A. [6.0 Points] Use substitution (Iterative) method to give tight asymptotic bound for the following recurrence relation:

$$T(n) = 7\,T\left(\frac{n}{2}\right) + c\,(n^2)\,, \quad \text{for } n > 1 \text{ and } T(1) = 1$$

Iter 1 : $7\ T(n/2) + C(n^2)$

Iter 2 : $7^2\ T(n/2^2) + 7\ C(n/2)^2 + C(n^2)$

Iter 3 : $7^3\ T(n/2^3) + 7^2\ C(n/2^2)^2 + 7\ C(n/2)^2 + C(n^2)$

Iter K : $7^k\ T(n/2^k) + 7^{k-1}\ C(n/2^{k-1})^2 + 7^{k-2}\ C(n/2^{k-2})^2 + C(n^2)$

$7^k\ T(n/2^k)$ ➔ highet order

$T(n) = O(7^{Log(n)})$ ➔ $O(n^{Log(7)})$

B. [11 Points] Describe O(n lg m)-time complexity Alg, for merging m sorted lists of objects into one sorted list, such that n represents the total number of objects in all the input lists.

```
function MergeSortedLists(lists):

  Create a min-heap (priority queue)

  for i from 1 to m:

    if lists[i] is not empty:

      Insert (lists[i][0], i) into the heap  # Store element and list index

  result = []

  while the heap is not empty:

    (value, listIndex) = ExtractMin from the heap

    Append value to result

    Advance the pointer for lists[listIndex]

    if lists[listIndex] is not empty:

      Insert (lists[listIndex][nextPointer], listIndex) into the heap

  return result
```

C. [3.0 Marks] Analyze Huffman algorithm to find its time complexity.

Build min-heap -> O(n)

Loop:

1. N for each operation
2. Extract form heap -> O(logn)

Alg. Take O(n logn)

D. [8.0 Marks] Let G = (V, E) be a simple graph with n vertices and the weight of every edge of G is equal to one. Compute in detail the weight of MST of G?

The weight of the Minimum Spanning Tree (MST) of a graph G=(V,E)G = (V, E)G=(V,E) with nnn vertices and all edge weights equal to 1 is n−1n - 1n−1, provided the graph is connected. This is because an MST contains n−1n - 1n−1 edges, and each edge has weight 1. If the graph is disconnected, an MST does not exist

E. [11.0 Marks] If S is an unsorted array of k integers (any element of M could be either positive or negative integer), design O(k lg k) worst-case time algorithm that searches two numbers $x, y \in M, x \neq y,$ such that $| \boldsymbol{x} + \boldsymbol{y} |$ is the minimum among all pairs in M.

```
function FindMinAbsSumPair(M):
  Sort array M in non-decreasing order  # O(k log k)
  i = 0
  j = len(M) - 1
  minAbsSum = ∞
  resultPair = (None, None)

  while i < j:
    s = M[i] + M[j]
    if |s| < minAbsSum:
      minAbsSum = |s|
      resultPair = (M[i], M[j])

    if s < 0:
      i = i + 1
    else if s > 0:
      j = j - 1
    else:
      break  # |s| = 0 is the smallest possible value

  return resultPair
```

F. [11 Points] Discuss how you can compute in_degree and out_degree of the nodes of a graph given when it is represented by adjacency list.

Definition:

Out-degree of a node: The number of edges leaving the node.

In-degree of a node: The number of edges arriving at the node.

Compute:

Out-degree of node u: The size of adj[u], O(1).

In-degree of node v: Count how many times v appears in the adjacency lists of all nodes, O(m).

Time Complexity is O(n + m)

n is the number of nodes

m is the number of edges