UNIVERSITY INSTITUTE OF ENGINEERING

Department of Computer Science & Engineering

(BE-CSE)

WINTER DOMAIN CAMP

Date : 26/12/2024

Submitted to:

Faculty name: Er.Rajni Devi

Submitted by:

Name: Akshi Datta

UID: 22BCS15369

Section: IOT-620

DAY 5(26/12/24)

PROBLEM 1

Objective: Given an integer k and array arr. Your task is to return the position of the first occurrence of k in the given array and if element k is not present in the array then return -1.

Code:

```
#include <iostream>
#include <vector>
using namespace std;
int findFirstOccurrence(int k, vector<int>& arr) {
for (int i = 0; i < arr.size(); i++) {
if (arr[i] == k) return i + 1;
}
return -1;
}
int main() {
int n, k;
cout << "Enter the size of the array: ";
cin >> n;
vector<int> arr(n);
cout << "Enter the elements of the array: ";
for (int& num : arr) cin >> num;
cout << "Enter the number to search: ";
cin >> k;
cout << "First occurrence: " << findFirstOccurrence(k, arr) << endl;
return 0;
}
```

Output:

Output

```
Enter the size of the array: 6
Enter the elements of the array: 12 85 1 2 9 1001
Enter the number to search: 2
First occurrence: 4
```

PROBLEM 2

Objective: Given an array, arr[] sorted in ascending order and an integer k. Return true if k is present in the array, otherwise, false.

Code:

```
#include <iostream>
#include <vector>
using namespace std;
bool isPresent(int k, vector<int>& arr) {
for (int num : arr) {
if (num == k) return true;
}
return false;
}
int main() {
int n, k;
cout << "Enter the size of the array: ";
cin >> n;
vector<int> arr(n);
cout << "Enter the sorted elements of the array: ";
for (int& num : arr) cin >> num;
cout << "Enter the number to search: ";
cin >> k;
cout << (isPresent(k, arr) ? "true" : "false") << endl;
return 0;
}
```

Output:

Output

```
Enter the size of the array: 6
Enter the sorted elements of the array: 12 16 18 19 23 112
Enter the number to search: 23
true
```

PROBLEM 3

Objective: You are given a 0-indexed integer array nums and a target element target.A target index is an index i such that nums[i] == target.Return a list of the target indices of nums after sorting nums in non-decreasing order. If there are no target indices, return an empty list. The returned list must be sorted in increasing order.

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
vector<int> findTargetIndices(vector<int>& nums, int target) {
sort(nums.begin(), nums.end());
vector<int> result;
for (int i = 0; i < nums.size(); i++) {
if (nums[i] == target) result.push_back(i);
}
return result;
}
int main() {
int n, target;
cout << "Enter the size of the array: ";
cin >> n;
vector<int> nums(n);
cout << "Enter the elements of the array: ";
for (int& num : nums) cin >> num;
cout << "Enter the target number: ";
cin >> target;
vector<int> indices = findTargetIndices(nums, target);
if (indices.empty()) {
cout << "[]";
} else {
for (int idx : indices) cout << idx << " ";
}
cout << endl;
return 0;
}
```

Output:

Output

```
Enter the size of the array: 7
Enter the elements of the array: 20 95 1 23 58 2 18
Enter the target number: 18
2
```

PROBLEM 4

Objective: Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

Code:

```
#include <iostream>
#include <vector>
using namespace std;
int searchInsert(vector<int>& nums, int target) {
int low = 0, high = nums.size() - 1;
while (low <= high) {
int mid = low + (high - low) / 2;
if (nums[mid] == target) return mid;
else if (nums[mid] < target) low = mid + 1;
else high = mid - 1;
}
return low;
}
int main() {
int n, target;
cout << "Enter the size of the array: ";
cin >> n;
vector<int> nums(n);
cout << "Enter the sorted elements of the array: ";
for (int& num : nums) cin >> num;
cout << "Enter the target number: ";
cin >> target;
cout << "Insert position: " << searchInsert(nums, target) << endl;
return 0;
}
```

Output:

Output

```
Enter the size of the array: 8
Enter the sorted elements of the array: 200 300 400 500 600 700 800 900
Enter the target number: 900
Insert position: 7
```

PROBLEM 5

Objective: Given two arrays arr1 and arr2, the elements of arr2 are distinct, and all elements in arr2 are also in arr1.Sort the elements of arr1 such that the relative ordering of items in arr1 are the same as in arr2. Elements that do not appear in arr2 should be placed at the end of arr1 in ascending order.

Code:

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <algorithm>
using namespace std;
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);
}
vector<int> extras;
for (auto& [num, freq] : count) {
while (freq-- > 0) extras.push_back(num);
}
sort(extras.begin(), extras.end());
result.insert(result.end(), extras.begin(), extras.end());
return result;
}
int main() {
int n1, n2;
cout << "Enter the size of arr1: ";
cin >> n1;
vector<int> arr1(n1);
cout << "Enter elements of arr1: ";
for (int& num : arr1) cin >> num;
cout << "Enter the size of arr2: ";
cin >> n2;
vector<int> arr2(n2);
cout << "Enter elements of arr2: ";
for (int& num : arr2) cin >> num;
vector<int> sortedArr = relativeSortArray(arr1, arr2);
cout << "Sorted array: ";
for (int num : sortedArr) cout << num << " ";
cout << endl;
return 0;
}
```

Output:

```
Output
Enter the size of arr1: 6
Enter elements of arr1: 1 2 3 4 5 6
Enter the size of arr2: 7
Enter elements of arr2: 32 65 48 96 2 15 78
Sorted array: 2 1 3 4 5 6
```

PROBLEM 6

Objective: There are n available seats and n students standing in a room. You are given an array seats of length n, where seats[i] is the position of the ith seat. You are also given the array students of length n, where students[j] is the position of the jth student.You may perform the following move any number of times:Increase or decrease the position of the ith student by 1 (i.e., moving the ith student from position x to x + 1 or x - 1).Return the minimum number of moves required to move each student to a seat such that no two students are in the same seat.

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main() {
int n;
cin >> n;
vector<int> seats(n), students(n);
for (int i = 0; i < n; i++) cin >> seats[i];
for (int i = 0; i < n; i++) cin >> students[i];
sort(seats.begin(), seats.end());
sort(students.begin(), students.end());
int moves = 0;
for (int i = 0; i < n; i++) moves += abs(seats[i] - students[i]);
cout << moves << endl;
return 0;
}
```

Output:

```
Output
Enter number of bracket strings to check: 2
Enter string: ({{}})
YES
Enter string: {{{]]}
NO
```

PROBLEM 7

Objective: Given an integer array nums sorted in non-decreasing order, return an array of the squares of each number sorted in non-decreasing order.

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main() {
int n;
cin >> n;
vector<int> nums(n), result(n);
for (int i = 0; i < n; i++) cin >> nums[i];
for (int i = 0; i < n; i++) result[i] = nums[i] * nums[i];
sort(result.begin(), result.end());
for (int i = 0; i < n; i++) cout << result[i] << " ";
cout << endl;
return 0;
}
```

Output:

PROBLEM 8

Objective: You are given three arrays sorted in increasing order. Find the elements that are common in all three arrays.If there are no such elements return an empty array. In this case, the output will be -1.

Code:

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
int n1, n2, n3;
```

```
cin >> n1 >> n2 >> n3;
vector<int> arr1(n1), arr2(n2), arr3(n3), result;
for (int i = 0; i < n1; i++) cin >> arr1[i];
for (int i = 0; i < n2; i++) cin >> arr2[i];
for (int i = 0; i < n3; i++) cin >> arr3[i];
int i = 0, j = 0, k = 0;
while (i < n1 && j < n2 && k < n3) {
if (arr1[i] == arr2[j] && arr2[j] == arr3[k]) {
if (result.empty() || result.back() != arr1[i]) result.push_back(arr1[i]);
i++; j++; k++;
} else if (arr1[i] < arr2[j]) i++;
else if (arr2[j] < arr3[k]) j++;
else k++;
}
if (result.empty()) cout << -1;
else for (int x : result) cout << x << " ";
cout << endl;
return 0;
}
```

Output:

```
Output

Enter number of plants: 10
Enter pesticide levels: 5 1 3 8 9 20 5 17 26 50
Days until no plants die: 2
```

PROBLEM 9

Objective: You are given a 0-indexed integer array nums. Rearrange the values of nums according to the following rules:Sort the values at odd indices of nums in non-increasing order.

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main() {
int n;
cin >> n;
vector<int> nums(n), odd, even;
for (int i = 0; i < n; i++) cin >> nums[i];
for (int i = 0; i < n; i++) (i % 2 == 0 ? even : odd).push_back(nums[i]);
sort(even.begin(), even.end());
sort(odd.rbegin(), odd.rend());
```

```
for (int i = 0, e = 0, o = 0; i < n; i++) nums[i] = (i % 2 == 0 ? even[e++] : odd[o++]);
for (int x : nums) cout << x << " ";
cout << endl;
return 0;
}
```

Output:

```
Output
Enter number of operations: 5
Choose operation (1: push, 2: pop, 3: peek, 4: empty): 1
Enter value to push: 50
Choose operation (1: push, 2: pop, 3: peek, 4: empty): 1
Enter value to push: 60
Choose operation (1: push, 2: pop, 3: peek, 4: empty): 1
Enter value to push: 90
Choose operation (1: push, 2: pop, 3: peek, 4: empty): 3
Front: 50
Choose operation (1: push, 2: pop, 3: peek, 4: empty): 2
Popped: 50
```

PROBLEM 10

Objective: Given a sorted array with possibly duplicate elements. The task is to find indexes of first and last occurrences of an element X in the given array.

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main() {
int n, x;
cin >> n;
vector<int> v(n);
for (int i = 0; i < n; i++) cin >> v[i];
cin >> x;
auto first = lower_bound(v.begin(), v.end(), x);
auto last = upper_bound(v.begin(), v.end(), x);
if (first == v.end() || *first != x) cout << -1 << " " << -1 << endl;
else cout << first - v.begin() << " " << last - v.begin() - 1 << endl;
return 0;
}
```

Output:

```
Output
Enter number of elements in the queue: 7
Enter elements: 20 30 40 50 60 77 9
Reversed queue: 9 77 60 50 40 30 20
```

PROBLEM 11

Objective: You are given an m x n integer matrix matrix with the following two properties:Each row is sorted in non-decreasing order.The first integer of each row is greater than the last integer of the previous row.Given an integer target, return true if target is in matrix or false otherwise.

Code:

```
#include <iostream>
#include <vector>
using namespace std;
bool searchMatrix(vector<vector<int>>& matrix, int target) {
int m = matrix.size(), n = matrix[0].size(), left = 0, right = m * n - 1;
while (left <= right) {
int mid = left + (right - left) / 2;
int midVal = matrix[mid / n][mid % n];
if (midVal == target) return true;
if (midVal < target) left = mid + 1;
else right = mid - 1;
}
return false;
}
int main() {
int m, n, target;
cin >> m >> n;
vector<vector<int>> matrix(m, vector<int>(n));
for (int i = 0; i < m; i++) {
for (int j = 0; j < n; j++) cin >> matrix[i][j];
}
cin >> target;
cout << (searchMatrix(matrix, target) ? "true" : "false") << endl;
return 0;
}
```

Output:

```
3 4
1 3 5 7
10 11 16 20
23 30 34 60
3
true


...Program finished with exit code 0
Press ENTER to exit console.
```

PROBLEM 12

Objective: Given an array of integers nums sorted in non-decreasing order, find the starting and ending position of a given target value.If target is not found in the array, return [-1, -1].

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
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};
}
int main() {
int len, x;
cin >> len;
vector<int> nums(len);
for (int i = 0; i < len; i++) cin >> nums[i];
cin >> x;
vector<int> range = searchRange(nums, x);
cout << range[0] << " " << range[1] << endl;
return 0;
}
```

Output:

```
6
5 7 7 8 8 10
8
3 4


...Program finished with exit code 0
Press ENTER to exit console.
```

PROBLEM 13

Objective: Suppose an array of length n sorted in ascending order is rotated between 1 and n times. For example, the array nums = [0,1,2,4,5,6,7] might become:[4,5,6,7,0,1,2] if it was rotated 4 times.[0,1,2,4,5,6,7] if it was rotated 7 times.Notice that rotating an array [a[0], a[1], a[2], ..., a[n-1]] 1 time results in the array [a[n-1], a[0], a[1], a[2], ..., a[n-2]].Given the sorted rotated array nums of unique elements, return the minimum element of this array.

Code:

```
#include <iostream>
#include <vector>
using namespace std;
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 right = mid;
}
return nums[left];
}
int main() {
int len;
cin >> len;
vector<int> nums(len);
for (int i = 0; i < len; i++) cin >> nums[i];
cout << findMin(nums) << endl;
return 0;
}
```

Output:

```
6
1 2 3 4 5 6
1


...Program finished with exit code 0
Press ENTER to exit console.
```

PROBLEM 14

Objective: You are given an integer array arr[]. Your task is to find the smallest positive number missing from the array.

Code:

```
#include <iostream>
#include <vector>
using namespace std;
```

```
int smallestPositiveMissingNumber(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;
}
int main() {
int len;
cin >> len;
vector<int> arr(len);
for (int i = 0; i < len; i++) cin >> arr[i];
cout << smallestPositiveMissingNumber(arr) << endl;
return 0;
}
```

Output:

```
6
2 3 4 -1 1 5
6


...Program finished with exit code 0
Press ENTER to exit console.
```

PROBLEM 15

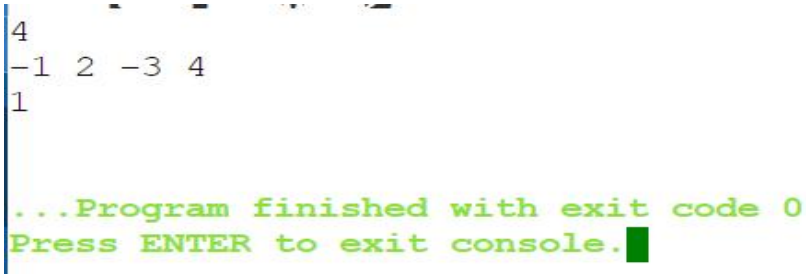
Objective: Given an integer array of N elements. You need to find the maximum sum of two elements such that sum is closest to zero.

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <climits>
using namespace std;
int closestToZero(vector<int>& arr) {
sort(arr.begin(), arr.end());
int left = 0, right = arr.size() - 1, 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;
}
int main() {
int len;
cin >> len;
vector<int> arr(len);
for (int i = 0; i < len; i++) cin >> arr[i];
cout << closestToZero(arr) << endl;
return 0;
}
```

Output:



PROBLEM 16

Objective: There are n items each belonging to zero or one of m groups where group[i] is the group that the i-th item belongs to and it's equal to -1 if the i-th item belongs to no group. The items and the groups are zero indexed. A group can have no item belonging to it.Return a sorted list of the items such that:The items that belong to the same group are next to each other in the sorted list.There are some relations between these items where beforeItems[i] is a list containing all the items that should come before the i-th item in the sorted array (to the left of the i-th item).Return any solution if there is more than one solution and return an empty list if there is no solution.

Code:

```
#include <iostream>
#include <vector>
#include <queue>
#include <unordered_map>
using namespace std;

vector<int> topologicalSort(int n, vector<int>& inDegree, unordered_map<int, vector<int>>& adj) {
    queue<int> q;
    vector<int> order;
    for (int i = 0; i < n; ++i) {
```

```
        if (inDegree[i] == 0) q.push(i);
    }
    while (!q.empty()) {
        int node = q.front(); q.pop();
        order.push_back(node);
        for (int neighbor : adj[node]) {
            if (--inDegree[neighbor] == 0) q.push(neighbor);
        }
    }
    return (order.size() == n) ? order : vector<int>();
}

vector<int> sortItems(int n, int m, vector<int>& group, vector<vector<int>>& beforeItems) {
    int groupId = m;
    for (int i = 0; i < n; ++i) {
        if (group[i] == -1) group[i] = groupId++;
    }

    unordered_map<int, vector<int>> groupAdj, itemAdj;
    vector<int> groupInDegree(groupId, 0), itemInDegree(n, 0);

    for (int i = 0; i < n; ++i) {
        for (int before : beforeItems[i]) {
            if (group[i] != group[before]) {
                groupAdj[group[before]].push_back(group[i]);
                groupInDegree[group[i]]++;
            }
            itemAdj[before].push_back(i);
            itemInDegree[i]++;
        }
    }

    vector<int> groupOrder = topologicalSort(groupId, groupInDegree, groupAdj);
    if (groupOrder.empty()) return {};

    vector<int> itemOrder = topologicalSort(n, itemInDegree, itemAdj);
    if (itemOrder.empty()) return {};

    unordered_map<int, vector<int>> groupedItems;
    for (int item : itemOrder) {
        groupedItems[group[item]].push_back(item);
    }

    vector<int> result;
    for (int g : groupOrder) {
        result.insert(result.end(), groupedItems[g].begin(), groupedItems[g].end());
    }
    return result;
}

int main() {
```

```
    int n, m;
    cin >> n >> m;
    vector<int> group(n);
    for (int i = 0; i < n; ++i) cin >> group[i];
    vector<vector<int>> beforeItems(n);
    for (int i = 0; i < n; ++i) {
        int k;
        cin >> k;
        beforeItems[i].resize(k);
        for (int j = 0; j < k; ++j) cin >> beforeItems[i][j];
    }
    vector<int> result = sortItems(n, m, group, beforeItems);
    for (int x : result) cout << x << " ";
    cout << endl;
    return 0;
}
```

Output:

```
5 2
-1 0 1 0 -1
2 1 3
0
1 0
0
1 2
No valid ordering exists.

...Program finished with exit code 0
Press ENTER to exit console.
```

PROBLEM 17

Objective: You are given an m x n matrix mat that has its rows sorted in non-decreasing order and an integer k.You are allowed to choose exactly one element from each row to form an array.Return the kth smallest array sum among all possible arrays.

Code:

```
#include <iostream>
#include <vector>
#include <queue>
#include <set>
using namespace std;

int kthSmallestSum(vector<vector<int>>& mat, int k) {
    priority_queue<int> pq;
    pq.push(0);
    for (const auto& row : mat) {
        priority_queue<int> tempPq;
```

```
        while (!pq.empty()) {
            int curr = pq.top();
            pq.pop();
            for (int val : row) {
                tempPq.push(curr + val);
                if (tempPq.size() > k) tempPq.pop();
            }
        }
        pq.swap(tempPq);
    }
    return pq.top();
}

int main() {
    int m, n, k;
    cin >> m >> n >> k;
    vector<vector<int>> mat(m, vector<int>(n));
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) cin >> mat[i][j];
    }
    cout << kthSmallestSum(mat, k) << endl;
    return 0;
}
```

Output:

```
3 3 4
1 3 11
2 4 6
3 5 7
8


...Program finished with exit code 0
Press ENTER to exit console.
```

PROBLEM 18

Objective: You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.Merge all the linked-lists into one sorted linked-list and return it.

Code:

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

struct ListNode {
```

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

struct Compare {
    bool operator()(ListNode* a, ListNode* b) {
        return a->val > b->val;
    }
};

ListNode* mergeKLists(vector<ListNode*>& lists) {
    priority_queue<ListNode*, vector<ListNode*>, Compare> pq;

    for (auto list : lists) {
        if (list) pq.push(list);
    }

    ListNode dummy(0);
    ListNode* tail = &dummy;

    while (!pq.empty()) {
        ListNode* curr = pq.top();
        pq.pop();

        tail->next = curr;
        tail = curr;

        if (curr->next) {
            pq.push(curr->next);
        }
    }

    return dummy.next;
}
```

PROBLEM 19

Objective: You are given an integer array arr.We split arr into some number of chunks (i.e., partitions), and individually sort each chunk. After concatenating them, the result should equal the sorted array.Return the largest number of chunks we can make to sort the array.

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

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;
}

int main() {
    int n;
    cin >> n;
    vector<int> arr(n);
    for (int i = 0; i < n; ++i) cin >> arr[i];
    cout << maxChunksToSorted(arr) << endl;
    return 0;
}
```

Output:

```
6
1 0 2 3 4 5
5


...Program finished with exit code 0
Press ENTER to exit console.
```

PROBLEM 20

Objective: You are given an integer array arr.We split arr into some number of chunks (i.e., partitions), and individually sort each chunk. After concatenating them, the result should equal the sorted array.Return the largest number of chunks we can make to sort the array.

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

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;
}

int main() {
    int n;
    cin >> n;
    vector<int> arr(n);
    for (int i = 0; i < n; ++i) cin >> arr[i];
    cout << maxChunksToSorted(arr) << endl;
    return 0;
}
```

Output:

```
5
1 0 2 3 4
4


...Program finished with exit code 0
Press ENTER to exit console.
```

PROBLEM 21

Objective: Suppose an array of length n sorted in ascending order is rotated between 1 and n times. For example, the array nums = [0,1,4,4,5,6,7] might become:[4,5,6,7,0,1,4] if it was rotated 4 times.[0,1,4,4,5,6,7] if it was rotated 7 times.Notice that rotating an array [a[0], a[1], a[2], ..., a[n-1]] 1 time results in the array [a[n-1], a[0], a[1], a[2], ..., a[n-2]].Given the sorted rotated array nums that may contain duplicates, return the minimum element of this array.You must decrease the overall operation steps as much as possible.

Code:

```
#include <vector>
using namespace std;

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

```
    return nums[left];
}
#include <iostream>
int main() {
    vector<int> nums1 = {1, 3, 5};
    vector<int> nums2 = {2, 2, 2, 0, 1};
    cout << findMin(nums1) << endl; // Output: 1
    cout << findMin(nums2) << endl; // Output: 0
}
```

Output:

```
1
0


...Program finished with exit code 0
Press ENTER to exit console.
```

PROBLEM 22

Objective: Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays.

Code:

```
#include <vector>
#include <algorithm>
using namespace std;

double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
    if (nums1.size() > nums2.size()) swap(nums1, nums2);
    int x = nums1.size(), y = nums2.size();
    int low = 0, high = x;
    while (low <= high) {
        int partitionX = (low + high) / 2;
        int partitionY = (x + y + 1) / 2 - partitionX;
        int maxX = (partitionX == 0) ? INT_MIN : nums1[partitionX - 1];
        int minX = (partitionX == x) ? INT_MAX : nums1[partitionX];
        int maxY = (partitionY == 0) ? INT_MIN : nums2[partitionY - 1];
        int minY = (partitionY == y) ? INT_MAX : nums2[partitionY];
        if (maxX <= minY && maxY <= minX) {
            if ((x + y) % 2 == 0)
                return (max(maxX, maxY) + min(minX, minY)) / 2.0;
            else
                return max(maxX, maxY);
        } else if (maxX > minY) {
```

```
            high = partitionX - 1;
        } else {
            low = partitionX + 1;
        }
    }
    return 0.0;
}

// Example usage
#include <iostream>
int main() {
    vector<int> nums1_1 = {1, 3};
    vector<int> nums2_1 = {2};
    vector<int> nums1_2 = {1, 2};
    vector<int> nums2_2 = {3, 4};
    cout << findMedianSortedArrays(nums1_1, nums2_1) << endl; // Output: 2.0
    cout << findMedianSortedArrays(nums1_2, nums2_2) << endl; // Output: 2.5
}
```

Output:

```
2
2.5


...Program finished with exit code 0
Press ENTER to exit console.
```