CF730B Minimum and Maximum

`https://codeforces.com/contest/730/problem/B`

I begin by trying the problem-solving strategy of simplifying the problem. What if I *only* needed to find the maximum of the array? Well, it is well-known that **exactly** $n-1$ comparisons are needed in order to find the maximum of an array—maintain a "running max" and note that to find the maximum, we only need to compare each next element with the current max so far.

```
ans = a[1]
for i from 2 to n:
    ans = max(ans, a[i])
```

Suppose that we run this solution and then get the maximum element of the array using $n-1$ comparisons. Can we then find the minimum element using the information from those queries plus $\approx n/2$ extra comparisons?

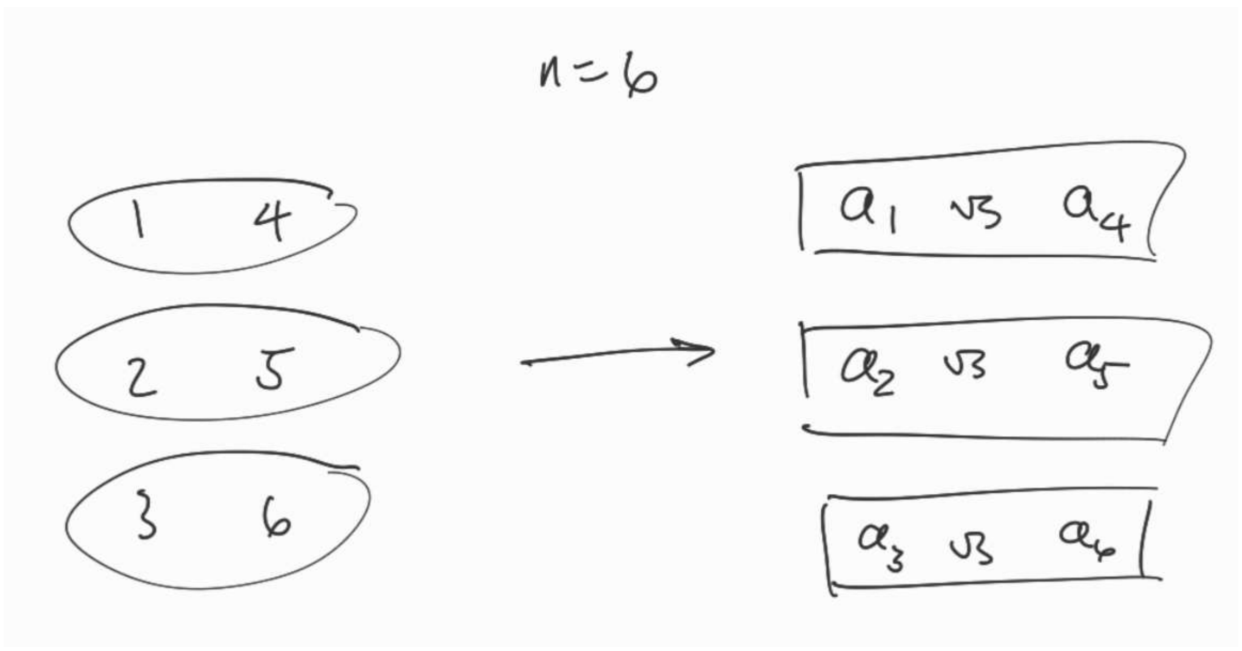
Unfortunately, I don't think so. Consider this worst case: Suppose that `a[1]` was already the maximum of the array. Then, all our comparisons will just tell us that `a[1]` is greater than all the other elements of the array. But then we didn't compare any of `a[2], a[3], ..., a[n]` with each other! So, we don't know anything else about them, and without any other structure imposed on them, we would need to spend $(n-1)-1$ comparisons in order to find the minimum among them. But $2n-3 > f(n)$ when $n \geq 4$, so this won't work.

I examine the given $f(n)$ in order to find inspiration. The most striking thing about it to me is the $3n/2$ term, specifically its factor of 3. By symmetry, I have an intuition that finding the maximum should take the same amount of effort as finding the minimum. But I've also shown that considering the two cases independently is not sufficient to solve the problem, either.

So, with that $3n/2$ term in mind, I conjecture that the solution should roughly have the following shape:

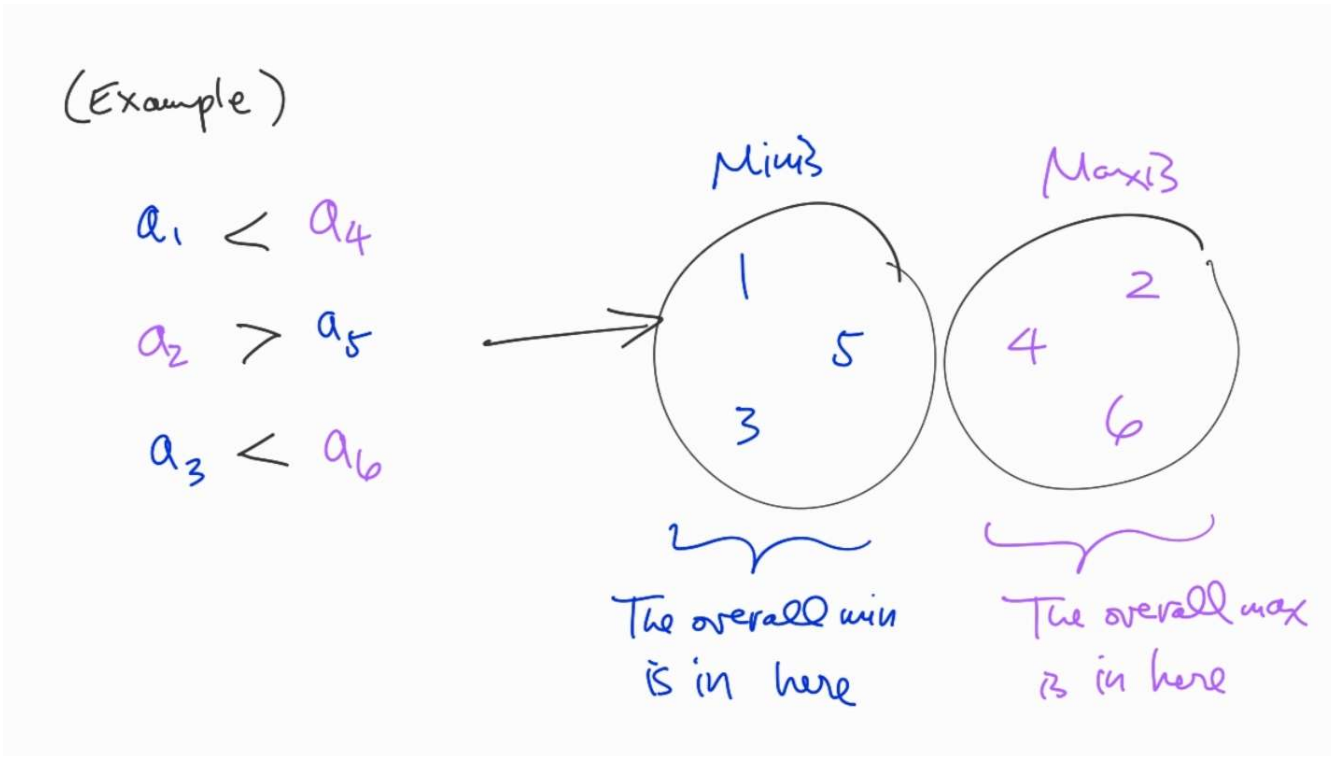
- Spend $n/2$ comparisons on some precomputation
- Then, spend $n/2$ comparisons finding the maximum
- Then, spend $n/2$ comparisons finding the minimum

Okay, what possible precomputations can I perform? What is the most "natural" way to perform $n/2$ comparisons on an array? The first "natural" idea that comes to mind is to pair up elements together somehow, and then compare the two elements in each pair.



Aha! Everything's falling into place now. Suppose I pair everyone off and then spend $n/2$ comparisons to compare each element with its partner. Now, I split the indices into groups, the "minis" and the "maxis". The index of each smaller element goes into the minis, and the index of each larger elements goes into the maxis (if two elements are equal, they can go anywhere, so just arbitrarily put one in each).

But note that the minimum of the entire array *must be a mini* (and similarly for maxis); a maxi can't be the minimum because we found an element that was smaller than it.



Now, I can just apply the running max/min idea on the maxis and then the minis, and it only takes $n/2 - 1$ comparisons for each. Our overall solution thus only takes $\approx 3n/2 - 2$ comparisons, so it seems we've solved the problem!

Let's be a bit more precise with counting the number of needed queries, just to really prove correctness.

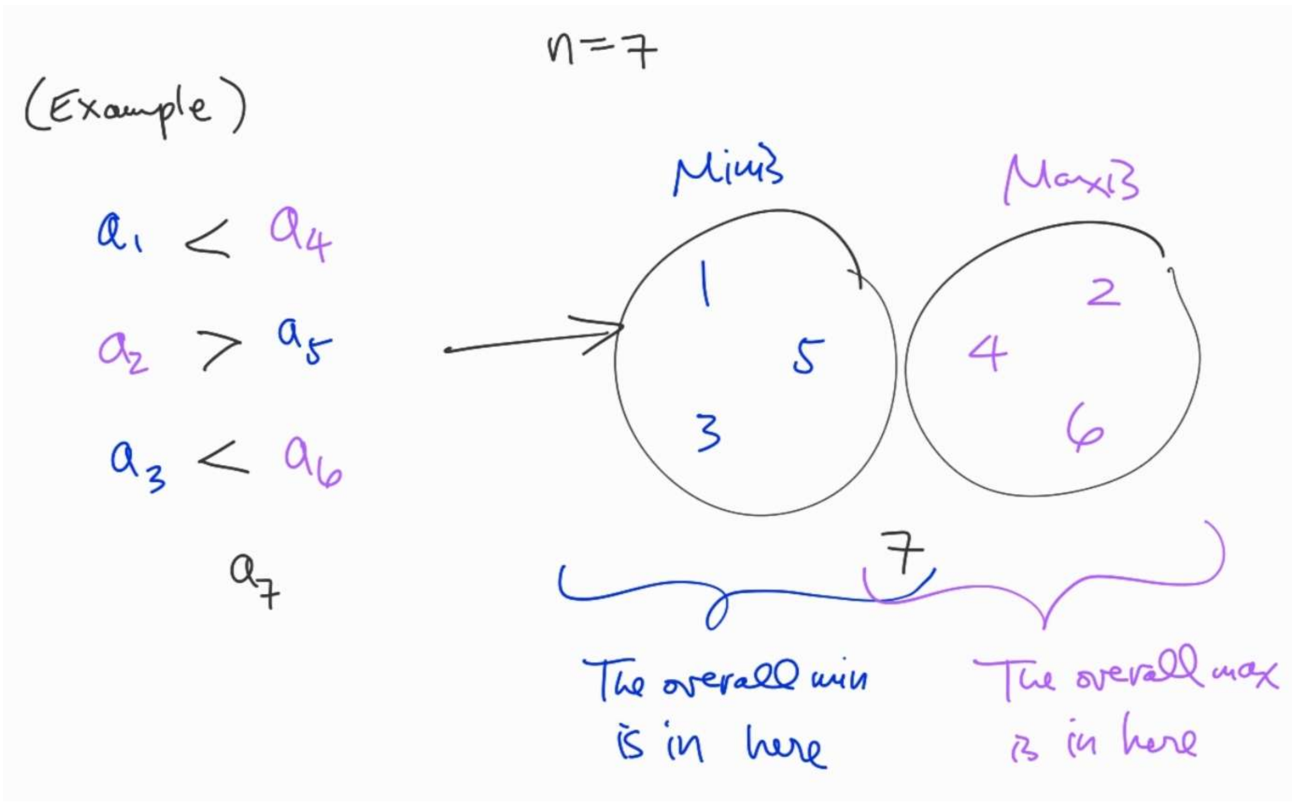
If n is even, then $f(n) = 3n/2 - 2$ exactly. Our solution proceeds exactly as we just described it.

- Pair up the elements and partition them into minis and maxis ($n/2$ comparisons)
- Find the minimum of the minis ($n/2 - 1$ comparisons)
- Find the maximum of the maxis ($n/2 - 1$ comparisons)

These steps take $3n/2 - 2$ comparisons in total, so we have solved the problem when n is even.

If n is odd, then $f(n) = (3n + 1)/2 - 2$ (because $3n$ is also going to be odd). We also have to adjust our solution a bit if n is odd.

- First, remove any element (say, element n) so that there are only $n-1$ elements remaining; note that $n-1$ is now even.
- Pair up the remaining $n-1$ elements and partition them into minis and maxis ($(n-1)/2$ comparisons).
- Note that there are $(n-1)/2$ minis, and $(n-1)/2$ maxis, and `a[n]` is uncategorized.
- Now, note that the minimum element must either be a mini, **or** it could be `a[n]` (similar logic for maximums)
- Find the minimum of the minis together with `a[n]` ($((n-1)/2+1)-1$ comparisons)
- Find the maximum of the maxis together with `a[n]` ($((n-1)/2+1)-1$ comparisons)



You can do the algebra and verify that $(3n+1)/2-2 = 3(n-1)/2$, and so we have also solved the problem when n is odd.

Useful takeaways for this problem:

- Simplifying-the-problem strategy
- Drawing inspiration from solutions to past related problems
- Incorporating defining/striking features of the problem into the solution