**Kaushik Dhruv Alok**
**22BCS10210**
**KPIT-901/A**
**Day 5 Questions**

**Q1. 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.

**Q2. 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.

**Q3. Search in 2D Matrix.**
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.

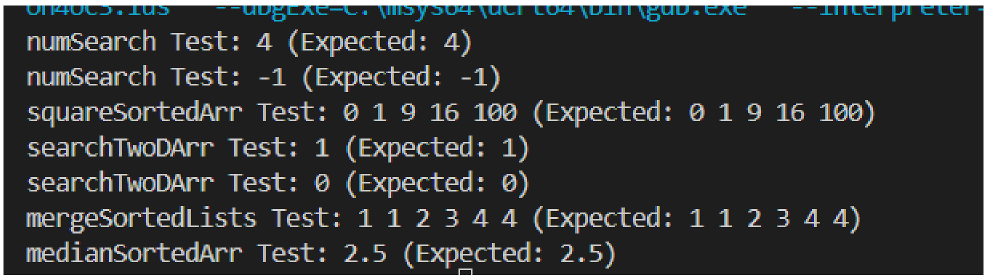
**Q4. Merge k Sorted Lists**
You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.
Merge all the linked-lists into one sorted linked-list and return it.

**Q5. Median of Two Sorted Arrays**
Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays.

**Code:**
```
#include <iostream>
#include <map>
#include <vector>
#include <math.h>
#include <algorithm>
#include <stdlib.h>

using namespace std;

int numSearch(vector<int>& nums, int target) {
   std::map<int, int> numMap;
   for (int i = 0; i < nums.size(); i++) {
      numMap[nums[i]] = i;
   }
   if (numMap.find(target) == numMap.end()) {
      return -1;
   }
   return numMap[target];
}
```

```
vector<int> squareSortedArr(vector<int>& nums) {
    for (size_t i = 0; i < nums.size(); i++) {
        nums[i] = pow(nums[i], 2);
    }
    sort(nums.begin(), nums.end());
    return nums;
}

bool searchTwoDArr(vector<vector<int>>& matrix, int target) {
    for (size_t i = 0; i < matrix.size(); i++) {
        for (size_t j = 0; j < matrix[i].size(); j++) {
            if (matrix[i][j] == target) {
                return true;
            }
        }
    }
    return false;
}

vector<int> mergeSortedLists(vector<int>& nums1, vector<int>& nums2) {
    vector<int> mergeList;
    mergeList.insert(mergeList.end(), nums1.begin(), nums1.end());
    mergeList.insert(mergeList.end(), nums2.begin(), nums2.end());
    sort(mergeList.begin(), mergeList.end());
    return mergeList;
}

double medianSortedArr(vector<int>& nums1, vector<int>& nums2) {
    vector<int> nums = mergeSortedLists(nums1, nums2);
    int size = nums.size();
    if (size == 0) return 0;

    if (size % 2 == 0) {
        return (nums[size/2] + nums[size/2 - 1]) / 2.0;
    }
    return nums[size/2];
}

int main() {
    // Test Q1
    vector<int> nums1 = {4, 5, 6, 7, 0, 1, 2};
    cout << "numSearch Test: " << numSearch(nums1, 0) << " (Expected: 4)" << endl;
    cout << "numSearch Test: " << numSearch(nums1, 3) << " (Expected: -1)" << endl;

    // Test Q2
    vector<int> nums2 = {-4, -1, 0, 3, 10};
    vector<int> squared = squareSortedArr(nums2);
    cout << "squareSortedArr Test: ";
    for (int num : squared) {
        cout << num << " ";
    }
    cout << "(Expected: 0 1 9 16 100)" << endl;

    // Test Q3
    vector<vector<int>> matrix = {
        {1, 3, 5, 7},
        {10, 11, 16, 20},
        {23, 30, 34, 60}
    };
    cout << "searchTwoDArr Test: " << searchTwoDArr(matrix, 3) << " (Expected: 1)" << endl;
    cout << "searchTwoDArr Test: " << searchTwoDArr(matrix, 13) << " (Expected: 0)" << endl;

    // Test Q4
    vector<int> nums3 = {1, 2, 4};
    vector<int> nums4 = {1, 3, 4};
    vector<int> merged = mergeSortedLists(nums3, nums4);
    cout << "mergeSortedLists Test: ";
    for (int num : merged) {
        cout << num << " ";
    }
    cout << "(Expected: 1 1 2 3 4 4)" << endl;

    // Test Q5
    vector<int> nums5 = {1, 2};
    vector<int> nums6 = {3, 4};
    cout << "medianSortedArr Test: " << medianSortedArr(nums5, nums6) << " (Expected: 2.5)" <<
endl;

    return 0;
}
```

**Output:**



```
on40c3.ids    --dbgExe=C:\msys64\ucrt64\bin\gdb.exe    --interpreter-
numSearch Test: 4 (Expected: 4)
numSearch Test: -1 (Expected: -1)
squareSortedArr Test: 0 1 9 16 100 (Expected: 0 1 9 16 100)
searchTwoDArr Test: 1 (Expected: 1)
searchTwoDArr Test: 0 (Expected: 0)
mergeSortedLists Test: 1 1 2 3 4 4 (Expected: 1 1 2 3 4 4)
medianSortedArr Test: 2.5 (Expected: 2.5)
```