CS217 – Algorithm Design and Analysis

Homework 3

Not Strong Enough

March 22, 2020

1

Given an array A of n items (numbers), we can find the maximum with $n-1$ comparisons (this is simple). Show that this is optimal: that is, any algorithm that does $n-2$ or fewer comparisons will fail to find the maximum on some inputs.

Proof. No matter how we make the comparisons and finally get the maximum, the process of finding the maximum forms a "comparision tree". Now we explain how the "comparison tree" is formed:

1. At the beginning, each item of A is regarded as a node, and all nodes are separated. Each node has a value such as $A[0], A[1], \ldots, A[n-1]$.
2. Each time we pick two nodes a and b such that both a and b have **no parent** in the tree.
3. Then we make a comparison between the values of a and b, and get the larger one.
4. We add a new node as the parent of both a and b. The value of the new node is set to $\max\{a, b\}$.
5. Repeat step 2 to step 4. Stop when there is only one node which has no parent, i.e., all nodes constitute a tree. Finally the value of the root is the maximum, which is what we want.

From the process, we know that the leaves of the tree are the very n origin nodes. And it is clear that every internal node, i.e., node that is not a leaf, has exactly two children, since the internal node is added after comparing its two children.

Now the number of comparisons equals the number of internal nodes in the "comparison tree". At the beginning, the number of nodes which have no parent is n, and there is no internal node. Everytime we make a comparison, we add a new node as the parent of two nodes which have no parent. So the number of nodes which have no parent decreases $2-1=1$, and the number of internal nodes increases 1. Finally there is only one node which is the root. So step 2 to step 4 repeats $n-1$ times, and number of internal nodes equals $n-1$.

Therefor, at least $n-1$ comparisons are needed. Otherwise we will get more than one tree in the end, and thus we cannot determine which value is the maximum.

Another explanation For each number x which is not the maximum, there must be at least one comparison between x and a number larger than or equal to x. If not, we may also regard x as the maximum since we don't ever find a number larger than or equal to it. Because there are $n-1$ numbers which are not the maximum, finding the maximum needs at least $n-1$ comparisons. □

2

Let A be an array of size n, where n is even. Describe how to find both the minimum and the maximum with at most $\frac{2}{3}n - 2$ comparisons. Make sure your solution is *simple*, in describe it in a clear and succinct way!

Solution. Since n is even, we can divide A into $n/2$ pairs, and the elements of the i-th pair are $A[2i]$ and $A[2i+1]$.

At the beginning, compare $A[0]$ and $A[1]$. Let *maximum* be the larger one and *minimum* be the smaller one.

For pair $i (i = 1, 2, \ldots, n-1)$, compare $A[2i]$ and $A[2i+1]$ at first. Then we compare the larger one with *maximum*. If the larger one is greater than *maximum*, let *maximum* be the larger one. Similarly, we then compare the smaller one with *minimum*, and if the smaller one is less than *minimum*, let *minimum* be the smaller one.

In the end we get both the *maximum* and the *minimum* of A.

Now we calculate the number of comparisons. For the 0-th pair, we made only 1 comparison. For other pairs, we made exactly 3 comparisons. So the total number of comparisons is $1 + (\frac{n}{2} - 1) \cdot 3 = \frac{3n}{2} - 2$, which satisfy the requirement. □

Here is the pseudocode of the algorithm above.

Algorithm 1 Find both the maximum and minimum of an array within $\frac{3n}{2} - 2$ comparisons.

Ensure: n is even

```
function FindMaxAndMin(A, n)
    m ← n/2
    if A[0] > A[1] then
        maximum ← A[0]
        minimum ← A[1]
    else
        maximum ← A[1]
        minimum ← A[0]
    end if
    for i ← 1 to (n − 1) do
        if A[2i] > A[2i + 1] then
            maximum ← max{maximum, A[2i]}
            minimum ← min{minimum, A[2i + 1]}
        else
            maximum ← max{maximum, A[2i + 1]}
            minimum ← min{minimum, A[2i]}
        end if
    end for
    return maximum, minimum
end function
```

3

Given an array A of size $n = 2^k$, find the second largest element with at most $n + \log_2(n)$ comparisons. Again, your solution should be *simple*, and you should explain it in a clear and succinct way!

Solution. Firstly we consider the following process of finding the maximum recursively.

We divide the array into 2 parts with the same length. Then we find the maximum of both left and right part recursively. And finally we return the larger one between the two maximums of the left and right part. According to exercise 1, it takes $n - 1$ comparisons to find the maximum.

Consider maximum of the whole array. During the process above, the maximum was compared to exactly $\log_2(n)$ element, since there are $\log_2(n) + 1$ levels in the "comparison tree" and in each level except the top one, the maximum was compared to another element.

Among the $\log_2(n)$ elements which were compared to the maximum, there must be the second largest element of the whole array. Otherwise assume the second largest element is x', and there is no direct comparison between the largest and the second largest element x'. Then either x' is not the second largest at all if there exists some "indirect comparison" between x' and the maximum, or according to the second explanation of exercise 1, we may also regard x' as the maximum since we didn't even find an element which is larger than or equal to x'!

So we just need to collect the $\log_2(n)$ elements which were compared to the maximum(this is not difficult because we can record down the elements compared to each element in each level of the "comparison tree"). Then we use $\log_2(n) - 1$ comparisons to find the maximum among the $\log_2(n)$ elements. What we get in the end is the second largest element of the whole array.

Therefore, the total number of comparisons is $(n - 1) + (\log_2(n) - 1) = n + \log_2(n) - 2$, which is less than the required number $n + \log_2(n)$. □

We also offer a pseudocode of this algorithm.

Algorithm 2 Find the second largest element with at most $n + \log_2(n)$ comparisons.

Ensure: $n = 2^k$

```
function FindMaximumIndex(A, left, right)
    if left + 1 = right then
        return left, [ ]
    end if
    mid ← (left + right)/2
    max_L, list_L ← FindMaximumIndex(A, left, mid)
    max_R, list_R ← FindMaximumIndex(A, mid, right)
    if A[max_L] > A[max_R] then
        append max_R to list_L
        return max_L, list_L
    else
        append max_L to list_R
        return max_R, list_R
    end if
end function

function FindSecondLargestElement(A, n)
    max, list ← FindMaximumIndex(A, 0, n)
    k ← log_2(n)
    secondLargest ← A[list[0]]
    for i ← 1 to (k − 1) do
        if A[list[i]] > secondLargest then
            secondLargest ← A[list[i]]
        end if
    end for
    return secondLargest
end function
```