

DOMAIN WINTER WINNING CAMP 2024

Assignment Day 5

1)Searching a Number

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.

Note: 1-based indexing is followed here.

Example1:

Input: k = 16 , arr = [9, 7, 16, 16, 4]

Output: 3

Explanation: The value 16 is found in the given array at positions 3 and 4, with position 3 being the first occurrence.

Example2:

Input: k=98 , arr = [1, 22, 57, 47, 34, 18, 66]

Output: -1

SOLUTION:

```
#include <iostream>
```

```
#include <vector>
```

```
using namespace std;
```

```
int searchNumber(int k, vector<int>& arr) {  
    for (int i = 0; i < arr.size(); i++) {  
        if (arr[i] == k) {  
            return i + 1; // Return 1-based index  
        }  
    }  
    return -1; // If k is not found in the array  
}
```

```
int main() {  
    // Example input  
    int k = 16;  
    vector<int> arr = {9, 7, 16, 16, 4};
```

```

int result = searchNumber(k, arr);
cout << result << endl;

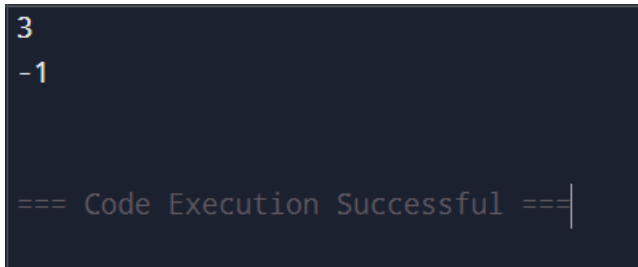
// Test another input
k = 98;
arr = {1, 22, 57, 47, 34, 18, 66};

result = searchNumber(k, arr);
cout << result << endl;

return 0;
}

```

OUTPUT:



```

3
-1

=== Code Execution Successful ===

```

2)Sorted array Search.

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

Example 1:

Input: arr[] = [1,2,3,4,6], k=6

Output: true

Explanation: Since, 6 is present in the array at index4 (0-based indexing), Output is true.

Example 2:

Input: arr[] = [1, 2, 4, 5, 6], k = 3

Output: false

Example 3:

Input: arr[] = [1, 2, 4, 5, 6], k = 6

Output: true

SOLUTION

```
#include <iostream>
#include <vector>
using namespace std;
bool binarySearch(const vector<int>& arr, int k) {
    int left = 0, right = arr.size() - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (arr[mid] == k) {
            return true;
        } else if (arr[mid] < k) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return false;
}

int main() {
    vector<int> arr = {1, 2, 4, 5, 6};
    int k = 6;

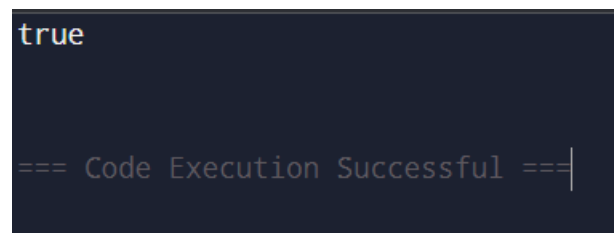
    bool result = binarySearch(arr, k);

    if (result) {
        cout << "true" << endl;
    } else {
```

```
        cout << "false" << endl;
    }

    return 0;
}
```

OUTPUT



```
true

=== Code Execution Successful ===
```

3)Find Target Indices After Sorting Array.

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.

Example 1:

Input: `nums = [1,2,5,2,3]`, `target = 2`

Output: `[1,2]`

Explanation: After sorting, `nums` is `[1,2,2,3,5]`.

The indices where `nums[i] == 2` are 1 and 2.

SOLUTION:

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

```

vector<int> targetIndices(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() {
    vector<int> nums = {1, 2, 5, 2, 3};
    int target = 2;

    vector<int> result = targetIndices(nums, target);
    cout << "[";
    for (size_t i = 0; i < result.size(); ++i) {
        cout << result[i];
        if (i < result.size() - 1) {
            cout << ", ";
        }
    }
    cout << "]" << endl;
    return 0;
}

```

OUTPUT:

```
[1, 2]
```

```
=== Code Execution Successful ===
```

4)Search Insert Position.

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.

You must write an algorithm with $O(\log n)$ runtime complexity.

Example 1:

Input: nums = [1,3,5,6], target = 5

Output: 2

Example 2:

Input: nums = [1,3,5,6], target = 2

Output: 1

SOLUTION

```
#include <iostream>
#include <vector>
using namespace std;
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; // Target found
        } else if (nums[mid] < target) {
            left = mid + 1; // Search in the right half
        } else {
            right = mid - 1; // Search in the left half
        }
    }
    return left;
}

int main() {
    vector<int> nums = {1, 3, 5, 6}; // Input array
```

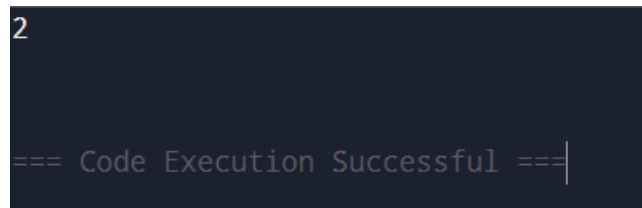
```

int target = 5;

int result = searchInsert(nums, target);
cout << result << endl;
return 0;
}

```

OUTPUT:



```

2

=== Code Execution Successful ===

```

5) Relative Sort Array.

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.

Example 1:

Input: arr1 = [2,3,1,3,2,4,6,7,9,2,19], arr2 = [2,1,4,3,9,6]

Output: [2,2,2,1,4,3,3,9,6,7,19]

SOLUTION:

```

#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> freqMap;
    for (int num : arr1) {

```

```

        freqMap[num]++;
    }
    vector<int> result;
    for (int num : arr2) {
        while (freqMap[num] > 0) {
            result.push_back(num);
            freqMap[num]--;
        }
    }
    vector<int> remaining;
    for (auto& entry : freqMap) {
        while (entry.second > 0) {
            remaining.push_back(entry.first);
            entry.second--;
        }
    }
    sort(remaining.begin(), remaining.end());
    result.insert(result.end(), remaining.begin(), remaining.end());

    return result;
}

int main() {
    vector<int> arr1 = {2, 3, 1, 3, 2, 4, 6, 7, 9, 2, 19};
    vector<int> arr2 = {2, 1, 4, 3, 9, 6};

    vector<int> result = relativeSortArray(arr1, arr2);

    for (int num : result) {
        cout << num << " ";
    }

    return 0;
}

```


OUTPUT:

```
2 2 2 1 4 3 3 9 6 7 19
```

```
=== Code Execution Successful ===
```

6) Minimum Number of Moves to Seat Everyone

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 i th seat. You are also given the array `students` of length n , where `students[j]` is the position of the j th student.

You may perform the following move any number of times:

Increase or decrease the position of the i th student by 1 (i.e., moving the i th 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.

Note that there may be multiple seats or students in the same position at the beginning.

Example 1:

Input: `seats = [3,1,5]`, `students = [2,7,4]`

Output: 4

Explanation: The students are moved as follows:

- The first student is moved from position 2 to position 1 using 1 move.
- The second student is moved from position 7 to position 5 using 2 moves.
- The third student is moved from position 4 to position 3 using 1 move.

In total, $1 + 2 + 1 = 4$ moves were used.

SOLUTION:

```
#include <iostream>
```

```
#include <vector>
```

```
#include <algorithm>
```

```
using namespace std;
```

```
int minMovesToSeat(vector<int>& seats, vector<int>& students) {
```

```
    sort(seats.begin(), seats.end());
```

```
    sort(students.begin(), students.end());
```

```
    int totalMoves = 0;
```

```

    for (int i = 0; i < seats.size(); ++i) {
        totalMoves += abs(seats[i] - students[i]);
    }

    return totalMoves;
}

int main() {
    vector<int> seats = {3, 1, 5};
    vector<int> students = {2, 7, 4};

    int result = minMovesToSeat(seats, students);
    cout << "Minimum number of moves: " << result << endl;

    return 0;
}

```

OUTPUT:

```
Minimum number of moves: 4
```

```
=== Code Execution Successful ===
```

7) Squares of a Sorted Array

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

Example 1:

Input: `nums = [-4,-1,0,3,10]`

Output: `[0,1,9,16,100]`

Explanation: After squaring, the array becomes `[16,1,0,9,100]`.

After sorting, it becomes `[0,1,9,16,100]`.

Example 2:

Input: `nums = [-7,-3,2,3,11]`

Output: `[4,9,9,49,121]`

SOLUTION:

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

vector<int> sortedSquares(vector<int>& nums) {
    int n = nums.size();
    vector<int> result(n);
    int left = 0, right = n - 1, index = n - 1;
    while (left <= right) {
        if (abs(nums[left]) > abs(nums[right])) {
            result[index--] = nums[left] * nums[left];
            left++;
        } else {
            result[index--] = nums[right] * nums[right];
            right--;
        }
    }

    return result;
}

int main() {
    vector<int> nums = {-4, -1, 0, 3, 10};
    vector<int> result = sortedSquares(nums);

    cout << "Sorted squares: ";
    for (int num : result) {
        cout << num << " ";
    }
    cout << endl;

    return 0;
}
```

OUTPUT:

```
Sorted squares: 0 1 9 16 100
```

```
=== Code Execution Successful ===
```

8) Common in 3 Sorted Arrays.

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.

Note: can you handle the duplicates without using any additional Data Structure?

Example1:

Input: arr1 = [1, 5, 10, 20, 40, 80] , arr2 = [6, 7, 20, 80, 100] , arr3 = [3, 4, 15, 20, 30, 70, 80, 120]

Output: [20, 80]

SOLUTION:

```
#include <iostream>
#include <vector>
using namespace std;
vector<int> commonInThreeSortedArrays(vector<int>& arr1, vector<int>& arr2,
vector<int>& arr3) {
    int i = 0, j = 0, k = 0;
    vector<int> result;
    while (i < arr1.size() && j < arr2.size() && k < arr3.size()) {
        // If arr1[i] == arr2[j] == arr3[k], it's a common element
        if (arr1[i] == arr2[j] && arr2[j] == arr3[k]) {
            // Add to result and move all three pointers
            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()) {
        result.push_back(-1);
    }
    return result;
}

int main() {
    vector<int> arr1 = {1, 5, 10, 20, 40, 80};
    vector<int> arr2 = {6, 7, 20, 80, 100};
    vector<int> arr3 = {3, 4, 15, 20, 30, 70, 80, 120};
    vector<int> result = commonInThreeSortedArrays(arr1, arr2, arr3);
    for (int num : result) {
        cout << num << " ";
    }
    cout << endl;

    return 0;
}

```

OUTPUT:

```
20 80
```

```
=== Code Execution Successful ===
```

9) Sort Even and Odd Indices Independently.

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.

For example, if `nums = [4,1,2,3]` before this step, it becomes `[4,3,2,1]` after. The values at odd indices 1 and 3 are sorted in non-increasing order.

Sort the values at even indices of `nums` in non-decreasing order.

For example, if `nums = [4,1,2,3]` before this step, it becomes `[2,1,4,3]` after. The values at even indices 0 and 2 are sorted in non-decreasing order.

Return the array formed after rearranging the values of `nums`.

Example 1:

Input: `nums = [4,1,2,3]`

Output: `[2,3,4,1]`

Explanation:

First, we sort the values present at odd indices (1 and 3) in non-increasing order. So, `nums` changes from `[4,1,2,3]` to `[4,3,2,1]`.

Next, we sort the values present at even indices (0 and 2) in non-decreasing order. So, `nums` changes from `[4,3,2,1]` to `[2,3,4,1]`.

Thus, the array formed after rearranging the values is `[2,3,4,1]`.

SOLUTION:

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

vector<int> sortEvenOddIndices(vector<int>& nums) {
    vector<int> evenValues, oddValues;
    for (int i = 0; i < nums.size(); i++) {
        if (i % 2 == 0) {
            evenValues.push_back(nums[i]); // Even index
        } else {
            oddValues.push_back(nums[i]); // Odd index
        }
    }
}
```

```

    sort(evenValues.begin(), evenValues.end());
    sort(oddValues.rbegin(), oddValues.rend());
    int evenIndex = 0, oddIndex = 0;
    for (int i = 0; i < nums.size(); i++) {
        if (i % 2 == 0) {
            nums[i] = evenValues[evenIndex++];
        } else {
            nums[i] = oddValues[oddIndex++];
        }
    }

    return nums;
}

int main() {
    vector<int> nums = {4, 1, 2, 3};
    vector<int> result = sortEvenOddIndices(nums);
    for (int num : result) {
        cout << num << " ";
    }
    cout << endl;

    return 0;
}

```

OUTPUT:

```
2 3 4 1
```

```
=== Code Execution Successful ===
```

10) Left most and Right most index.

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.

Note: If the element is not present in the array return {-1,-1} as pair.

Example1:

Input: N = 9

v[] = {1, 3, 5, 5, 5, 5, 67, 123, 125}

X = 5

Output:2 5

Explanation:

Index of first occurrence of 5 is 2

and index of last occurrence of 5 is 5.

SOLUTION:

```
#include <iostream>
```

```
#include <vector>
```

```
using namespace std;
```

```
// Function to find the first occurrence of X
```

```
int findFirstOccurrence(const vector<int>& arr, int X) {
```

```
    int low = 0, high = arr.size() - 1, first = -1;
```

```
    while (low <= high) {
```

```
        int mid = low + (high - low) / 2;
```

```
        if (arr[mid] == X) {
```

```
            first = mid; // Found X, but continue searching in the left half
```

```
            high = mid - 1;
```

```
        } else if (arr[mid] < X) {
```

```
            low = mid + 1;
```

```
        } else {
```

```
            high = mid - 1;
```

```
        }
```

```
    }
```



```

    return first;
}

// Function to find the last occurrence of X
int findLastOccurrence(const vector<int>& arr, int X) {
    int low = 0, high = arr.size() - 1, last = -1;

    while (low <= high) {
        int mid = low + (high - low) / 2;

        if (arr[mid] == X) {
            last = mid; // Found X, but continue searching in the right half
            low = mid + 1;
        } else if (arr[mid] < X) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return last;
}

```

```

pair<int, int> findLeftRightIndices(const vector<int>& arr, int X) {
    int first = findFirstOccurrence(arr, X);
    int last = findLastOccurrence(arr, X);

    // If the element is not found, return {-1, -1}
    if (first == -1) {
        return {-1, -1};
    }
    return {first, last};
}

```

```

int main() {
    vector<int> arr = {1, 3, 5, 5, 5, 5, 67, 123, 125};
}

```

```
int X = 5;

// Get the leftmost and rightmost indices of X
pair<int, int> result = findLeftRightIndices(arr, X);

// Output the result
cout << result.first << " " << result.second << endl;

return 0;
}
```

OUTPUT:

```
2 5

=== Code Execution Successful ===
```