Name : Rudraksh Mishra
UID   : 22BCS10607
Class : KPIT - 901 / A

Q1. 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. 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. You are given an m x n integer matrix matrix with the following two properties:

- Each row is sorted in non-decreasing order.
- The 1st value in each row is greater than the last value of the previous row.

Given an integer target, return true if target is in matrix or false otherwise. You must write a solution in O(log(m * n)) time complexity.

Q4. You are given 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. 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.

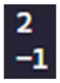
Notice that rotating an array [a[0], a[1], a[2], ..., a[n-1]] 1-time results in the array [a[n-1], a[0], a[1], a[2], ..., a[n-2]]. Given the sorted rotated array called ‘nums’ that may contain duplicates, return the minimum element of this array. You must decrease the overall operation steps as much as possible.

Solutions :

A1. Searching a Number

```
#include <iostream>
#include <vector>

using namespace std;

int search_k(vector<int> nums, int k) {
    for (int i = 0; i < nums.size(); i++) {
        if (nums[i] == k) { return i; }
    }
    return -1;
}

int main(int argc, char* argv[]) {
    vector<int> a1 = {9,7,16,16,4};
    vector<int> a2 = {1,22,57,47,34,18,66};
    cout << search_k(a1, 16) << endl;
    cout << search_k(a2, 98) << endl;
    return 0;
}
```

Output :

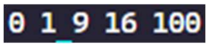


```
2
-1
```

A2. Squares of a Sorted Array

```
#include <algorithm>
#include <iostream>
#include <vector>

using namespace std;

vector<int> Sorted_Squares(vector<int>& arr) {
    vector<int> res;
    for (const int a : arr) { res.push_back(a * a); }
    sort(res.begin(), res.end());
    return res;
}

int main(int argc, char* argv[]) {
    vector<int> t = {-4,-1,0,3,10};
    vector<int> r = Sorted_Squares(t);
    for (const int a : r) { cout << a << " "; }
    return 0;
}
```

Output :



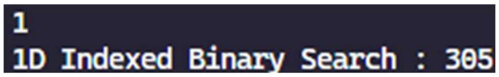
```
0 1 9 16 100
```

A3. Search in 2D Matrix

```
#include <iostream>
#include <vector>
#include <chrono>

using namespace std;

bool search_2d_matrix_sbs(vector<vector<int>>& matrix, int target) {
    bool retval = false;

    int m  = matrix.size();
    int n  = matrix[0].size();
    int lo = 0;
    int hi = m * n - 1;
    int mi = lo + (hi - lo) / 2;

    while (lo <= hi) {
        int row = mi / n;
        int col = mi % n;
        if (matrix[row][col] == target) {
            retval = true;
            break;
        }
        else if (matrix[row][col] < target) { lo = mi + 1; }
        else { hi = mi; }
        mi = lo + (hi - lo) / 2;
    }

    return retval;
}
```

```
int main(int argc, char* argv[]) {
    vector<vector<int>> t = {
        {1,  3,  5,  7},
        {10, 11, 16, 20},
        {23, 30, 34, 60}
    };

    auto start = chrono::high_resolution_clock::now();

    cout << search_2d_matrix_sbs(t, 3) << endl;

    auto stop = chrono::high_resolution_clock::now();

    auto duration = chrono::duration_cast<chrono::microseconds>(stop - start);
    cout << "1D Indexed Binary Search : " << duration.count() << endl;

    return 0;
}
```

Output :



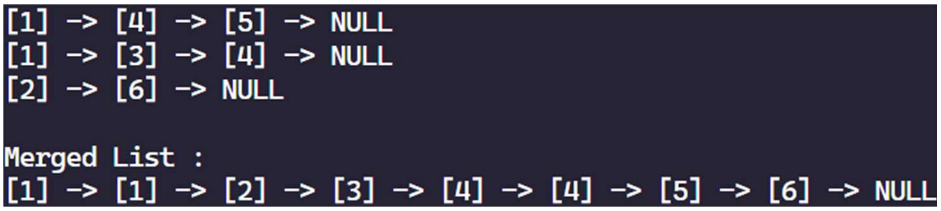
```
1
1D Indexed Binary Search : 305
```

A4. Merge k Sorted Lists.

```
#include <iostream>
#include <vector>

using namespace std;

struct Node {
    int data;
    Node* next;
    Node(int data) : data(data) , next(nullptr) {}
};

Node* Merge_2(Node* l1, Node* l2) {
    Node dummy(0);
    Node* tail = &dummy;

    while (l1 && l2) {
        if (l1->data <= l2->data) {
            tail->next = l1;
            l1 = l1->next;
        } else {
            tail->next = l2;
            l2 = l2->next;
        }
        tail = tail->next;
    }
    tail->next = l1 ? l1 : l2;

    return dummy.next;
}
```

```
Node* Merge(vector<Node*>& lists) {
    if (lists.empty()) return nullptr;

    int k = lists.size();
    int interval = 1;

    while (interval < k) {
        for (int i = 0; i < k - interval; i += interval * 2) {
            lists[i] = Merge_2(lists[i], lists[i + interval]);
        }
        interval *= 2;
    }

    return lists[0];
}

int main() {
    Node L11(1), L12(4), L13(5);
    Node L21(1), L22(3), L23(4);
    Node L31(2), L32(6);

    L11.next = &L12;
    L12.next = &L13;
    L21.next = &L22;
    L22.next = &L23;
    L31.next = &L32;

    vector<Node*> L = {&L11, &L21, &L31};

    cout << "Lists : \n";
    for (auto* List : L) {
        while (List != nullptr) {
            cout << "[" << List->data << "] -> ";
            List = List->next;
        }
        cout << "NULL" << endl;
    }

    Node* M = Merge(L);

    cout << "\nMerged List : \n";
    while (M != nullptr) {
        cout << "[" << M->data << "] -> ";
        M = M->next;
    }
    cout << "NULL" << endl;

    return 0;
}
```

Output :

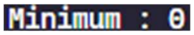


```
[1] -> [4] -> [5] -> NULL
[1] -> [3] -> [4] -> NULL
[2] -> [6] -> NULL

Merged List :
[1] -> [1] -> [2] -> [3] -> [4] -> [4] -> [5] -> [6] -> NULL
```

A5. Find Minimum in Rotated Sorted Array

```
#include <iostream>
#include <vector>

using namespace std;

int Find_Minimum(vector<int>& nums) {
    int lo = 0;
    int hi = nums.size() - 1;

    if (nums[lo] < nums[hi]) { return nums[lo]; }

    while (lo < hi) {
        int mid = lo + (hi - lo) / 2;

        if (nums[mid] == nums[hi]) {
            lo++;
            continue;
        }

        if (nums[mid] > nums[hi])      { lo = mid + 1; }
        else if (nums[mid] < nums[hi]) { hi = mid; }
        else                           { hi--; }
    }

    return nums[lo];
}

int main(int argc, char* argv[]) {
    vector<int> t = {2,2,2,0,1};
    cout << "Minimum : " << Find_Minimum(t) << endl;
    return 0;
}
```

Output :



```
Minimum : 0
```