

Day-5

Name – Ishita

Faculty Name- Er. Rajni Devi

UID- 22BCS15353

Date- 26 Dec ,2024

Section – 620-B

Very Easy

Ques 1:

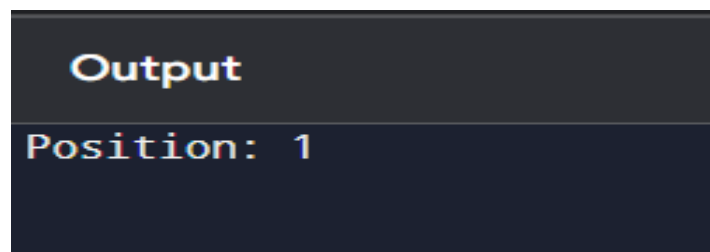
Searching a Number

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.

Code:

```
#include <iostream>
using namespace std;
int findFirstOccurrence(int arr[], int n, int k) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == k) {
            return i; }
    }
    return -1; }
int main() {
    int arr[] = {2, 4, 6, 8, 4};
    int n = sizeof(arr) / sizeof(arr[0]); // Calculate the size of the array
    int k = 4;
    int position = findFirstOccurrence(arr, n, k);
    cout << "Position: " << position << endl;
    return 0;
}
```

Output:

A screenshot of a terminal window with a dark background. The word "Output" is displayed in a light blue, monospace-style font at the top. Below it, the text "Position: 1" is displayed in a light green, monospace-style font.

Ques 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.

Code:

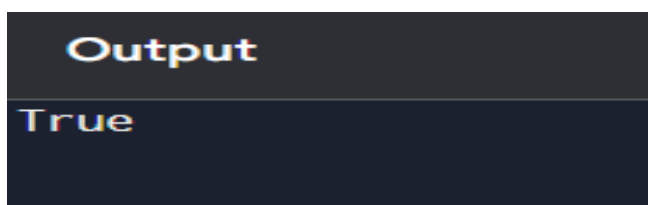
```
#include <iostream>

using namespace std;

bool isPresent(int arr[], int n, int k) {
    int left = 0, right = n - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2; // To avoid overflow
        if (arr[mid] == k) {
            return true; // k is found
        } else if (arr[mid] < k) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return false;
}

int main() {
    int arr[] = {1, 3, 5, 7, 9};    int k = 5;
    int n = sizeof(arr) / sizeof(arr[0]);
    if (isPresent(arr, n, k)) {
        cout << "True" << endl;
    } else {
        cout << "False" << endl;
    }
    return 0;
}
```

Output:

A screenshot of a terminal window with a dark background. The word "Output" is displayed in a light blue, monospace-style font at the top. Below it, the word "True" is displayed in the same font and color, indicating the result of the search operation.

Easy

Ques 3:

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.

Code:

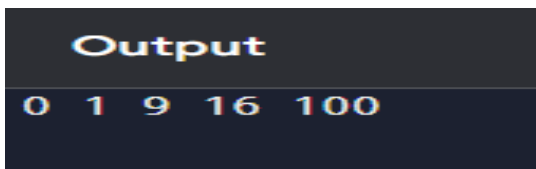
```
#include <iostream>

using namespace std;

void sortedSquares(int nums[], int n, int result[]) {
    int left = 0, right = n - 1;
    int index = n - 1; // Start filling from the end of the result array
    while (left <= right) {
        int leftSquare = nums[left] * nums[left];
        int rightSquare = nums[right] * nums[right];
        if (leftSquare > rightSquare) {
            result[index--] = leftSquare;
            left++;
        } else {
            result[index--] = rightSquare;
            right--;
        }
    }
}

int main() {
    int nums[] = {-4, -1, 0, 3, 10};
    int n = sizeof(nums) / sizeof(nums[0]);
    int result[n];
    sortedSquares(nums, n, result);
    for (int i = 0; i < n; i++) {
        cout << result[i] << " ";
    }
    cout << endl;
    return 0;
}
```

Output:



```
Output
0 1 9 16 100
```

Ques 4:

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.

Code:

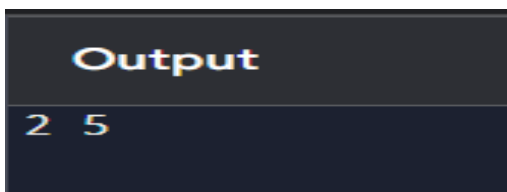
```
#include <iostream>

using namespace std;

pair<int, int> findFirstAndLast(int arr[], int n, int x) {
    int first = -1, last = -1;
    for (int i = 0; i < n; i++) {
        if (arr[i] == x) {
            first = i;
            break;}}
    for (int i = n - 1; i >= 0; i--) {
        if (arr[i] == x) {
            last = i;
            break;}}
    return {first, last};
}

int main() {
    int arr[] = {1, 3, 5, 5, 5, 5, 67, 123, 125};
    int n = sizeof(arr) / sizeof(arr[0]);
    int x = 5;
    pair<int, int> result = findFirstAndLast(arr, n, x);
    cout << result.first << " " << result.second << endl;
    return 0;
}
```

Output:



Medium

Ques 5:

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.

Code:

```
#include <iostream>

using namespace std;

int findFirst(int arr[], int n, int target) {
    int left = 0, right = n - 1, first = -1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target) {
            first = mid;
            right = mid - 1; // Search in the left half
        } else if (arr[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return first;
}

int findLast(int arr[], int n, int target) {
    int left = 0, right = n - 1, last = -1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target) {
            last = mid;
```

```

        left = mid + 1; // Search in the right half
    } else if (arr[mid] < target) {
        left = mid + 1;
    } else {
        right = mid - 1;
    }
}
return last;
}

int main() {
    int n, target;
    cout << "Enter the size of the array: ";
    cin >> n;
    int arr[n];
    cout << "Enter " << n << " elements in sorted order: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    cout << "Enter the target value: ";
    cin >> target;
    int first = findFirst(arr, n, target);
    int last = findLast(arr, n, target);
    cout << "[" << first << ", " << last << "]" << endl;
    return 0;
}

```

Output:

Output

```

Enter the size of the array: 6
Enter 6 elements in sorted order: 5 7 7 8 8 10
Enter the target value: 8
[3, 4]

```

Ques 6:

Smallest Positive Missing Number.

You are given an integer array arr[]. Your task is to find the smallest positive number missing from the array.

Code:

```
#include <iostream>

using namespace std;

int findSmallestMissingPositive(int arr[], int n) {
    // Mark non-positive numbers and numbers greater than n as invalid
    for (int i = 0; i < n; i++) {
        if (arr[i] <= 0 || arr[i] > n) {
            arr[i] = n + 1;
        }
        for (int j = 0; j < n; j++) {
            int num = abs(arr[j]);
            if (num <= n) {
                arr[num - 1] = -abs(arr[num - 1]);
            }
        }
        for (int i = 0; i < n; i++) {
            if (arr[i] > 0) {
                return i + 1;
            }
        }
        return n + 1;
    }
}

int main() {
    int n;
    cout << "Enter the size of the array: ";
    cin >> n;
    int arr[n];
    cout << "Enter " << n << " elements: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    int result = findSmallestMissingPositive(arr, n);
    cout << "Smallest Positive Missing Number: " << result << endl;
    return 0;
}
```

Output:

Output

```
Enter the size of the array: 6
Enter 6 elements: 2 -3 4 1 1 7
Smallest Positive Missing Number: 3
```

Hard

Ques 7:

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.

Code:

```
#include <iostream>

#include <queue>

using namespace std;

linked list.

struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) : val(x), next(nullptr) {}
};

for priority queue

struct compare {
    bool operator()(ListNode* a, ListNode* b) {
        return a->val > b->val;
    }
};
```



```

ListNode* mergeKLists(ListNode* lists[], int k) {
    priority_queue<ListNode*, vector<ListNode*>, compare> minHeap;

    for (int i = 0; i < k; i++) {
        if (lists[i]) minHeap.push(lists[i]);
    }

    ListNode* dummy = new ListNode(0);
    ListNode* current = dummy;
    while (!minHeap.empty()) {
        current->next = minHeap.top();
        minHeap.pop();
        current = current->next;
        if (current->next) minHeap.push(current->next);
    }
    return dummy->next;
}

ListNode* createList(int arr[], int n) {
    ListNode* head = new ListNode(0), *temp = head;
    for (int i = 0; i < n; i++) {
        temp->next = new ListNode(arr[i]);
        temp = temp->next;
    }
    return head->next;
}

// Print the linked list

```

```

void printList(ListNode* head) {
    while (head) {
        cout << head->val << " ";
        head = head->next;
    }
    cout << endl;
}

int main() {
    int k;

    cout << "Enter number of lists: ";
    cin >> k;

    ListNode* lists[k];

    for (int i = 0; i < k; i++) {
        int n;

        cout << "Enter number of elements for list " << i + 1 << ": ";
        cin >> n;

        int arr[n];

        cout << "Enter elements: ";
        for (int j = 0; j < n; j++) cin >> arr[j];

        lists[i] = createList(arr, n);
    }

    ListNode* mergedList = mergeKLists(lists, k);

    cout << "Merged List: ";
    printList(mergedList);

    return 0;
}

```

Output:

Output

```
Enter number of lists: 3
Enter number of elements for list 1: 3
Enter elements: 1 4 5
Enter number of elements for list 2: 5
Enter elements: 2 6 7 1 2
Enter number of elements for list 3: 2
Enter elements: 2 4
Merged List: 1 2 2 4 4 5 6 7 1 2
```

Ques 8:

Max Chunks To Make Sorted II

You are given an integer array arr.

We split arr into some number of chunks (i.e., partitions), and individually sort each chunk. After concatenating them, the result should equal the sorted array.

Return the largest number of chunks we can make to sort the array.

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int maxChunksToSorted(vector<int>& arr) {
    vector<int> sortedArr = arr;
    sort(sortedArr.begin(), sortedArr.end());
    int chunkCount = 0;
    int maxLeft = 0;
    for (int i = 0; i < arr.size(); i++) {
        maxLeft = max(maxLeft, arr[i]);
        if (maxLeft == sortedArr[i]) {
```

```

        chunkCount++;
    }
}

return chunkCount;
}

int main() {
    int n;
    cout << "Enter the size of the array: ";
    cin >> n;

    vector<int> arr(n);
    cout << "Enter the elements of the array: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    cout << "Max chunks to sort: " << maxChunksToSorted(arr) << endl;

    return 0;
}

```

Output:

Output

```

Enter the size of the array: 5
Enter the elements of the array: 5 4 3 2 1
Max chunks to sort: 1

```

Very Hard

Ques 9:

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.

The overall run time complexity should be $O(\log(m+n))$.

Code:

```
#include <iostream>

#include <algorithm>

using namespace std;

double findMedianSortedArrays(int nums1[], int m, int nums2[], int n) {

    if (m > n) {

        // Swap arrays to ensure nums1 is the smaller array

        return findMedianSortedArrays(nums2, n, nums1, m);}

    int left = 0, right = m, halfLen = (m + n + 1) / 2;

    while (left <= right) {

        int i = (left + right) / 2; // Partition for nums1

        int j = halfLen - i; // Partition for nums2

        if (i < m && nums1[i] < nums2[j - 1]) {

            left = i + 1;

        } else if (i > 0 && nums1[i - 1] > nums2[j]) {

            right = i - 1;

        } else {

            int maxLeft = 0;

            if (i == 0) maxLeft = nums2[j - 1];

            else if (j == 0) maxLeft = nums1[i - 1];

            else maxLeft = max(nums1[i - 1], nums2[j - 1]);
```

```

        if ((m + n) % 2 == 1) return maxLeft; // If odd length, return the max of left partition

        int minRight = 0;

        if (i == m) minRight = nums2[j];

        else if (j == n) minRight = nums1[i];

        else minRight = min(nums1[i], nums2[j]);

        return (maxLeft + minRight) / 2.0; }}

return 0.0; }

int main() {

    int m, n;

    cout << "Enter size of first array: ";

    cin >> m;

    int nums1[m];

    cout << "Enter elements of the first array: ";

    for (int i = 0; i < m; i++) {

        cin >> nums1[i];}

    cout << "Enter size of second array: ";

    cin >> n;

    int nums2[n];

    cout << "Enter elements of the second array: ";

    for (int i = 0; i < n; i++) {

        cin >> nums2[i];

    }

    double median = findMedianSortedArrays(nums1, m, nums2, n);

    cout << "Median: " << median << endl;

    return 0;

}

```

Output:

```
Output
Enter size of first array: 2
Enter elements of the first array: 1 2
Enter size of second array: 1
Enter elements of the second array: 3
Median: 2
```

Ques 10:

Kth Smallest Product of Two Sorted Arrays.

Given two sorted 0-indexed integer arrays `nums1` and `nums2` as well as an integer `k`, return the `k`th (1-based) smallest product of `nums1[i] * nums2[j]` where $0 \leq i < \text{nums1.length}$ and $0 \leq j < \text{nums2.length}$.

Code:

```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;
struct Product {
    int value;
    int i, j;
    Product(int v, int x, int y) : value(v), i(x), j(y) {}
    bool operator>(const Product& p) const {
        return value > p.value; // Min-heap
    }
};
int kthSmallestProduct(vector<int>& nums1, vector<int>& nums2, int k) {
    int m = nums1.size(), n = nums2.size();
    priority_queue<Product, vector<Product>, greater<Product>> minHeap;
    for (int i = 0; i < m; i++) {
        minHeap.push(Product(nums1[i] * nums2[0], i, 0));
    }
}
```

```

for (int count = 0; count < k - 1; count++) {
    Product top = minHeap.top();
    minHeap.pop();
    if (top.j + 1 < n) {
        minHeap.push(Product(nums1[top.i] * nums2[top.j + 1], top.i, top.j + 1));}
    return minHeap.top().value;}

int main() {
    int m, n, k;
    cout << "Enter size of first array: ";
    cin >> m;
    vector<int> nums1(m);
    cout << "Enter elements of first array: ";
    for (int i = 0; i < m; i++) {
        cin >> nums1[i];}
    cout << "Enter size of second array: ";
    cin >> n;
    vector<int> nums2(n);
    cout << "Enter elements of second array: ";
    for (int i = 0; i < n; i++) {
        cin >> nums2[i];}
    cout << "Enter value of k: ";
    cin >> k;
    int result = kthSmallestProduct(nums1, nums2, k);
    cout << "The " << k << "-th smallest product is: " << result << endl;
    return 0;}

```

Output:

Output
Enter size of first array: 2
Enter elements of first array: 2 5
Enter size of second array: 2
Enter elements of second array: 3 4
Enter value of k: 2
The 2-th smallest product is: 8