

CS 498 MCS Algorithms ✧ Fall 2026

🌀 Homework 4 🌀

Due **Thursday, October 1, 2026** at 9pm Central Time

- **Submit your solutions electronically on Gradescope as PDF files.**
 - Submit a separate PDF file for each numbered problem.
 - You can find a $\text{\LaTeX}$ solution template on the course web site, which we encourage you to use to typeset your homework.
 - If you plan to submit scanned handwritten solutions, please use dark ink (not pencil) on white unlined paper (not notebook or graph paper), and use a scanner or a scanning app to create a high-quality PDF for submission (not a raw photo). We reserve the right to reject submissions that are difficult to read.
 - If you plan to use a tablet and a note-taking app, please make sure your submitted PDF is broken into standard US-letter sized pages (not a long scroll).
- For each numbered problem, your solution must include three explicit declarations.
 - **Sources:** Every source you consulted, either paper or electronic, excluding the official course materials. (If you did not consult any external sources, write “None.”)
 - **Collaborators:** Every person you worked with, excluding the instructors and the homework group listed on the first page of your solution. (If you did not work with any other people, write “None.”)
 - **AI use:** An explicit list of every AI tool that you used, and exactly how you used it. (If you did not use any AI tools, write “None.”)

Solutions without all three declarations will not be graded.

See the course web site for more information.

If you have any questions about these policies,
please don't hesitate to ask in lecture, in office hours, or online.

Required problems

- Suppose you are given an array $A[1..n]$ and you are asked to compute the length of a longest **palindrome** subsequence of A . Recall that a sequence $B[1..\ell]$ is a *palindrome* if $B[i] = B[\ell - i + 1]$ for every index i . For example, given the array

$$A = \langle \underline{3}, 1, 4, 1, 5, 9, \underline{2}, 6, 5, \underline{3}, 5, 8, \underline{9}, \underline{7}, \underline{9}, \underline{3}, \underline{2}, \underline{3}, 8, 4, 6, 2, 7 \rangle$$

your algorithm should compute the integer 9, because $\langle 3, 2, 3, 9, 7, 9, 3, 2, 3 \rangle$ is a palindrome subsequence of A , and no other palindrome subsequence of A is longer. (There are several other palindrome subsequences of length 9.)

- Design a *recursive* algorithm that solves this problem. Don't worry about the running time of your algorithm.
- Identify the subproblem structure in your recursive algorithm for part (a). How many subproblems are there for an input array of size n ?

Now consider the following variant. The given array A contains non-negative integers, you want to compute the length of a longest palindromic subsequence **whose elements sum to a multiple of 3**. For example, given the array

$$\langle 3, 1, 4, 1, 5, 9, \underline{2}, \underline{6}, 5, 3, 5, \underline{8}, \underline{9}, \underline{7}, \underline{9}, 3, 2, 3, \underline{8}, 4, \underline{6}, \underline{2}, 7 \rangle$$

your algorithm should return 9, because $\langle 2, 6, 8, 9, 7, 9, 8, 6, 2 \rangle$ is the longest palindromic subsequence whose sum $2 + 6 + 8 + 9 + 7 + 9 + 8 + 6 + 2 = 57$ is divisible by 3.

- Design a *recursive* algorithm that solves this problem. Again, don't worry about the running time of your algorithm.
 - Identify the subproblem structure in your recursive algorithm from part (c). How many subproblems are there for an input array of size n ?
- Again, recall that a string is called a **palindrome** if it is equal to its reversal, like **I**, **POOP**, **REDIVIDER**, **AMANAPLANACANALPANAMA**, or the empty string.

Any string can be decomposed into a sequence of palindromes. For example, the string **BUBBASEESABANANA** ("Bubba sees a banana.") can be broken into palindromes in the following ways (and 65 others):

$$\begin{aligned} & \text{BUB} \bullet \text{BASEESAB} \bullet \text{ANANA} \\ & \text{B} \bullet \text{U} \bullet \text{BB} \bullet \text{ASEESA} \bullet \text{B} \bullet \text{ANANA} \\ & \text{BUB} \bullet \text{B} \bullet \text{A} \bullet \text{SEES} \bullet \text{ABA} \bullet \text{N} \bullet \text{ANA} \\ & \text{B} \bullet \text{U} \bullet \text{BB} \bullet \text{A} \bullet \text{S} \bullet \text{EE} \bullet \text{S} \bullet \text{A} \bullet \text{B} \bullet \text{A} \bullet \text{NAN} \bullet \text{A} \\ & \text{B} \bullet \text{U} \bullet \text{B} \bullet \text{B} \bullet \text{A} \bullet \text{S} \bullet \text{E} \bullet \text{E} \bullet \text{S} \bullet \text{A} \bullet \text{B} \bullet \text{A} \bullet \text{N} \bullet \text{A} \bullet \text{N} \bullet \text{A} \end{aligned}$$

Describe and analyze an algorithm to find the smallest number of palindromes that make up a given input string. For example, given the input string **BUBBASEESABANANA**, your algorithm should return 3.

3. *Vankin's Mile* is an American solitaire game played on an $n \times n$ square grid. The player starts by placing a token on any square of the grid. Then on each turn, the player moves the token either one square to the right or one square down. The game ends when player moves the token off the edge of the board. Each square of the grid has a numerical value, which could be positive, negative, or zero. The player starts with a score of zero; whenever the token lands on a square, the player adds its value to his score. The object of the game is to score as many points as possible.

For example, given the grid below, the player can score $8 - 6 + 7 - 3 + 4 = 10$ points by placing the initial token on the 8 in the second row, and then moving down, down, right, down, down. (This is *not* the best possible score for these values.)

-1	7	-8	10	-5
-4	-9	8	-6	0
5	-2	-6	-6	7
-7	4	7	-3	-3
7	1	-6	4	-9

- (a) Describe and analyze an efficient algorithm to compute the maximum possible score for a game of Vankin's Mile, given the $n \times n$ array of values as input.
- (b) In the Canadian version of this game, appropriately called *Vankin's Kilometer*, the player can move the token either one square down, one square right, or one square left in each turn. However, to prevent infinite scores, the token cannot land on the same square more than once. Describe and analyze an efficient algorithm to compute the maximum possible score for a game of Vankin's Kilometer, given the $n \times n$ array of values as input.¹

Additional practice problems (not for submission)

Feel free to ask us about these problems after class, in office hours, or on Ed Discussion!

4. It's almost time to show off your flippin' sweet dancing skills! Tomorrow is the big dance contest you've been training for your entire life, except for that summer you spent with your uncle in Alaska hunting wolverines. You've obtained an advance copy of the list of n songs that the judges will play during the contest, in chronological order. Yessssssss!

You know all the songs, all the judges, and your own dancing ability extremely well. For each integer k , you know that if you dance to the k th song on the schedule, you will be awarded exactly $Score[k]$ points, but then you will be physically unable to dance for the next $Wait[k]$ songs (that is, you cannot dance to songs $k + 1$ through $k + Wait[k]$). The dancer with the highest total score at the end of the night wins the contest, so you want your score to be as high as possible.

Describe and analyze an efficient algorithm to compute the maximum total score you can achieve. The input to your sweet algorithm is the pair of arrays $Score[1..n]$ and $Wait[1..n]$.

¹If we also allowed upward movement, the resulting game (Vankin's Fathom?) would be NP-hard.

5. Describe and analyze an algorithm to compute the length of a longest **convex** subsequence of a given array $A[1..n]$ of integers. A sequence $B[1..\ell]$ is *convex* if $B[i] - B[i-1] > B[i-1] - B[i-2]$ for every index $i \geq 3$.

For example, given the array

$\langle \underline{3}, \underline{1}, 4, \underline{1}, 5, 9, \underline{2}, 6, 5, 3, \underline{5}, 8, \underline{9}, 7, 9, 3, 2, 3, 8, 4, 6, 2, 7 \rangle$

your algorithm should return the integer 6, because $\langle 3, 1, 1, 2, 5, 9 \rangle$ is a longest convex subsequence (one of many).

6. Suppose you are given an array $A[1..n]$ of numbers, which may be positive, negative, or zero, and which are **not** necessarily integers.

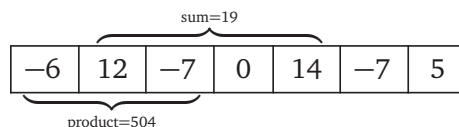
- (a) Describe and analyze a *dynamic programming* algorithm that finds the largest sum of elements in a contiguous subarray $A[i..j]$.

This has been a standard computer science interview question since at least the mid-1980s; it even has its own [Wikipedia page](#). Don't just regurgitate the well-known solution; instead, *derive* the algorithm from scratch, following the dynamic programming design process.

- (b) Suppose you are also given a positive integer $L \leq n$. Describe and analyze a dynamic programming algorithm that finds the largest sum of elements in a contiguous subarray $A[i..j]$ with length at least L , meaning $j \geq i + L - 1$.

- (c) Describe and analyze a dynamic programming algorithm that finds the largest *product* of elements in a contiguous subarray $A[i..j]$.

For example, given the array $[-6, 12, -7, 0, 14, -7, 5]$ as input, your algorithm from part (a) should return 19, and your algorithm from part (c) should return 504.



Given the one-element array $[-67]$ as input, your algorithm from part (a) should return 0, and your algorithm from part (b) should return 1. (The empty interval is still an interval!) For purposes of analysis, assume that comparing, adding, or multiplying any two numbers takes $O(1)$ time.

7. A *shuffle* of two strings X and Y is formed by interleaving the characters into a new string, keeping the characters from each string in their original order. For example, the string **BANANAANAS** is a shuffle of the strings **BANANA** and **ANANAS** in several different ways.

BANANA**ANANAS** **BAN****ANA****ANANAS** **B****AN****AN****A****ANANAS**

Similarly, the strings **PRODGYRNAMAMMIINCG** and **DYPRONGARMAMMicing** are both shuffles of **DYNAMIC** and **PROGRAMMING**:

PRO**D**G**Y**R**N**AM**M**I**N**C**G**

DY**P**RO**N**G**A**R**M**AM**M**I**C**ING

- (a) Given three strings $A[1..m]$, $B[1..n]$, and $C[1..m+n]$, describe and analyze an algorithm to determine whether C is a shuffle of A and B .
- (b) A *smooth* shuffle of X and Y is a shuffle of X and Y that never uses more than two consecutive symbols of either string. For example,
- **P**R**D**O**Y**G**N**A**R**A**M**M**I**C**N**G is a smooth shuffle of the strings **DYNAMIC** and **PROGRAMMING**.
 - **D**Y**P**R**N**O**G**R**A**A**M**M**I**C**I**NG is a shuffle of **DYNAMIC** and **PROGRAMMING**, but it is not a smooth shuffle (because of the substrings **OGR** and **ING**).
 - **XX****X****X****X****XX****XX****XX****X****X** is a smooth shuffle of **XXXXXX** and **XXXXXXXXXX**.
 - The strings **XXXX** and **XXXXXXXXXXXX** have no smooth shuffles.

Describe and analyze an algorithm to decide, given three strings X , Y , and Z , whether Z is a smooth shuffle of X and Y .

The colors and offsets in the examples above are only meant to show you how the strings are shuffled; this is not part of the input data. For example, given the input strings $X = \text{DYNAMIC}$, $Y = \text{PROGRAMMING}$, and $Z = \text{DYPRNOGRAAMMIMING}$, your algorithm for part (a) should return **TRUE**, but your algorithm for part (b) should return **FALSE**.