You have 120 minutes to answer five questions.

**Write your answers in the separate answer booklet.**

Please return this question sheet and your cheat sheet with your answers.

1. (a) Solve the following recurrences:
   - $A(n) = A(2n/3) + O(\sqrt{n})$
   - $B(n) = 8B(n/4) + O(n^{3/2})$
   - $C(n) = C(n/2) + C(n/3) + O(n)$

   (b) Describe an appropriate memoization structure and evaluation order for the following (meaningless) recurrences, and state the running time of the resulting iterative algorithm to compute the requested function value.
   - Compute $Pleb(n, 1)$ where

$$Pleb(i,k) = \begin{cases} i & \text{if } k \le 0 \\ k & \text{if } i > n \\ \max \left\{ \begin{array}{c} i + k + Pleb(i-1, k+1) \\ i + Pleb(i-1, k) \\ k + Pleb(i, k+1) \end{array} \right\} & \text{otherwise} \end{cases}$$

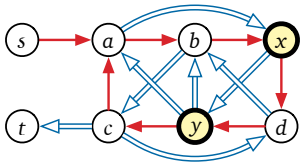
   - Compute $Nom(1, n)$ where

$$Nom(i,k) = \begin{cases} 0 & \text{if } k = i \\ (X[k] - X[i]) + \min \left\{ \begin{array}{l} Nom(i,j) \\ + Nom(j,k) \end{array} \middle| \; i < j < k \right\} & \text{otherwise} \end{cases}$$

2. Suppose you are given a directed graph $G$ in which every edge is either red or blue, and a subset of the vertices are marked as *special*. A walk in $G$ is *legal* if color changes happen only at special vertices. That is, for any two consecutive edges $u \to v \to w$ in a legal walk, if the edges $u \to v$ and $v \to w$ have different colors, the intermediate vertex $v$ must be special.

   Describe and analyze an algorithm that either returns the length of the shortest legal walk in $G$ from vertex $s$ to vertex $t$, or correctly reports that no such walk exists.

   For example, if you are given the following graph as input (where single arrows are "red" and double arrows are "blue"), with special vertices $x$ and $y$, your algorithm should return the integer 8, which is the length of the shortest legal walk $s \to x \to a \to b \to x \Rightarrow y \Rightarrow b \Rightarrow c \Rightarrow t$. The shorter walk $s \to a \to b \Rightarrow c \Rightarrow t$ is not legal, because vertex $b$ is not special.

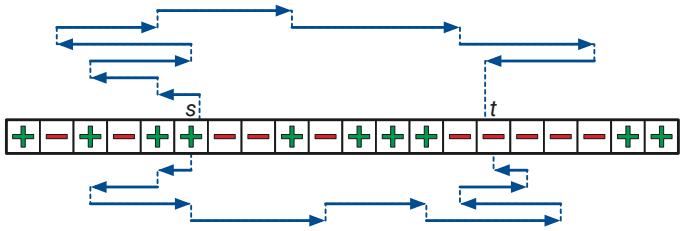



*Problem 3 is on the back of this page.*

3. ***Ink-deck*** is a solitaire puzzle game played on a row of $n$ squares, each marked with either $+$ or $-$. Your goal is to move a token from a specified start square $s$ to a specified target square $t$ using a sequence of *moves*. Each move translates the token either left or right along the row to a new square. The *length* of a move is the distance that the token moves; for example, a move from square 5 to square 12 has length 7. Moves are subject to the following rules:

   - The first move must have length 1.
   - If the token is on a square marked $+$, your next move must be one square longer than your previous move.
   - If the token is on a square marked $-$, your next move must be one square shorter than your previous move. (In particular, if your previous move had length 1, then you cannot move at all!)
   - You are never allowed to move the token off either end of the row.

   The following figure shows an example ink-deck puzzle, along with two solutions.




An ink-deck puzzle with two solutions.

   (a) The *total length* of a solution is the sum of the lengths of the moves. For example, the top solution in the figure above has total length $1+2+3+4+3+4+5+4+3 = 29$, and the bottom solution has total length $1+2+3+4+3+4+3+2+1 = 23$.

   Describe an algorithm that, given an ink-deck puzzle, either finds a solution whose *total* length is as *small* as possible, or correctly reports that there is no solution.

   (b) The *maximum length* of a solution is length of its longest move. For example, the top solution above has maximum length 5, and the bottom solution has maximum length 4.

   Describe an algorithm that, given an ink-deck puzzle, either finds a solution whose *maximum* length is as *large* as possible, or correctly reports that there is no solution.

Your input to both algorithms consists of an array $ID[1..n]$, where $ID[i] \in \{-1, +1\}$ for each index $i$, along with two indices $1 \le s \le n$ and $1 \le t \le n$.

*Problems 4 and 5 are on the next page.*

4. Recall that an *arithmetic progression* is any sequence of real numbers $x_1, x_2, \ldots, x_n$ such that $x_{i+1} - x_i = x_i - x_{i-1}$ for every index $2 \le i \le n-1$.

   Suppose we are given a sorted array $X[1..n]$ containing an arithmetic sequence *with one element repeated once*. Describe and analyze an algorithm to find the repeated element as quickly as possible.

   For example, given the input array $X = [2, 4, 6, 6, 8, 10, 12]$, your algorithm should return 6, and given the input array $X = [1, 1, 1, 1]$, your algorithm should return 1.

5. Suppose you are given a string of symbols, representing a message in some foreign language that you do not understand, in an array $T[1..n]$. You have access to a black-box subroutine IsWord that takes a string $w$ as input and decides in $O(|w|)$ time whether $w$ is a word.

   You eagerly implement and run the text-splitting algorithm we saw in class, only to discover that the given string *cannot* be split into words! Apparently, as a crude form of cryptography, the author of the message added extra symbols at the beginning and end.

   Describe and analyze an algorithm to find the length of the *longest substring* of $T$ that can be split into words.

   For example, suppose IsWord($w$) returns True if and only if $w$ is an English word with at least four letters. Given the input string STURDYNAMICEXTRAPROGRAMBLE, your algorithm should return the integer 19, which is the length of the substring DYNAMICEXTRAPROGRAM, which can be split into the words DYNAMIC, EXTRA, and PROGRAM:

   STUR | DYNAMIC | EXTRA | PROGRAM | BLE

   The words STURDY, MICE, TRAP, and RAMBLE have larger total length 20, but there are gaps between them; so they can't be formed by splitting a *substring* of $T$.

   | STURDY | NA | MICE | X | TRAP | ROG | RAMBLE |

*Nothing to see here.*