Homework 3 - Problem 2 (Approach 1)

```
Given Set A is the set containing a sequence of numbers
Let size be the size of Set A
Let maxLen be 0
Let longestSubsequence be an empty array

while size > 0:
    Let smallest be MAX_VALUE
    Let smallestIndex be -1

    for index from 0 to size - 1:
        if A at index < smallest:
            smallest = A at index
            smallestIndex = index

    output smallest

    Remove A at smallestIndex from A

    size = size - 1
```

Time Estimate: $O(n^2)$

Explanation:

Search for the minimum element in the remaining sequence, which requires comparing every element with the minimum element. This is $O(n^2)$ worst case scenario.

Approach 1 does not work:

Example:

Set A = [8, 2, 5, 7, 3, 4, 9, 6, 10]
Approach Output = [2, 3, 4, 5, 6, 8, 9, 10]
Correct Output = [2, 3, 4, 9, 10] or [2, 5, 7, 9, 10]

This outputs the numbers from smallest to largest without guaranteeing the longest increasing subsequence. The issue is it isn't comparing the indexes of the values properly.

Homework 3 - Problem 2 (Approach 2)

```
Given Set A is the set containing a sequence of numbers
Let size be the size of Set A
Let maxLen be 0
Let longestSubsequence be an empty array

for l from 1 to size:
    Let index be l
    Let currentLen be 1
    Let currentSubsequence be an array containing index at A

    while index is less than size:

        Let Vi be index at A
        Let Vs be subIndex at A

        Find the smallest subIndex such that subIndex > index and
         Vi is less than subIndex at A

        If no such subIndex exists, exit

        Set index = subIndex
        currentLen++
        Append index at A to currentSubsequence

    if currentLen is greater than maxLen:
        Set maxLen to currentLen
        Set longestSubsequence to currentSubsequence

Output longestSubsequence
```

Time Estimate: $O(n^2)$

Explanation:

The inner loop, which searches for the smallest subIndex, can potentially check each element in the sequence. The outer loop iterates through all possible starting points. This is O(n * n) or $O(n^2)$

Approach 2 works:

Example:

Set A = [8, 2, 5, 7, 3, 4, 9, 6, 10]
Approach Output = [2, 5, 7, 9, 10]
Correct Output = [2, 3, 4, 9, 10] or [2, 5, 7, 9, 10]

This works because it correctly compares the index of the numbers as well as the numbers themselves, something that Approach 1 did not do.