Jordan Allard
CSCI 264-01
Homework 3

HW3 Problem 2 Writeup

Approach #1: Smallest Number

a) Pseudocode

```
Let A be a sequence of numbers with length n
minIndex = 0
newMinFound = true
while (newMinFound):
      newMinFound = false
      for (int i = minIndex; i < n; i++):
            if A[i] < A[minIndex]:
                  minIndex = i
                  newMinFound = true
      if (newMinFound):
            output A[minIndex]
      minIndex += 1
```

b) Running Time Estimate

The running time of this algorithm is O(n^2). In the worst possible input, the input where every number in the array is already in increasing order, the while loop will run n times (since the algorithm will end up finding and printing n minimums). Each iteration would remove one number from the input sequence on which the for loop operates, so the total running time for the worst possible input would be n + (n - 1) + (n - 2) + . . . + 2 + 1 = (n * (n - 1))/2, which is equlivalent to O(n^2).

c) Evaluation of Correctness

This algorithm is not correct for every input, for the simple reason that the minimum number in the sequence will not always be the best number from which to start searching for an increasing subsequence. A counterexample would be the sequence 2, 3, 4, 5, 1, 6. This algorithm would output 1, 6, while the true longest increasing subsequence would be 2, 3, 4, 5, 6.

Approach #2: All Sequences

a) Pseudocode

```
Let A be a sequence of numbers with length n
L = empty array of length n
longestSequenceIndex = 0
longestSequenceLength = 0
for (int i = 0; i < n; i++):
      L[i][0] = A[i]
      sequenceLength = 1
      j = i + 1
      while (j < n):
            while j < n && A[j] <= A[i]:
                  j++
            if (j < n):
                  L[i][sequenceLength] = A[j]
                  sequenceLength += 1
                  if sequenceLength > longestSequenceLength:
                        longestSequenceIndex = i
                        longestSequenceLength = sequenceLength
```

```
                j += 1
      for (int k = 0; k < n; k++)
            output L[longestSequenceIndex][k]
```

b) Running Time Estimate

The running time estimate for this algorithm is O(n^2). The outer for loop will run n times, since it calculates a sequence for every number in the array. The nested while loops contain the same condition of j < n, and within those loops j starts at i and can only increase. This means that each interation is guaranteed to bring j at least one step closer to the break condition, and thus the maximum times the nested while loops will run is n - i, where i = 1, 2... n. So the time complexity of this algorithm would be n + (n - 1) + (n - 2) + . . . + 2 + 1 = (n * (n - 1)) / 2 = O(n^2).

c) Evaluation of Correctness

This implementation of the algorithm is not correct for every input. It fails to account for the multiple different increasing subsequences that are possible for the same starting number. By taking the first larger number, the algorithm will not always find the longest possible subsequence for each number. An example input where the algorithm fails would be 2, 9, 3, 4, 5. The algorithm would calculate the sequences (2, 9), (9), (3, 4, 5), (4, 5), and (5), and output 3, 4, 5. However, the longest increasing subsequence is actually 2, 3, 4, 5.