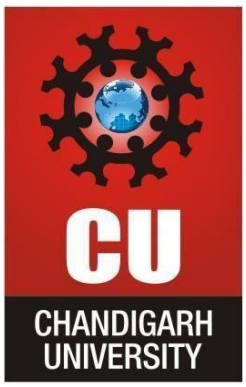

Winter Winning Camp

ASSIGNMENT – 5

Student Name: Ishan Jain

UID: 22BCS13653

Date of Submission: 26/12/2024

Branch: BE-CSE

Section/Group: 615-**B**

DAY – 5

```
#include <iostream>
```

```
#include <vector>
#include <unordered_map>
#include <algorithm>
#include <queue>
#include <stack>
#include <numeric>
#include <cmath> using
namespace std;
```

// Very Easy Questions

// Q1: Searching a Number

```
int searchNumber(int k, vector<int>& arr) {     for
(int i = 0; i < arr.size(); i++) {         if (arr[i] == k)
return i + 1; // 1-based indexing
    }
return -1;
}
```

// Q2: Sorted Array Search

```
bool sortedArraySearch(vector<int>& arr, int k) {
return find(arr.begin(), arr.end(), k) != arr.end();
}
```

// Q3: Find Target Indices After Sorting Array

```
vector<int> targetIndices(vector<int>& nums, int target) {
sort(nums.begin(), nums.end());     vector<int> indices;
```

```
    for (int i = 0; i < nums.size(); i++) {
if (nums[i] == target) {
indices.push_back(i);
        }     }
return indices;
}
```

// Q4: Search Insert Position

```
int searchInsert(vector<int>& nums, int target) {
int left = 0, right = nums.size() - 1;     while (left
<= right) {         int mid = left + (right - left) / 2;
if (nums[mid] == target) return mid;         else if
(nums[mid] < target) left = mid + 1;         else
right = mid - 1;
    }     return left; // Position to
insert
}
```

// Q5: Relative Sort Array

```
vector<int> relativeSortArray(vector<int>& arr1, vector<int>& arr2) {
unordered_map<int, int> count;     for (int num : arr1) {
count[num]++;
    }     vector<int> result;
for (int num : arr2) {
while (count[num] > 0) {
result.push_back(num);
count[num]--;
```

```
        }
    }
    for (auto& [num, cnt] : count) {
while (cnt > 0) {
result.push_back(num);
            cnt--;
        }     }
return result;
}
```

// Easy Questions

// Q6: Minimum Number of Moves to Seat Everyone

```
int minMovesToSeat(vector<int>& seats, vector<int>& students) {
sort(seats.begin(), seats.end());     sort(students.begin(),
students.end());     int moves = 0;     for (int i = 0; i < seats.size();
i++) {         moves += abs(seats[i] - students[i]);
    }
    return moves;
}
```

// Q7: Squares of a Sorted Array

```
vector<int> sortedSquares(vector<int>& nums) {
vector<int> result(nums.size());     for (int i = 0; i
< nums.size(); i++) {         result[i] = nums[i] *
nums[i];
    }     sort(result.begin(),
result.end());     return result;
```

```
}
```

// Q8: Common in 3 Sorted Arrays vector<int>

```
commonInThreeSortedArrays(vector<int>& arr1, vector<int>& arr2,
vector<int>& arr3) {     vector<int> result;     int i = 0, j = 0, k = 0;     while (i <
arr1.size() && j < arr2.size() && k < arr3.size()) {         if (arr1[i] == arr2[j] &&
arr2[j] == arr3[k]) {             result.push_back(arr1[i]);             i++; j++; k++;
       } else if (arr1[i] < arr2[j]) {
i++;
       } else if (arr2[j] < arr3[k]) {
j++;        } else {             k++;
       }
   }
   return result.empty() ? vector<int>{-1} : result;
}
```

// Q9: Sort Even and Odd Indices Independently

```
vector<int> sortEvenOdd(vector<int>& nums) {
vector<int> even, odd;     for (int i = 0; i <
nums.size(); i++) {         if (i % 2 == 0)
even.push_back(nums[i]);         else
odd.push_back(nums[i]);

   }
   sort(even.begin(), even.end());
sort(odd.rbegin(), odd.rend());     for (int i = 0; i <
nums.size(); i++) {         nums[i] = (i % 2 == 0) ?
even[i / 2] : odd[i / 2];
```

```
    }
    return nums;
}
```

// Q10: Left Most and Right Most Index

```
vector<int> findFirstAndLast(vector<int>& nums, int target) {
    int first = -1, last = -1;      for (int i = 0; i <
nums.size(); i++) {          if (nums[i] == target) {
if (first == -1) first = i; // First occurrence
last = i; // Last occurrence
        }     }     return
{first, last};
}
```

// Medium Questions

// Q11: Search in 2D Matrix

```
bool searchMatrix(vector<vector<int>>& matrix, int target) {
int m = matrix.size(), n = matrix[0].size();     int left = 0, right
= m * n - 1;     while (left <= right) {          int mid = left +
(right - left) / 2;
        if (matrix[mid / n][mid % n] == target) return true;
else if (matrix[mid / n][mid % n] < target) left = mid + 1;
else right = mid - 1;
    }     return
false;
}
```

// Q12: Find First and Last Position of Element in Sorted Array

```
vector<int> searchRange(vector<int>& nums, int target) {     int left =
lower_bound(nums.begin(), nums.end(), target) - nums.begin();     int right =
upper_bound(nums.begin(), nums.end(), target) - nums.begin() - 1;     if (left <=
right) return {left, right};     return {-1, -1};
}
```

// Q13: Find Minimum in Rotated Sorted Array int

```
findMin(vector<int>& nums) {
   int left = 0, right = nums.size() - 1;     while (left
< right) {         int mid = left + (right - left) / 2;
if (nums[mid] > nums[right]) left = mid + 1;
else if (nums[mid] < nums[right]) right = mid;
else right--; // Handle duplicates
   }     return
nums[left];
}
```

// Q14: Smallest Positive Missing Number

```
int smallestMissingPositive(vector<int>& arr) {
   int n = arr.size();
   for (int i = 0; i < n; i++) {         while (arr[i] > 0 && arr[i] <= n
&& arr[arr[i] - 1] != arr[i]) {             swap(arr[i], arr[arr[i] - 1]);
      }     }     for (int i = 0; i < n;
i++) {         if (arr[i] != i + 1)
return i + 1;
   }     return n
+ 1;
```

```
}
```

// Q15: Pair Sum Closest to 0

```
int closestToZero(vector<int>& arr) {
sort(arr.begin(), arr.end());     int left
= 0, right = arr.size() - 1;
   int closestSum = INT_MAX;
   while (left < right) {         int
sum = arr[left] + arr[right];
      if (abs(sum) < abs(closestSum) || (abs(sum) == abs(closestSum) && sum > closestSum))
{            closestSum =
sum;
      }
      if (sum < 0) left++;
else right--;
   }
   return closestSum;
}
```

// Hard Questions

// Q16: Sort Items by Groups Respecting Dependencies

```
vector<int> sortItems(int n, int m, vector<int>& group, vector<vector<int>>& beforeItems) {
vector<vector<int>> groupItems(m);     vector<vector<int>> graph(n);     vector<int>
inDegree(n, 0);     for (int i = 0; i < n; i++) {         if (group[i] == -1) {             group[i] =
m++;
      }
      groupItems[group[i]].push_back(i);
```

```
    }     for (int i = 0; i < n; i++) {
for (int before : beforeItems[i]) {
graph[before].push_back(i);
inDegree[i]++;
        }
    }     vector<int> result;     for (int i = 0; i < m; i++) {
vector<int> items;         for (int item : groupItems[i]) {
if (inDegree[item] == 0) items.push_back(item);
        }
        while (!items.empty()) {             int curr =
items.back();             items.pop_back();
result.push_back(curr);             for (int next : graph[curr])
{                 inDegree[next]--;                 if (inDegree[next]
== 0) items.push_back(next);             }

        }
    }     return result.size() == n ? result :
vector<int>();
}
```

// Q17: Find the Kth Smallest Sum of a Matrix With Sorted Rows

```
int kthSmallest(vector<vector<int>>& mat, int k) {
priority_queue<int, vector<int ```cpp >,
greater<int>> minHeap;     int m = mat.size(), n =
mat[0].size();     vector<int> indices(m, 0);
minHeap.push(0); // Initial sum is 0
```

```
    for (int i = 0; i < k; i++) {         int
currentSum = minHeap.top();
minHeap.pop();

        if (i == k - 1) return currentSum;

        for (int j = 0; j < m; j++) {             if (indices[j] < n)
{                 int newSum = currentSum +
mat[j][indices[j]];                 minHeap.push(newSum);
                indices[j]++;
            }
        }
    }
    return -1; // Should not reach here
}
```

// Q18: Merge k Sorted Lists

```
struct ListNode {
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(NULL) {}
};

ListNode* mergeKLists(vector<ListNode*>& lists) {     auto cmp = [](ListNode*
a, ListNode* b) { return a->val > b->val; };     priority_queue<ListNode*,
vector<ListNode*>, decltype(cmp)> minHeap(cmp);
```

```
    for (ListNode* list : lists) {
if (list) minHeap.push(list);
    }
    ListNode dummy(0);
    ListNode* tail = &dummy;

    while (!minHeap.empty()) {        ListNode*
node = minHeap.top();        minHeap.pop();
tail->next = node;        tail = tail->next;        if
(node->next) minHeap.push(node->next);
    }
    return dummy.next;
}
```

// Q19: Max Chunks To Make Sorted II

```
int maxChunksToSorted(vector<int>& arr) {
    int n = arr.size();     vector<int>
maxLeft(n), minRight(n);     maxLeft[0] =
arr[0];     for (int i = 1; i < n; i++) {
maxLeft[i] = max(maxLeft[i - 1], arr[i]);
    }
    minRight[n - 1] = arr[n - 1];     for (int i = n -
2; i >= 0; i--) {        minRight[i] =
min(minRight[i + 1], arr[i]);
    }     int chunks = 0;     for (int i = 0; i <
n - 1; i++) {        if (maxLeft[i] <=
minRight[i + 1]) {            chunks++;
        }
```

```
    }
    return chunks + 1; // Add the last chunk
}
```

// Q20: Find Minimum in Rotated Sorted Array II

```
int findMinII(vector<int>& nums) {
int left = 0, right = nums.size() - 1;
while (left < right) {         int mid =
left + (right - left) / 2;         if
(nums[mid] > nums[right]) {
left = mid + 1;
        } else if (nums[mid] < nums[right]) {
right = mid;
        } else {
            right--; // Handle duplicates
        }
    }     return
nums[left];
}
```

// Q21: Median of Two Sorted Arrays

```
double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
if (nums1.size() > nums2.size()) {         swap(nums1, nums2);
    }
    int m = nums1.size(), n = nums2.size();     int
left = 0, right = m, halfLen = (m + n + 1) / 2;
double maxOfLeft, minOfRight;
```

```
    while (left <= right) {         int i = left +
(right - left) / 2;         int j = halfLen - i;
if (i < m && nums1[i] < nums2[j - 1]) {
left = i + 1; // i is too small
        } else if (i > 0 && nums1[i - 1] > nums2[j]) {
right = i - 1; // i is too big         } else { // i is perfect
maxOfLeft = 0;             if (i == 0) maxOfLeft = nums2[j -
1];             else if (j == 0) maxOfLeft = nums1[i - 1];
else maxOfLeft = max(nums1[i - 1], nums2[j - 1]);


            if ((m + n) % 2 == 1) return maxOfLeft; // Odd case

            minOfRight = 0;             if (i == m) minOf
```cpp Right = nums2[j];             else if (j == n)
minOfRight = nums1[i];             else minOfRight =
min(nums1[i], nums2[j]);


            return (maxOfLeft + minOfRight) / 2.0; // Even case
        }
    }
    return 0.0; // Should not reach here
}
```

// Q22: Create Sorted Array through Instructions

```
int createSortedArray(vector<int>& instructions) {
const int MOD = 1e9 + 7;     vector<int>
count(100001, 0);     long long totalCost = 0;     for
(int num : instructions) {         int less = 0, greater
```
```

```
= 0;        for (int i = 1; i < num; i++) {            less
+= count[i];
        }
        for (int i = num + 1; i < 100001; i++) {
greater += count[i];
        }
        totalCost = (totalCost + min(less, greater)) % MOD;
count[num]++;
    }     return
totalCost;
}
```

// Q23: Kth Smallest Product of Two Sorted Arrays

```
long long kthSmallestProduct(vector<int>& nums1, vector<int>& nums2, int k) {     long
long left = -1e10, right = 1e10;     while (left < right) {         long long mid = left + (right -
left) / 2;         long long count = 0;         for (int num1 : nums1) {             count +=
upper_bound(nums2.begin(), nums2.end(), mid / num1) - nums2.begin();
        }
        if (count < k) left = mid + 1;
else right = mid;
    }     return
left;
}
```

// Q24: Sorted GCD Pair Queries vector<int> sortedGcdPairQueries(vector<int>& nums, vector<int>& queries) {

```
vector<int> gcdPairs;     int n = nums.size();     for (int i = 0; i < n; i++) {         for
(int j = i + 1; j < n; j++) {             gcdPairs.push_back(__gcd(nums[i], nums[j]));
```

```
        }
    }
    sort(gcdPairs.begin(), gcdPairs.end());
vector<int> result;     for (int q : queries)
{       result.push_back(gcdPairs[q]);
    }
    return result;
}
```