ESc101 : Practice Sheet 1

Q1. QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. The pivot element actually divides the array into two parts.
There are many ways of picking up the pivot element, in the given code we are picking the last element of array as the pivot element.

The key process in quickSort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x.

Example : { 8, 6 , 3 , 5 , 2 , 4 } is the array, we take 4 as pivot and partition the array by keeping 4 at its correct position, so we get : { 3 , 2 , **4** , 8 , 6 , 5 }

Partition Algorithm :
. We start from the leftmost element and keep track of index of smaller (or equal to x) elements as i.
. While traversing, if we find a smaller element, we swap current element with arr[i].
. Otherwise we ignore current element.
. At the end we swap the pivot element(last element) with arr[i+1]

Below is the code for partition algorithm for quick sort, fill the red blanks to implement the partition algorithm.

```
void swap(int* a, int* b) // swap the two elements
{
        int t = *a;
        *a = *b;
        __ = __;
}

// l is the  index of first element and h is the index of last element of array
int partition (int arr[], int l, int h)
{
        int x = arr[ __ ]; // pivot  element
        int i = (l – 1); // Index of smaller element
        for (int j = l; j <= h- 1; j++)
        {
                // If current element is smaller than or equal to pivot
                if ( __  <= __ )
                {
                        i++; // increment index of smaller element
                        swap(&arr[i], &arr[j]); // Swap current element with index
                }
        }
        swap( ____ , ____);

        return (i + 1);
}
```

Q2. C**ounting Sort** is an algorithm for sorting a collection of integers according to keys that are small integers. It operates by counting the number of elements that have each distinct key value, and using arithmetic on those counts to determine the positions of each key value in the output sequence.

Example : Consider the array in range 0 – 9 : { 3 , 1 , 2 , 8 , 5 ,4 , 1 }
1) Take a count array to store the count of each unique object.

Index:	0	1	2	3	4	5	6	7	8	9
Count:	0	2	1	1	1	1	0	0	1	0

2) Modify the count array such that each element at each index stores the sum of previous counts.

Index:	0	1	2	3	4	5	6	7	8	9
Count:	0	2	3	4	5	6	6	6	7	7

The modified count array indicates the position of each object in the output sequence.

3) Output each object from the input sequence followed by decreasing its count by 1.

Process the input data: 3 , 1 , 2 , 8 , 5 ,4 , 1.
Position of 3 is 4.
Put data 3 at index 4 in output , decrease its count by 1.
Position of 1 is 2.
Put data 1 at index 2 in output.
Decrease count by 1 to place next data 1 at an index 1 smaller than this index.

Below is the code for counting sort function, fill the red blanks to implement the algorithm.

```
#include <stdio.h>
#define RANGE 9

// The function that sort the given array ,  n is size of array
void countSort(int arr[] , int n)
{
        // The output  array that will have sorted arr
        int output[ n ];

        // Create a count array to store count of inidividul integers

        int count[RANGE + 1], i;
        for ( _________ )   // initialize all elements of count array as 0
                ___ = _____  ;

        // Store count of each element
        for(i = 0;  ___ ; ++i)
                ++count[arr[i]];

        // Change count[i] so that count[i] now contains actual position of
        // this integer in output array
        for (i = 1; i <= RANGE; ++i)
                count[i] += count[ __ ];
        // Build the output array
        for (i = 0; i < n ; ++i)
        {
                output[count[ ___ ] = arr[i];
                --count[str[i]];
        }

        // Copy the output array to arr, so that arr now
        // contains sorted integers
        for (i = 0; i < n ; ++i)
        ___ = _____
}
```