Daily Coding Problem #235

Problem

This problem was asked by Facebook.

Given an array of numbers of length `N`, find both the minimum and maximum using less than `2 * (N - 2)` comparisons.

Solution

A naive solution would involve iterating through and comparing each element to a running minimum and a running maximum. Since we would have to compare each element twice, this does not satisfy our constraint.

The trick here is to notice that each comparison actually provides two pieces of information: the smaller element cannot possibly be the maximum of the list, and the larger element cannot be the minimum. So if we take successive pairs of the list, we only need to compare the smaller one to our running minimum, and the larger one to our running maximum.

For example, take the input `[4, 2, 7, 5, -1, 3, 6]`. To start out, both the minimum and maximum can be initialized as the first element of the list, 4. Then, we examine the next pair to find that 2 is smaller and 7 is bigger. So we update our running minimum to be `min(4, 2) = 2` and update our running maximum to be `max(4, 7) = 7`. Carrying along like this, we would eventually find the minimum and maximum to be `-1` and 7, respectively.

Since it takes `n / 2` comparisons to order each pair, and each element will be compared to either the running minimum or maximum, this algorithm uses about `3 * n / 2` comparisons.

```
def min_and_max(arr):
    min_element = max_element = arr[0]
    compare = lambda x, y: (x, y) if y > x else (y, x)

    # Make the list odd so we can pair up the remaining elements neatly.
    if len(arr) % 2 == 0:
        arr.append(arr[-1])

    for i in range(1, len(arr), 2):
        smaller, bigger = compare(arr[i], arr[i + 1])
        min_element = min(min_element, smaller)
        max_element = max(max_element, bigger)

    return min_element, max_element
```

An alternative approach is to use a divide and conquer algorithm.

Our base cases are as follows:

- If there is only one element in the array, return that element for both the minimum and maximum.
- If there are two elements, return the smaller one as the minimum, and the larger one as the maximum.

For the general case, we recursively apply our algorithm to the left and right halves of our array, and return the minimum and maximum of the results.

```
def min_and_max(arr):
    if len(arr) == 1:
        return arr[0], arr[0]
    elif len(arr) == 2:
        return (arr[0], arr[1]) if arr[0] < arr[1] else (arr[1], arr[0])
    else:
        n = len(arr) // 2
        lmin, lmax = min_and_max(arr[:n])
        rmin, rmax = min_and_max(arr[n:])
        return min(lmin, rmin), max(lmax, rmax)
```

To be more concrete, here are the intermediate steps when this is applied to the example above:

```
# Recursively break down array
[4, 2, 7, 5, -1, 3, 6]
[[4, 2, 7, 5], [-1, 3, 6]]
[[[4, 2], [7, 5]], [[-1, 3], [6]]]

# Base case: reorder so that smaller comes before larger
[[[2, 4], [5, 7]], [[-1, 3], [6, 6]]]

# Merge to find min and max
[[min(2, 5), max(4, 7)], [min(-1, 6), max(3, 6)]]
[min(2, -1), max(7, 6)]
[-1, 7]
```

We can derive the complexity of this algorithm as follows. For each array of size `N`, we are breaking down the problem into two subproblems of size `N / 2`, plus 2 additional comparisons. More formally, `T(N) = 2T(N / 2) + 2`, with base case `T(2) = 1`. When `N` is a power of two, this recurrence relation resolves exactly to `T(N) = 3 * N / 2 - 2`; otherwise, it will take a few more steps.