot is $q = 11$)

Second partition:
{34, 22, 32, 28, 43, 82, 25, 27, 34, 19, 18, 20, 33, 33, 21, 30, 47}

To find the 6-th smallest element we would have to recurse our search in the first partition.

14

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 6th 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.