1. In a knock out tennis tournament with n players, how many matches are required to find the winner? □

2. An array consisting of positive integers is given (assume that no two numbers are the same). We want to find the second largest number in the array. □

    1. Suppose your program is given as inputs a) a positive integer n, b) an array A of n numbers and c) a "candidate" number x. Your program must **verify** whether x is the second largest number in A. Can you do the verification with at most n *comparisons*? Your algorithm must return 1 if x is the second largest number in A, and must return 0 otherwise.
    2. Suppose your inputs are only a) a positive integer n and b) an array A of n numbers (no "candidate" is given now!), can you **solve** the problem of finding the second largest number in A with $2n-3$ comparisons?
    3. Can you **solve** the problem of finding the second largest number in A with less than $2n-3$ comparisons?

3. A group of n people have to elect a leader. A leader is a person in the group with support from $> \frac{n}{2}$ people within the group. An electronic voting is being done to determine the leader. The voting program contains an array A of n elements and the entry $A[i]$ is set to the value j, if the i^{th} voter has voted in favour of candidate j. For example, if the second voter has voted for candidate 10, then $A[2] = 10$. Note that it is possible that no person has enough votes to be a leader; but more than one person can never get the majority for leadership. That is, if a leader exists, then (s)he is unique. □

    1. The inputs to your program are: a) a positive integer n, b) an array A of n numbers and c) a "candidate" j (again, a positive integer). You want to **verify** whether j is the leader. Your algorithm must return 1 if j is the leader and return 0 otherwise. Can you design a verification algorithm that requires at most n comparisons?
    2. Can you **solve** the problem of finding the leader? That is, your inputs are a) a positive integer n and b) an array A of n numbers. Your algorithm must return the leader if one exists; and must return 0 otherwise. The number of comparisons your algorithm takes must be $O(n)$.
    3. Suppose now that you are given the following **constraint**: You are not permitted to declare any array in your program (of course, the array A which is part of the input is given, but you are not permitted to write into A and tamper its values). Can you **solve** the problem, still requiring only $O(n)$ comparisons?

4. This question studies the CNFSAT problem (Conjunctive Normal Form Boolean Satisfiability problem). □

    1. Is the following boolean formula in conjunctive normal form (CNF), **satisfiable?** If so, find all truth assignments that satisfy the formula. The formula is:
    $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3)$.
    2. Design an algorithm that takes as inputs: a) a positive integer n, b) a boolean formula ϕ in n variables $x_1, x_2, \ldots x_n$ and c) a "candidate" truth assignment τ to n variables (for example, if $n = 3$, $\tau = (T, F, T)$ is a truth assignment to 3 variables that sets $x_1 = T, x_2 = F, x_3 = T$). Your algorithm must **verify** whether τ is a truth assignment that satisfies ϕ. That is, if τ is a truth assignment to n variables that satisfies ϕ then your algorithm must return 1. Otherwise, your algorithm must return 0. (You need not write the whole code - just describe the strategy). What will be the time complexity of your algorithm?
    3. Design an algorithm that takes as input a) a positive integer n and b) a boolean formula ϕ in n variables $x_1, x_2, \ldots x_n$. Your algorithm must **solve** the satisfiability problem. That is, your algorithm must return 1 if ϕ is satisfiable and return 0 otherwise. What is the complexity of your algorithm?