Quiz1 I Solutions

K. Subramani
Department of Computer Science and Electrical Engineering,
West Virginia University,
Morgantown, WV
ksmani@csee.wvu.edu

1 Quiz1

Algorithm (1.1) presents the required *Divide-And-Conquer* strategy.

Function MAX-AND-MIN($\mathbf{A}, p, q$)
1: Let $n = q - p + 1$ {n is the number of elements in the array}
2: This function will return 2 values max and min. It should be called from the main program as MAX-AND-MIN($\mathbf{A}, 1, n$)
3: **if** $(n = 1)$ **then**
4: There is only one element in the array
5: **return**($\mathbf{A}[p], \mathbf{A}[p]$)
6: **end if**
7: **if** $(p < q)$ **then**
8: **if** $(n = 2)$ **then**
9: **if** ($A[p] > A[q]$) **then**
10: **return**($A[p], A[q]$)
11: **else**
12: **return**($A[q], A[p]$)
13: **end if**
14: **else**
15: $mid = \frac{p+q}{2}$
16: (max_1, min_1) =MAX-AND-MIN($\mathbf{A}, p, mid$)
17: (max_2, min_2) =MAX-AND-MIN($\mathbf{A}, mid + 1, q$)
18: **if** $(max_1 > max_2)$ **then**
19: $max = max_1$
20: **else**
21: $max = max_2$
22: **end if**
23: **if** $(min_1 < min_2)$ **then**
24: $min = min_1$
25: **else**
26: $min = min_2$
27: **end if**
28: **return**(max, min)
29: **end if**
30: **end if**

Algorithm 1.1: Max-and-Min

Observe the similarity of MAX-AND-MIN with MERGE-SORT. The key idea is that if there is only one element, then nothing needs to be done. If there are two elements, then exactly one comparison is required. Otherwise, we break up the array into two equal parts and recurse to find their respective minimum and maximum elements respectively. Once that is done, we need one comparison to find the minimum of the two minimums and the maximum of the two maximums.

Accordingly, we get the following recurrence relation

$$\begin{aligned} T(2) &= 1 \\ T(n) &= 2.T(\frac{n}{2}) + 2 \end{aligned} \tag{1}$$

Thus, we have

$$\begin{aligned} T(n) &= 2.[2.T(\frac{n}{4}) + 2] + 2 \\ &= 4.T(\frac{n}{4}) + 4 + 2 \\ &= 8.T(\frac{n}{8}) + 8 + 4 + 2 \\ &= 2^{k-1}.T(\frac{n}{2^{k-1}}) + 2^{k-1} + \ldots + 8 + 4 + 2 \end{aligned}$$

where we let $n = 2^k$. Thus $\frac{n}{2^{k-1}} = 2$ and we have that $T(\frac{n}{2^{k-1}}) = T(2) = 1$. Hence,

$$\begin{aligned} T(n) &= 2^{k-1} + 2^{k-1} + \ldots + 8 + 4 + 2 \\ &= 2^{k-1} + 2.[1 + 2 + 4 + \ldots 2^{k-2}] \\ &= 2^{k-1} + 2.\frac{[2^{k-1} - 1]}{2 - 1} \\ &= 2^{k-1} + 2.[2^{k-1} - 1] \\ &= 2^{k-1} + 2^k - 2 \\ &= \frac{2^k}{2} + 2^k - 2 \\ &= \frac{3.2^k}{2} - 2 \\ &= \frac{3.n}{2} - 2 \end{aligned}$$

since $n = 2^k$. The text-book obtains the same bound; however it uses an iterative procedure to process pairs of elements at a time.

2 Homework (Bonus)

The key idea is building a tournament tree as shown in Figure (1).

Pair up the elements as (a_1, a_2), (a_3, a_4) and so on. Compare the elements in each pair and declare the smaller one as the winner. Thus at the end of the first round we have exactly $\frac{n}{2}$ elements. Repeat the procedure $\log_2 n$ times so that the root of the tree has the minimum element. Clearly a total of $n - 1$ comparisons have been made thus far. Observe that at the root of the tree, there is one element from the left child and one element from the right child. Let us say that the minimum element came from the left child. (The same argument works if it came from the right child too.) Then the loser at the root is the only candidate for second minimum from the entire right child. (All other elements in the right child are less than it.) Take this candidate element and trace the path back through the tree along all elements that were eliminated by the minimum element. At each level, test whether the current candidate is smaller than the loser at that node. If so, retain the current element,

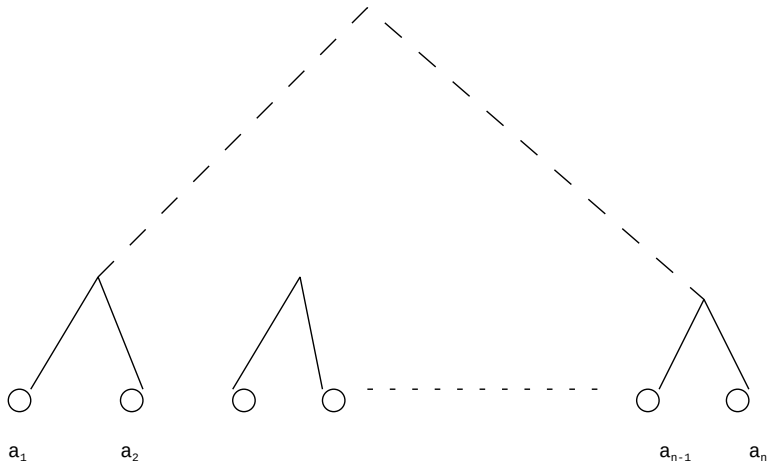

Figure 1: Tree of losers

otherwise make the element at the node, the current candidate. Continue till you hit a leaf. You have then found the second-minimum element.

Hence the total number of comparisons is $n - 1 + \log_2 n - 1 = n + \log n - 2$.

2.1 Divide and Conquer Approach

We now approach the problem from a *Divide-And-Conquer* perspective. We need the following two observations:

Observation: 2.1 *When there are just two elements there is exactly one candidate for second minimum (the larger of the two!) and the second minimum can be determined with one comparison.*

Observation: 2.2 *Let us say that we have two sets S_1 and S_2. Each S_i has an element min_i which is the smallest element in that set and a list l_i which contains the candidates for second minimum in that set. If $min_1 < min_2$, then no element of l_2 can be a candidate for second-minimum of the whole set and vice versa.*

We formalize the above observations into the *Divide-and-Conquer* algorithm (2.1).

So what is the size of the candidate list l. The following recurrences describe the growth of the candidate list.

$$\begin{aligned} Size(2) &= 1 \\ Size(n) &= Size(\frac{n}{2}) + 1 \end{aligned}$$

I hope you can work out the recurrence to conclude that $S(n) = \log_2 n$. In other words, the SECOND-MINIMUM-CAND Algorithm provides the minimum of the array and a candidate list of size $\log_2 n$. The minimum of the candidate list, which is in fact the second minimum of the input array can be found using $\log n - 1$ comparisons in the usual way!

Function SECOND-MINIMUM-CAND($\mathbf{A}, p, q$)

1: Let $n = q - p + 1$ {n is the number of elements in the array}
2: This function will return the *min* value of the set and a candidate list (which can be a linked list or array) of elements that could be the second-minimum. It should be called from the main program as SECOND-MINIMUM-CAND($\mathbf{A}, 1, n$)
3: **if** ($n = 1$) **then**
4: There is only one element in the array
5: **return**($\mathbf{A}[p], \{\mathbf{A}[p]\}$)
6: **end if**
7: **if** ($p < q$) **then**
8: **if** ($n = 2$) **then**
9: **if** ($A[p] > A[q]$) **then**
10: **return**($A[p], \{A[q]\}$)
11: **else**
12: **return**($A[q], \{A[p]\}$)
13: **end if**
14: **else**
15: $mid = \frac{p+q}{2}$
16: (min_1, l_1) =SECOND-MINIMUM-CAND($\mathbf{A}, p, mid$)
17: (min_2, l_2) =SECOND-MINIMUM-CAND($\mathbf{A}, mid + 1, q$)
18: **if** ($min_1 < min_2$) **then**
19: $min = min_1$
20: Append $\{min_2\}$ to l_1 creating l
21: **else**
22: $min = min_2$
23: Append $\{min_1\}$ to l_2 creating l
24: **end if**
25: **return**(min, l)
26: **end if**
27: **end if**

Algorithm 2.1: Second Minimum