CSCE 629

Jessica Salguero/UIN: 937005006 **Homework 1** January 31, 2025

Common Guidelines for submitting Homework & Project

- **No late homework/project submission will be accepted. Students should double check each submission to make sure correct files are submitted.**
- **By default, all work is solo; no collaboration allowed unless stated otherwise. Searching for solutions to homework problems from external resources (e.g., search online) is strictly forbidden.**
- Ensure clarity and conciseness in your work; avoid confusing and verbose explanations as they will not make any impact on your grade.
- Instead of using specific numerical examples, provide rigorous mathematical reasoning or a clear explanation demonstrating the correctness of your algorithm for all cases.
- In case of confusion, seek clarification from the TA or professor. DO NOT MAKE ASSUMPTIONS ABOUT EXPECTATIONS WHEN UNCERTAIN.
- You may use the provided LaTeX template to submit your work. Simply make a copy of this project on Overleaf, and if you encounter any problems using it, please contact the TA.

Problem Statement

Problem 1

Let S = (s1, s2, · · · , sn) be a sequence of n numbers. A “contiguous subsequence” of S is a subsequence si, si+1, · · · , sj (for some 1 i j n), which is made up of consecutive elements of S. (For example, if S = (6, 16, 29, 11, 4, 41, 11), then 16, -29, 11 is a “contiguous subsequence”, but 6, 16, 41 is not.) Given a sequence S, our objective is to find a contiguous subsequence whose sum if maximized. (For example, if S = (6, 16, 29, 11, 4, 41, 11), such a contiguous subsequence would be 11, -4, 41, 11, whose sum is 59.)

Our “Maximum-Sum Contiguous Subsequence Problem” is defined as follows:

Input: A sequence of numbers S = (s1, s2, · · · , sn).

Output: A contiguous subsequence of S whose sum is maximized.

Your task: Design an algorithm of time complexity O(n) for the above problem. (Remember: as we emphasized in class, whenever you design an algorithm, you need to: (1) explain the main idea of the algorithm, (2) present its pseudo-code, (3) prove its correctness, (4) analyze its time complexity. For dynamic programming, (1) and (3) can often be the same.) (Hint: For each j 1, 2, · · · , n, consider contiguous subsequences ending exactly at position j.)

Main Idea

When trying to find the maximum sum of contiguous numbers in a sequence, the simplest answer is to add together every possible combination of sequence subsets. However, as your initial sequence gets longer, this approach can take a very long time and get to a time complexity of O(n-squared).

For this specific homework problem, the most important aspect of the algorithm is to maintain the time complexity at O(n). This means that each item in the sequence can only be looked at once. With dynamic programming in mind, the key to solving this problem was to break down the main issue into smaller and smaller problems. That means finding ways to make this one large sequence actually seem like smaller, more manageable sequences throughout each step.

While iterating through the sequence, trying to find the overall max sum of items in the sequence broke down into two main issues:

- Should we continue adding numbers to the current sum of contigous numbers?
- Or should we restart and begin adding together a new subset of sequence values?

This then broke down to additional problems:

- What criteria should be used to determine if adding the next value in the sequence is a good choice?
- Is there a current cumulative sum that would suggest that restarting the contiguous sum is the better choice?

Possible brainstorming solutions included:

- What if the cumulative sum is less than the next number in the sequence?
  - However, this wouldn't work in all cases, because an all positive sequence could have numbers in non-numeric order but still want to keep adding them together.
- What if the cumulative sum is a negative value?
  - This seemed like the right path to venture down. If the cumulative sum is negative, then continuing to add (value x) to the cumulative sum would result in a number lower than the original (value x). This is when it's probably better to just restart the contiguous sum.

After figuring out the issue of determining when it is ideal to stop adding new items to the contiguous sum, another important issue was keeping track of the maximum sum regardless of where it is within the original sequence. That's where the important of global variables comes in, because you need to be able to keep track of the previous contiguous sums so that you don't have to revisit the previous numbers in the sequence.

Pseudocode

Algorithm 1 Maximum-Sum Contiguous Subsequence

```
function MAX CONTIGUOUS SUM(void)
    Initialize global variables prior to beginning loop:
        • (current i) is equal to the first position in the sequence (either 0 or 1, depending on the indexing used with the language the algorithm is being written in)
        • (current j) is equal to the first position in the sequence
        • (max i) is equal to the first position in the sequence
        • (max j) is equal to the first position in the sequence
        • (max sum) is equal to the first value (S1) in the sequence
        • (current sum) is equal to the first value in the sequence
    Loop through each of the individual positions in the sequence (range will be from 2nd position in the sequence to the last position in the sequence):
        • Is (current sum) a negative number?
            – Yes, (current sum) is negative.
                ∗ set (current i) equal to the current position in the sequence
                ∗ set(current j) equal to the current position in the sequence
                ∗ set (current sum) equal to the value of this specific position in the sequence
                ∗ Is (current sum) smaller than (max sum)?
                    · If (current sum) is smaller than (max sum), restart the contiguous sequence you're adding together. Make no adjustments to any max values.
                    · If (current sum) is larger than (max sum), restart the contiguous sequence you're adding together. Do make adjustments to max values
                    · set (max i) equal to the current position in the sequence
                    · set (max j) equal to the current position in the sequence
                    · set (max sum) equal to the value corresponding to this specific position in the sequence
            – No, (current sum) is a positive number. Continue adding to the running sum
                ∗ add current value in the sequence to (current sum)
                ∗ set (current j) equal to the position in the sequence
                ∗ Is (current sum) larger than (max sum)?
                    · Yes, (current sum) is larger than (max sum)
                    · set (max sum) equal to (current sum)
                    · set (max i) equal to (current i)
                    · set (max j) equal to (current j)
                    · No, (current sum) is not larger than (max sum)
                    · Do not adjust any values stored in the global variables
    After loop is finished: Print the subset of the sequence that spans from positions (max i) to (max j). Also print the value stored in (max sum).
end function
```

Proof of Correctness

Clearly articulate the key steps and reasoning behind your algorithm's correctness. Prove that your algorithm always finds a correct solution for all possible instances. **Specific examples may be used to help explain your proof, but they cannot replace the rigorous proof.**

The key steps involved in this algorithm are:

- Choosing if you should keep adding to the current sum or start a new sum of contiguous elements in the sequence.
  - Checking if the current sum is negative is a good criteria for checking this case. If the current sum is negative, then adding the next element of (value x) will only result in your current sum now being less than the original (value x); this also means that there is no chance of the max sum being greater than that (value x).
  - However, if you instead decide to start a new sum of contiguous, then the (value x) will start as the current sum. The next element in the sequence (value y) would then have the chance to continue to positively add to that current sum, without the previous elements in the sequence negatively impacting the sum.
  - I walked through this logic in Figure 1 and walked through part of an example in FIgure 2.
- Keeping track of the maximum sum, regardless of where it occurs in the sequence.
  - Keeping global variables with the current working sum and the highest recorded sum makes it easier to conduct comparisons between the two values.
  - Additional global variables for keeping track of the current and max starting (i) and end (j) points also for the max subset and sum to be printed at the end of the function.

Time Complexity

We often look at the outermost loop to help calculate the time complexity. Since this algorithm contains only one loop that goes through the entire sequence of length (n) once, there will only be (n) loops. This results in the desired time complexity of O(n).

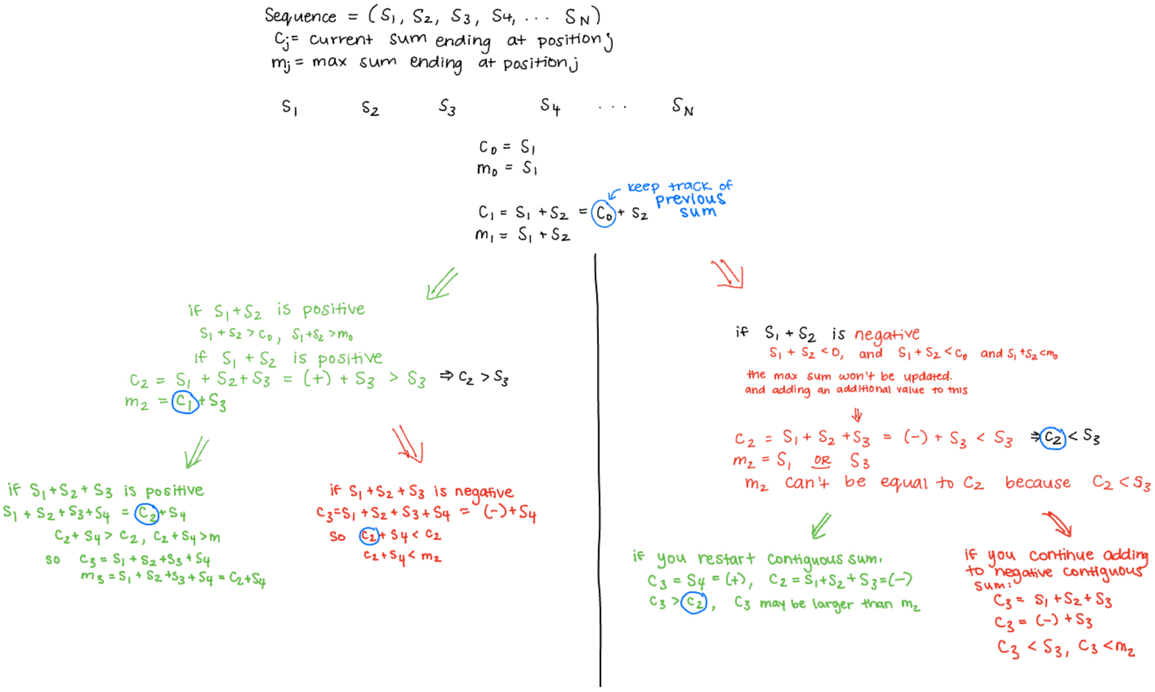
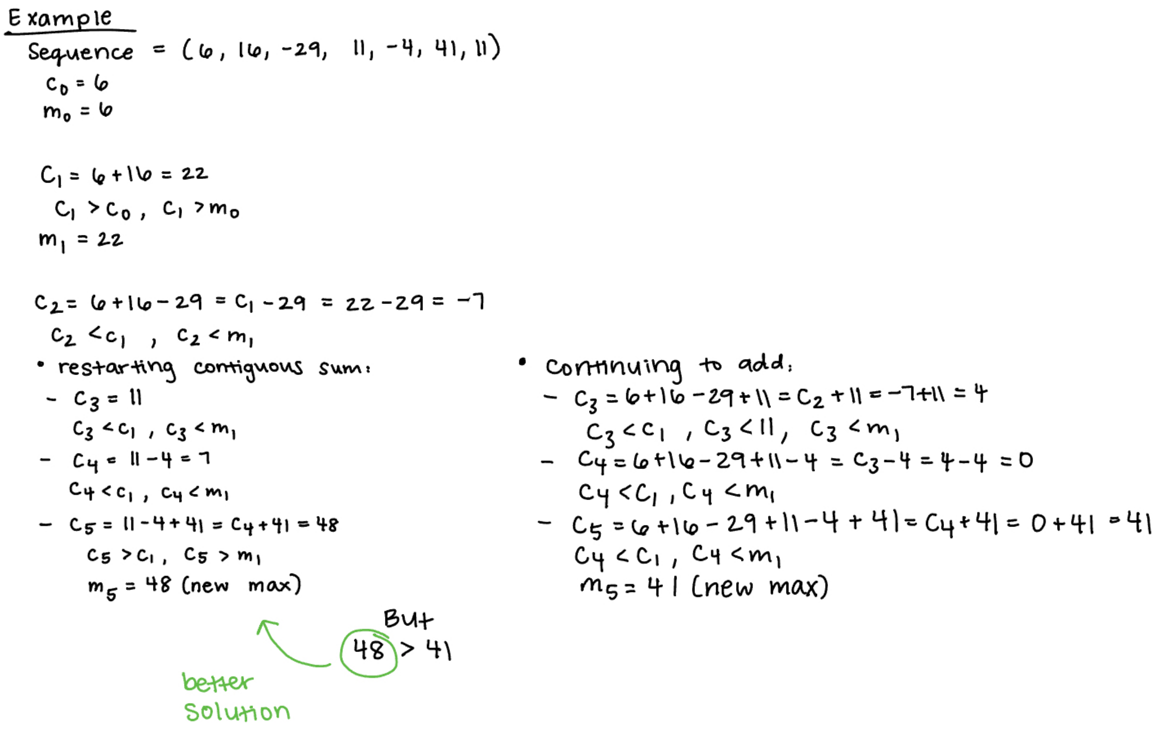

Sequence = (S1, S2, S3, S4, . . . SN)
Cj= current sum ending at position j
mj = max sum ending at position j
S1 S2 S3 S4 . . . SN
C0 = S1
m0 = S1
C1 = S1 + S2 = C0 + S2
keep track of previous sum
m1 = S1 + S2
if S1+S2 is positive
S1 + S2 > C0 , S1+S2 > m0
if S1 + S2 is positive
C2 = S1 + S2+S3 = (+) + S3 > S3 ⇒ C2 > S3
m2 = C1 + S3
if S1+S2+ S3 is positive
S1 + S2 + S3+S4 = C2+S4
C2+ S4 > C2 , C2 + S4 > m
so C3 = S1 + S2 +S3 + S4
m3 = S1 + S2 +S3 + S4 = C2+S4
if S1 +S2 + S3 is negative
C3=S1 +S2 + S3 + S4 = (-)+S4
so C2+ S4 < C2
C2 + S4 < m2
if S1 + S2 is negative
S1 + S2 < 0, and S1 + S2 < C0 and S1 +S2 < m0
the max sum won't be updated.
and adding an additional value to this
C2 = S1 + S2 +S3 = (-) + S3 < S3 ⇒ C2 < S3
m2 = S1 OR S3
m2 can't be equal to C2 because C2 < S3
if you restart contiguous sum.
C3 = S4 = (+), C2 = S1+S2+S3=(-)
C3 > C2 , C3 may be larger than m2
if you continue adding to negative contiguous sum:
C3 = S1 +S2+ S3
C3 = (-) + S3
C3 < S3, C3 < m2

Figure 1: Logic flowchart



<u>Example</u>

Sequence = (6, 16, -29, 11, -4, 41, 11)

$c_0 = 6$
$m_0 = 6$

$c_1 = 6+16 = 22$
$c_1 > c_0$, $c_1 > m_0$
$m_1 = 22$

$c_2 = 6+16-29 = c_1 - 29 = 22-29 = -7$
$c_2 < c_1$, $c_2 < m_1$

- restarting contiguous sum:
  - $c_3 = 11$
    $c_3 < c_1$, $c_3 < m_1$
  - $c_4 = 11-4 = 7$
    $c_4 < c_1$, $c_4 < m_1$
  - $c_5 = 11-4+41 = c_4+41 = 48$
    $c_5 > c_1$, $c_5 > m_1$
    $m_5 = 48$ (new max)

- Continuing to add:
  - $c_3 = 6+16-29+11 = c_2+11 = -7+11 = 4$
    $c_3 < c_1$, $c_3 < 11$, $c_3 < m_1$
  - $c_4 = 6+16-29+11-4 = c_3 - 4 = 4-4 = 0$
    $c_4 < c_1$, $c_4 < m_1$
  - $c_5 = 6+16-29+11-4+41 = c_4+41 = 0+41 = 41$
    $c_4 < c_1$, $c_4 < m_1$
    $m_5 = 41$ (new max)

But $\boxed{48} > 41$

better solution

Figure 2: Example case