**Experiment No. – 3.1**

**Student Name: Deepak Saini UID: 20BCS4066**

**Branch: 20BCC1 Section/Group: A**

**Semester:**  **5th Date of Performance: 11/10/2022**

**Subject Name: ADVANCED PROGRAMMING LAB**

**Subject Code: 20CSP-334**

1. **Aim/Overview of the practical:**

Design a quick sort with random pivoting using Lomuto partition scheme.

1. **Task to be done:**

Design a quick sort with random pivoting using Lomuto partition scheme.

1. **Steps for practical**:

**Lomuto’s Partition Scheme:**

This algorithm works by assuming the pivot element as the last element. If any other element is given as a pivot element then swap it first with the last element. Now initialize two variables i as low and j also low, iterate over the array and increment i when arr[j] <= pivot and swap arr[i] with arr[j] otherwise increment only j. After coming out from the loop swap arr[i] with arr[hi]. This i stores the pivot element.

partition(arr[], lo, hi)

    pivot = arr[hi]

    i = lo     // place for swapping

    for j := lo to hi – 1 do

        if arr[j] <= pivot then

            swap arr[i] with arr[j]

            i = i + 1

    swap arr[i] with arr[hi]

    return i

partition\_r(arr[], lo, hi)

    r = Random Number from lo to hi

    Swap arr[r] and arr[hi]

    return partition(arr, lo, hi)

quicksort(arr[], lo, hi)

    if lo < hi

        p = partition\_r(arr, lo, hi)

        quicksort(arr, lo , p-1)

        quicksort(arr, p+1, hi)

**4 . Code:**

#include <bits/stdc++.h>

using namespace std;

/\*This function takes last element as pivot, places

the pivot element at its correct position in sorted

array, and places all smaller (smaller than pivot)

to left of pivot and all greater elements to right

of pivot\*/

int partition(int arr[], int low, int high)

{

// pivot

int pivot = arr[high];

// Index of smaller element

int i = (low - 1);

for (int j = low; j <= high - 1; j++)

{

// If current element is smaller

// than or equal to pivot

if (arr[j] <= pivot) {

// increment index of

// smaller element

i++;

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

}

}

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

return (i + 1);

}

// Generates Random Pivot, swaps pivot with

// end element and calls the partition function

int partition\_r(int arr[], int low, int high)

{

// Generate a random number in between

// low .. high

srand(time(NULL));

int random = low + rand() % (high - low);

// Swap A[random] with A[high]

swap(arr[random], arr[high]);

return partition(arr, low, high);

}

/\* The main function that implements

QuickSort

arr[] --> Array to be sorted,

low --> Starting index,

high --> Ending index \*/

void quickSort(int arr[], int low, int high)

{

if (low < high) {

/\* pi is partitioning index,

arr[p] is now

at right place \*/

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

// Separately sort elements before

// partition and after partition

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

/\* Function to print an array \*/

void printArray(int arr[], int size)

{

int i;

for (i = 0; i < size; i++)

cout<<arr[i]<<" ";

}

// Driver Code

int main()

{

int arr[] = { 10, 7, 8, 9, 1, 5 };

int n = sizeof(arr) / sizeof(arr[0]);

printf("Input array: \n");

printArray(arr, n);

cout<<endl;

quickSort(arr, 0, n - 1);

printf("Sorted array: \n");

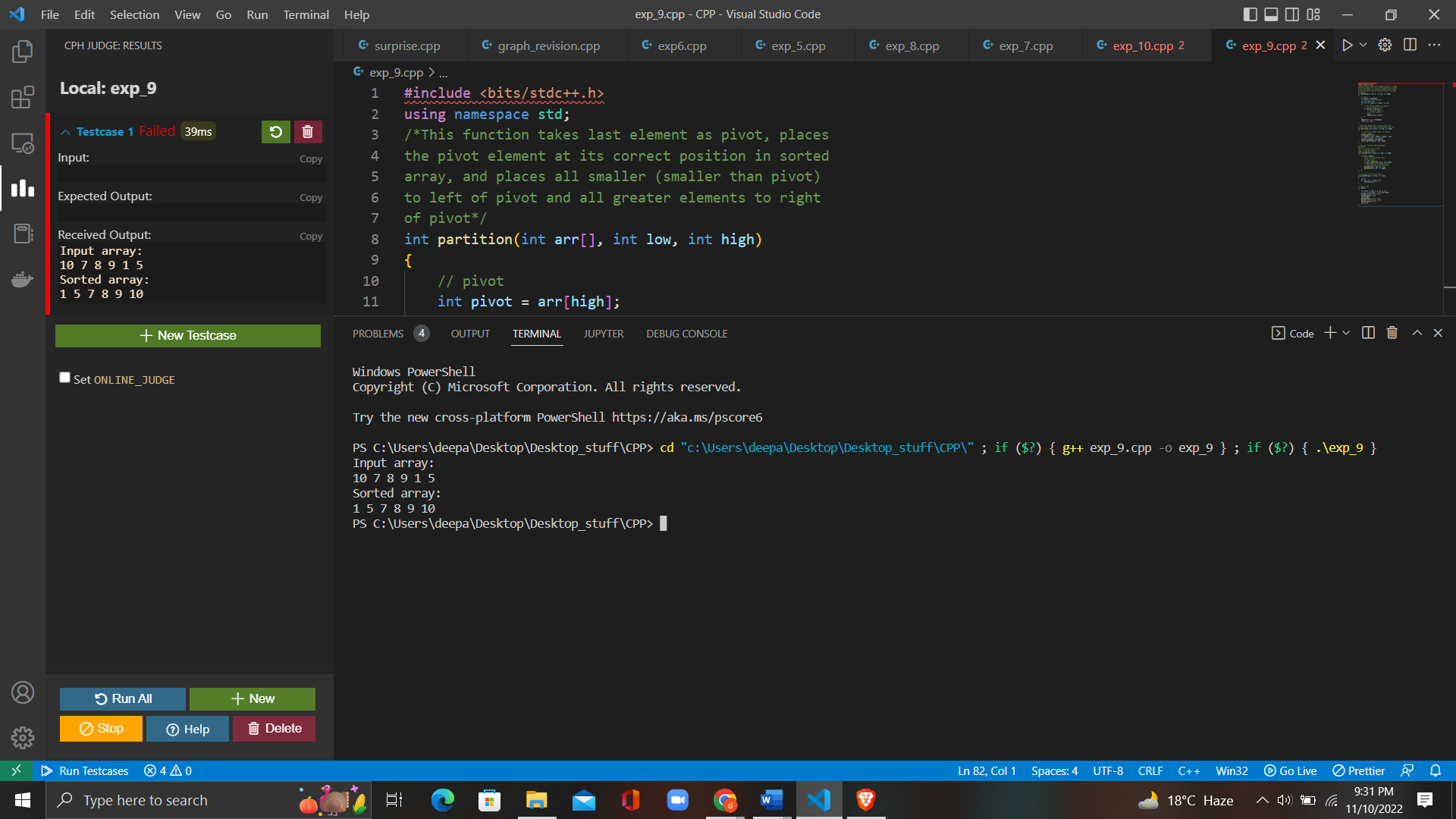
printArray(arr, n);

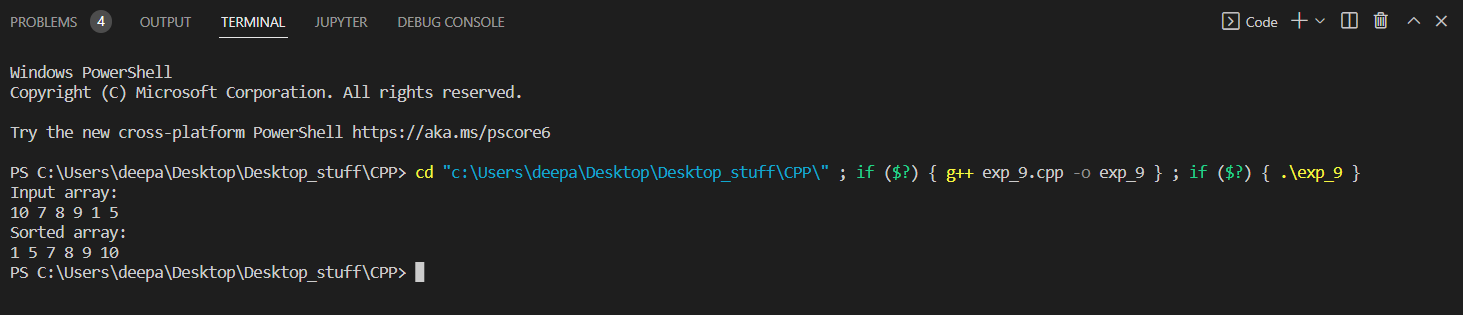
return 0;

}

**5.** **Output:**

**a)**





**6. Learning Outcomes:**

* Decide and implement an appropriate graph algorithm and hashing function in computer networks for data security.
* Learn Lomuto Partition Scheme
* Revise the basics of Quick Sort algorithm

**Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. No. | Parameters | Marks Obtained | Maximum Marks |
| 1. |  |  |  |
| 2. |  |  |  |
| 3. |  |  |  |