CP 312, Fall 2018
Assignment 4 (5% of the final grade)
(due Monday, November 19, at 10:30 PM)

There are four questions in this assignment.

1. [8 marks] **Greedy algorithms**

   In the IntervalCover problem, the input is a set of n points on the line $0 \leq p_1 < p_2 < \cdots < p_n \leq M$, and the valid solution is the minimum number k of unit intervals $[x_1, x_1 + 1], \ldots, [x_k, x_k + 1]$ required to cover all n points. (A point p_i is *covered* by the interval $[x_j, x_j + 1]$ if $x_j \leq p_i \leq x_j + 1$.

   (a) Show that the greedy algorithm that at each step selects an interval that covers the largest number of still-uncovered points does not solve the problem.

   (b) Describe a greedy algorithm that solves the IntervalCover problem. Prove that your algorithm always returns a valid solution.

2. [10 marks] **Dynamic programming**

   In the SpacedSum problem, the input is an array A of n positive integers and the valid solution for this input is the largest possible value that can be obtained by summing a subset of the entries of A such that the subset contains at most one of any two consecutive entries $A[i], A[i+1]$ for every $1 \leq i \leq n-1$.

   In this question, you will design and analyze a dynamic programming algorithm that solves the SpacedSum problem.

   (a) Provide an example to show that in some cases, the optimal solution can skip two consecutive entries in the array.

   (b) Give an example to show that the greedy algorithm of choosing the entries in order of decreasing value (from largest to smallest while satisfying constrains) does not always give the best solution.

   (c) Give a precise definition of a subproblem that can be used to solve the SpacedSum problem with a dynamic programming algorithm.

   (d) Describe and justify the recurrence rule that will be used to compute the solutions to the subproblems defined in part (c).

   (e) Provide the pseudocode for your dynamic programming algorithm that solves the SpacedSum problem using the subproblems and recurrence rule defined above.

   (f) Analyze the time complexity of the algorithm obtained in part (e).

3. [6 marks] **Divide-and-Conquer**. Suppose you have an unsorted array $A[1..n]$ of elements. You can only do equality tests on the elements (e.g. they are large GIFs). In particular this means that you cannot sort the array. You want to find (if it exists) a majority element, i.e., an element that appears more than half the time. For example in [a, b, a] the majority element is a, but [a, b, c] and [a, a, b, c] have no majority element. Consider the following approach to finding a majority elememt: Recursively find the majority element y in the first half of the array and the majority element z in the second half of the array...

   (a) prove that if x is a majority element in A then it is a majority element in the first half of the array or the second half of the array (or both).

   (b) Using part (a) give a divide-and-conquer algorithm that runs in time $O(n \log n)$. Detailed pseudocode is required. Be sure to argue correctness and analyze the run time.

4. [6 marks] **Diameter of a tree**

   The *diameter* of a tree is the length of the longest simple path between nodes of the tree. Design an efficient divide and conquer algorithm to compute the diameter of a rooted binary tree. Your algorithm may use the following operations on trees that each have time complexity $\Theta(1)$.

   - Root(T) — Returns the root node of the tree T.
   - LeftChild(T, v) — Returns the node that is the left child of v in T; or $\emptyset$ if v does not have a left child.
   - RightChild(T, v) — Returns the node that is the right child of v in T; or $\emptyset$ if v does not have a right child.

   Provide justification of the correctness and a its running time complexity.