Solutions CSE 330 Winter 2020 – Homework 1

Task 1:

(1) What is the maximal contiguous subsum for a vector of only positive values? (Explain in general, not by specific example.) – 3 points

Correct Answer: the sum of all numbers in the vector.

(2) What is the maximal contiguous subsum for a vector of only negative values? (Explain in general, not by specific example.)

Two Correct Answers: (a) the largest of the negative values contained in the vector, if you go by the formulation of the problems without consideration of possible implementations; (b) zero, if you go by presented implementation of the function that solves the problem (int maxSum is initialized to 0 as we tend to do prior to accumulation of sums; with all negative vector values, no local sum will ever exceed maxSum; the latter will remain at zero and be returned as result.

(3) Imagine a stage of problem solving in the context of Algorithm 4: previous iterations have determined some candidate (so far seen) maximal subsum; in the current iteration, a local sum has been accumulated and its value is negative. What is the significance of this fact for this current iteration relative to our goal (= finding the overall maximal subsum)?

Correct Answer: When the value of a local sum that is being accumulated within an iteration is negative, this negative value will bring down whatever value the localSum can achieve in the continuation of the iteration.

Grading Task 2:

Write one brief paragraph, in which you **explain in your own words** what the clever insight is that allows Algorithm4 to solve the problem in a single for-loop (= single pass over the input). Write plainly, clearly, and concisely. The answers from Task 1 should be helpful.

Correct Answer (related to Task 1(3)): (this one is much longer than expected or required)

Your vector contains a mix of positive and negative numbers (otherwise the problem is not very interesting as you have found out in Task 1 (1) and (2)). We follow the idea of finding the maximal contiguous subsum <u>in a single iteration over the vector …</u> Imagine the case that the vector starts with a negative number (e.g., vec[0] is negative). We know right away that the maximal subsum will not involve this negative number … why? Well if the maximal subsum were M and negative v[0] were one of the summands, then M – v[0] would be bigger than M and that larger number will be the maximal sum … so we can safely ignore the negative value at i = 0 and *start the sum at index i = 1. If v[1] is also negative, let's ignore that position and restart at index i = 3 …. Etc. Sooner or later, as we move from left to right, there will be some positive value to start the sum … From that point, we keep on summing into localSum and we save away (in maxSum) any maximal sum we have produced so far. MaxSum will either stay the same or increase; localsum will go up and down, but as long as it is positive there is a chance that values at the higher indices will push it above the current value of maxSum (so we keep going). Since the vector contains positive and negative values, it is perfectly possible that at some iteration, localSum assumes a negative value. Let's take a look at this scenario:*

maxSum = <some positive value, largest so far>

localsum = <negative number> that emerged after iteration k-1

vector values left to visit at indices   k, k+1, k+2,…. N

KEY INSIGHT: Out continued effort to find the maximal subsum for the entire vector is now equivalent to the problem of finding the maximal subsum for a vector with values

<negative number>,v[k],v[k+1],….. v[N]

And we have already established (see above) that a leading negative vector value has nothing to contribute to a maximal subsume that may yet result from values at v[k],v[k+1] …. Therefore, it is again safe to ignore <negative number> at index [k-1]. Just restart the localSum (i.e., reinitialize it to 0) and resume summing with index k …. When all values have been visited

once, going over them left to right, maxSum will inevitably hold the maximal subsum for the given number series. DONE! (If one wanted to do this more formally, it would a proof by induction.)