**Practical 11**

**Problem Statement :**

Given an unsorted array of integers, design an algorithm and implement it using a program to sort an array of elements by partitioning the array into two subarrays based on a pivot element such that one of the sub array holds values smaller than the pivot element while another sub array holds values greater than the pivot element. Pivot element should be selected randomly from the array. Your program should also find number of comparisons and swaps required for sorting the array.  
  
**Algorithm:**

**Step 1.** Function QUICK\_SORT(arr, low, high)  
**Step 2.** If low < high then:  
**Step 3.** Call PARTITION(arr, low, high) to find the pivot index pi  
**Step 4.** Recursively call QUICK\_SORT(arr, low, pi - 1)  
**Step 5.** Recursively call QUICK\_SORT(arr, pi + 1, high)  
**Step 6.** End If  
**Step 7.** End Function

**Step 1.** Function PARTITION(arr, low, high)  
**Step 2.** Set pivot ← arr[high]  
**Step 3.** Initialize i ← low - 1  
**Step 4.** For j ← low to high - 1 do:  
**Step 5.** Increment comparisons by 1  
**Step 6.** If arr[j] < pivot then:  
**Step 7.** Increment i by 1  
**Step 8.** Swap arr[i] and arr[j]  
**Step 9.** Increment swaps by 1  
**Step 10.** End If  
**Step 11.** End For  
**Step 12.** Swap arr[i + 1] and arr[high]  
**Step 13.** Increment swaps by 1  
**Step 14.** Return i + 1 (pivot index)  
**Step 15.** End Function

**Code :**

#include <iostream>

#include <vector>

using namespace std;

int comparisons = 0;

int swaps = 0;

void swap(int &a, int &b) {

int temp = a;

a = b;

b = temp;

swaps++;

}

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

int pivot = arr[high];

int i = low - 1;

for (int j = low; j < high; j++) {

comparisons++;

if (arr[j] < pivot) {

i++;

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

}

}

swap(arr[i + 1], arr[high]);

return (i + 1);

}

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

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

void printArray(const vector<int> &arr) {

for (int num : arr) {

cout << num << " ";

}

cout << endl;

}

int main() {

int choice;

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

cin >> choice;

while (choice--) {

int n;

cout << "\nEnter the size of the array: ";

cin >> n;

vector<int> arr(n);

cout << "Enter the elements of the array: \n";

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

cin >> arr[i];

}

comparisons = 0;

swaps = 0;

quickSort(arr, 0, n - 1);

cout << "\nSorted array: \n";

printArray(arr);

cout << "Number of comparisons: " << comparisons << endl;

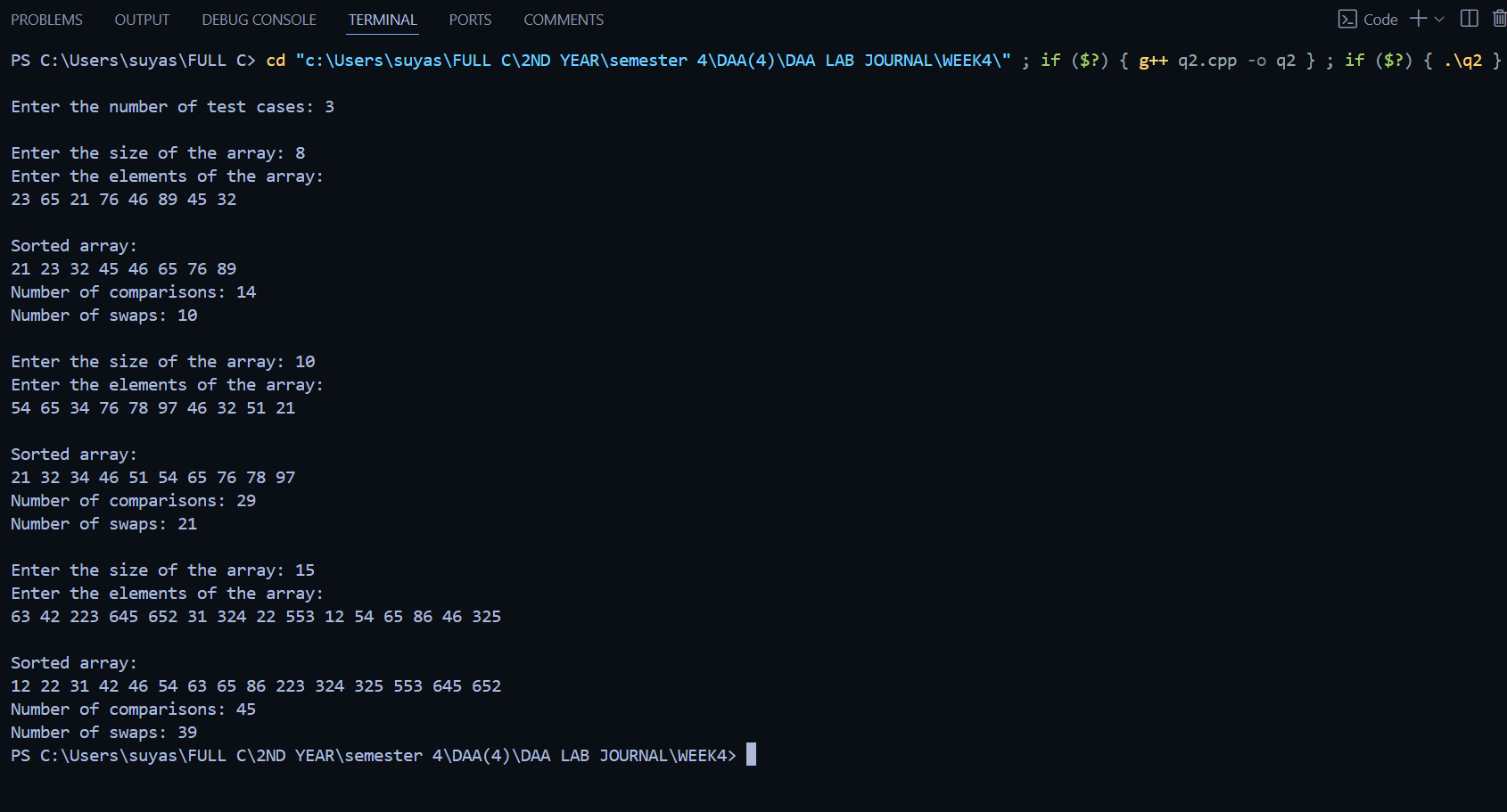
cout << "Number of swaps: " << swaps << endl;

}

return 0;

}

**OUTPUT**

**Practical 12**

**Problem Statement:**

Given an unsorted array of integers, design an algorithm and implement it using a program to find Kth smallest or largest element in the array. (Worst case Time Complexity = O(n))

**Algorithm:**

**Step 1**. Function PROCESS\_ARRAY(arr, n, k)  
**Step 2**. Initialize:  
**Step 3**. maxe ← arr[0]  
**Step 4**. mine ← arr[0]  
**Step 5**. For i ← 1 to n - 1 do:  
**Step 6**. If arr[i] > maxe then:  
**Step 7**. maxe ← arr[i]  
**Step 8.** If arr[i] < mine then:  
**Step 9.** mine ← arr[i]  
**Step 10.** End For  
**Step 11**. Compute range ← maxe - mine + 1  
**Step 12.** Initialize freq[range] ← {0}  
**Step 13**. For i ← 0 to n - 1 do:  
**Step 14.** index ← arr[i] - mine  
**Step 15**. freq[index]++  
**Step 16.** End For  
**Step 17**. For i ← 1 to range - 1 do:  
**Step 18**. freq[i] ← freq[i] + freq[i - 1]  
**Step 19.** End For  
**Step 20**. Initialize temp[n]  
**Step 21**. For i ← n - 1 to 0 do:  
**Step 22.** index ← arr[i] - mine  
**Step 23**. pos ← --freq[index]  
**Step 24.** temp[pos] ← arr[i]  
**Step 25.** End For  
**Step 26**. If k > 0 and k ≤ n then:  
**Step 27**. Print "The k-th smallest element is:" temp[k - 1]  
**Step 28.** Print "The k-th largest element is:" temp[n - k]  
**Step 29**. Else Print "Invalid value of k"  
**Step 30.** End Function

**Code:**

#include <iostream>

using namespace std;

int main()

{

int choice;

cout << "ENTER NUMBER OF TEST CASES : ";

cin >> choice;

while (choice--)

{

cout << "ENTER THE SIZE OF THE ARRAY : ";

int n;

cin >> n;

cout << "ENTER THE ARRAY ELEMENTS : ";

int arr[n];

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

{

cin >> arr[i];

}

int maxe = arr[0];

int mine = arr[0];

for (int i = 1; i < n; i++)

{

if (arr[i] > maxe)

maxe = arr[i];

if (arr[i] < mine)

mine = arr[i];

}

int range = maxe - mine + 1;

int freq[range] = {0};

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

{

int index = arr[i] - mine;

freq[index]++;

}

for (int i = 1; i < range; i++)

{

freq[i] = freq[i] + freq[i - 1];

}

int temp[n];

for (int i = n - 1; i >= 0; i--)

{

int index = arr[i] - mine;

int pos = --freq[index];

temp[pos] = arr[i];

}

int k;

cout << "ENTER THE VALUE OF k: ";

cin >> k;

if (k > 0 && k <= n)

{

cout << "The " << k << "-th smallest element is: " << temp[k - 1] << endl;

cout << "The " << k << "-th largest element is: " << temp[n - k] << endl;

}

else

{

cout << "Invalid value of k" << endl;

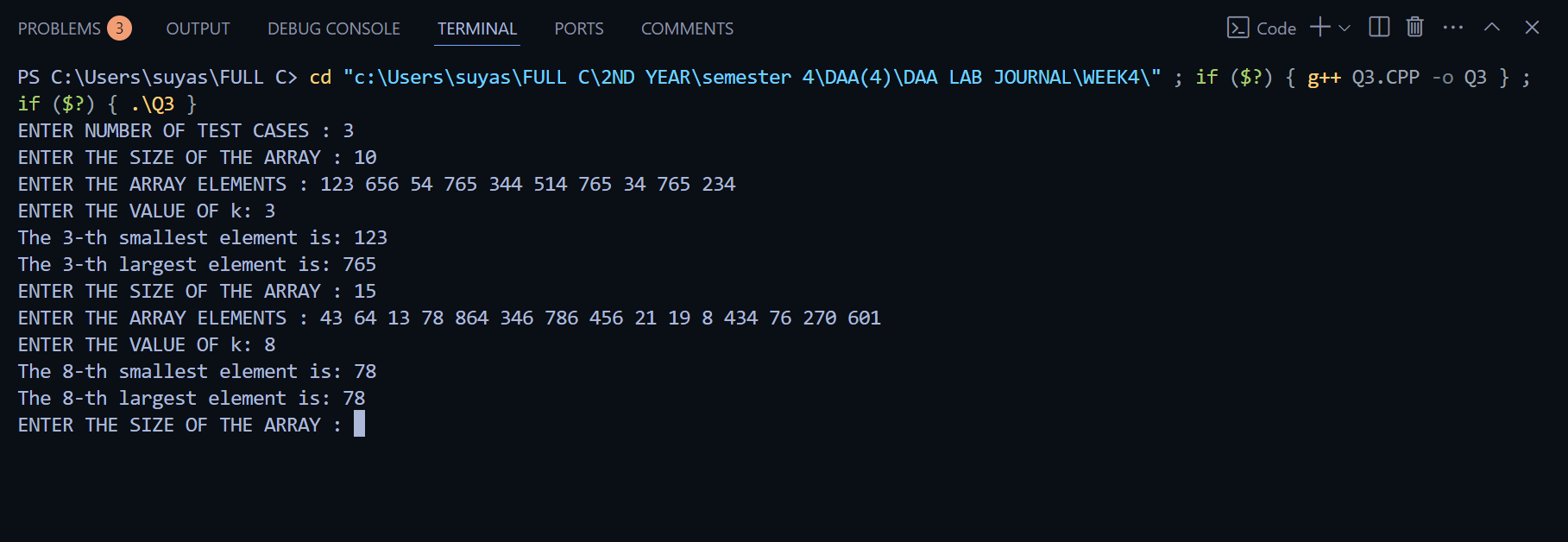
}

}

return 0;

}

**OUTPUT**

****

**Practical 13**

**Problem Statement :**

Given an unsorted array of alphabets containing duplicate elements. Design an algorithm and implement it using a program to find which alphabet has maximum number of occurrences and print it. (Time Complexity = O(n)) (Hint: Use counting sort)

**Algorithm:**

**Steps 1.** Start function findMaxOccurrenceAlphabet(n, arr).  
**Steps 2.** Initialize freq[26] to 0 for all indices.  
**Steps 3.** Loop from i = 0 to n - 1:  
**Steps 4.** Increment freq[arr[i] - 'a'].  
**Steps 5.** Initialize max = 0 and max\_char = 'a'.  
**Steps 6.** Loop from i = 0 to 25:

**Steps 7**. If freq[i] > max:  
**Steps 8.** Set max = freq[i].  
**Steps 9.** Set max\_char = 'a' + i.  
**Steps 10**. If max == 1:  
**Steps 11.** Print "NO DUPLICATES FOUND".  
**Steps 12.** Else:  
**Steps 13.** Print max\_char and max.  
**Steps 14.** End function.

**Code :**

#include <iostream>

using namespace std;

void findMaxOccurrenceAlphabet(int n, char arr[]) {

int freq[26] = {0};

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

freq[arr[i] - 'a']++;

}

int max = 0;

char max\_char = 'a';

for (int i = 0; i < 26; i++) {

if (freq[i] > max) {

max = freq[i];

max\_char = 'a' + i;

}

}

if (max == 1) {

cout << "NO DUPLICATES FOUND" << endl;

} else {

cout << "Alphabet with maximum occurrences: " << max\_char << " - " << max << endl;

}

}

int main() {

int choice;

cout << "ENTER THE NUMBER OF TEST CASES: ";

cin >> choice;

while (choice--) {

int n;

cout << "ENTER THE ARRAY SIZE: ";

cin >> n;

char arr[n];

cout << "ENTER THE ARRAY ELEMENTS: ";

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

cin >> arr[i];

}

findMaxOccurrenceAlphabet(n, arr);

}

return 0;

}

**OUTPUT**

****

**Practical 14**

**Problem Statement :**

Given an unsorted array of integers, design an algorithm and implement it using a program to find whether two elements exist such that their sum is equal to the given key element. (Time Complexity = O(n log n))

**Algorithm:**

**Steps 1**. Start function findPairWithSum(arr, n, key).  
**Steps 2**. Sort the array in ascending order (time complexity = O(n log n)).  
**Steps 3**. Initialize two pointers:  
**Steps 4**. left = 0   
**Steps 5.** right = n - 1  
**Steps 6**. While left < right:  
**Steps 7.** Compute sum = arr[left] + arr[right].  
**Steps 8.** If sum == key:  
**Steps 9**. Print the pair arr[left] and arr[right].  
**Steps 10.** Return (pair found).  
**Steps 11.** Else if sum < key:  
**Steps 12** Increment left by 1.  
**Steps 13**. Else:  
**Steps 14**. Decrement right by 1.  
**Steps 15** If no pair is found, print "No pair found".  
**Steps 16.** End function.

**Code :**

#include <iostream>

#include <algorithm>

using namespace std;

void findPairWithSum(int arr[], int n, int key) {

sort(arr, arr + n);

int left = 0, right = n - 1;

while (left < right) {

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

if (sum == key) {

cout << "Pair found: " << arr[left] << ", " << arr[right] << endl;

return;

} else if (sum < key) {

left++;

} else {

right--;

}

}

cout << "No pair found" << endl;

}

int main() {

int choice ;

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

cin >> choice ;

while (choice--) {

int n, key;

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

cin >> n;

int arr[n];

cout << "Enter the array elements: ";

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

cin >> arr[i];

}

cout << "Enter the key sum: ";

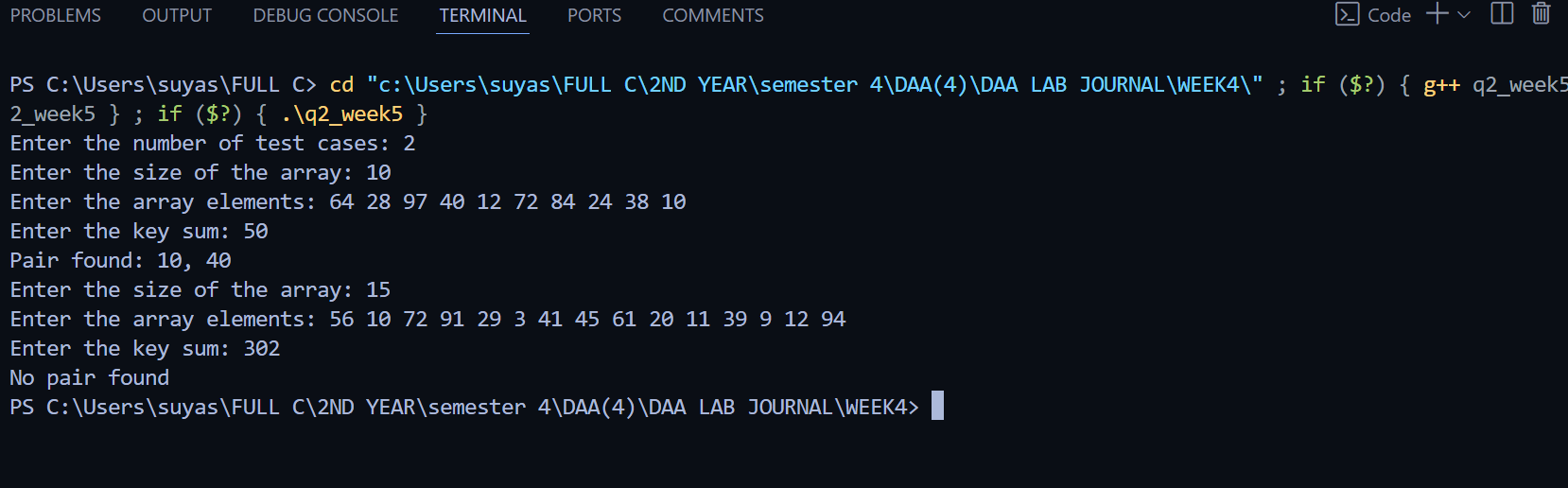
cin >> key;

findPairWithSum(arr, n, key);

}

return 0;

}

**OUTPUT**