

CSE 101 Homework 4

Kreshiv Chawla, Brian Masse, Taira Sakamoto, Emily Xie, Annabelle Coles

October 31, 2025

1. Maximum consecutive sum

Here is an algorithm that, given an array of integers (not necessarily positive) $A[1..n]$, finds the maximum sum of a consecutive subarray, $\max_{1 \leq I \leq J \leq n} \sum_{K=I}^{K=J} A[K]$.

(a) Initialize $\text{MaxSumEndingAt}[1..n]$.

(b) $\text{MaxSumEndingAt}[1] = A[1]$

(c) For $J = 2$ to n do:

(d) $\text{MaxSumEndingAt}[J] = \max(A[J], A[J] + \text{MaxSumEndingAt}[J - 1])$

(e) $\text{MaxConsSum} = \text{MaxSumEndingAt}[1]$.

(f) For $J = 2$ to n do:

(g) $\text{MaxConsSum} = \max(\text{MaxConsSum}, \text{MaxSumEndingAt}[J])$

(h) Return MaxConsSum

(a) Give a mathematical formula for the value stored in $\text{MaxSumEndingAt}[J]$.

let S_j be $\text{MaxSumEndingAt}[J]$. Let A_i be $A[i]$

$$S_j = A_j + \max(S_{j-1}, 0)$$

(b) Prove that is actually what is stored there as a loop invariant . (10 points)

From the previous question, prove the loop invariant: $S_j = A_j + \max(S_{j-1}, 0)$

i. Base Case: Trivially, $S_1 = A_1 = A[1]$

ii. Loop Invariant: Assume for some $k, 1 < k < n, S_k = \text{MaxSumEndingAt}[k]$, show $S_{k+1} = \text{MaxSumEndingAt}[k + 1]$:

During the $k + 1$ iteration, there are 2 possibilities:

A. if $\text{MaxSumEndingAt}[k] > 0$:

$$\begin{aligned} &\implies \max(A[k + 1], A[k + 1] + \text{MaxSumEndingAt}[k]) \\ &= A[k + 1] + \text{MaxSumEndingAt}[k] \\ &= A_{k+1} + S_k = A_{k+1} + \max(S_k, 0) \\ &= S_{k+1} \end{aligned}$$

B. if $\text{MaxSumEndingAt}[k] \leq 0$:

$$\begin{aligned} &\implies \max(A[k + 1], A[k + 1] + \text{MaxSumEndingAt}[k]) \\ &= A[k + 1] + 0 \\ &= A[k + 1] + \max(0, \text{MaxSumEndingAt}[k]) \\ &= A_{k+1} \cdot \max(S_k, 0) \\ &= S_{k+1} \end{aligned}$$

- (c) Use this to prove the algorithm is correct. (5 points)

Based on the loop invariant, after i iterations, S_i is the largest consecutive sum ending at i .

After n iterations, all S_1, \dots, S_n are the largest consecutive sums ending at $1 \dots n$.

Because the largest consecutive sum must end at some $i \in 1 \dots n$, $\max(S_1, \dots, S_n)$, is the largest consecutive sum in the array.

- (d) Give a time analysis for this algorithm. (5 points)

Both loops run in $O(n)$ time. The work in the two loops is both constant time. Thus, the runtime for the entire algorithm is: $O(2n) \in O(n)$

2. Maximum within sliding window

You are given an array $A[1..n]$ of integers, and an integer $1 \leq k \leq n$. You wish to compute $\text{MaxInWindow}[I] = \max_{I \leq J \leq \min(I+k, n)} A[J]$ for all $1 \leq I \leq n$. By answering the questions below, you'll develop an efficient algorithm for this problem. A helpful abstraction will be to consider a window set at each I , $WS[I] = \{(J, A[J]) | I \leq J \leq \min(I + k, n)\}$.

In terms of the window set at I , what is $\text{MaxInWindow}[I]$? (2 points)

What is the difference between $WS[I]$ and $WS[I + 1]$? (3 points)

Based on your answers to the first two questions, what data structure operations would we want a data structure for WS to support? (3 points)

What data structure could we use for WS ? (3 points)

What is the maximum size of this data structure? (2 points)

How long do the different data structure operations take? (3 points)

Give pseudo-code for an algorithm to solve this problem using data structure operations. (5 points)

Give a time analysis for your algorithm, using answers to previous questions.(4 points).

3. Subway stops

Underneath a city road, there are n subway stops s_1, \dots, s_n and k subway lines. Each subway line k stops at some of the stops, in the same order, with line k stopping at $s_{i_{k,1}}, \dots, s_{i_{k,t_k}}$, where $i_{k,1} < i_{k,2} \dots < i_{k,t_k}$. You want to go from s_1 to s_n making as few transfers between lines as possible.

For example, there might be five stops, and three lines. The first line might stop at stops 1, 3, the second at 2, 4, 5, and the third at 2, 3, 4. Then we could board on line 1, take it to stop 3, switch to line 3, take it to stop 4, switch to line 2 and take it to the end.

Below, you will describe how to use a graph algorithm from class to solve the problem.

- (a) What graph will you use to solve the problem? Be sure to specify the set of vertices in your graph, the set of edges, whether the edges are directed or undirected, and what weights edges have, if any. (5 points)
- (b) How will you create the graph from the information given? What format will you use for the graph? How long does it take to create the graph? (3 points)
- (c) How many vertices does your graph have, at most? Give this in terms of n and k . (2 points)
- (d) How many edges does your graph have, at most? Give this in terms of n and k (3 points)
- (e) How do paths in your graph relate to ways of transferring? What is the relationship between the length of paths and the number of transfers? (4 points)
- (f) What algorithm from class will you run on the graph? Be sure to specify all inputs to this algorithm, and say how you use the results. (4 points)
- (g) What is the total time complexity of using this algorithm from class to solve the subway transfer problem? This should be given in terms of n and k . (4 points)

4. Vertex costs

Say we are given a graph G where both vertices and edges have positive integer weights and two vertices u, v . We want to find a path from u to v that minimizes the total weights of both edges and vertices along the path. Give an efficient algorithm to solve this problem. You can use any algorithm from class as given, but need to relate the correctness guarantee proved for that algorithm to correctness for this new problem. (10 points clear algorithm description, 10 points correctness argument, 5 points time analysis and efficiency)