Name: ______________________ **Entry number**: ______________

COL351 Major Exam Duration: 2 hours

Read the following instructions before you begin writing.

1. Keep a pen, your identity card, and optionally a water bottle with you. Keep everything else away from you, at the place specified by the invigilators.

2. Do not detach sheets. Write your entry number and name on every page. (We will detach sheets prior to grading.) This is to be done before the exam ends. No extra time will be given for this.

3. Answer only in the designated space. The designated space for solving a problem is the space between the statement of the problem and the statement of the next problem. Think before you use this space. No additional space will be provided for writing answers.

4. The last page is designated for rough work. Additional sheets will be provided for rough work on demand. Space designated for rough work cannot be used for writing answers.

5. No clarifications will be given during the exams. If something is unclear or ambiguous, make reasonable assumptions and state them clearly. The instructor reserves the right to decide whether your assumptions were indeed reasonable.

6. You may use any result discussed in class or tutorials without proving it. Similarly, you can use any algorithm discussed in class or tutorials as a "black-box" i.e., without reproducing any details of how it works.

7. As per the instructions of the Associate Dean, Curriculum, you are required to read and follow the following honour code.

> **As a student of IIT Delhi, I will not give or receive aid in examinations. I will do my share and take an active part in seeing to it that others as well as myself uphold the spirit and letter of the Honour Code.**

---

1. A fundamental problem in machine learning is clustering, where you are given a data set of n points in $\mathbb{R}^m$ as input, and you must choose k points ($k \leq n$), also known as cluster centers, that "represent" the data set. Specifically, the *k-means clustering* problem is defined as follows. Given a set X of data points and a set of cluster centers C, the contribution of a data point to the cost of C is the square of the distance between the data point and its closest point in C. The cost of C is, thus, equal to $\sum_{x \in X} \min_{c \in C} d(x, c)^2$, where d denotes the Euclidean distance. Given a set $X = \{x_1, \ldots, x_n\}$ of data points and a number k, the objective is to find a set $C \subseteq \mathbb{R}^m$ of k cluster centers that minimizes cost. We will solve the problem when $m = 1$, that is, all the data points and cluster centers lie on the real number line. Note that points not in the set X also qualify to be cluster centers.

(a) (2 points) First, consider the easy case where $k = 1$, that is, only one center c needs to be chosen so that $\sum_{i=1}^{n} d(x_i, c)^2 = \sum_{i=1}^{n} (x_i - c)^2$ is minimized. Derive an expression for such a c in terms of $x_1, \ldots, x_n$.

Define $f(c) = \sum_{i=1}^{n} (x_i - c)^2$. Then $f'(c) = -2\sum_{i=1}^{n} (x_i - c)$ and $f''(c) = 2$ $\therefore$ c is a local minimum iff $f'(c) = 0$ ie $c = \frac{\sum_i x_i}{n}$.

Since this is a unique local minimum, it is a global minimum.

(b) (10 points) Design a polynomial time algorithm that, given a sorted array $X = [x_1, \ldots, x_n]$ of data points in $\mathbb{R}$ and a number $k \leq n$, computes the cost of a cost-minimizing set of k cluster centers. You can use the following fact without proving it: for any choice C of cluster centers and $c \in C$, the data points which have c as their closest cluster center are contiguous in X.

If $[y_1 \cdots y_m]$ is an array of points ~~p~~clustered into a single cluster, ~~its~~ the cluster center is $\frac{\sum_i y_i}{m}$, and the cost of these points ~~=~~is $\sum_{j=1}^{m} \left(y_j - \frac{\sum_i y_i}{m} \right)^2$.

Denote this ~~by~~ quantity by $cost([y_1 \cdots y_m])$.

We design a dynamic programming algorithm.

We use a 2D table T with rows indexed by $0 \cdots n$ and columns by $1 \cdots k$. $T[i][j]$ is the optimum cost of clustering $[x_1 \cdots x_i]$ using j clusters.

Base cases ~~of j = 1 is hard~~: $T[i][1] = cost([y_1 \cdots y_i])$ if $i > 0$ ~~(0 if i = 0)~~.

$T[0][j] = 0$

Recurrence for $T[i][j]$:

$$T[i][j] = \min_{i' \in \{0, \dots, i-1\}} T[i'][j-1] + \text{cost}([x_{i'+1}, \dots, x_i]).$$

Finally, we return $T[n][k]$.

Running time = size of table × time per entry

$= O(nk) \times O(n) = O(n^2 k)$.

2. A researcher has written n papers, and her i'th paper has a_i citations. The *h-index* of a researcher is defined as the largest number k such that at least k papers written by that researcher have at least k citations each.

   In each of the following algorithm design problems concerning h-index computation, write proofs of correctness and running time only if you think they are not obvious from the way you write the algorithm. Ideally, in the interest of time, you should write algorithms in such a way that proofs become obvious and unnecessary.

   (a) (5 points) Suppose the array $A = [a_1, \ldots, a_n]$ of citation counts is sorted in non-increasing order, that is, $a_1 \geq \cdots \geq a_n$. Design an $O(\log n)$-time algorithm to compute the h-index.

```
$a_0 \leftarrow \infty$, $a_{n+1} \leftarrow 0$.
left $\leftarrow$ 0, right $\leftarrow$ n+1.
while right - left > 1 :
    (Loop invariant: a[left] > left, and a[right] < right)
    mid $\leftarrow \left\lfloor \frac{\text{left + right}}{2} \right\rfloor$

    If a[mid] = mid :
            return mid
            (papers 1...mid have $\geq$ mid citations.
             no paper other than 1...mid-1 has > mid
             citations.)
    ElseIf a[mid] < mid :
            right $\leftarrow$ mid.
    Else (a[mid] > mid)
            left $\leftarrow$ mid
return left (loop invariant ensures left $\leq$ hindex < right.
             termination condition ensures right = left+1)
```

$T(n) = T(n/2) + 1$ $\therefore$ $T(n)$ is $O(\log n)$

(b) (5 points) Now suppose the array $A = [a_1, \ldots, a_n]$ of citation counts isn't necessarily sorted. Design an $O(n)$-time algorithm to compute the h-index.

Using one pass over A, compute an array $C = [c_1 \ldots c_n]$, where c_i = # papers with exactly i citations, for $i < n$

c_n = # papers with $\geq n$ citations.

Using one pass over C, compute an array $S = [s_1 \ldots s_n]$, where $s_n = c_n$; $s_i = s_{i+1} + c_i$ for $i < n$, so that s_i = # papers with #citations $\geq i$.

Using a linear scan over S, find the largest k such that $s_k \geq k$.

Return k.

Name: ________________________ **Entry number:** ________________

COL351 Major Exam Duration: 2 hours

3. (10 points) Recall that we discussed in class how to find a minimum cardinality vertex cover of a bipartite graph efficiently. But what if some vertices are costlier than others?

   You are given a bipartite graph $G = (L, R, E)$ and a function $c : L \cup R \longrightarrow \mathbb{R}_{\geq 0}$ that assigns a non-negative cost to every vertex of G. The cost of a set of vertices is simply the sum of the costs of vertices in that set. Design a strongly polynomial time algorithm that, given G and c, computes a minimum cost vertex cover. Give a short proof of correctness and strongly polynomial running time.

Construct a network from G as follows.

- Add vertex s, connect s to every $u \in L$ with an edge of capacity $c(u)$
- Add vertex t, connect every $v \in R$ to t with an edge of capacity $c(v)$.
- Set capacity of all edges in E, oriented from L to R, to ∞.

Find a~~n min~~ s-t mincut, say S in this ~~net~~ network.

Return $(\overset{L\setminus S}{\cancel{S\setminus t}}) \cup (S \cap R)$.

Claim: Let $S \subseteq L \cup R \cup \{s, t\}$ be a set such that $s \in S$ and $t \notin S$. Then S has a finite capacity iff $(\overset{L\setminus S}{\cancel{S\setminus t}}) \cup (S \cap R)$ is a vertex cover of G.

Proof: S has finite capacity $\Leftrightarrow \nexists\, u \in S \cap L$ and $v \in R \setminus S$ such that $\{u, v\} \in E \Leftrightarrow (S \cap L) \cup (R \setminus S)$ is an independent set in $G \Leftrightarrow (L \cup R) \setminus [(S \cap L) \cup (R \setminus S)] = (\overset{L\setminus S}{\cancel{S\setminus t}}) \cup (S \cap R)$ is a vertex cover of G.

Moreover, if S is finite capacity cut, then its capacity is equal to the weight of $(L \setminus S) \cup (S \cap R)$

Also, if C is a vertex cover, then its ~~capacity~~ weight is equal to the capacity of the cut $\{s\} \cup (L \setminus C) \cup (R \cap C)$.

This proves that the algorithm is correct.

4. (8 points) Recall the (partial) story of Alibaba told to you in class: Alibaba manages to sneak into the cave of the forty thieves and manages to bring home a knapsack full of indivisible precious items. The next episode in the story is as follows. The news of Alibaba's fortune reaches Qasim, his greedy brother. Qasim carries k identical knapsacks, each of an integral capacity W, and manages to sneak into the cave. Inside, Qasim sees n (indivisible) items having weights $w_1, \ldots, w_n$. Given the numbers k and W and the array $[w_1, \ldots, w_n]$, Qasim wants to find out whether it is possible to pack all the n items into his k knapsacks such that the total weight of items packed into each knapsack is at most W. Prove that this problem is `NP-complete`. (Qasim tries to solve the problem by brute force and, by the time he computes the answer, he forgets the password for getting out. You know the rest of the story.)

NP membership: The verifier accepts an array y of size n as a proof. It treats $y[i]$ as the identity of the knapsack in which item i is put, ~~in~~ in some feasible packing.

Verifier: On input k, W, $[~~y~~w_1 \ldots w_n]$, $[y_1 \ldots y_n]$:

- If $y_i \notin \{1 \ldots k\}$ for some i, then reject.
- If $\sum_{i: y[i]=j} w_j > W$ for some $j \in \{1 \ldots k\}$, reject
- Accept.

NP hardness: Reduction from SetBisection.

On input $A = [a_1, \ldots a_n]$;

$k \leftarrow 2$, $W \leftarrow \Sigma_i di/2$

Output (k, W, A).

If A has a bisection, say (A_1, A_2), pack A_1 in one knapsack and A_2 in the other.

If a packing exists, both knapsacks must be full. Let A_1 (resp. A_2) be the set of items in knapsack 1 (resp. 2). Then (A_1, A_2) is a bisection of A.