Divide and Conquer: Finding the top k elements of an array[1]

In the first lecture, we saw that the maximum of an array $A[1:n]$ with distinct integers can be found in $n-1$ comparisons. Naively generalizing, the maximum *and* minimum can be found in $2n-3$ comparisons, but we saw how to actually do it in $\leq \frac{3n}{2}$ comparisons. A similar idea also led to a $\frac{3n}{2}$-comparison algorithm for finding the maximum and the second maximum.

We now see how recursion allows us to get an even faster algorithm for finding the first and the second maximum. As in the MAX-AND-MIN case, we *pair* the elements up, compare the pairs, and send the *larger* of the two elements in B. If A has odd number of elements, then the last element would be unpaired, and we send it to B as well. When we pair the elements up, we remember the partners using a helper partner function. So an element e is paired with $\mathsf{partner}(e)$, and an unpaired element e with have $\mathsf{partner}(e) = \perp$.

Now, where does the *maximum* element of A lie? Well, as in the MAX-AND-MIN case, it lies in the list B. Therefore $\max(A) = \max(B)$. Where does the *second maximum* of A lie? One guess may be that $\mathsf{2ndmax}(A) = \mathsf{2ndmax}(B)$. But that is not correct. Consider the array $A = [19, 20, 1, 2]$; in this case, $B = [20, 2]$ but the second maximum of B is not the second maximum of A. And then you make the *key* observation. The second maximum of A is *either* the second maximum of B, *or* it is the *partner* of the maximum. Let's write it as a claim.

Claim 1. Let $A[1:n]$ be an array of n elements, and let B be an array of $\lceil n/2 \rceil$ elements obtained as described above. Let $b^* = \max(B) = \max(A)$. Then, $\mathsf{2ndmax}(A) = \max(\mathsf{2ndmax}(B), \mathsf{partner}(b^*))$.

Proof. Suppose b^*, the maximum of both B and A, lies at $A[i]$. Suppose the second-max of A lies at $A[j]$. If $A[j] \neq \mathsf{partner}(b^*)$ (that is, if $j \neq i+1$ or $i-1$ depending on i), then $A[j]$ *must* be bigger than its partner. It is the second max after all, and the only element that can "defeat" it is $A[i]$. □

Now, we are ready to state our algorithm. Given A, we form the array B. We find the $b_1 = \max(B)$ and $b_2 = \mathsf{2ndmax}(B)$. We are guaranteed $b_1 = \max(A)$. And by the above claim, we are guaranteed $a_2 = \max(b_2, \mathsf{partner}(b_1))$ is the second-max. How do we find b_1 and b_2? But that is *exactly* the same question asked on B. So we recurse. Boom!

```
1: procedure MAX-AND-2ND-MAX($A[1:n]$):
2:     ▷ Returns the max-and-second-max of an $n$ element array as a tuple.
3:     if $n \leq 2$ then:
4:         return the max and second max by at most one comparison. ▷ 1 comparison
5:     We pair the elements $(A[1], A[2])$, $(A[3], A[4])$ and so on. If $n$ is odd, then the last element is
       left unpaired.
6:     We maintain a pointer partner where $\mathsf{partner}(e)$ returns its pair. This is $\perp$ if $e$ is unpaired.
7:     We compare $e$ and $\mathsf{partner}(e)$ and send the larger element to a list $B$. ▷ $\lfloor n/2 \rfloor$ comparisons.
8:     If an element was unpaired, then it is sent to $B$.
9:     $(b_1, b_2) \leftarrow$ MAX-AND-2ND-MAX($B$).
10:    return $(b_1, \max(b_2, \mathsf{partner}(b_1)))$ ▷ 1 comparison
```

[1] *Lecture notes by Deeparnab Chakrabarty. Last modified : 19th Mar, 2022*
These have not gone through scrutiny and may contain errors. If you find any, or have any other comments, please email me at deeparnab@dartmouth.edu. Highly appreciated!

Analysis. Let $T(n)$ be the worst-case number of comparisons on an array of length n. We have the base cases:

$$T(1) = 0 \quad T(2) = 1$$

Note that for running Line 4, we really don't need both max and min; just a single comparison will tell us the answer.

For a general $n > 2$, let us see what the number of comparisons are. The only comparisons are made in Line 7, in the recursive call Line 9, and finally, in Line 10 (when we compare b_2 and partner(b_1)). Therefore, we get the recurrence

$$\text{For } n \geq 3, \quad T(n) \leq \lfloor n/2 \rfloor + T(\lceil n/2 \rceil) + 1 \tag{1}$$

where we have used the fact that the number of elements in B is exactly $\lceil n/2 \rceil$ (including the unpaired elements).

Now, we can solve this recurrence using the kitty method. Note that we should not apply the Master theorem as that would give a *coarse* answer of $O(n)$ (you see this, right?) But we really want a much more fine-grained answer, and then we have to resort to the kitty method (or the open-up the inequalities method).

For simplicity, let us assume $n = 2^\ell$ so that the pesky ceilings and floors don't bother us (we'll take care of this at the end). So, we get

$$\begin{aligned}
T(n) &\leq T(n/2) + \left(\frac{n}{2} + 1\right) \\
&\leq T(n/4) + \left(\frac{n}{4} + 1\right) + \left(\frac{n}{2} + 1\right) \\
&\leq T(n/8) + \left(\frac{n}{8} + 1\right) + \left(\frac{n}{4} + 1\right) + \left(\frac{n}{2} + 1\right) \\
&\vdots \\
&\leq T(n/2^{\ell-1}) + \left(\frac{n}{2^{\ell-1}} + 1\right) + \left(\frac{n}{2^{\ell-2}} + 1\right) + \cdot + \left(\frac{n}{4} + 1\right) + \left(\frac{n}{2} + 1\right)
\end{aligned}$$

We stop when at $(\ell - 1)$, because then $n/2^{\ell-1}$ becomes 2 (we have assumed $n = 2^\ell$), and in that case we get $T(n/2^{\ell-1}) = T(2) = 1$, which we know by the base case. Putting this together, we get (at least for $n = 2^\ell$),

$$T(n) \leq 1 + \left(\frac{n}{2^{\ell-1}} + 1\right) + \left(\frac{n}{2^{\ell-2}} + 1\right) + \cdot + \left(\frac{n}{4} + 1\right) + \left(\frac{n}{2} + 1\right)$$

To make sense of the RHS, let us see how many $+1$s are there. You will see there are exactly ℓ many of them, and $\ell = \log_2 n$. And then we are left with the geometric series: $\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots + \frac{n}{2^{\ell-1}}$. Again, when $n = 2^\ell$, this is simply $2 + 4 + 8 + \cdots + 2^{\ell-1} = 2^\ell - 2 = n - 2$. Adding both these terms, we get

$$\text{When } n = 2^\ell, \text{ we get} \quad T(n) \leq n + \log_2 n - 2$$

How about the case of n when it's not a power of 2? We now show a way to solve (1) by another method of solving recurrences called "guess and prove". Once we can guess the "shape" of a recurrence, then we could assert it and prove by induction.

Theorem 1. MAX-AND-2ND-MAX makes at most $n + \lceil \log_2 n \rceil - 2$ comparisons.

Proof. We need to show that the solution to (1) satisfies $T(n) \leq n + \lceil \log_2 n \rceil - 2$, for all $n \geq 2$. We proceed by induction. The base case holds for all $n = 2$ since $T(2) = 1$ and $2 + \lceil \log_2 2 \rceil - 2 = 1$.

$$T(n) \leq T(\lceil n/2 \rceil) + \lfloor n/2 \rfloor + 1$$

and since $\lceil n/2 \rceil < n$ (since $n > 2$), by the induction hypothesis, we get

$$T(n) \leq \Big(\lceil n/2 \rceil + \lceil \log_2 (\lceil n/2 \rceil) \rceil - 2 \Big) + \lfloor n/2 \rfloor + 1$$

Rearranging, we get

$$T(n) \leq \Big(\lceil n/2 \rceil + \lfloor n/2 \rfloor \Big) + \lceil \log_2 (\lceil n/2 \rceil) \rceil - 1$$

To complete the proof, we need to prove the following algebraic claim. Then the inductive case would also be proved.

Claim 2. $\lceil \log_2 (\lceil n/2 \rceil) \rceil - 1 \leq \lceil \log_2 n \rceil - 2$ for any n

Proof. Equivalently, we need to show for any n, $\lceil \log_2 n \rceil \geq \lceil \log_2 (\lceil n/2 \rceil) \rceil + 1$. Now, if $n = 2^\ell$ is a power of 2, then we indeed have equality. So, $\ell < \log_2 n < \ell + 1$. $n = 2^\ell + q$ for some $q < 2^\ell$. So, $\lceil \log_2 n \rceil = \ell + 1$.

On the other hand, $n/2 = 2^{\ell-1} + \frac{q}{2}$. Now, we want to put an upper bound on $n/2$. The largest q can be is $2^\ell - 1$, and in that case, we have $\lceil n/2 \rceil = 2^\ell$. Thus, $\lceil n/2 \rceil \leq 2^\ell$ implying $\log_2 \lceil n/2 \rceil \leq \ell$. And thus, $\lceil \log_2 (\lceil n/2 \rceil) \rceil \leq \ell$ as well. This proves that $\lceil \log_2 n \rceil = \ell + 1 \geq \lceil \log_2 (\lceil n/2 \rceil) \rceil$. □

□

Finding the first 3, or in general the first k numbers. If you have understood the above algorithm, then you should see an algorithm for finding the largest 3 numbers as well. Indeed, let's describe an algorithm for finding the largest k. The idea is the same: we form the array B from A by pairing and comparing. We find the largest k numbers in B recursively. Let this list be L. These may not be the largest k numbers in A, but the largest k numbers of A are in L or $\mathsf{partner}(L) := \{\mathsf{partner}(e) : e \in L\}$. Do you see this? The following claim establishes it.

Claim 3. Let $A[1 : n]$ be an array of n elements, and let B be an array of $\lceil n/2 \rceil$ elements obtained as described pairing A and taking the larger number from each pair, and an unpaired numbered if any. Let $b_1 > b_2 > \cdots > b_k$ be the k largest numbers of B. Let $a_i := \mathsf{partner}(b_i)$ for $1 \leq i \leq k-1$. Then, the largest k numbers of A lie in the set $S := \{a_1, a_2, \ldots, a_{k-1}, b_1, b_2, \ldots, b_k\}$.

Proof. Let me sketch a proof. Suppose not. Suppose there is some element e of A which is in the largest k elements but it's not in the set S. First observe that $e \notin B$, for otherwise there are k elements (namely, $b_1, b_2, \ldots, b_k$) which are larger than e, and thus e is not in the top k elements of A. So $\mathsf{partner}(e) \in B$. But since $e \notin \{a_1, \ldots, a_{k-1}\}$, the partner $\mathsf{partner}(e) \notin \{b_1, \ldots, b_{k-1}\}$. That is, there are at least $(k-1)$ numbers in B bigger than $\mathsf{partner}(e)$. And since $\mathsf{partner}(e) > e$ (for $\mathsf{partner}(e)$ was sent to B), we again obtain $\geq k$ numbers bigger than e. This contradicts e was in the top k elements of A. □

And so, we get the recursive algorithm.

```
1: procedure K-LARGEST($A[1:n]$):
2:     ▷ Returns the largest $k$ elements of $A$
3:     if $n \le k$ then:
4:         Sort $A$. ▷ $C_k = O(k \log k)$ comparisons
5:     Form $B$ as in MAX-AND-2ND-MAX. ▷ $\lfloor n/2 \rfloor$ comparisons.
6:     $L \leftarrow$ k-Largest($B$).
7:     $S \leftarrow L \cup \mathsf{partner}(L)$
8:     Sort $S$ and return the largest $k$ elements in $S$. ▷ $C_k = O(k \log k)$ comparisons
```

The recurrence inequality governing the above algorithm is the following. Here C_k is some fixed constant depending on k, and is $O(k \log k)$

$$T(n) \le C_k, \text{ if } n \le k, \quad \text{for } n \ge k, \quad T(n) \le T(\lceil n/2 \rceil) + \lfloor n/2 \rfloor + C_k \tag{2}$$

For $n = 2^\ell$, a power of 2, you can solve the recurrence using the kitty method, and then also use the inductive argument above to prove the following theorem.

Theorem 2. For any integer k, the K-LARGEST algorithm makes $n + C_k \cdot \lceil \log_2(n/k) \rceil$ many comparison, where C_k is the maximum number of comparisons needed to find the largest k elements in an array of length $(2k-1)$. In short, this is $n + O(k \log k \log(n/k))$.