CISC 380 **Instructor: Sarah Miracle**

Homework 3

Student: SOLUTIONS

1. (5 points) Recall that a subarray is similar to a subsequence except that the elements must be consecutive. For instance, if A is $[5, 15, -30, 10, -5, 40, 10]$ then $[15, -30]$ is a subarray but $[5, 15, 40]$ is not. Use **dynamic programming** to give a linear-time algorithm for the following task:

   Input: A list of numbers, $a_1, a_2, \ldots, a_n$.
   Output: The **sum** of the subarray with maximum sum
   Note that if the array contains only negative numbers then the answer should be negative. For example, if the input is `[-1, -2, -3]` then your method should return -1 (i.e. a subarray has length at least 1).

Solution:

1. Let $MAXSS(i)$ be sum of the subarray with maximum sum among the first i terms of the sequence *and both including and ending at the ith element.* In other words, this subproblem only measures the subarrays that end at and include a_i.

2. For the recurrence, we have two cases depending on whether the subarray has 1 or more than one elements (note that these are the only possibilities since we are not allowing 0-length subarrays). If the max subarray ending at i only includes one element then it must include only a_i, and has sum a_i. If instead the max subarray end at i has more than one element then it must include a_i, a_{i-1} and possibly other elements. $MAXSS(i-1)$ gives us the sum of subarray with maximum sum that ends at and includes a_{i-1} (which is exactly what we need!). Thus if there is more than one element then $MAXSS(i) = a_i + MAXSS(i-1)$.

   We will choose the case that gives us the larger sum, so

   $$MAXSS(i) = \max(a_i, a_i + MAXSS(i-1)).$$

   For our base case, if $i = 1$ then there is only one choice (there are no previous elements to include) so $MAXSS(1) = a_1$.

   Since a subarray must end somewhere, the maximum value across all $MAXSS(i)$ will give us the value of the max subarray sum.

   Stylistic note: We can keep track of the maximum as we fill in the $MAXSS$ table if we want to do it all in one pass, or we can come back and retrieve it from the array afterwards. I will do the latter.

3. Note that for this problem, even the straightforward recursive implementation is $O(n)$. Here is a solution using the dynamic programming paradigm and iteration.

```
function MAXSS(Array $a[1 \dots n]$)
    Array $S[1 \dots n]$
    $S[1] = a[1]$                                        ▷ base case
    for $i = 2$ to $n$ do
        $S[i] = \max(S[i-1] + a[i], a[i])$
                    ▷ At this point, the optimum sum is the maximum of the $S[i]$.
    $maxSoFar = S[1]$
    for $j = 2$ to $n$ do
        $max = \max(maxSoFar, S[j])$
    return $maxSoFar$    ▷ However you decide to output this, it will take $O(n)$.
```

4. We do two loops that both do $O(n)$ work, which leads to an $O(n)$ algorithm.

2. **(PRACTICE)** You are going on a long trip. You start on the road at mile post 0. Along the way there are n hotels, at mile posts $a_1 < a_2 < \ldots < a_n$, where each a_i is measured from the starting point. The only places you are allowed to stop are at these hotels, but you can choose which of the hotels you stop at. You must stop at the final hotel (at distance a_n), which is your destination.

You'd ideally like to travel 200 miles a day, but this may be impossible to do exactly (depending on the spacing of the hotels). If you travel x miles during a day, the penalty for that day is $(200 - x)^2$. You want to plan your trip so as to minimize the total penalty - that is, the sum, over all travel days, of the daily penalties. Give an efficient algorithm that determines the optimal sequence of hotels at which to stop.

Solution:

1. Let $MinTripCost(i)$ be the minimum possible cost incurred up to hotel i, provided you stop there.
2. Set $Mintripcost(0) = 0$. Consider where you stopped right before i, at some hotel $x < i$. Then you can see that $MinTripCost(i) = MinTripCost(x) + (200 - (a_i - a_x))^2$. You don't know apriori what x is, so you choose the minimum of this value over all possible locations a_x : there are i of these choices.
3.

```
function MINTRIPCOST(Array $a[0 \dots n]$)
    Array $S[0 \dots n]$ initialized to $+\infty$
    Array $prev[1 \dots n]$ ▷ This will be used to aid in reconstructing the sequence of
stops
    $S[0] \leftarrow 0$
    for $i = 1$ to $n$ do
        $x \leftarrow i - 1$                          ▷ A tracker for the previous stop
        for $x = 1$ to $i - 1$ do
            if $S[i] > S[x] + (200 - (a_i - a_x))^2$ then
                $S[i] \leftarrow S[x] + (200 - (a_i - a_x))^2$
                $prev[i] = x$                        ▷ Set the previous stop to x
```

```
                                   ▷ After this, $S[n]$ has the min cost of the total trip
if $S[n] = \infty$ then           ▷ If the distance between any adjacent hotels $> 200$
    return "No possible route"
$i \leftarrow n$
$Answer \leftarrow$ ''             ▷ An empty linked list of indices to store our answer
while $i \neq 0$ do
    $Answer = a[i] + Answer$       ▷ Prepend the index of the current stop
    $i = prev[i]$
return $Answer$
```

4. We have two nested for loops that do at most $O(n)$ work each, which leads to an $O(n^2)$ algorithm.