1. (15 points) Solve the following recurrence relations.
   (a) \( T(n) = 3T(n/3) + cn \), \( T(1) = c \)
   (b) \( T(n) = 3T(n/3) + cn^2 \), \( T(1) = c \)
   (c) \( T(n) = 3T(n-1) + 1 \), \( T(1) = 1 \)

2. (20 points) Consider the following problem: You are given a pointer to the root \( r \) of a binary tree, where each vertex \( v \) has pointers \( v.lc \) and \( v.rc \) to the left and right child, and a value \( Val(v) > 0 \). The value NIL represents a null pointer, showing that \( v \) has no child of that type. You wish to find the path from \( r \) to some leaf that maximizes the total values of vertices along that path. Give an algorithm to find the maximum sum of vertices along such a path along with a proof of correctness and runtime analysis.

3. (20 points) One ordered pair \( v = (v_1, v_2) \) dominates another ordered pair \( u = (u_1, u_2) \) if \( v_1 \geq u_1 \) and \( v_2 \geq u_2 \). Given a set \( S \) of ordered pairs, an ordered pair \( u \in S \) is called Pareto optimal for \( S \) if there is no \( v \in S \) such that \( v \) dominates \( u \). Give an efficient algorithm that takes as input a list of \( n \) ordered pairs and outputs the subset of all Pareto-optimal pairs in \( S \). Provide a proof of correctness along with the runtime analysis.

4. (20 points) Given a list of distinct integers \( a[1...n] \), an inversion is a pair of elements \( a[i], a[j] \) such that \( a[i] > a[j] \) and \( i < j \).
   Example: the list \([41, 72, 3, 74, 31]\) has 5 inversions, namely \((41, 3), (41, 31), (72, 3), (72, 31), (74, 31)\).
   Design a \( O(n \log n) \) divide and conquer algorithm to count the number of inversions of a list of length \( n \) with distinct integers.
   (Hint: alter mergesort and keep count of the inversions during the merge part.) (No justification for correctness needed. Please give a justification of the runtime.)

5. (25 points) An array \( A[1...n] \) is said to have a majority element if more than half of its entries are the same. Given an array, the task is to design an efficient algorithm to tell whether the array has a majority element, and, if so, to find that element. The elements of the array are not necessarily from some ordered domain like the integers, and so there can be no comparisons of the form “is \( A[i] \geq A[j] \)?” (Think of the array elements as GIF files, say.) However you can answer questions of the form: “is \( A[i] = A[j] \)?” in constant time.
   (a) Show how to solve this problem in \( O(n \log n) \) time. (Hint: Split the array \( A \) into two arrays \( A_1 \) and \( A_2 \) of half the size. Does knowing the majority elements of \( A_1 \) and \( A_2 \) help you figure out the majority element of \( A \)? If so, you can use a divide-and-conquer approach.) Provide a runtime analysis and proof of correctness.
   (b) Can you give a linear-time algorithm? (Hint: Here’s another divide-and-conquer approach:
   • Pair up the elements of \( A \) arbitrarily, to get \( n/2 \) pairs
   • Look at each pair: if the two elements are different, discard both of them; if they are the same, keep just one of them
   Show that after this procedure there are at most \( n/2 \) elements left, and that if \( A \) has a majority element then it’s a majority in the remaining set as well)