Answer all the questions. Give a correctness proof and run time analysis for all the algorithms.
Problem 1: [15pts] A list \( A = \langle a_1, \ldots, a_n \rangle \) is said to have a majority element if more than half of its entries are the same. Consider the following procedure: Pair up the elements of \( A \) arbitrarily to get \( n/2 \) pairs (assume \( n \) is even). For each pair do the following: if the two elements of the pair are different, discard both of them, and if they are the same then keep just one of them.

- Show that at most \( n/2 \) elements are left after this procedure, and that they have a majority element if \( A \) does.
- Use this procedure to describe a linear-time algorithm to determine whether \( A \) has a majority element, and, if so, return that element.
Problem 2: [20pts] Let $G = (V, E)$ be a directed connected graph, with $|V| = n$ and $|E| = m$, $c : E \to \mathbb{R}$ the edge-weight function ($c(e)$ may be negative), and $t \in V$ a vertex. For each vertex $v \in V$, a value $\Delta(v)$ (which can be any real number) is also given.

Describe a linear-time algorithm to verify whether $\Delta(v)$ is the cost of the shortest path from node $v$ to $t$, for all $v \in V$. Note that you are not being asked to compute the cost of the shortest paths; you are just being asked to verify.
Problem 3: [25pts] A quack is a data structure combining properties of both stacks and queues. It can be viewed as a list of elements written left to right such that three operations are possible:

- QUACKPUSH(x): add a new item x to the left end of the list;
- QUACKPOP(): remove and return the item on the left end of the list;
- QUACKPULL(): remove the item on the right end of the list.

Implement a quack using three stacks and \( O(1) \) additional memory, so that the amortized time for any QUACKPUSH, QUACKPOP, or QUACKPULL operation is \( O(1) \). In particular, each element in the quack must be stored in exactly one of the three stacks. The component stacks can be accessed only through the standard stack functions PUSH and POP.

(If you can’t solve the above problem, to get partial credit show how to implement QUACKPUSH(x) and QUACKPULL using two stacks so that each of the two operations takes \( O(1) \) time.)
Problem 4: [25pts] A sequence is called a palindrome if it is the same as its reversal, e.g., RACECAR or AMANAPLANACAPANALPANA. Describe an \(O(n^2)\)-time algorithm that takes a sequence \(A = a_1 \cdots a_n\) (stored in an array) as an input and returns the length of the longest palindrome subsequence in \(x\). For example, the longest palindrome subsequence in \text{MAHDYNOAMICPROGRAMZLETMESHOWYOUTHHEM} is \text{MHYMRORMYHM}.

(Hint: For \(i \leq j\), let \(L(i, j)\) be the length of the longest palindrome subsequence in \(a_i \cdots a_j\). Write a recurrence relation for \(L(i, j)\) and use dynamic programming.)
Problem 5: [15pts] A kite is a graph on an even number of vertices, say $2n$, in which $n$ of the vertices form a clique and the remaining $n$ vertices are connected in a tail that consists of a path joined to one of the vertices of the clique. Let

$$KITE = \{(G, k) \mid G \text{ contains a kite of size } 2k \text{ as a subgraph}\}.$$ 

Show that $KITE$ is NP-Complete.