.1

CS 218 : Design and Analysis of Algorithms

Lecture 5.1: Finding minimum and maximum element in an array

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering   Indian Institute of Technology, Bombay

1 Finding the Maximum and Minimum

The next problem we consider is finding both the maximum and the minimum elements in an array. The obvious solution of first finding the minimum and then the maximum takes $2n-2$ comparisons. Can we do better? This is a question which will haunt us through this course. There will be times when you encounter the *Algorithm Designer's block.* You do not know where to start. Here is a tip to tide you through such moments.

Tip : When lost, try small examples.

How many comparisons does one require for four elements? *Hint:* The answer is four. *Answer:* Compare x_1 with x_2 and x_3 with x_4. Now compare the minimum of x_1 with x_2 and the minimum of x_3 with x_4 to output the minimum. One more comparison suffices to find the maximum. This tell us that the naive bound may not be correct. So, what is the correct bound for n elements? Can you figure out how this is done? Can you generalize what we did for 4 elements? What would you do for 8 elements? 6 elements? Try before you read ahead.

There are two ways to generalize this.

Here is one way to solve this problem.

1. Split the array into two equal parts.
2. Recurse and find the maximum and minimum of both the parts.
3. Now compare the two minimums(maximums) to output the minimum (maximum.)

How many comparisons does this algorithm make?

We detail the solution below. Read it carefully. This is the way we approach bounding the running time of any algorithm.

1.0.1 Bounding running times

Let $T(n)$ denote the number of comparisons made by the algorithm. $T(n)$ then satisfies:

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

$2T(n/2)$ is the time taken by two recursive calls. In addition, we make 2 comparisons. Also we will assume

$$T(1) = 0, T(2) = 1$$

To solve such recurrences, expand a few terms and see how the espression looks like after k iterations for a generic k. Then choose a suitable k.
$T(n) \leq 2(2T(n/2^2) + 2) + 2 = 2^2T(n/2^2) + 2^2 + 2$
$= 2^kT(n/2^k) + 2^k + 2^{k-1} + ... + 2^2 + 2.$
We let $n/2^k = 2$;

So we get $T(n) = n/2 + (2^{k+1} - 2) = 3n/2 - 2$;
Now notice that we could have let $n/2^k = 1$. What answer do you get in this case? What is the message in this problem? Please think awhile on this.

Warning. There is a subtle bug in what we have just done. You will correct it in your homework.

We outline the main design steps. These steps are important enough to state separately. We will use this with other problems.

1. Split the problem into two (equal) parts.

2. Solve each recursively.

3. Now put the solutions together.

Exercise: How many assignments does the above algorithm make?

Here is the second way to generalize. Solve the problem on the first $n - 2$ elements. One comparison finds the maximum and minimum of the last two elements. Now compare the maximums to output the maximum and the minimums to output the minumums. The recurrence is $T(n) = T(n - 2) + 3$. Make sure you understand why the 3.
Exercise. Solve this recurrence.

If the recursion is unrolled you get the following algorithm.

1. Pair up each element at an odd index with the next element at an even index (assuming indices start at 1) and find the maximum and minimum of each pair.

2. Find the minimum (and separately the maximum) of the $n/2$ minimums (maximums) from the previous step.

The total number of comparisons is $n/2 + n/2 - 1 + n/2 - 1 = 3n/2 - 2$.

Exercise: See what happens when you split as $n - 4$ and 4.

There could be many ways of recursing on subproblems. Recursing on subproblems of equal size often yields better running times. However here putting the solutions together may not be that obvious. This is called divide and conquer. Chapter 5 in KT.