- The instructions are the same as in Homework 1, 2, and 3.

There are 7 questions for a total of 100 points.

---

1. (5 points) Solve the following recurrence relations. You may use the Master theorem wherever applicable.
   (a) $T(n) = 3 \cdot T(n/3) + O(n)$; $T(1) = O(1)$
   (b) $T(n) = 3 \cdot T(n/3) + O(n^2)$; $T(1) = O(1)$
   (c) $T(n) = 3 \cdot T(n-1) + 1$; $T(1) = 1$

2. (5 points) In this question, we consider the mergesort algorithm: *given an array $A[1...n]$, we split the array into two parts, A_L, and A_R; recursively sort both parts, then merge.* The recurrence relation for the running time you will find in most literature is $T(n) = 2 \cdot T(n/2) + O(n); T(1) = O(1)$. However, this recurrence relation makes sense only for even values of n. For an odd value, such as $n = 5$, the recurrence does not make sense since $T(2.5)$ is not well defined. For arrays with odd sizes, the split cannot be equal but roughly equal. To be more precise, for odd n, one of A_L, A_R is of size $\lceil n/2 \rceil$ and the other is of size $\lfloor n/2 \rfloor$. So, the correct recurrence relation that holds for all values of n is

$$T(n) = T\left(\lceil \frac{n}{2} \rceil\right) + T\left(\lfloor \frac{n}{2} \rfloor\right) + O(n)$$

   Clearly, the master method is not applicable for this recurrence relation. So, we go back to the first principle, the recursion tree method. Use the recursion tree method to solve the above recurrence relation and show that the solution is the same as that for $T(n) = 2 \cdot T(n/2) + O(n)$, which is applicable for n that is a power of 2.

3. (10 points) Use the ideas from the previous question to solve the recurrence:

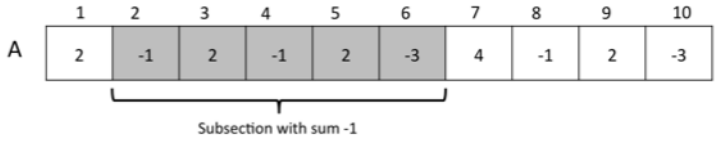
$$T(n) \leq 2 \cdot T\left(\lceil n/2 \rceil\right) + T\left(\lceil n/2 \rceil + 1\right) + O(n); \quad T(1) = 1.$$

   Note that this is the accurate recurrence relation for the Karatsuba algorithm.

4. (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.

5. (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.

6. (20 points) Given a sequence of integers (positive or negative) in an array $A[1...n]$, the goal is to find a subsection of this array such that the sum of integers in the subsection is maximized. A subsection is a contiguous sequence of indices in the array. For example, consider the array and one of its subsections in the figure below. The sum of integers in this subsection is -1.



You have to design an algorithm that takes as input an array $A[1...n]$ and outputs a pair of indices (i, j) such that i and j are the starting and ending indices of the maximum subsection of the array A.

Design a divide-and-conquer algorithm with running time $O(n \log n)$. Discuss the correctness and running time of your algorithm.
(*Note that there is an $O(n)$ time algorithm for this problem! That algorithm uses another algorithm design technique called Dynamic Programming, which we will see in upcoming lectures.*)

7. 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.
   (*Hint: 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.
   (*Hint: 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)*