ASSIGNMENT – 05

# 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

**Example2:**

**Input:** k=9 , arr = [1, 22, 57, 47, 34, 9, 66]

# Output: 6

 **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 k1 = 16;

vector<int> arr1 = {9, 7, 16, 16, 4};

cout << findFirstOccurrence(k1, arr1) << endl;

int k2 = 98; vector<int> arr2 = {1, 22, 57,

47, 34, 18, 66}; cout <<

findFirstOccurrence(k2, arr2) << endl;

int k3 = 9; vector<int> arr3 = {1, 22, 57,

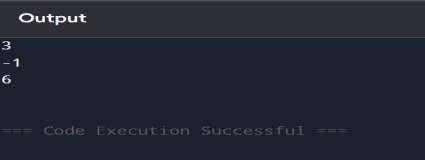
47, 34, 9, 66}; cout <<

findFirstOccurrence(k3, arr3) << endl;

return 0;

}

# Output:

****

1. **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]

# Constraints:

* 1 <= nums.length <= 104
* -104 <= nums[i] <= 104
* nums is sorted in non-decreasing order.

# Code:

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

<algorithm> using

namespace std;

vector<int> sortedSquares(vector<int>& nums) { for (int i = 0; i < nums.size(); i++) {

nums[i] = nums[i] \* nums[i];

}

sort(nums.begin(), nums.end()); return nums;

}

int main() {

vector<int> nums1 = {-4, -1, 0, 3, 10}; vector<int> result1 = sortedSquares(nums1);

for (int x : result1) { cout << x << " ";

}

cout << endl;

vector<int> nums2 = {-7, -3, 2, 3, 11}; vector<int> result2 = sortedSquares(nums2);

for (int x : result2) { cout << x << " ";

}

cout << endl;

return 0;

}

# Output:

****

1. **Find First and Last Position of Element in Sorted Array.**

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].

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

# Example 1:

**Input:** nums = [5,7,7,8,8,10], target = 8

**Output:** [3,4]

# Example 2:

**Input:** nums = [5,7,7,8,8,10], target = 6

**Output:** [-1,-1]

# Example 3:

**Input:** nums = [], target = 0

**Output:** [-1,-1]

# Constraints:

0 <= nums.length <= 105

-109 <= nums[i] <= 109

nums is a non-decreasing array.

-109 <= target <= 109

# Code:

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

vector<int> searchRange(vector<int>& nums, int target) { int left = 0, right = nums.size() - 1, first = -1, last = -1;

// Find the first occurrence while (left <= right) { int mid

= left + (right - left) / 2; if (nums[mid] == target) {

first = mid; right = mid - 1;

} else if (nums[mid] < target) { left = mid + 1;

} else { right = mid - 1;

}

}

left = 0, right = nums.size() - 1;

// Find the last occurrence

while (left <= right) { int mid

= left + (right - left) / 2; if (nums[mid] == target) {

last = mid; left = mid + 1;

} else if (nums[mid] < target) {

left = mid + 1;

} else { right = mid - 1;

}

}

return {first, last};

}

int main() {

vector<int> nums1 = {5, 7, 7, 8, 8, 10}; int target1 = 8;

vector<int> result1 = searchRange(nums1, target1);

cout << "[" << result1[0] << ", " << result1[1] << "]" << endl;

vector<int> nums2 = {5, 7, 7, 8, 8, 10}; int target2 = 6;

vector<int> result2 = searchRange(nums2, target2);

cout << "[" << result2[0] << ", " << result2[1] << "]" << endl;

vector<int> nums3 = {}; int target3 = 0;

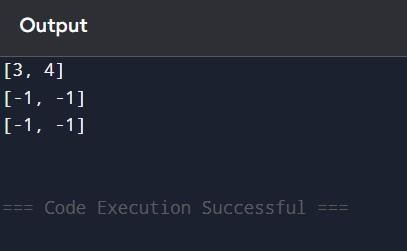
vector<int> result3 = searchRange(nums3, target3);

cout << "[" << result3[0] << ", " << result3[1] << "]" << endl;

return 0;

}

# Output:

****

1. **Find the Kth Smallest Sum of a Matrix With Sorted Rows.**

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.

# Example 1:

**Input:** mat = [[1,3,11],[2,4,6]], k = 5

# Output: 7

**Explanation**: Choosing one element from each row, the first k smallest sum are: [1,2], [1,4], [3,2], [3,4], [1,6]. Where the 5th sum is 7.

# Example 2:

**Input:** mat = [[1,3,11],[2,4,6]], k = 9

# Output: 17

**Example 3:**

**Input:** mat = [[1,10,10],[1,4,5],[2,3,6]], k = 7

# Output: 9

**Explanation:** Choosing one element from each row, the first k smallest sum are: [1,1,2], [1,1,3], [1,4,2], [1,4,3], [1,1,6], [1,5,2], [1,5,3]. Where the 7th sum is 9.

# Constraints:

* 1. m == mat.length
  2. n == mat.length[i]
  3. 1 <= m, n <= 40
  4. 1 <= mat[i][j] <= 5000
  5. 1 <= k <= min(200, nm) VI. mat[i] is a non-decreasing array.

# Code:

#include <iostream> #include <vector> #include <queue>

#include <set> #include <tuple> using namespace std;

int kthSmallestSum(vector<vector<int>>& mat, int k) { int m = mat.size(), n = mat[0].size();

priority\_queue<pair<int, vector<int>>, vector<pair<int, vector<int>>>, greater<>> minHeap; set<vector<int>> visited;

vector<int> initial(m, 0); int initialSum = 0; for (int i = 0; i < m; ++i) {

initialSum += mat[i][0];

}

minHeap.push({initialSum, initial}); visited.insert(initial);

while (!minHeap.empty()) {

auto [currentSum, indices] = minHeap.top(); minHeap.pop();

--k;

if (k == 0) return currentSum; for (int i = 0; i < m; ++i) {

if (indices[i] + 1 < n) {

vector<int> newIndices = indices; newIndices[i]++;

if (visited.find(newIndices) == visited.end()) {

int newSum = currentSum - mat[i][indices[i]] + mat[i][newIndices[i]]; minHeap.push({newSum, newIndices});

visited.insert(newIndices);

}

}

}

}

return -1; // This should never be reached

}

int main() {

vector<vector<int>> mat1 = {{1, 3, 11}, {2, 4, 6}}; int k1 = 5;

cout << kthSmallestSum(mat1, k1) << endl; // Output: 7

vector<vector<int>> mat2 = {{1, 3, 11}, {2, 4, 6}}; int k2 = 9;

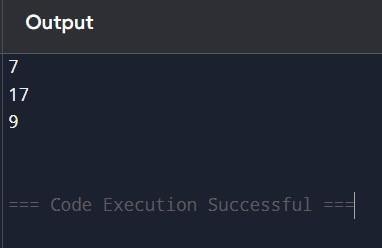
cout << kthSmallestSum(mat2, k2) << endl; // Output: 17

vector<vector<int>> mat3 = {{1, 10, 10}, {1, 4, 5}, {2, 3, 6}}; int k3 = 7;

cout << kthSmallestSum(mat3, k3) << endl; // Output:

9 return 0; }

# Output:

****

1. **Find Minimum in Rotated Sorted Array II.**

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.

# Example 1:

**Input:** nums = [1,3,5]

# Output: 1

**Example 2:**

**Input:** nums = [2,2,2,0,1]

# Output: 0

**Constraints:**

1. n == nums.length
2. 1 <= n <= 5000
3. -5000 <= nums[i] <= 5000
4. nums is sorted and rotated between 1 and n times.

# 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 if (nums[mid] < nums[right])

{ right = mid; } else {

right--; // Handle duplicates

} } return nums[left]; } int main() { vector<int> nums1 =

{1, 3, 5};

cout << findMin(nums1) << endl; // Output: 1

vector<int> nums2 = {2, 2, 2, 0, 1};

cout << findMin(nums2) << endl; // Output: 0

vector<int> nums3 = {10, 1, 10, 10, 10};

cout << findMin(nums3) << endl; // Output: 1

return 0;

}



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

* **Output:**

****