CPSC 320 2020S2: Assignment 4

This assignment is due at **Wed Aug 5 at 10pm PDT**. Assignments submitted within 24 hours after the deadline will be accepted, but a penalty of 15% will be applied. Please follow all of the guidelines provided for Assignment 1.

1 List full names of all students in your group (as recorded on Canvas

Provide the list here. This is worth 1 mark.

2 Statement on collaboration and use of resources

To develop good practices in doing homeworks, citing resources and acknowledging input from others, please complete the following.

1. All group members have read and followed the guidelines for groupwork on assignments in CPSC320 (see `https://www.students.cs.ubc.ca/~cs-320/2020S2/coursework.html`, under Assignments).

   ◯ Yes ◯ No

2. We used the following resources (list books, online sources, etc. that you consulted):

3. One or more of us consulted with course staff during office hours.

   ◯ Yes ◯ No

4. One or more of us collaborated with other CPSC 320 students; none of us took written notes during our consultations and we took at least a half-hour break afterwards.

   ◯ Yes ◯ No

   If yes, please list their name(s) here:

5. One or more of us collaborated with or consulted others outside of CPSC 320; none of us took written notes during our consultations and we took at least a half-hour break afterwards.

   ◯ Yes ◯ No

   If yes, please list their name(s) here:

3 One Digit to Rule Them All

There are several popular ways to represent a positive integer m - from simply m, to m in base 2, or as a product of primes. In this question you will determine how to express such an integer m using only the digit '1' along with addition and multiplication operations. Parentheses indicate order of operations. Our goal is to find such an expression using as few as 1's as possible. We let $T(m)$ denote this minimum number of 1's in an expression for m. Throughout, assume that $m \geq 1$.

For example, consider $m = 6$. Using five 1's, we can express m as $6 = (1 + 1)(1 + 1 + 1)$.

1. Consider $m = 12$. Give three expressions of m, including an expression with the minimum number of 1's. (For this example, you can use brute force to find the expression with the minimum number of 1's. Use these solutions to think about tackling this problem by constructing smaller subproblems.)
2. Using the addition and multiplication operators there are two ways we can decompose m, either as $m = x + y$ or $m = (x)(y)$, where $1 \leq x, y < m$. Using these relations, we can define $T(m)$ in terms of smaller instances of this problem. For both cases, define $T(m)$ recursively when $m \geq 2$.
3. Using your recursive definition above, plus the base case that $T(1) = 1$, give a memoization algorithm for computing $T(m)$. What is the runtime of your algorithm?

4 Moving Mattresses

The following problem description is verbatim from tutorial 7.

You have been hired by *thekingboomattress.com*, a mattress selling company, to preform a data analysis operation. Over the next several months, the company has decided to host weekend sales where some number of items will be on sale. Each weekend, the company can either have a *small*, *medium*, or *large* sale, where a small, medium, and large number of items will be on sale.

For each weekend you are given an estimate of the profit that will be made for each type of sale. That is, for weekend i, you are given values s_i, m_i, l_i that represent the respective profits for each type of sale. Note that these values may be negative if a loss would be incurred. If no sale occurs on weekend i, the the profit for that weekend is zero.

The Boo-King himself however is concerned about having too many consecutive medium and large sales may result in the website running out of inventory for the following weekend! As such the following constraints are present:

- If a large sale occurs on weekend i, then on both weekends $i - 1$ and $i - 2$, no sale can occur.
- If a medium sale occurs on weekend i, then on weekend $i - 1$ no sale can occur.

There is sufficient inventory for the first two weekends so **the above constraints only apply for weekends** $i \geq 3$.

Given the estimated profit for each sale type over a period of n weekends, your goal is to determine what sale should occur for each weekend, if any, such that the estimated profit is maximized and the above constraints are realized.

1. Let *maxProfit*(i) denote the maximum profit that will be achieved over i weekends. Give a recurrence relationship to define *maxProfit*(i). Be sure to cover the base case(s) in your solution. We recommend that you include the case where $i = 0$ as a base case.

2. Using your recurrence above, give a memoization algorithm that returns the maximum profit possible for an instance of the Moving Mattresses problem of size n (that is, profit estimates for n weekends are provided). Your algorithm should use an array $maxProfit(0, \ldots, n)$. Use big-O notation to describe both the time and space (memory) used by your algorithm as a function of n.

5 Data Sequencing

You are designing a system that relays data to a remote server. Given a sequence of data items of various lengths, you can partition the sequence into (nonempty) packets of some bounded total length, and then send the packets to the server. You are told that on the server's end, the server will run more slowly if the packets have inconsistent lengths. Thus you wish to determine a partition of the data such that each packet will roughly have the same length.

Formally, let the items in the sequence be consecutively numbered from 1 to n, and let l_i be the length of data item $i, 1 \leq i \leq n$. Let L be the maximum allowable length of a packet. A packet $S_{i,j}$ of the items $i, i+1, \cdots, j$, where $i \leq j$ is *legal* if

$$\sum_{k=i}^{j} l_k \leq L.$$

We define $C_{i,j} = L - \sum_{k=i}^{j} l_k \geq 0$ as the *cost* of this legal packet. We wish to partition the sequence of items into legal packets such that the total sum of the packet costs will be minimized. Note that the ordering of data must be maintained when packets are created. You can assume that all of the lengths l_i are less than or equal to L.

1. We start by considering how many possible packets are there, legal or not. Remember that a packet $S_{i,j}$ is a sequence of data items $i, i+1, \ldots j$ with $i \leq j$.

   We start by considering how many partitions of n data items are possible, legal or not. How many partitions exist? (Remember that a partition must maintain item order.)

2. We have defined the *cost* of a packet above, but only if it is legal. Below you will give an algorithm that determines how to partition S using these costs. Such an algorithm will need to account for the cost of each packet in some way. For any packet that is not legal, what is an appropriate value for its cost?

3. Let $optCost(j)$ denote the optimal cost of partitioning the first j data items into packets. Give a recursive definition for $optCost(j)$.

4. Describe an efficient dynamic programming algorithm to compute the minimum cost partition of S. What is the runtime of your algorithm?

5. Using the values in $optCost$, give a backtracking algorithm that yields the partition corresponding to the optimal cost in order.

6 Where to Warehouse

The online seller *Lamazon* has decided to open up warehouses in the mountainous province of Ordona. The company has the option to open a warehouse in any city of the region at some cost. These warehouses will then be able to serve the city they are located in, as well as any nearby city. The company would like to choose the cities where warehouses will be opened, so as to minimize building costs as well as to ensure that all cities are serviced by at least one warehouse.

Formally, an instance of the *where to warehouse* problem is an undirected graph $G = (V, E)$ where vertices correspond to cities and edges indicate which two cities are nearby (e.g. a road exists), and for each city $i \in V$, $c[i] > 0$ is the cost of building a warehouse in city i. The goal is to minimize the cost of building warehouses such that all cities will have a warehouse or will be nearby a city that does.

As it turns out, due to the mountainous terrain, the graph corresponding the Ordona province is a **line**: We'll assume that the nodes are $\{1, 2, \ldots, n\}$ and that there is an edge $(i, i+1), 1 \leq i < n$. We denote the cost of an optimal solution for this line graph by $Opt(n)$.

In addition to the cost of the optimal solution, namely $Opt(n)$, we'll define two additional quantities:

- $OptIn(n)$ is the min cost of a solution that *must include* n.
- $OptOut(i)$ is the min cost of a solution $\{1, \ldots, i\}$ that *must exclude* n.

Examples: Consider the line graph with $n = 3$, where $c[1] = 7$, $c[2] = 15$, and $c[3] = 6$. Then $Opt(n)$ is 13, achieved by putting warehouses at cities 1 and 3. Also, $OptIn(n)$ is 13, while $OptOut(n)$ is 15, since if there is no warehouse at city 1, there must be one at city 2 and that warehouse can also service city 3.

If instead we have $c[1] = 7$, $c[2] = 6$, and $c[3] = 15$, then $OptIn(n)$ is still 13, achieved by putting warehouses at cities 1 and 2, while $Opt(n)$ is 6 and $OptOut(n)$ is 6.

Our algorithm will compute $Opt(n)$ from solutions to smaller subproblems. For each i, $0 \leq i \leq n$, the line graph containing just nodes $1, \ldots, i$, along with the costs $c[1], c[2], \ldots, c[i]$ is a subproblem. (The case where $i = 0$ is the trivial case with no cities, and the case where $i = n$ is the full problem.)

1. What is the optimal solution for the following instance, and what is the cost of this solution? In this instance, $n = 5$ and $c[1..5] = [8, 6, 15, 7, 4]$.
2. What is the optimal solution for the following instance, and what is the cost of this solution? In this instance, $n = 5$ and $c[1..5] = [12, 6, 7, 15, 4]$.
3. For $i \geq 3$, it is the case that

   $$OptIn(i) = c[i] + \min\{OptIn(i-1), Opt(i-2)\}.$$

   Explain why this is true.
4. For $i \geq 2$, write a recurrence for $OptOut(i)$, and briefly justify your recurrence.
5. We can express $\text{Opt}(n)$ as $\min\{OptIn(n), OptOut(n)\}$. Use this to write an algorithm that computes $\text{Opt}(n)$ in $O(n)$ time. The algorithm takes as input n and an array $c[1..n]$, where $c[i]$ is the cost of putting a warehouse at node i. Your algorithm should use memoization and recursion to compute both $OptIn(i)$ and $OptOut(i)$ for $1 \leq i \leq n$, and use these quantities to compute $\text{Opt}(n)$. Make sure you consider the base cases carefully. (No justification is needed, just the algorithm.)
6. Using a recursion tree, explain why the running time of your algorithm of part 2 is $\Theta(n)$.