Monsoon 2021 | Graduate Algorithms: End-sem | *Max: 40 points*

Due: **4 December** | | *Duration: 120 mins*

20% if you write **"I don't know"** for any (sub)question, with a cap of 6.
"I don't know" can be written for subquestions that are marked with "partial IDK".
Avoid detailed pseudocode; instead, explain in words.
Common algorithms like binary search, sorting, graph algorithms done in course can be used directly.
Only 1 page is allowed for every question from Q1–Q6, except Q5(d) which is allowed 1 page of its own.

Q1 [4+4=8 points (partial IDK)] Consider a data structure that maintains a forest of rooted trees, and further supports three operations.

1. $MakeTree(x)$: Given an element x, create a tree whose only node is x.
2. $Depth(x)$: Given a node x, return the depth of x in the tree containing x (depth of the root node of any tree is defined to be 0).
3. $Join(r, x)$: Given a root node r, and another node x not in the same tree as that of r (x need not be a root), make r the child of x.

Suppose that we use reversed trees to implement this data structure (similar to what we did for disjoint-sets): we use $v.p$ to denote the parent of a node v, and set $v.p = v$ if v is a root node. Suppose further that we implement $Join(r, x)$ by setting $r.p = x$ and $Depth(x)$ by following the path from x up to a root, returning a count of all nodes on the path other than x. Clearly, the complexities of $MakeTree$ and $Join$ are $O(1)$ and that of $Depth$ is linear in the depth of the given node.

Show that the worst-case total running time of any sequence of m operations (consisting of a mix of $MakeTree()$, $Depth()$, $Join()$) (a) is $O(m^2)$ and (b) is $\Omega(m^2)$. Think carefully what each of the bounds mean.

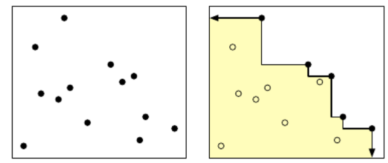
Q2 [1+1+1+1=4 points] Fill in the blanks below to complete a recursive algorithm that identifies Hamiltonian cycle in an undirected graph G (or returns NULL if none exists). The function is initially called as `FindHC(G)`. The algorithm may make calls to another algorithm $HasHC(H)$ that determines if H has a Hamiltonian cycle. Assuming that $HasHC$ runs in polynomial time, $FindHC$ should also run in polynomial time. Fill in only the steps (a)–(d) below; write "no-op" if nothing should be done for any of the steps. `FindHC(G)` should return a *sequence* of vertices on any Hamiltonian cycle (e.g., for a cycle $a - b - c - d - a$, the function can return any of $[a, b, c, d], [b, c, d, a], [c, d, a, b], [d, a, b, c]$); however, *only one* sequence should be returned. Feel free to use known polynomial-time graph algorithms, and avoid detailed pseudocode/Java/C++/etc. You cannot use any global variable or modify the input/output signature of the function. You may add a 2-3 line explanation for your steps.

```
def FindHC(graph H): // should return NULL if no cycle exists

    // (a) fill the base case

    // (b) add anything you want to perform before the for-loop

    for edge e in H:
        // there is a Hamiltonian cycle in H after removing the edge e
        if HasHC(H-{e}) returns true:
            // (c) fill what to do here

        else:
            no-op // I am not kiddin'

    // (d) anything you want to perform after the loop, including a return statement
```

Q3 [5 points] Show that for any integer r, it is not possible to construct a polynomial time r-absolute-approximation algorithm for 0-1 Knapsack with integer values and integer weights (unless $P \neq NP$).

Q4 [1+3+1=5 points] Suppose you are given a list L of n distinct points in a two-dimensional plane; there could be points that have common X coordinates or common Y coordinates, but not both, and further, the points in L are in the order of increasing X coordinates.

A point $p \in L$ is defined to be maximal in L if no other point in L is **both** above p and to the right of p. In the example below, L has 5 maximal points.

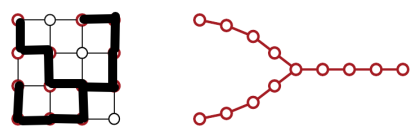


```
def CountMaximal(list L of points sorted in X coord):
    // (a) add base case

    LL = left half of points in L
    LR = right half of points in L

    // (b) finish the rest, must call CountMaximal(LL) and CountMaximal(LR)
```

Complete the divide-and-conquer algorithm above that returns the set of maximal points in L in $O(n \log n)$ time; if you require, you can design `CountMaximal` to compute and return other information as well. The subproblems for the divide step has been computed for you; fill in the rest. You may add 2-3 lines explaining your idea. (c) You should explain the running time with the help of a recurrence relation (no need to solve the recurrence).

Q5 [1+3+1+7=12 (partial IDK)] A "tee" is a subgraph that consists of three equal-length paths connected to one node (all the three paths should have the same number of edges).

The weight of a tee is the sum of the weights of its edges. The figure shows a graph on the left with edge weights 1 on all edges and a tee on the right with weight 12. We want to find a tee in a weighted undirected graph with the largest weight – call this the FINDTEE problem.



(a) Show how to define a decision problem named DECTEE corresponding to FINDTEE. (b) Explain how a polynomial algorithm for DECTEE can be used to solve FINDTEE in polynomial time; explain both algorithm and complexity analysis in words. (c) Show that DECTREE is an NP problem by describing a verifier. (d) Prove that DECTREE is NP-complete (hint: $stHamPath(G, s, t)$, that decides if there is a Hamiltonian path from s to t in a graph G, is NP-complete).

Q6 [6 points] A computer has to pick one of two tasks, Task-0 and Task-1, for each of the next n time slots. A table $U[0,1][1 \dots n]$ is available in which $U[0][i]$ indicates the amount of utility that will be generated if Task-0 is scheduled for the ith slot, and $U[1][i]$ indicates the utility generated by scheduling Task-1 for the i-slot. However, the computer also has to pay a preemption penalty of S utility everytime one task is preempted for another. The computer has to figure out which task should be scheduled in which slot so that the total utility after the nth slot is maximized.

	$n = 3, S = 2$		
$U[0] =$	3	1	2
$U[1] =$	1	4	2

In this example, the schedule (Task-0,Task-0,Task-1) has utility $3 + 1 + 2 = 6$ and (Task-0,Task-1,Task-0) has utility $3 - 2 + 4 - 2 + 2 = 5$ due to switching penalties.

Design either a dynamic programming algorithm or a graph reduction algorithm to solve this optimization problem given the table U as input **(use only one of the approaches)**.

For a DP-based approach, **(a [1 point])** specify the English description of a function, **(b [3 points])** a recursive formula (inluding a base case) to compute that function, **(c [1 point])** describe how to compute the optimal utility from the function values, and **(d [1 point])** explain the time complexity for doing that. Add 2-3 lines explaining the correctness of your formula in (b).

For a graph reduction based approach, explain how to construct a graph G on which we can run an algorithm for longest (with the largest weight) path on G between **one pair of vertices** to solve the scheduling question. Describe **(a [2 point])** what do the vertices, edges and their weights represent with respect to the scheduling problem, **(b [1 point])** the number of vertices and edges of G (they should be polynomial in n). Suppose that you have access to an algorithm $LP(G, s, t)$ to return the longest weighted path in G from s to t, **(c [1 point])** explain how to solve the scheduling problem with the help of $LP()$ and finally **(d [1 point])** explain in 4-5 lines how to implement $LP()$ so that the entire algorithm for task scheduling takes polynomial time, **(e [1 point])** explain and derive the total complexity for the task scheduling problem using the above graph reduction approach.