Mr. Brilliant's Algorithm

Approach 1

Description

Find the smallest number in the sequence if there are more than one, select the one with the smallest index. Suppose the number is a. Output the number. Repeat the previous steps for the input sequence $a_{l+1}, a_{l+2}, \ldots, a_n$, until the input sequence is empty.

Psudocode

```
def brilliants_algo: array<number>(A:array):
    out:array<number> = []
    while len(A) > 0
        smallest: number = A[0]
        smallest_index: int = 0
        for index from 0 to len(A):
            if A[index] < smallest:
                smallest = A[index]
                smallest_index = index
        out.append(smallest)
        if len(A) > 0:
            A = A[smallest_index+1:] # Trim A to only include elements
after smallest_index
    return out
```

Runtime

```
def brilliants_algo: array<number>(A:array):
    out:array<number> = [] # O(1)
    while len(A) > 0 # O(n^2)
        smallest: number = A[0] # O(1)
        smallest_index: int = 0 # O(1)
        for index from 0 to len(A): # O(n)
            if A[index] < smallest: # O(1)
                smallest = A[index] # O(1)
                smallest_index = index # O(1)
        out.append(smallest) # O(1)
        if len(A) > 0: # O(n) / O(1)
            A = A[smallest_index+1:] # O(n)
    return out # O(1)
```

In the worst case, this algorithm will take $O(n^2)$ time, since it will go through the top layer while loop n times, then through the inner for loop, which will take $O(n)$ time to run.

Functionality

This algorithm doesn't work. A counterexample would be $A = [5, 6, 7, 2, 1]$ In this example, the algorithm would first, pick the final element as the smallest, and remove the elements before it, returning only the final element, when in actuality the solution would be $[5, 6, 7]$.

Approach 2

Description

Try the following with $\ell = 1, 2, ..., n$: Let $i = \ell$. Include a_i in the subsequence. Find the smallest j such that $j > i$ and $a_i < a_j$. If there is no such j, stop. Else, let $i = j$ and repeat the steps described in the last two sentences. Once you are done trying all possible ℓ\s, out of the n obtained subsequences output one with the longest length.

Psudocode

```
def brilliants_algo(A:array<number>):
    let n = A.length
    let subsequences = an empty 2d array
    for l = 0 to n:
        let i = l
        let subsequence = new array
        subsequence.append(A[i])
        let j = 0
        while j < n:
            j ++
            found = false
            while j < n and not found:
                if A[j] > A[i]:
                    next_index = j;
                    found = true
                else:
                    j++
            if not found:
                j = n;
            else:
                i = next_index
                subsequence.append(A[i])
        subsequences[l] = subsequence
    best_len = 0
    best_i
    for i = 0 to n:
        if subsequences[i].length > best_len:
            best_i = i
            best_len = subsequences[i].length
    return subsequences[i]
```

Runtime

The outer loop runs n times and the inner loop runs in O(n) time in the worst case. This means that the total time it takes to run is $O(n^2)$.

Functionality

This algorithm does not work for every case, if we try the array $A = [1, 4, 2, 3]$, we first make our ℓ arrays, $\ell_1 = [1, 4], \ell_2 = [4], \ell_3 = [2, 3], \ell_4 = [3]$. We then find the longest ℓ array, which is ℓ_1, and we return $[1, 4]$. The actual correct answer would be $[1, 2, 3]$.