COMP 2080 Summer 2024 – Assignment 6

This assignment is due by 11:59pm on **Monday June 17**.

Your assignment will be submitted in Crowdmark and *not* in UM Learn. A Crowdmark invitation will be sent to your U of M email.

Proofs will be graded on both correctness and presentation. Clearly explain all of your steps.

The total number of points is 19.

Questions

Questions 1 – 3 are about the **maximum sum sub-array problem**, which we previously saw in Assignment 5. Let A be an integer array of length n. The entries in A can be positive, negative, or zero. We want to find the indices a, b, where $0 \leq a \leq b < n$, so that the sum

$$A[a] + A[a+1] + \cdots + A[b]$$

is maximized. For simplicity, we will only find the maximal sum S, not the indices a and b.

In assignment 5, we developed a divide-and-conquer algorithm to solve this problem. In particular, the maximum sum sub-array problem can be solved in $O(n \log n)$ steps using a divide-and-conquer algorithm (see the sample solutions if your algorithm took more than this!). Now, we would like to see if we can improve on this using dynamic programming.

First, we will develop a recursive, exhaustive solution. `maxSum(A,j)` will be a recursively defined function that returns the maximum sum of a (continuous) sub-array of A that ends at position j.

1. **[4 marks]** Write a pseudocode implementation of the function `maxSum(A,j)`. Do not introduce any dynamic programming yet, just implement the recursive strategy. Make sure you include the base case and the recursive cases.

2. **Memoization**

   (a) **[4 marks]** Now, we will write a memoization algorithm for the maximum sum sub-array problem. Write a function `maxSumMemo(A,j,T)` by adapting your pseudocode from Question 1 to use memoization (using T for the table instead of A, since A is the array). Then write a function `maxSum(A)`, which returns the maximum sum of a (continuous) sub-array of A overall, using `maxSumMemo()`.

   (b) **[2 marks]** What is the cost of your algorithm from part (a)? Briefly explain why (you do not need to prove it, an intuitive explanation is fine).

3. **Tabulation**

   (a) [**3 marks**] Finally, we will write a tabulation algorithm for the maximum sum sub-array problem. Write a function `maxSumTabulate(A)` that returns the maximum sum of a (continuous) sub-array of A overall, by adapting your pseudocode from Question 1 to use the tabulation strategy. Your code should start from the base case and systemically fill in an array T with answers to subproblems.

   (b) [**2 marks**] What is the cost of your algorithm from part (a)? Briefly explain why (you do not need to prove it, an intuitive explanation is fine).

4. Assume that the function `random(n)`returns an integer selected uniformly at random from the set $\{1, 2, \ldots, n\}$. For example, `random(4)` would return one of 1, 2, 3, or 4, all with equal probability. Also, for any $k \geq 1$, assume that the function `weightedCoin(k)` returns either 0 or 1. The value 0 occurs with probability $\frac{k-1}{k}$ and the value 1 occurs with probability $\frac{1}{k}$.

   Now, I have written the following (bad) function to add two positive integers:

```
// Pre: a,b >= 1
baddAdd(int a, int b):
    if(weightedCoin(a) == 0):
        return a + b
    else:
        n ← random(b)
        return a + n
```

   (a) [**1 mark**] Given inputs a and b, `badAdd(a, b)` returns **the correct answer** if the return value is $a+b$. What is the probability that `badAdd(a, b)` returns the correct answer?

   (b) [**3 marks**] Let X be the random variable that represents the value returned by the function call `badAdd(a, b)`. Calculate $E(X)$, the expected value of X.