DAY 5

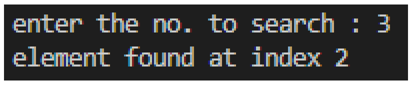
Student Name: Akarsh **UID: 22BCS15248**
Branch: BE-CSE **Section/Group: 620 - A**
Date of Performance:26/12/24

Problem 1

1. **Aim: Searching a Number**
2. **Code:**

```
int main()
{
    int arr[] = {1,2,3,4,5,6};
    int n;
    cout<<"enter the no. to search : ";
    cin>>n;
    for (int i = 0; i < 6; i++)
    {
        if (arr[i] == n)
        {
            cout<<"element found at index "<<i;
            break;
            }
    }
    return 0;
}
```

3. **Output:**



```
enter the no. to search : 3
element found at index 2
```

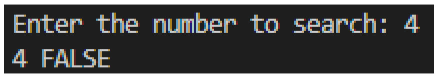
Problem 2

1. **Aim: Sorted array Search**

2. Code:

```
bool isPresent(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, 3, 5, 7, 9, 11};
    int k;
    cout << "Enter the number to search: ";
    cin >> k;
    if (isPresent(arr, k)) {
        cout << k << " TRUE" << endl;
    } else {
        cout << k << " FALSE" << endl;
    }
    return 0;
}
```

3. Output:



```
Enter the number to search: 4
4 FALSE
```

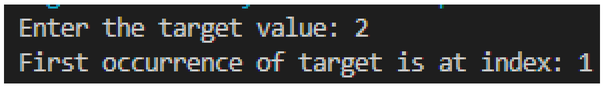
Problem 3

1. Aim: .Find Target Indices After Sorting Array

2. Code:

```
int findFirstOccurrence(const vector<int>& arr, int target) {
    int left = 0, right = arr.size() - 1;
    int result = -1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target) {
            result = mid;
            right = mid - 1;
        } else if (arr[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return result;
}
int main() {
    vector<int> arr = {1, 2, 2, 2, 3, 4, 5};
    int target;
    cout << "Enter the target value: ";
    cin >> target;
    int index = findFirstOccurrence(arr, target);
    if (index != -1) {
        cout << "First occurrence of target is at index: " << index << endl;
    } else {
        cout << "Target not found in the array." << endl;
    }
    return 0;
}
```

3. Output:



```
Enter the target value: 2
First occurrence of target is at index: 1
```

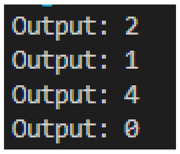
Problem 4

1. Aim: Search Insert Position

2. Code:

```
int searchInsert(vector<int>& nums, int target) {
    int left = 0;
    int 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;
}
int main() {
    vector<int> nums = {1, 3, 5, 6};
    int target = 5;
    int result = searchInsert(nums, target);
    cout << "Output: " << result <<endl;
    target = 2;
    result = searchInsert(nums, target);
    cout << "Output: " << result << std::endl;
    target = 7;

    result = searchInsert(nums, target);

    cout << "Output: " << result <<endl;

    target = 0;

    result = searchInsert(nums, target);

    cout << "Output: " << result <<endl;

    return 0;

}
```

3. Output:



```
Output: 2
Output: 1
Output: 4
Output: 0
```

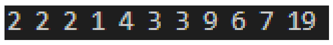
Problem 5

1. Aim: Relative Sort Array

2. Code:

```
vector<int> relativeSortArray(vector<int>& arr1, vector<int>& arr2) {
    unordered_map<int, int> orderMap;
    for (int i = 0; i < arr2.size(); ++i) {
        orderMap[arr2[i]] = i;
    }
    auto comparator = [&orderMap](int a, int b) {
        bool aInArr2 = orderMap.find(a) != orderMap.end();
        bool bInArr2 = orderMap.find(b) != orderMap.end();
        if (aInArr2 && bInArr2) {
            return orderMap[a] < orderMap[b];
        } else if (aInArr2) {
            return true;
        } else if (bInArr2) {
            return false;
        } else {
            return a < b;
        }
    };
    sort(arr1.begin(), arr1.end(), comparator);
    return arr1;
}

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> sortedArray = relativeSortArray(arr1, arr2);
    for (int num : sortedArray) {
        cout << num << " ";
    }
    cout << endl;
    return 0;
}
```

3. Output:



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

Problem 6

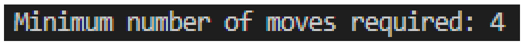
1. Aim: Sum of Odd Numbers up to N

2. Code:

```
#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 required: " << result << endl;
    return 0;
}
```

3. **Output:**



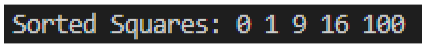
```
Minimum number of moves required: 4
```

Problem 7

1. **Aim: Squares of a Sorted Array**
2. **Code:**

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

int main() {
    int num, rev_num = 0;
    cout << "Enter a number: ";
    cin >> num;
    while (num != 0) {
        rev_num = rev_num * 10 + num % 10;
        num /= 10;
    }
    cout << "Reversed Number: " << rev_num << endl;
    return 0;
}
```

3. **Output:**



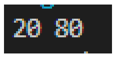
```
Sorted Squares: 0 1 9 16 100
```

Problem 8

1. **Aim: Common in 3 Sorted Arrays**
2. **Code:**

```
#include <iostream>
#include <vector>
std::vector<int>  commonInThreeSortedArrays(const  std::vector<int>&
arr1, const std::vector<int>& arr2, const std::vector<int>& arr3) {
    std::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]) {
            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()) {
        return {-1};
    }
    return result;
}
int main() {
    std::vector<int> arr1 = {1, 5, 10, 20, 40, 80};
    std::vector<int> arr2 = {6, 7, 20, 80, 100};
    std::vector<int> arr3 = {3, 4, 15, 20, 30, 70, 80, 120};
    std::vector<int>                    commonElements                    =
commonInThreeSortedArrays(arr1, arr2, arr3);
    if (commonElements.size() == 1 && commonElements[0] == -1) {
        std::cout << -1 << std::endl;
    } else {
        for (int num : commonElements) {
            std::cout << num << " ";
        }
        std::cout << std::endl;
    }

    return 0;
}
```

3. Output:



```
20 80
```

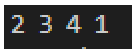
Problem 9

1. **Aim: Sort Even and Odd Indices Independently.**

2. **Code:**

```
#include <iostream>
#include <vector>
#include <algorithm>
std::vector<int> sortEvenOdd(std::vector<int>& nums) {
    std::vector<int> oddIndices;
    std::vector<int> evenIndices;
    for (int i = 0; i < nums.size(); ++i) {
        if (i % 2 == 0) {
            evenIndices.push_back(nums[i]);
        } else {
            oddIndices.push_back(nums[i]);
        }
    }
    std::sort(evenIndices.begin(), evenIndices.end());
    std::sort(oddIndices.rbegin(), oddIndices.rend());
    for (int i = 0, j = 0, k = 0; i < nums.size(); ++i) {
        if (i % 2 == 0) {
            nums[i] = evenIndices[j++];
        } else {
            nums[i] = oddIndices[k++];
        }
    }
    return nums;
}
int main() {
    std::vector<int> nums = {4, 1, 2, 3};
    std::vector<int> result = sortEvenOdd(nums);
    for (int num : result) {
        std::cout << num << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

3. **Output:**



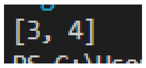
```
2 3 4 1
```

Problem 10

1. **Aim: Find First and Last Position of Element in Sorted Array.**
2. **Code:**

```
#include <iostream>
#include <vector>
using namespace std;
class Solution {
public:
    vector<int> searchRange(vector<int>& nums, int target) {
        vector<int> result = {-1, -1};
        result[0] = findFirstPosition(nums, target);
        result[1] = findLastPosition(nums, target);
        return result;
    }

private:
    int findFirstPosition(const vector<int>& nums, int target) {
        int left = 0, right = nums.size() - 1;
        int firstPos = -1;

        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (nums[mid] == target) {
                firstPos = mid; // Found target, continue searching on the left side
                right = mid - 1;
            } else if (nums[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return firstPos;
    }

    int findLastPosition(const vector<int>& nums, int target) {
        int left = 0, right = nums.size() - 1;
        int lastPos = -1;

        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (nums[mid] == target) {
                lastPos = mid; // Found target, continue searching on the right
side
                left = mid + 1;
            } else if (nums[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return lastPos;
    }
};

int main() {
    Solution solution;
    vector<int> nums = {5, 7, 7, 8, 8, 10};
    int target = 8;
    vector<int> result = solution.searchRange(nums, target);

    cout << "[" << result[0] << ", " << result[1] << "]" << endl; // Output: [3,
4]
    return 0;
}
```

3. Output:



```
[3, 4]
```