

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

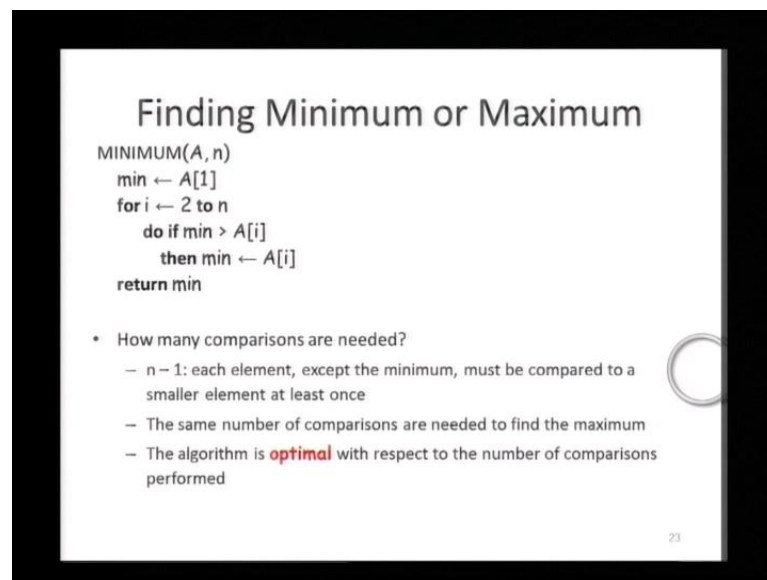
Module - 4

Lecture - 31

Content

Finding minimum and maximum in an array
Simultaneous min and max in an array: analysis
Example of simultaneous min and max

(Refer Slide Time: 00:06)



Finding Minimum or Maximum

```
MINIMUM(A, n)
min ← A[1]
for i ← 2 to n
  do if min > A[i]
    then min ← A[i]
return min
```

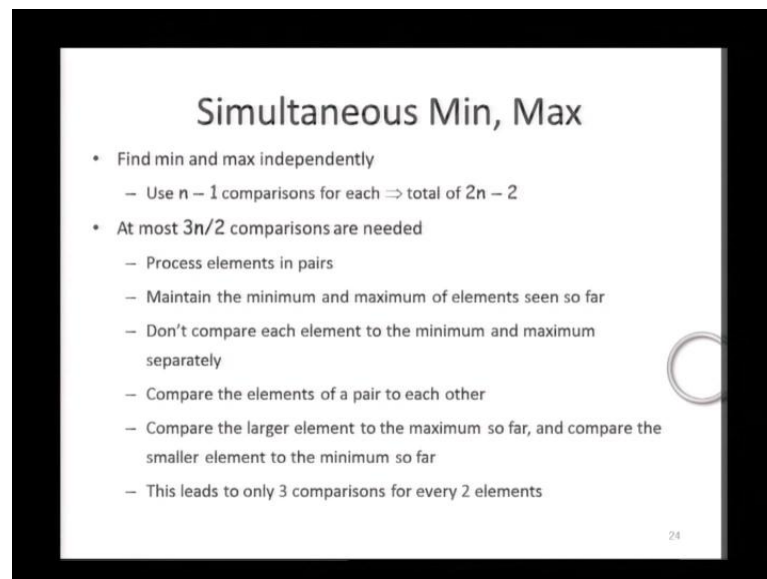
- How many comparisons are needed?
 - $n - 1$: each element, except the minimum, must be compared to a smaller element at least once
 - The same number of comparisons are needed to find the maximum
 - The algorithm is **optimal** with respect to the number of comparisons performed

23

And compare the current minimum to the new data; that is entered into the sub problem. In other words in the second iteration, we consider the second element and compare it with the minimum, which was the first element. If indeed the second element is smaller than the current minimum, then it indeed becomes the minimum among the array, which consist of the elements a of 1 and a of 2. So, this is an iterative procedure, where one scans the array from the element index by the smallest value up to the element index by the largest value in this case 1 to n , and start of with the, an estimate of the minimum value to be the first element and update this estimate by a comparison, by one comparison in each iteration. It is very clear that we will have to perform n minus 1 comparisons. It is not possible to perform anything less n minus 1 comparisons in this approach, unless we know something else about the array. Therefore, these algorithms

that we have just discussed, is an optimal algorithm with respect to the number of comparisons. The number of comparisons is you need $n - 1$ comparison in the algorithm above, performs exactly $n - 1$ comparison; therefore, this is an optimal algorithm.

(Refer Slide Time: 01:27)



Simultaneous Min, Max

- Find min and max independently
 - Use $n - 1$ comparisons for each \Rightarrow total of $2n - 2$
- At most $3n/2$ comparisons are needed
 - Process elements in pairs
 - Maintain the minimum and maximum of elements seen so far
 - Don't compare each element to the minimum and maximum separately
 - Compare the elements of a pair to each other
 - Compare the larger element to the maximum so far, and compare the smaller element to the minimum so far
 - This leads to only 3 comparisons for every 2 elements

24

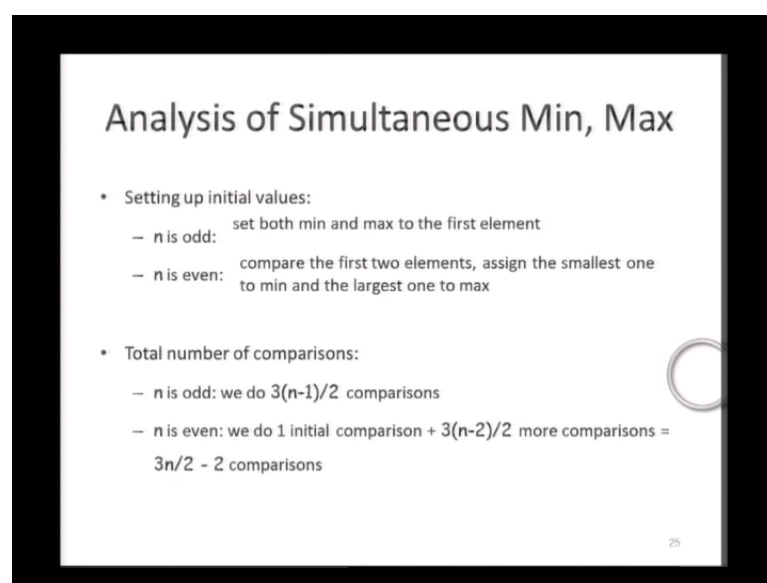
it is the very interesting exercise to ask, if one thing compute, both the minimum and the maximum elements in the array by efficiently, but more importantly by in a simultaneous fashion; that is we want the algorithm at every point of time to keep estimates of both the min and max in the array, and update both this estimates, and finally, conclude that min and max of min found. Here is one way of doing it. Use $n - 1$ comparison for each of thing that is can the elements from the first element to the last element.

Assume that the first element is both the minimum and the maximum, and compare the current minimum and the maximum in every iteration with, the current element. This is two comparisons for element, there are $n - 1$ elements that there are compared against the minimum and the maximum values, and therefore, refer from $2n - 2$ comparisons. We can interested in coming up with a better algorithm which uses strictly smaller than $2n - 2$ comparisons, and we presents an algorithm which uses at most $3n/2$ comparisons. So, let us just understand how this can be done. In this straight

forward approach, which is listed in the first item. We have estimates of min and max and we compare both min and max, with the next element in the array; that is in the i th iteration we compare the i th element with both min and max, to check if the i th element replaces the minimum value or the maximum value. Of course, one can use the fact that, we have computed two values which are min and max, and if we compare min and max with a pair of values, then min is to be compared only with the smaller of the two, and max needs to be compared only with the larger of the two, and therefore, you can compute the min and max among these four elements with 3 comparisons. This is the whole idea.

So, the whole idea is, to process the elements in the array in pairs and maintain the minimum and the maximum values, in each iteration that has been calculated so far, and compare the minimum element with the smaller element of a pair, and compare the maximum to the maximum element in the pair. Observe that the inner pair; the smaller element can be identified with one comparison, and after that we count only two comparisons therefore, among four elements we are able to compute the minimum and maximum, using just 3 comparisons, and this is the trick that we generalise, to reduce the total number of comparisons .

(Refer Slide Time: 04:48)



Analysis of Simultaneous Min, Max

- Setting up initial values:
 - n is odd: set both min and max to the first element
 - n is even: compare the first two elements, assign the smallest one to min and the largest one to max
- Total number of comparisons:
 - n is odd: we do $3(n-1)/2$ comparisons
 - n is even: we do 1 initial comparison + $3(n-2)/2$ more comparisons = $3n/2 - 2$ comparisons

25

So, here is the algorithm which is described. Initially the values, when n is odd \min and \max are taken to be the first element, and when n is even \min and \max are taken to be the first two elements. This takes one comparison; \min is taken to be the smaller of the first two elements, and \max is taken to be the larger of the first two elements. This is done at the cost of one comparison.

(Refer Slide Time: 05:19)

Example

- $n = 5$ (odd), array $A = \{2, 7, 1, 3, 4\}$
 - Set $\min = \max = 2$
 - Compare elements in pairs:
 - $1 < 7$; compare 1 with \min and 7 with \max
 - Now $\min = 1$ and $\max = 7$, 3 comparisons used
 - $3 < 4$; compare 3 with \min and 4 with \max
 - Now $\min = 1$, $\max = 7$, 3 more comparisons

$6 \text{ comparisons} = 3(n-1)/2$

26

And then based on this algorithm that we have outlined, there will be a total of n by 2 comparisons, or if n is odd, there will be a n minus 1 by 2 comparisons. And if n is even, there will be n minus 2 by 2 comparisons. Comparison pairs and each of them requires 3 comparisons to identify or update the minimum and maximum. Therefore, the total numbers of comparisons which are made, are 3 times n minus 1 by 2 plus the first comparison and 3 times n minus 2 by 2 plus the first comparison. This is the total number of comparisons which are made. So, this is the whole idea. Let us run it on a single example, where there are five elements in the array. The array has elements 2 7 1 3 and 4, and we just illustrate how this simultaneous \min and \max calculation happens. Initially, because n is odd \min and \max are taken to be 2.

Then we compare the elements, by considering the pairs one and 7, and the pair 3 and 4. In one iteration, we consider the pair one and 7; one is smaller than 7, this involves one

comparison. And after that one is compared with a current minimum, and 7 is compared with the current maximum. As you can see this is sufficient for us, to very easily extract the minimum and maximum among the elements 2 7 and 1. Now min and max are updated to be one and 7 respectively. They are different from the earlier estimates, which was both 2. And we have now use 3, addition 3 comparisons. Then 3 and four are brought into the whole exercise. Now 3 is compared with the current minimum, because it is a smaller of the 2, and four is compared with the current maximum, because it is larger of 3 and four. Already one comparison is used to identify which of 3 and four is smaller; therefore, we use three more comparisons, and total of six comparisons are used which is three times n minus 1 by 2 comparisons

(Refer Slide Time: 07:36)

Example

- $n = 6$ (even), array $A = \{2, 5, 3, 7, 1, 4\}$
 - Compare 2 with 5: $2 < 5$, Set **min** = 2, **max** = 5
 - 1 comparison
 - $3 < 7$; compare 3 with **min** and 7 with **max**
 - **min** = 2, **max** = 7, 3 comparisons
 - $1 < 4$; compare 1 with **min** and 4 with **max**
 - **min** = 1, **max** = 7, 3 comparisons

Total of $3n/2 - 2$ comparisons, 7 in this case

Similarly, when n is even, we use one more comparison to identify this smaller of the first two elements, and subsequently we have three times n minus 2 by 2 comparison. In this case it is very easy to see that there are 7 comparisons that have been made. So, this is the whole idea, let us run it on a single example, where there are five elements in the array. The array has elements 2 7 1 3 and 4, and we just illustrate have this simultaneous min and max calculation happens. Initially because n is odd, min and max are taken to be two then... We compare the elements by considering the pairs one and 7, and the pair three and four. In one iteration, we consider the pair 1 and 7. One is smaller than 7. This

involves one comparison, and after that one is compared with current minimum and 7 is compared with the current maximum. As you can see this is sufficient for us, to very easily extract the minimum and maximum, among the elements 2 7 and 1. Now min and max are updated to be 1 and 7 respectively. They are different from the earlier estimate, which was both two, and we have now use three addition, three comparisons. Then three and four are brought into the whole exercise.

Now there is compared with the current minimum, because it is a smaller of the two, and four is compared with the current maximum, because it is large of three and four. Already one comparison is used up to identify which of three and four is smaller; therefore, we use three more comparisons, and total of six comparisons are use, which is three times n minus 1 by 2 comparisons. Similarly when n is even, we use one more comparison to identify this smaller of the first two elements, and subsequently we have three times n minus 2 by 2 comparisons. In this case it is very easy to see that there are 7 comparisons that have been made.