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.

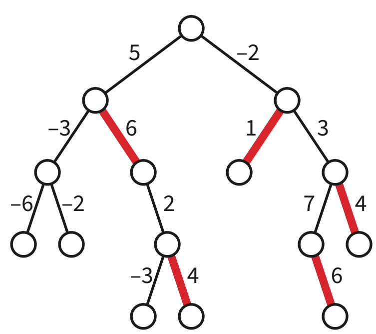
   (a) Describe a fast algorithm that either computes an index i such that $A[i] = i$ or correctly reports that no such index exists.

   (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. Suppose we are given a ***binary*** tree T with weighted edges; each edge weight could be positive, negative, or zero. A subset M of edges of T is called a *matching* if every vertex of T is incident to *at most* one edge in M.

   Describe and analyze an algorithm to find a matching in T with maximum total weight.

   For example, given the binary tree shown below, your algorithm should return the integer 21, which is the total weight of the indicated matching.



[Questions 3 and 4 are on the back.]

3. The *Hamming distance* between two bit strings is the number of positions where the strings have different bits. For example, the Hamming distance between the strings `01101001` and `11010001` is 4.

   Suppose we are given two bit strings $P[1..m]$ (the "pattern") and $T[1..n]$ (the "text"), where $m \leq n$. Describe and analyze an algorithm to find the minimum Hamming distance between P and a substring of T of length m. For full credit, your algorithm should run in $O(n \log n)$ time.

   For example, if $P =$ `1100101` and $T =$ `1111111010101000000`, your algorithm should return `1`, which is the Hamming distance between P and the substring `1110101` of T:

```
1111111010101000000
    1100101
```

   [Hint: Consider `0`s and `1`s separately.]

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:

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

   Here is a larger example:

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

   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.