COMPSCI 311: Introduction to Algorithms **Fall 2022**

Discussion 7

Your Name: ____________________ Collaborators: ____

You will be randomly assigned groups to work on these problems in discussion section.

Problem 1. Master theorem. Solve the following recurrences using the Master theorem:

(a) $T(n) = 4T(n/5) + O(n)$

(b) $T(n) = 5T(n/4) + O(n)$

(c) $T(n) = 16T(n/2) + O(n^3)$

(d) $T(n) = 16T(n/2) + O(n^4)$

(e) $T(n) = 16T(n/2) + O(n^5)$

Problem 2. Choosing between algorithms. Suppose you are choosing between the following three algorithms:

- Algorithm A solves problems by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.
- Algorithm B solves problems of size n by recursively solving two subproblems of size $n - 1$ and then combining the solutions in constant time.
- Algorithm C solves problems of size n by dividing them into nine subproblems of size $n/3$, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.

What are the running times of each of these algorithms (in big-O notation), and which would you choose?

Problem 3. Proof by Induction for Recurrences. Consider the following recurrence, which describes an algorithm that divides a problem of size n into two equal-sized subproblems, but then does $O(n \log n)$ outside of the recursive calls:

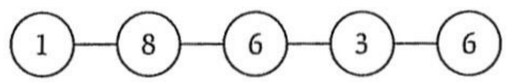
$$T(n) \leq 2T(n/2) + cn \log n,$$

We again assume the recurrence holds for $n \geq 2$ and that $T(2) \leq c$. Prove by induction that $T(n) \leq cn(\log n)^2$. **Hint**: see p. 213 of the book and slides from Lecture 11. The algebra for this proof is slightly longer but follows the same pattern.

Problem 4. Maximum Independent Set. Let $G = (V, E)$ be an undirected graph with n nodes. Recall that a subset of the nodes is called an **independent set** if no two of them are joined by an edge. Finding large independent sets is difficult in general; but here we'll see that it can be done efficiently if the graph is "simple" enough.

Call a graph $G = (V, E)$ a **path** if its nodes can be written as $v_1, v_2, \cdots, v_n$ with an edge between v_i and v_j if and only if the numbers i and j differ by exactly 1. With each node v_i, we associate a positive integer weight w_i.

Consider, for example, the following five-node path. The weights are the numbers drawn inside the nodes.



The goal in this question is to solve the following problem: *Find an independent set in a path G whose total weight is as large as possible.*

(a) What is the maximum-weight independent set in the example?

(b) Give an example to show that the following algorithm does not always find an independent set of maximum total weight.

Start with S equal to the empty set
while some node remains in G **do**
    Pick a node v_i of maximum weight
    Add v_i to S
    Delete v_i and its neighbors from G
return S

(c) Give an example to show that the following algorithm also does not always find an independent set of maximum total weight.

Let S_1 be the set of all v_i where i is an odd number
Let S_2 be the set of all v_i where i is an even number
(Note that S_1 and S_2 are both independent sets)
Determine which of S_1 or S_2 has greater total weight, and **return** this one

(d) Give an algorithm with $O(n)$ running time that takes an n-node path G with weights and returns the weight of the largest independent set.

(e) Now give an $O(n)$ algorithm to compute the maximum-weight independent set itself. Your algorithm should use values stored in the memoization array from your previous algorithm to trace back through the array and construct the optimal solution. (Hint: given the memoization arrary, how can you tell whether node v_n is in the optimal solution?)