Dept. of Computer Science **University of Toronto**

Assignment # 1 (Due May 29, 11:59 pm)

CSC 373H1 **Summer 2021**

Worth: 10%
[Please make sure your solution is no more than 2 regular A4/Letter sized pages per problem, i.e., your full submission should be no more than 8 pages.]

1. [20 marks]
   Consider the problem of choosing cell phone tower locations on a long straight road. We are given a list of locations along the road at which there are customers' houses, say $\{x_k\}_{k=1}^K$. No towers have yet been built (so no customers currently have service). However, a survey of the road has provided a set of J possible tower locations, $\{t_j\}_{j=1}^J$, where these potential tower locations are also measured in terms of the distance along the road. (These indexed lists of house and tower locations might be in any order. You can assume that all these distances are integers.)

   Each customer will get service (at home) if and only if they are within a range $R > 0$ of at least one cell phone tower that gets built, that is, the house at x_k will have service iff there exists a tower that has been built at some t_j with $|x_k - t_j| \le R$.

   You can assume that if all the towers were built then every home would get service. However, such a solution could be overly expensive for the phone company. How can we minimize the number of towers that need to be built but still provide service at every house? Note that a suitable solution may still leave some parts of the road without cell service, even though each customers' house will have service.

   (a) [7 points]
   Write a greedy algorithm for choosing the minimum number M of cell phone towers required, along with a suitable set of locations, say $T = \{t_{j(m)}\}_{m=1}^M$, such that every house has service. Present your algorithm as a pseudocode.

   (b) [10 points]
   Prove the correctness of your algorithm.

   (c) [3 points]
   Compute the complexity of your algorithm.

2. [20 marks]
   Consider a variant on the problem of Interval Partitioning.

   Suppose we have **two** resources, and we want to schedule as many requests as we can. The input is $(s_1, f_1), (s_2, f_2), \cdots, (s_n, f_n)$, where $n \ge 1$ and all $s_i < f_i$ are nonnegative integers. The integers s_i and f_i represent the start time and finish time, respectively, of request r_i.

   A *schedule* is now defined as a pair of sets (A_1, A_2), the intuition being that A_i is the set of requests scheduled on resource i. A schedule must satisfy the obvious constraints: $A_1 \subseteq \{1, 2, \ldots, n\}$, $A_2 \subseteq \{1, 2, \ldots, n\}$, $A_1 \cap A_2 = \emptyset$, and for all $i \ne j$ such that $i, j \in A_1$ or $i, j \in A_2$, requests i and j do not overlap.

   (a) [10 marks]
   Design an algorithm (write a pseudocode) to solve the above problem in time $o(n^2)$, *i.e.*, strictly less than $\mathcal{O}(n^2)$. Compute the complexity of your algorithm.

   (b) [10 marks]
   Prove that the above algorithm is guaranteed to compute an optimal schedule.

3. [20 marks]
   Consider the problem of making change, given a finite number of coins with various values. Formally:

Input: A list of positive integer coin values c_1, c_2, $\cdots$, c_m (with repeated values allowed) and a positive integer amount A.

Output: A selection of coins $\{i_1, i_2, \ldots, i_k\} \subseteq \{1, 2, \ldots, m\}$ such that $c_{i_1} + c_{i_2} + \cdots + c_{i_k} = A$ and k is as small as possible. If it is impossible to make change for amount A exactly, then the output should be the empty set $\emptyset$.

(a) [5 points]
Describe a natural greedy strategy you could use to try and solve this problem, and show that your strategy does not work.

(The point of this question is **not** to try and come up with a really clever greedy strategy — rather, we simply want you to show why the "obvious" strategy fails to work.)

(b) [10 points]
Give a detailed dynamic programming algorithm to solve this problem. Follow the steps outlined in class, and include a brief (but convincing) argument that your algorithm is correct.

(c) [5 points]
What is the worst-case running time of your algorithm? Justify briefly.

4. [20 marks]
During the renovations at Union Station, the work crews excavating under Front Street found veins of pure gold ore running through the rock! They cannot dig up the entire area just to extract all the gold: in addition to the disruption, it would be too expensive. Instead, they have a special drill that they can use to carve a single path into the rock and extract all the gold found on that path. Each crew member gets to use the drill once and keep the gold extracted during their use. You have the good luck of having an uncle who is part of this crew. What's more, your uncle knows that you are studying computer science and has asked for your help, in exchange for a share of his gold!

The drill works as follows: starting from any point on the surface, the drill processes a block of rock 10cm × 10cm × 10cm, then moves on to another block 10cm below the surface and connected with the starting block either directly or by a face, edge, or corner, and so on, moving down by 10cm at each "step". The drill has two limitations: it has a maximum depth it can reach and an initial hardness that gets used up as it works, depending on the hardness of the rock being processed; once the drill is all used up, it is done even if it has not reached its maximum depth.

The good news is that you have lots of information to help you choose a path for drilling: a detailed geological survey showing the hardness and estimated amount of gold for each 10cm × 10cm × 10cm block of rock in the area. To simplify the notation, in this homework, you will solve a two-dimensional version of the problem defined as follows.

Input: A positive integer d (the initial *drill hardness*) and two $[m \times n]$ matrices H, G containing non-negative integers. For all $i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\}$, $H[i, j]$ is the *hardness* and $G[i, j]$ is the *gold content* of the block of rock at location i, j (with $i = 1$ corresponding to the surface and $i = m$ corresponding to the maximum depth of the drill).

There is one constraint on the values of each matrix: $H[i, j] = 0 \implies G[i, j] = 0$ (blocks with hardness 0 represent blocks that have been drilled already and contain no more gold).

Figure 1 below shows the general form of the input. Figure 2 shows a sample input.

Output: A drilling path $j_1, j_2, \ldots, j_\ell$ for some $\ell \leq m$ such that:

- $\boxed{1 \leq j_k \leq n \textbf{ for } k = 1, 2, \ldots, \ell \textbf{ (each coordinate on the path is valid);}}$

- $j_{k-1} - 1 \le j_k \le j_{k-1} + 1$ for $k = 2, \ldots, \ell$ (each block is underneath the one just above, either directly or diagonally);
- $H[1, j_1] + H[2, j_2] + \cdots + H[\ell, j_\ell] \le d$ (the total hardness of all the blocks on the path is no more than the initial drill hardness);
- $G[1, j_1] + G[2, j_2] + \cdots + G[\ell, j_\ell]$ is maximum (the path collects the maximum amount of gold possible).

Follow the dynamic programming paradigm given in class (the "five step" process) to give a detailed solution to this problem. Make sure to keep a clear distinction between each of the steps, and to explain what you are doing and why at each step — you will **not** get full marks if your answer contains only equations or algorithms with no explanation or justification.

$H[1,1], G[1,1]$	$H[1,2], G[1,2]$	$\cdots$	$H[1,n], G[1,n]$
$H[2,1], G[2,1]$	$H[2,2], G[2,2]$	$\cdots$	$H[2,n], G[2,n]$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$H[m,1], G[m,1]$	$H[m,2], G[m,2]$	$\cdots$	$H[m,n], G[m,n]$

Figure 1: General form of the input matrix.

2,2	**2,7**	0,0	2,3	1,8
2,2	**0,0**	1,2	2,0	1,1
1,4	1,1	**2,6**	2,1	3,8

Figure 2: Sample input with optimum path shown in **bold** (for $d = 4$).