You have 120 minutes to answer four questions.

Write your answers in the separate answer booklet.

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

1. Suppose we are given an array $A[1..n]$ of n distinct integers, which could be positive, negative, or zero, sorted in increasing order.

   (a) Describe a fast algorithm that either computes an index i such that $A[i] = i$ or correctly reports that no such index exists.

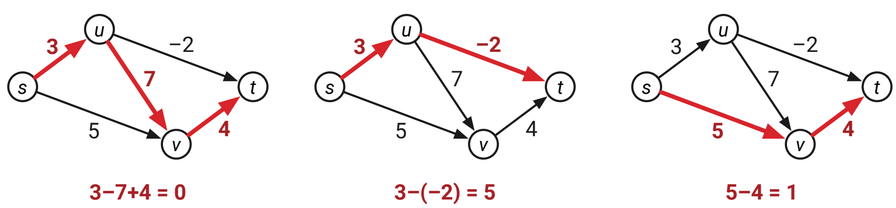
   (b) Suppose we know in advance that $A[1] > 0$. Describe an even faster algorithm that either computes an index i such that $A[i] = i$ or correctly reports that no such index exists.

2. Let G be a directed **acyclic** graph, in which every edge $e \in E$ has a weight $w(e)$, which could be positive, negative, or zero. We define the ***alternating length*** of any path in G to be the weight of the first edge, ***minus*** the weight of the second edge, ***plus*** the weight of the third edge, and so on. More formally, for any path $P = v_0 \to v_1 \to v_2 \to \cdots \to v_\ell$ in G, we define

$$AltLen(P) = \sum_{i=0}^{\ell-1} (-1)^i \cdot w(v_i \to v_{i+1}).$$

   Describe an algorithm to find a path from s to t with the largest alternating length, given the graph G, the edge weights $w(e)$, and vertices s and t as input.

   For example, given the graph shown below, your algorithm should return 5, which is the alternating length of the path $s \to u \to t$.



[Questions 3 and 4 are on the back.]

3. Suppose we are given an array $B[1..n]$ of n boolean values, which is *supposed* to be sorted (all FALSES before all TRUES) but isn't. An *inversion* in B is a pair of indices (i, j) such that $i < j$ and $B[i] = \text{TRUE}$ and $B[j] = \text{FALSE}$.

   (a) Describe and analyze an efficient algorithm to compute the number of inversions in the input array B.

   (b) The *length* of an inversion (i, j) is the difference $j - i$. By definition, the length of an inversion is an integer between 1 and $n-1$. Describe and analyze an efficient algorithm to compute, for every integer $1 \le \ell \le n-1$, the number of inversions of length ℓ in the input array B.

   For example, if the input array is $B = [T, F, F, T, F, F, T]$, your first algorithm should return the integer 6, and your second algorithm should return the array $[2, 2, 0, 1, 1, 0]$.

4. The StupidScript language includes a binary operator `@` that computes the *average* of its two arguments. For example, the StupidScript code `print(3 @ 6)` would print `4.5`, because $(3+6)/2 = 4.5$.

   Expressions like 4 @ 7 @ 3 that use the @ operator more than once yield different results when they are evaluated in different orders:

   $$(4 \,@\, 7) \,@\, 3 = 5.5 \,@\, 3 \;=\; 4.25 \qquad \text{but} \qquad 4 \,@\, (7 \,@\, 3) = 4 \,@\, 5 \;=\; 4.5$$

   Here is a larger example:

   $$\begin{aligned} ((((8 \,@\, 6) \,@\, 7) \,@\, 5) \,@\, 3) \,@\, (0 \,@\, 9) &= 4.5 \\ ((8 \,@\, 6) \,@\, (7 \,@\, 5)) \,@\, ((3 \,@\, 0) \,@\, 9) &= 5.875 \\ (8 \,@\, (6 \,@\, (7 \,@\, (5 \,@\, (3 \,@\, 0))))) \,@\, 9 &= 7.890625 \end{aligned}$$

   Describe and analyze an algorithm to compute, given a sequence of integers separated by @ signs, the ***smallest*** possible value the expression can take by adding parentheses. Your input is an array $A[1..n]$ listing the sequence of integers.

   For example, if your input sequence is $[4, 7, 3]$, your algorithm should return 4.25, and if your input sequence is $[8, 6, 7, 5, 3, 0, 9]$, your algorithm should return 4.5. Assume all arithmetic operations (including @) can be performed exactly in $O(1)$ time.