Find the minimum and maximum value in an array

Understanding the Problem

Problem Description: Given an array A[] of size n, you need to find the maximum and minimum element present in the array. Your algorithm should make the minimum number of comparisons.

For Example:

Input: A[] = { 4, 2, 0, 8, 20, 9, 2}

Output: Maximum: 20, Minimum: 0

Input: A[] = {-8, -3, -10, -32, -1}

Output: Maximum: -1, Minimum: -32

Possible follow-up questions to ask the interviewer :-

1. Are the array elements necessarily positive? (**Ans:** No, they can be positive, negative, or zero)
2. Are the array element sorted ? (Ans: No, they can be in any order)
3. Can the array contain duplicates? (**Ans:** Sure, that's a possibility.)

Problem Note:- The interviewer would not judge your algorithm for this question based on the time complexity as all solution has the time complexity of O(n). The bottleneck parameter in this problem is the number of comparisons that your algorithm requires to determine the maximum and the minimum element. You need to decrease the number of comparisons as much as you can.

Solutions

1. Searching linearly: Increment the loop by 1
2. Divide and Conquer: Tournament Method
3. Comparison in pairs: Increment the loop by 2

1. Searching linearly: Increment the loop by 1

We initialize both minimum and maximum element to the first element and then traverse the array, comparing each element and update minimum and maximum whenever necessary.

Pseudo-Code

```
int[] getMinMax(int A[], int n)
{
    int max = A[0]
    int min = A[0]
    for ( i = 1 to n-1 )
    {
        if ( A[i] > max )
            max = A[i]
        else if ( A[i] < min )
            min = A[i]
    }
    // By convention, let ans[0] = maximum and ans[1] = minimum
    int ans[2] = {max, min}
    return ans
}
```

Complexity Analysis

At every step of the loop, we are doing 2 comparisons in the worst case. Total no. of comparisons (in worst case) = 2*(n-1) = 2n - 2

Time complexity = O(n), Space complexity = O(1)

In the best case, a total of n-1 comparisons have been made. (**How?**)

Critical ideas to think!

- We have initialized maximum and minimum with the first element of the array - why?
- What would be the best case and worst case input?
- How can we decrease the number of comparisons made here?

2. Divide and Conquer : Tournament Method

Another way to do this could be by following the divide and conquer strategy. Just like the merge sort, we could divide the array into two equal parts and recursively find the maximum and minimum of those parts. After this, compare the maximum and minimum of those parts to get the maximum and minimum of the whole array.

Solution Steps

1. Write a recursive function accepting the array and its start and end index as parameters
2. The base cases will be

- If array size is 1, return the element as both max and min
- If array size is 2, compare the two elements and return maximum and minimum

3. The recursive part is

- Recursively calculate and store the maximum and minimum for left and right parts
- Determine the maximum and minimum among these by 2 comparisons

4. Return max and min.

Pseudo Code

```
int[] findMinMax(int A[], int start, int end)
{
    int max;
    int min;
    if ( start == end )
    {
        max = A[start]
        min = A[start]
    }
    else if ( start + 1 == end )
    {
        if ( A[start] < A[end] )
        {
            max = A[end]
            min = A[start]
        }
        else
        {
            max = A[start]
            min = A[end]
        }
    }
    else
    {
        int mid = start + (end - start)/2
        int left[] = findMinMax(A, start, mid)
        int right[] = findMinMax(A, mid+1, end)
        if ( left[0] > right[0] )
            max = left[0]
        else
            max = right[0]
        if ( left[1] < right[1] )
            min = left[1]
        else
            min = right[1]
    }
    // By convention, we assume ans[0] as max and ans[1] as min
    int ans[2] = {max, min}
   return ans
}
```

Complexity Analysis

For counting the number of comparisons, since this is a recursive function, let us define the recurrence relation :

$T(n) = 2\ T(n/2) + 2$

$T(2) = 1$

$T(1) = 0$

We can solve **this** recurrence relation by master method/recursion tree method.

if n is a power of 2
$T(n) = 3n/2 - 2$
Time complexity = O(n) and space complexity = O(logn) (For recursion call stack)

If n is a power of 2, the algorithm needs exactly 3n/2–2 comparisons to find min and max. If it's not a power of 2, it will take a few more(not significant).

Critical ideas to think!

- How do we analyze the recursion by the master's theorem and recursion tree method?
- How is the space complexity derived to be O(logn)?
- Why there are 2 base cases? What if we remove the base case with array size 2?
- Why prefer mid = start + (end - start)/2 over (start + end)/2 when calculating middle of the array ?
- Can the number of comparisons be decreased further?

3. Comparison in Pairs : Increment the loop by 2

In this approach, we pick array elements in pairs and update the min and max. If the array size is odd, we initialize the first element as both min and max, and if it's even, we compare the first two elements and initialize min and max accordingly.

Solution Steps

1. Create max and min variables.
2. Check for the size of the array

- If odd, initialize min and max to the first element
- If even, compare the elements and set min to the smaller value and max to the bigger value

3. Traverse the array in pairs

4. For each pair, compare the two elements and then

- Compare the bigger element with max, update max if required.
- Compare the smaller element with min, update min if required.

5. Return max and min.

Pseudo Code

```
int[] findMinMax(int A[], int n)
{
   int max, min
   int i
   if ( n is odd )
   {
      max = A[0]
      min = A[0]
      i = 1
   }
   else
   {
      if ( A[0] < A[1] )
      {
            max = A[1]
            min = A[0]
        }
        else
        {
            max = A[0]
            min = A[1]
        }
        i = 2
    }
    while ( i < n )
    {
        if ( A[i] < A[i+1] )
        {
            if ( A[i] < min )
                min = A[i]
            if ( A[i+1] > max )
                max = A[i+1]
        }
        else
        {
            if ( A[i] > max )
                max = A[i]
            if ( A[i+1] < min )
                min = A[i+1]
        }
        i = i + 2
    }
    // By convention, we assume ans[0] as max and ans[1] as min
    int ans[2] = {max, min}
   return ans
}
```

Complexity Analysis

Time Complexity is O(n) and Space Complexity is O(1).

For each pair, there are a total of three comparisons, first among the elements of the pair and the other two with min and max.

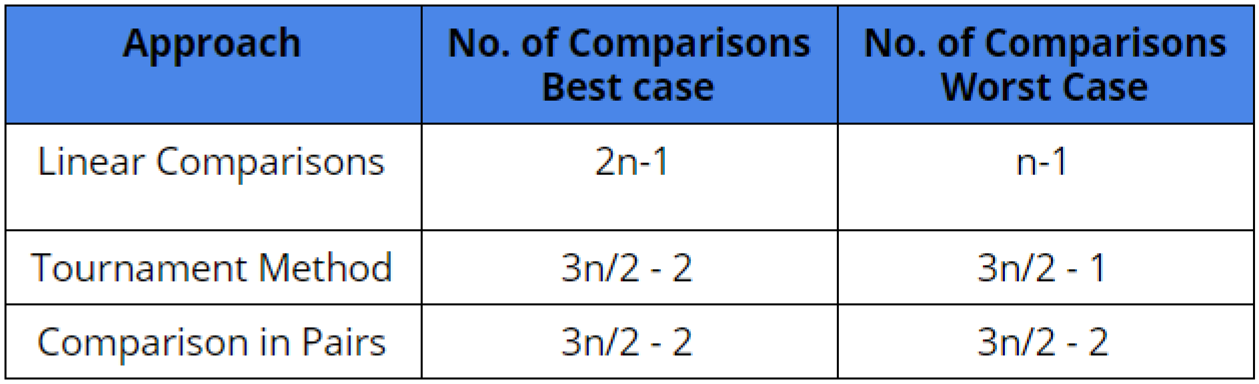
Total number of comparisons:-

- If n is odd, 3 * (n-1) / 2
- If n is even, 1 + 3*(n-2)/2 = 3n/2-2

Critical ideas to think!

- Why min and max are initialized differently for even and odd sized arrays?
- Why incrementing the loop by 2 help to reduce the total number of comparsion ?
- Is there any other way to solve this problem? Think.
- In which case, the number of comparisons by method 2 and 3 is equal?

Comparison of different solutions



Approach	No. of Comparisons Best case	No. of Comparisons Worst Case
Linear Comparisons	2n-1	n-1
Tournament Method	3n/2 - 2	3n/2 - 1
Comparison in Pairs	3n/2 - 2	3n/2 - 2