**1**

**#include <iostream>**

**#include <vector>**

**using namespace std;**

**pair<bool, int> linearSearch(vector<int>& arr, int key) {**

**int comparisons = 0;**

**for (int i = 0; i < arr.size(); ++i) {**

**comparisons++;**

**if (arr[i] == key) {**

**return make\_pair(true, comparisons);**

**}**

**}**

**return make\_pair(false, comparisons);**

**}**

**int main() {**

**int test\_cases;**

**cout << "Enter the number of test cases: ";**

**cin >> test\_cases;**

**for (int t = 0; t < test\_cases; ++t) {**

**int size;**

**cout << "Enter the size of the array: ";**

**cin >> size;**

**vector<int> arr(size);**

**cout << "Enter the elements of the array separated by space: ";**

**for (int i = 0; i < size; ++i) {**

**cin >> arr[i];**

**}**

**int key;**

**cout << "Enter the key element to search: ";**

**cin >> key;**

**pair<bool, int> result = linearSearch(arr, key);**

**if (result.first) {**

**cout << "The key element " << key << " is present in the array." << endl;**

**} else {**

**cout << "The key element " << key << " is not present in the array." << endl;**

**}**

**cout << "Total number of comparisons: " << result.second << endl << endl;**

**}**

**return 0;**

**}**

**2**

**#include <iostream>**

**#include <vector>**

**using namespace std;**

**// Function to perform binary search**

**int binarySearch(const vector<int>& arr, int key, int& comparisons) {**

**int left = 0;**

**int right = arr.size() - 1;**

**while (left <= right) {**

**int mid = left + (right - left) / 2;**

**comparisons++; // Increment comparison count**

**// Check if key is present at mid**

**if (arr[mid] == key)**

**return mid;**

**// If key is greater, ignore left half**

**else if (arr[mid] < key)**

**left = mid + 1;**

**// If key is smaller, ignore right half**

**else**

**right = mid - 1;**

**}**

**return -1; // Key not found**

**}**

**int main() {**

**int t; // Number of test cases**

**cin >> t;**

**while (t--) {**

**int n; // Size of array**

**cin >> n;**

**vector<int> arr(n); // Input array**

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

**cin >> arr[i];**

**int key; // Key element to search**

**cin >> key;**

**int comparisons = 0; // Initialize comparison count**

**int result = binarySearch(arr, key, comparisons);**

**if (result != -1)**

**cout << "Present " << comparisons << endl;**

**else**

**cout << "Not Present " << comparisons << endl;**

**}**

**return 0;}**

**3**

**#include <iostream>**

**#include <vector>**

**#include <cmath>**

**using namespace std;**

**int jumpSearch(const std::vector<int>& arr, int key, int& comparisons) {**

**int n = arr.size();**

**int step = sqrt(n); // Determine the step size by taking the square root of the array size.**

**int prev = 0; // Initialize the previous index.**

**comparisons = 0; // Initialize the comparisons counter.**

**// Perform the jump until the value at the current step is less than the key.**

**while (arr[std::min(step, n) - 1] < key) {**

**prev = step; // Update the previous index.**

**step += sqrt(n); // Move the step forward.**

**if (prev >= n) {**

**return -1; // If the previous index exceeds or equals the array size, the key is not present.**

**}**

**comparisons++; // Increment comparisons counter.**

**}**

**// Perform linear search in the current block for the key.**

**while (arr[prev] < key) {**

**prev++; // Move to the next element in the current block.**

**if (prev == std::min(step, n)) {**

**return -1; // If the previous index reaches the end of the current block, the key is not present.**

**}**

**comparisons++; // Increment comparisons counter.**

**}**

**// If the key is found, return its index.**

**if (arr[prev] == key) {**

**return prev;**

**}**

**// If the key is not found, return -1.**

**return -1;**

**}**

**int main() {**

**int test\_cases;**

**cout << "Enter the number of test cases: ";**

**cin >> test\_cases;**

**for (int t = 0; t < test\_cases; ++t) {**

**int n;**

**std::cout << "Enter the size of the array: ";**

**std::cin >> n;**

**std::vector<int> arr(n);**

**std::cout << "Enter the sorted array elements: ";**

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

**std::cin >> arr[i];**

**}**

**int key;**

**std::cout << "Enter the key to search: ";**

**std::cin >> key;**

**int comparisons;**

**// Call the jumpSearch function to find the key in the array.**

**int result = jumpSearch(arr, key, comparisons);**

**// Print the result based on whether the key is found or not.**

**if (result != -1) {**

**std::cout << "Element " << key << " found at index " << result << ". ";**

**} else {**

**std::cout << "Element " << key << " not found. ";**

**}**

**std::cout << "Comparisons required: " << comparisons << std::endl;**

**}**

**return 0;**

**}**

**4**

**#include <iostream>**

**#include <vector>**

**using namespace std;**

**pair<bool, int> binarySearch(const vector<int>& arr, int key) {**

**int n = arr.size();**

**int low = 0, high = n - 1;**

**int first\_occurrence = -1;**

**// Binary search to find the first occurrence of the key**

**while (low <= high) {**

**int mid = low + (high - low) / 2;**

**if (arr[mid] == key) {**

**first\_occurrence = mid;**

**high = mid - 1;**

**} else if (arr[mid] < key) {**

**low = mid + 1;**

**} else {**

**high = mid - 1;**

**}**

**}**

**// If key is not found, return false**

**if (first\_occurrence == -1) {**

**return {false, 0};**

**}**

**// Find the last occurrence of the key**

**int last\_occurrence = first\_occurrence;**

**while (last\_occurrence + 1 < n && arr[last\_occurrence + 1] == key) {**

**last\_occurrence++;**

**}**

**// Calculate the number of copies of the key**

**int num\_copies = last\_occurrence - first\_occurrence + 1;**

**return {true, num\_copies};**

**}**

**int main() {**

**int T;**

**cout << "Enter the number of test cases: ";**

**cin >> T;**

**while (T--) {**

**int n;**

**cout << "Enter the size of the array: ";**

**cin >> n;**

**vector<int> arr(n);**

**cout << "Enter the sorted array elements: ";**

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

**cin >> arr[i];**

**}**

**int key;**

**cout << "Enter the key element to search: ";**

**cin >> key;**

**pair<bool, int> result = binarySearch(arr, key);**

**if (result.first) {**

**cout << key << " - " << result.second << endl;**

**} else {**

**cout << "Key not present" << endl;**

**}**

**}**

**return 0;**

**}**

**5**

**#include <iostream>**

**#include <vector>**

**#include <algorithm>**

**using namespace std;**

**int binary\_search(vector<int>& arr, int left, int right, int x) {**

**while (left <= right) {**

**int mid = left + (right - left) / 2;**

**if (arr[mid] == x) {**

**return mid;**

**} else if (arr[mid] < x) {**

**left = mid + 1;**

**} else {**

**right = mid - 1;**

**}**

**}**

**return -1; // element not found**

**}**

**bool find\_triplet\_indices(vector<int>& arr) {**

**int n = arr.size();**

**for (int i = 0; i < n - 1; ++i) {**

**for (int j = i + 1; j < n; ++j) {**

**int sum = arr[i] + arr[j];**

**int index = binary\_search(arr, j + 1, n - 1, sum);**

**if (index != -1) {**

**cout << i + 1 << ", " << j + 1 << ", " << index + 1 << endl;**

**return true;**

**}**

**}**

**}**

**cout << "No sequence found." << endl;**

**return false;**

**}**

**int main() {**

**int T;**

**cout<<"enter the number of test case:";**

**cin >> T;**

**while (T--) {**

**int n;**

**cout<<"enter the size of arr:";**

**cin >> n;**

**vector<int> arr(n);**

**cout<<"enter the sorted array:";**

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

**cin >> arr[i];**

**}**

**// Find triplet indices**

**find\_triplet\_indices(arr);**

**}**

**return 0;**

**}**

**6**

#include <bits/stdc++.h>

#include <vector>

#include <unordered\_set>

using namespace std;

int count\_pairs\_with\_difference(vector<int>& arr, int key) {

    unordered\_set<int> num\_set;

    int count = 0;

    for (int num : arr) {

        if (num\_set.find(num - key) != num\_set.end() || num\_set.find(num + key) != num\_set.end()) {

            count++;

        }

        num\_set.insert(num);

    }

    return count;

}

int main() {

    int T;

    cout<<"enter the number of test cases:";

    cin >> T;

    while (T--) {

        int n;

        cout<<"enter the size of array:";

        cin >> n;

        vector<int> arr(n);

        cout<<"enter the array:";

        for (int i = 0; i < n; ++i) {

            cin >> arr[i];

        }

        int key;

        cout

        <<"enter the value of key:";

        cin >> key;

*// Count pairs with difference equal to key*

        int result = count\_pairs\_with\_difference(arr, key);

*// Print output*

        cout << result << endl;

    }

    return 0;

}

**7**

#include <iostream>

#include <vector>

using namespace std;

*// Function to perform insertion sort and count comparisons and shifts*

pair<int, int> insertionSort(vector<int>& arr) {

    int comparisons = 0;

    int shifts = 0;

    for (int i = 1; i < arr.size(); ++i) {

        int key = arr[i];

        int j = i - 1;

        comparisons++;

        while (j >= 0 && key < arr[j]) {

            arr[j + 1] = arr[j];

            j--;

            shifts++;

            comparisons++;

        }

        arr[j + 1] = key;

    }

    return make\_pair(comparisons, shifts);

}

int main() {

    int T;

    cout<<"enter the test case :";

    cin >> T;

    for (int t = 0; t < T; ++t) {

        int n;

        cout<<"enter the size of array:";

        cin >> n;

        vector<int> arr(n);

        cout<<"enter the array:";

        for (int i = 0; i < n; ++i) {

            cin >> arr[i];

        }

*// Sorting the array using insertion sort and getting comparisons and shifts*

        pair<int, int> counts = insertionSort(arr);

*// Output*

        cout << "Sorted Array:" << endl;

        for (int i = 0; i < n; ++i) {

            cout << arr[i] << " ";

        }

        cout << endl;

        cout << "comparisons = " << counts.first << endl;

        cout << "shifts = " << counts.second << endl;

    }

    return 0;

}

**8**

#include <iostream>

#include <vector>

using namespace std;

*// Function to perform selection sort and count comparisons and swaps*

pair<int, int> selectionSort(vector<int>& arr) {

    int comparisons = 0;

    int swaps = 0;

    for (int i = 0; i < arr.size() - 1; ++i) {

        int minIndex = i;

        for (int j = i + 1; j < arr.size(); ++j) {

            comparisons++;

            if (arr[j] < arr[minIndex]) {

                minIndex = j;

            }

        }

        if (minIndex != i) {

            swap(arr[i], arr[minIndex]);

            swaps++;

        }

    }

    return make\_pair(comparisons, swaps);

}

int main() {

    int T;

    cout<<"enter the cases :";

    cin >> T;

    for (int t = 0; t < T; ++t) {

        int n;

        cout<<"enterthe size of array:";

        cin >> n;

        vector<int> arr(n);

        cout<<"enter the element of array:";3

        for (int i = 0; i < n; ++i) {

            cin >> arr[i];

        }

*// Sorting the array using selection sort and getting comparisons and swaps*

        pair<int, int> counts = selectionSort(arr);

*// Output*

        cout << "Sorted Array:" << endl;

        for (int i = 0; i < n; ++i) {

            cout << arr[i] << " ";

        }

        cout << endl;

        cout << "comparisons = " << counts.first << endl;

        cout << "swaps = " << counts.second << endl;

    }

    return 0;

}

**9**

#include <iostream>

#include <vector>

using namespace std;

*// Merge function for mergesort*

void merge(vector<int>& arr, int left, int mid, int right) {

    int n1 = mid - left + 1;

    int n2 = right - mid;

    vector<int> L(n1);

    vector<int> R(n2);

    for (int i = 0; i < n1; ++i) {

        L[i] = arr[left + i];

    }

    for (int j = 0; j < n2; ++j) {

        R[j] = arr[mid + 1 + j];

    }

    int i = 0, j = 0, k = left;

    while (i < n1 && j < n2) {

        if (L[i] <= R[j]) {

            arr[k++] = L[i++];

        } else {

            arr[k++] = R[j++];

        }

    }

    while (i < n1) {

        arr[k++] = L[i++];

    }

    while (j < n2) {

        arr[k++] = R[j++];

    }

}

*// Mergesort function*

void mergeSort(vector<int>& arr, int left, int right) {

    if (left < right) {

        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);

        mergeSort(arr, mid + 1, right);

        merge(arr, left, mid, right);

    }

}

*// Function to check for duplicates in a sorted array*

string checkDuplicates(vector<int>& arr) {

    mergeSort(arr, 0, arr.size() - 1); *// Sort the array using mergesort*

    for (int i = 0; i < arr.size() - 1; ++i) {

        if (arr[i] == arr[i + 1]) {

            return "YES"; *// Duplicates found*

        }

    }

    return "NO"; *// No duplicates found*

}

int main() {

    int T;

    cout<<"enter the test caes:";

    cin >> T;

    for (int t = 0; t < T; ++t) {

        int n;

        cout<<"enter the size of array:";

        cin >> n;

        vector<int> arr(n);

        cout<<"enter the element of array:";

        for (int i = 0; i < n; ++i) {

            cin >> arr[i];

        }

*// Check for duplicates in the array*

        string result = checkDuplicates(arr);

*// Output*

        cout << result << endl;

    }

    return 0;

}

**10**

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

long long merge(vector<int>& arr, int low, int mid, int high) {

    vector<int> temp(high - low + 1);

    int i = low, j = mid + 1, k = 0;

    long long inversions = 0;

    while (i <= mid && j <= high) {

        if (arr[i] <= arr[j]) {

            temp[k++] = arr[i++];

        } else {

*// Inversion found*

            inversions += mid - i + 1;

            temp[k++] = arr[j++];

        }

    }

    while (i <= mid) {

        temp[k++] = arr[i++];

    }

    while (j <= high) {

        temp[k++] = arr[j++];

    }

    for (int i = low, k = 0; i <= high; i++, k++) {

        arr[i] = temp[k];

    }

    return inversions;

}

long long mergeSort(vector<int>& arr, int low, int high) {

    long long inversions = 0;

    if (low < high) {

        int mid = low + (high - low) / 2;

        inversions += mergeSort(arr, low, mid);

        inversions += mergeSort(arr, mid + 1, high);

        inversions += merge(arr, low, mid, high);

    }

    return inversions;

}

int main() {

    int t;

    cout << "Enter the number of test cases: ";

    cin >> t;

    while (t--) {

        int n;

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

        cin >> n;

        vector<int> arr(n);

        cout << "Enter the array elements: ";

        for (int i = 0; i < n; i++)

            cin >> arr[i];

        long long inversions = mergeSort(arr, 0, n - 1);

        cout << "Sorted array: ";

        for (int i = 0; i < n; i++)

            cout << arr[i] << " ";

        cout << endl;

        cout << "Comparisons = " << n \* log2(n) << endl; *// Number of comparisons in merge sort*

        cout << "Inversions = " << inversions << endl;

    }

    return 0;

}

**11**

#include <iostream>

#include <vector>

#include <cstdlib> // for rand() and srand()

#include <ctime>   // for time()

using namespace std;

// Function to partition the array and return the index of the pivot element

int partition(vector<int> &arr, int low, int high, int &comparisons, int &swaps)

{

    int pivot = arr[low + rand() % (high - low + 1)]; // Selecting random pivot

    int i = low - 1;

    for (int j = low; j <= high; j++)

    {

        comparisons++;

        if (arr[j] < pivot)

        {

            i++;

            swap(arr[i], arr[j]);

            swaps++;

        }

    }

    swap(arr[i + 1], arr[high]);

    swaps++;

    return i + 1;

}

// Function to perform QuickSort

void quickSort(vector<int> &arr, int low, int high, int &comparisons, int &swaps)

{

    if (low < high)

    {

        int pivotIndex = partition(arr, low, high, comparisons, swaps);

        quickSort(arr, low, pivotIndex - 1, comparisons, swaps);

        quickSort(arr, pivotIndex + 1, high, comparisons, swaps);

    }

}

int main()

{

    int T;

    cin >> T;

    while (T--)

    {

        int n;

        cin >> n;

        vector<int> arr(n);

        for (int i = 0; i < n; ++i)

        {

            cin >> arr[i];

        }

        int comparisons = 0, swaps = 0;

        quickSort(arr, 0, n - 1, comparisons, swaps);

        cout << "Sorted array: ";

        for (int i = 0; i < n; ++i)

        {

            cout << arr[i] << " ";

        }

        cout << endl;

        cout << "comparisons = " << comparisons << endl;

        cout << "swaps = " << swaps << endl;

    }

    return 0;

}

**12**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int kthSmallestUsingCountSort(vector<int>& arr, int k) {

*// Find the maximum element in the array*

    int maxElement = \*max\_element(arr.begin(), arr.end());

*// Initialize a frequency array to count occurrences of each element*

    vector<int> freq(maxElement + 1, 0);

*// Count the occurrences of each element in the array*

    for (int num : arr) {

        freq[num]++;

    }

*// Iterate through the frequency array to find the Kth element*

    int count = 0;

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

        count += freq[i];

        if (count >= k) {

            return i;

        }

    }

    return -1; *// Kth element not found*

}

int main() {

    int t;

    cin >> t;

    while (t--) {

        int n, k;

        cin >> n;

        vector<int> arr(n);

        cout << "Enter the array elements: ";

        for (int i = 0; i < n; i++)

            cin >> arr[i];

        cout << "Enter the value of K: ";

        cin >> k;

        int kthSmallest = kthSmallestUsingCountSort(arr, k);

        if (kthSmallest != -1)

            cout << "Kth smallest element is: " << kthSmallest << endl;

        else

            cout << "not present" << endl;

    }

    return 0;

}

**13**

#include <iostream>

#include <unordered\_map>

#include <vector>

#include <string>

using namespace std;

int main() {

    int T;

    cin >> T;

    while (T--) {

        int n;

        cin >> n;

*// Map to store alphabet occurrences*

        unordered\_map<char, int> freqMap;

*// Read array elements and count occurrences*

        string arr;

        cin.ignore(); *// Ignore newline character in input buffer*

        getline(cin, arr);

        for (char c : arr) {

            if (c != ' ') *// Ignore spaces*

                freqMap[c]++;

        }

        int maxCount = 0;

        char maxChar = '\0'; *// Initialize with null character*

        bool noDuplicates = true;

*// Find alphabet with maximum occurrences*

        for (auto it : freqMap) {

            if (it.second > maxCount) {

                maxCount = it.second;

                maxChar = it.first;

                noDuplicates = false;

            }

        }

*// Print the result*

        if (noDuplicates) {

            cout << "No Duplicates Present" << endl;

        } else {

            cout << maxChar << " - " << maxCount << endl;

        }

    }

    return 0;

}

**14**

#include <iostream>

#include <vector>

using namespace std;

void merge(vector<int>& arr, int low, int mid, int high) {

    int n1 = mid - low + 1;

    int n2 = high - mid;

    vector<int> left(n1), right(n2);

    for (int i = 0; i < n1; i++)

        left[i] = arr[low + i];

    for (int j = 0; j < n2; j++)

        right[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = low;

    while (i < n1 && j < n2) {

        if (left[i] <= right[j])

            arr[k++] = left[i++];

        else

            arr[k++] = right[j++];

    }

    while (i < n1)

        arr[k++] = left[i++];

    while (j < n2)

        arr[k++] = right[j++];

}

void mergeSort(vector<int>& arr, int low, int high) {

    if (low < high) {

        int mid = low + (high - low) / 2;

        mergeSort(arr, low, mid);

        mergeSort(arr, mid + 1, high);

        merge(arr, low, mid, high);

    }

}

void findElementsWithSum(vector<int>& arr, int key) {

    int n = arr.size();

    mergeSort(arr, 0, n - 1);

    int left = 0;

    int right = n - 1;

    while (left < right) {

        int sum = arr[left] + arr[right];

        if (sum == key) {

            cout << arr[left] << " " << arr[right] << endl;

            return;

        } else if (sum < key) {

            left++;

        } else {

            right--;

        }

    }

    cout << "No Such Elements Exist" << endl;

}

int main() {

    int t;

    cout << "Enter the number of test cases: ";

    cin >> t;

    while (t--) {

        int n;

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

        cin >> n;

        vector<int> arr(n);

        cout << "Enter the array elements: ";

        for (int i = 0; i < n; i++)

            cin >> arr[i];

        int key;

        cout << "Enter the key: ";

        cin >> key;

        findElementsWithSum(arr, key);

    }

    return 0;

}

**15**

#include <iostream>

#include <unordered\_map>

#include <vector>

using namespace std;

// Function to find the intersection of two unsorted arrays

void findIntersection(const vector<int>& arr1, const vector<int>& arr2) {

    unordered\_map<int, int> hashTable;

    // Insert elements of the first array into the hash table

    for (int num : arr1) {

        hashTable[num]++;

    }

    // Traverse the second array and check for elements in the hash table

    cout << "Intersection of the two arrays: ";

    for (int num : arr2) {

        if (hashTable[num] > 0) {

            cout << num << " ";

            hashTable[num]--;

        }

    }

    cout << endl;

}

int main() {

    int size1, size2;

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

    cin >> size1;

    vector<int> arr1(size1);

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

    for (int i = 0; i < size1; i++) {

        cin >> arr1[i];

    }

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

    cin >> size2;

    vector<int> arr2(size2);

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

    for (int i = 0; i < size2; i++) {

        cin >> arr2[i];

    }

    // Find and print the intersection of the two arrays

    findIntersection(arr1, arr2);

    return 0;

}