CS142, Entrance Exam Report **Due: 23:59, Tue 01/17/2023**

Name: Majd Kawak / NetID: mkawa025 / CodeForces ID: mkawa025 / Student ID: 862273310

**** I would like to use 1 day grace period for this assignment ****

A. Whomping Willow

Submission#: 188920595

In this program, firstly I declared four *int* to hold a, b, r, n given in question. Then I go and create an *array* of string to hold ”yes” or ”no” upon computing result, two *int* c, d to hold x, y to check coordinate. My program uses *for* loop n times to take input and compute answer, storing each answer at it’s rightful index. Total time complexity = $O(n)$.

B. Chocolate Frogs

Submission#: 188924073

In this program, firstly I declared a *map* to hold card number and number of assurances as a pair of *key* and *value*. Using a simple *for* loop upon input to both take input and calculate number of assurances to each card. Total time complexity = $O(n)$.

C. Patronus Charm

Submission#: 188936680

In this program, firstly I declared an *int* that would hold *vector_size*, a *vector* that would hold input sequence of numbers. Then I go and declare a function to compute *LIS* with *MAX_SUM* called *compute_LIS*. I pass my input *vector* to *compute_LIS*, using 1D method to compute *LIS*; function compute values at each index using Recurrence Relation below:

$$l_i = max \begin{cases} 1 \\ \underset{0 < j < i, a_j < a_i}{max} \left\{ (l_j + l_i) \right. \end{cases}$$

Total time complexity = $O(n^2)$.

D. Black Family Tree

Submission#: 189709202

In this program, firstly I declared two *int* n, m that would hold number of nodes and traitors respectively. Then I go and create a *vector* to hold all nodes that are roots(in case a node gets detached from tree and form its own root tree). I declare a *vector* of *vectors* to hold my *AdjacencyList* tree representation. After that I declare a *vector* with size n to hold each node’s parent at its rightful index, assign value -1 if a node is a root. I go on and input nodes parents and push them at the right index to represent nodes parent as well as push nodes into $AdjacencyList$. Upon inputing traitors I look for parent and children to erase node from tree, children would become they're own tree and parent would lose traitor node. lastly I pass all roots to a function to calculate max number of connected nodes from each root, using queue pushing and popping each element to get to result. Total time complexity $= O(n) + O(m) + O(roots) = O(n)$.