Monsoon 2023 Graduate Algorithms: End-sem *Max: 100 points*
Due: **5 December** *Duration: 180 mins*

10% if you write **"I don't know"** for any (sub)question, with a cap of 5.
Avoid detailed pseudocode; instead, explain in words.
Common algorithms like binary search, sorting, graph algorithms done in course can be used directly.

Q1 [5+7=12 points] This question is about solving two problems using DP.

(a) The input consists of two arrays $X[1 \dots m]$ and $Y[1 \dots n]$ where $m \leq n$. We want to determine whether X occurs as two disjoint subsequences of Y. For example, the string PPAP appears as two disjoint subsequences in the string **P**EN**P**INE**A**PPLEAPPLE**P**EN. Define a function $L(i, j, k)$ (where $0 \leq i, j \leq m$ and $0 \leq k \leq n$) to be true if $X[1 \dots i]$ and $X[1 \dots j]$ appear inside $Y[1 \dots k]$ at disjoint locations, and to be false otherwise. For solving the problem we need to compute $L(m, m, n)$.

(b) The input is a weighted directed graph G on n vertices and m edges; weights can be negative, but there are no negative weighted cycles. We want to compute the weight of the shortest paths between each pair of vertices. Define a function $SP(u, v, k)$ (u, v, are vertices and $k \geq 0$ is an integer) as the weight of the shortest path from u to v using at most 2^k edges.

For each of the above functions $L(i, j, k)$ and $SP(u, v, k)$, (i) write a recursive formula for the function – explicitly mention all cases and write the mathematical formula for each case, (ii) write the size of the memo required to solve the problem, and (c) analyse the time complexity required to solve the problem. You do not need to write the base cases.

Q2 [5+5=10 points] This question asks you to compute DFT and IDFT. Write the answers as lists/arrays. Write only the final answer; intermediate steps are not required.

(a) Suppose we want to multiply the polynomial $P(x) = 2x$ by $Q(x) = x/2$ using FFT. Do the first step: Compute the DFT of $P(x)$.

(b) The pointwise-product of the DFTs of $P(x)$ and $Q(x)$ is $R = [1, -1, 1, -1]$. Compute the inverse Fourier transform of R.

Q3 [3+5=8 points] Consider a disjoint-set implementation using reversed-trees (every non-root node has a pointer to its parent) with union-by-depth. Now, suppose an application has made n `MakeSet` and m `Union` calls. Answer these questions with brief (2-3 sentence) explanations.

(a) What is the largest *depth* of any node? 1. $O(n)$ 2. $O(\log n)$ 3. $O(m)$ 4. $O(\log m)$ 5. Other; specify.

(b) Further, suppose it is known that $m = O(n)$. What is the total complexity of this for-loop: `for i=1...n: Find(i)`? 1. $O(n^2)$ 2. $O(n \log n)$ 3. $O(n)$ 4. Other; specify.

Bin-Packing is a very important combinatorial problem that has found its uses across multiple industrial domains, e.g., transportation, logistics, and VLSI. You will be solving several questions related to Bin-Packing during this exam. Sadly, there is no known polynomial time algorithm for this problem. Let's first define the BINPACKING problem formally. Its inputs are are n items with costs $A[1 \dots n]$ and there are t bins. The task is to distribute the items in the bins such that the cost of the costliest bin is minimized, and the objective of the problem is to output the items in each bin; a natural data structure for this is a collection of disjoint-sets over $\{1, 2, \dots n\}$ – each set represents the items in the corresponding bin).

Here is an example of the Bin-Packing problem: There are 4 items with costs $A = [10, 5, 6, 12]$ and there are $t = 3$ bins. There could be many ways to pack those items in the bins. One strategy is : bin-1 has item 1, bin-2 has items $\{2, 4\}$, and bin-3 has item-3; bin-2 is costliest with cost 17. Another stragegy is: bin-1 has items $\{1, 4\}$, bin-2 has $\{2, 3\}$, and there is nothing in bin-3; here, bin-1 is costliest with cost 22. So, the first strategy is better among these two.

Q4 [2+3+5=10 points] Let DECBP be a decision version of the BINPACKING problem.

(a) Write a formal definition of DECBP in terms of its input and objective.

(b) Show that DECBP is in NP by designing a verification algorithm. Write the pseudocode along with its explanation. Proof of correctness is not required.

(c) Show that DECBP is NP-hard; for reduction use the well-known NP-complete PARTITION problem. Proof of correctness is not required.

Q5 [15 points] Consider the above BINPACKING problem but with 2 bins. Design a dynamic programming algorithm to return the optimal cost – cost of the costliest bin in the optimal solution. For complexity analysis, assume that cost of each item is represented by a c-bit integer. *Hint: You may find it easier to first think about a decision version of the problem.*

Alternatively, if you cannot design a DP, you can get partial points upto 10 for designing a recursive backtracking algorithm for the same.

Q6 [10+5=15 points] Consider the above BINPACKING problem.

(a) Prove that the following algorithm for BINPACKING has 2-relative approximation ratio:

```
for i=1 ... n: add item i to the least costliest bin at that moment
```

Hint: OPT denotes the cost of the costliest bin among all bins of the optimal solution, and APPROX denotes the cost of the costliest bin among all bins of the approximate solution. Let bin t be the costliest bin at the end of the algorithm, and let $A[k]$ be the last item added to bin t. Let C be the cost of bin t before $A[k]$ was added. To relate OPT and APPROX, use $A[k]$, C, and the average cost of the bins at the end of the algorithm.

(b) Show that a polynomial-time 1.25-(relative-)approximation algorithm for BINPACKING can be used to design a polynomial-time algorithm for PARTITION.

Q7 [5+10=15 points] The dual problem of Bin-packing is **Bin-covering**. Bin-covering takes as input n items with values $V[1 \dots n]$, and puts them the in the largest number of bins possible while ensuring that the value of each bin is at least s; both V and s are inputs, and you can think of the output as disjoint-sets that partition $\{1 \dots n\}$. We will look at 2 other versions. The optimization version **Opt-bin-covering** returns only the largest number of bins (instead of the bin-wise itemsets). The decision version **Is-bin-covering** takes an additional input t, and decides if the items can be distributed in t bins such that each bin has value at least s; this problem is NP-complete.

(a) Suppose there is a polynomial-time algorithm $DecBC1(values, t)$ to solve Is-bin-covering (s is global); let $T1(n)$ denote the time-complexity of $DecBC1$ on n items. Show that Opt-bin-covering can be solved in polynomial time. Design an efficient algorithm for the latter and analyse its complexity.

(b) Suppose there is a polynomial-time algorithm $DecBC2(values)$ to solve Is-bin-covering (both s and t are global); let $T2(n)$ denote the time-complexity of $DecBC2$ on n items. Show that Bin-covering can be solved in polynomial time. Design an efficient algorithm for the latter and analyse its complexity.

Q8 [15 points] Consider a scenario of the bin-packing problem with t bins, but now with a different cost model. There are infinite supplies of two types of items - type-A and type-B. You have to fill the bins one-by-one, starting from bin-1, then bin-2, and so on, using *exactly one* item. However, there is a cost-table given, $U[1,2][1 \dots t]$ in which $U[1][i]$ is the value of putting a type-A item in the i-th bin and $U[2][i]$ is the value of putting a type-B item in the i-th bin. To make matters worse, there is a penalty S_{AB} for switching from a type-A item to a type-B item and S_{BA} for switching from a type-B item to a type-A item. All the entries in U and the penalties are positive. Your task is to figure out how to fill the bins that maximizes the total value.

$t = 3, S_{AB} = 2, S_{BA} = 1$

bin	1	2	3
$U[1]$	3	1	2
$U[2]$	1	4	2

Example:
Value of $[A, A, A] = 3 + 1 + 2 = 6$ (type-A item in each bin).
Value of $[A, B, B] = 3 + (-2) + 4 + 2 = 7$ (type-A in bin-1, type-B in the rest).
Value of $[A, B, A] = 3 + (-2) + 4 + (-1) + 2 = 6$ (type-B in bin-2, type-A in rest).

In this question you will complete the below graph-reduction algorithm to solve the above problem.

```
def OptimalSequence(U[1,2][1...t]):
    G = construct graph from U
    s,w = identify vertices in G

    L = longest path (i.e., with maximum weight) in G from s to w
    S = compute sequence of items from L
    return S
```

(a) Show how to construct the directed weighted graph G, and how to select the two vertices s and w in G such that we can run an algorithm for longest path (path with the largest weight) on G from s to w to compute the optimal sequence. Describe (i) what do the vertices, edges and their weights represent with respect to the bin-packing problem, and (ii) how many vertices and edges are in G (they should be polynomial in t).

(b) Explain briefly but clearly how to construct the optimal sequence S from L. Analyse the time-complexity of this step.

(c) What would be the complexity of computing L from G, s, w? Explain briefly in 3-4 sentences.