CS 3000: Algorithms & Data — Summer 1 '20 — Tim LaRock

Homework 2
Due Tuesday, May 19 at 11:59pm via Canvas

Name:
Collaborators:

- Make sure to put your name on the first page. If you are using the LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- This assignment is due Tuesday, May 19 at 11:59pm via Canvas. Make sure to submit something before the deadline.
- Solutions must be typeset in LaTeX. If you need to draw any diagrams, you may draw them by hand as long as they are embedded in the PDF. I recommend using the source file for this assignment to get started.
- I encourage you to work with your classmates on the homework problems. *If you do collaborate, you must write all solutions by yourself, in your own words.* Do not submit anything you cannot explain. Please list all your collaborators in your solution for each problem by filling in the `yourcollaborators` command.
- Finding solutions to homework problems on the web, or by asking students not enrolled in the class, is strictly forbidden.

Problem 1. *Recurrences*

Suppose we have algorithms with running times $T(n)$ given by the recurrences:

$$T(n) = 36T(n/6) + n^2$$
$$T(n) = 8T(n/3) + n$$
$$T(n) = 7T(n/7) + n$$
$$T(n) = 11T(n/5) + n^2$$
$$T(n) = 3T(n/3) + 1$$

Solve each recurrence using the master theorem and put these running times in ascending order $T_1, T_2, \ldots, T_5$ so that $T_i = O(T_{i+1})$.

Solution:

Problem 2. *Babysitting*

You are babysitting your niece and before she will go to bed she insists on playing the following guessing game:

1. She picks a number x in $1, 2, \ldots, n$.
2. You make a guess y_1, and she simply says *correct* or *incorrect*.
3. You make a sequence of guesses $y_2, y_3, \ldots$. If your guess $y_i = x$ then your neice says *correct* and goes to bed. If your guess y_i is closer to x than the previous guess y_{i-1}, then she says *warmer* and if y_{i+1} is farther than the previous guess, then she says *colder*.

Your goal is to find x with as few guesses as possible so that your niece will go to bed. Design a divide-and-conquer algorithm that guesses your niece's number using $O(\log n)$ guesses.[1]

(a) Describe your algorithm in pseudocode.

Solution:

(b) Prove by induction that your algorithm correctly guesses the number.

Solution:

(c) Write a recurrence describing the number of guesses made by the algorithm, and solve the recurrence. You may solve the recurrence using any method you like.

Solution:

[1] **Hint:** You are allowed to guess negative numbers and numbers larger than n, although there are also correct solutions that do not use this option.

Problem 3. *Addition Chains*

An *addition chain* for an integer n is a list of integers $\langle x_0, x_1, ..., x_l \rangle$ such that $x_0 = 1, x_l = n$ and for all indices $k > 0$ there exist i, j such that $i \leq j < k$ and $x_i + x_j = k$. For example, an addition chain for $n = 24$ could be $\langle 1, 2, 3, 6, 12, 24 \rangle$.

The length l of an addition chain is the number of elements minus 1; we don't bother to count the first entry, since it is the same for every chain. As another example, an addition chain of length $l = 11$ for $n = 374$ could be $\langle 1, 2, 3, 5, 10, 20, 23, 46, 92, 184, 187, 374 \rangle$.

In this problem, you will propose and analyze a solution to find a minimum length addition chain for any positive integer n using a recursive *backtracking* strategy.

(a) Give a minimum length addition chain for each of the following numbers:

$$n = \{3, 4, 5, 20, 21, 100, 101\}$$

Solution:

(b) Design a recursive backtracking algorithm that computes a minimum length addition chain for any positive integer n. Include a proof of correctness, but do not worry about runtime yet (e.g. do not worry if you think your algorithm is very slow, as long as it uses backtracking).

Solution:

(c) Explain informally what makes your algorithm a backtracking algorithm. You may want to reference section 2.4 of the Erickson book to help guide your explanation. It is also okay to draw a picture if that helps, just please embed your entire solution in LaTeX (look up the macro `\includegraphics` to embed a photo).

Solution:

(d) Analyze the runtime of your algorithm. Do you think it is possible to speed it up? Explain.

Solution:

Problem 4. *Strategery*

Alice and Bob play the following game. There is a row of n tiles with values $a_1, \ldots, a_n$ written on them. Starting with Alice, Alice and Bob take turns removing either the first or last tile in the row and placing it in their pile until there are no tiles remaining. For example, if Alice takes tile 1, Bob can take either tile 2 or tile n on the next turn. At the end of the game, each player receives a number of points equal to the sum of the values of their tiles minus that of the other player's tiles. Specifically, if Alice takes tiles $A \subseteq \{1, \ldots, n\}$ and Bob takes tiles $B = \{1, \ldots, n\} \setminus A$, then their scores are

$$\sum_{i \in A} a_i - \sum_{i \in B} a_i \quad \text{and} \quad \sum_{i \in B} a_i - \sum_{i \in A} a_i,$$

respectively. For example, if $n = 3$ and the tiles have numbers $10, 2, 8$ then taking the first tile guarantees Alice a score of at least $10 + 2 - 8 = 4$, whereas taking the last tile would only guarantee Alice a score of at least $8 + 2 - 10 = 0$.

In this question, you will design an algorithm to determine the maximum score that Alice can guarantee for herself, assuming Bob plays optimally to maximize his score.[2]

(a) Describe the set of subproblems that your dynamic programming algorithm will consider. Your solution should look something like "For every ..., we define $opt(\ldots)$ to be ..."

Solution:

(b) Give a recurrence expressing the solution to each subproblem in terms of the solution to smaller subproblems.

Solution:

(c) Explain in English a valid order to fill your dynamic programming table in a "bottom-up" implementation of the recurrence.

Solution:

(d) Describe in pseudocode an algorithm that finds the maximum score that Alice can guarantee for herself. Your implementation may be either "bottom-up" or "top-down."

Solution:

(e) Analyze the running time and space usage of your algorithm.

Solution:

[2] **Hint:** Note that the sum of their scores is always 0, so if Bob is playing optimally to maximize his own score, then he is also playing optimally to minimize Alice's score.