- The instructions are the same as in Homework-0, 1, 2.

There are 6 questions for a total of 80 points.

---

1. (5 points) Solve the following recurrence relations. You may use the Master theorem wherever applicable.
   (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 minimizes the total values of vertices along that path. Give an algorithm to find the minimum sum of vertices along such a path along with a proof of correctness and runtime analysis.

3. (20 points) Let S and T be sorted arrays each containing n elements. Design an algorithm to find the n^{th} smallest element out of the $2n$ elements in S and T . Discuss running time, and give proof of correctness.

4. An array $A[1...n]$ is said to have a majority element if more than half (i.e., $> n/2$) 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) (10 points) Show how to solve this problem in $O(n \log n)$ time. Provide a runtime analysis and proof of correctness.
   (<u>Hint</u>: *Split the array A into two arrays $A1$ and $A2$ of half the size. Does knowing the majority elements of $A1$ and $A2$ help you figure out the majority element of A? If so, you can use a divide-and-conquer approach.*)
   (b) (10 points) Design a linear time algorithm. Provide a runtime analysis and proof of correctness.
   (<u>Hint</u>: *Here is another divide-and-conquer approach:*
   - *Pair up the elements of A arbitrarily, to get $n/2$ pairs (say n is even)*
   - *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)*

5. Consider the following algorithm for deciding whether an undirected graph has a *Hamiltonian Path* from x to y, i.e., a simple path in the graph from x to y going through all the nodes in G exactly once. ($N(x)$ is the set of neighbors of x, i.e. nodes directly connected to x in G).

```
HamPath(graph G, node x, node y)
  - If x = y is the only node in G return True.
  - If no node in G is connected to x, return False.
  - For each z ∈ N(x) do:
        - If HamPath(G − {x}, z, y), return True.
  - return False
```

(a) (7 points) Give proof of correctness of the above backtracking algorithm.

(b) (8 points) If every node of the graph G has degree (number of neighbors) at most 4, how long will this algorithm take at most? *(Hint: you can get a tighter bound than the most obvious one.)*

6. (*This question has been set up as an extra credit question*) Design an $O(poly(n) \cdot 2^{n/4})$-time backtracking algorithm for the maximum independent set problem for graphs with bounded degree 3 (these are graphs where all vertices have degree at most 3). Give running time analysis and proof of correctness.