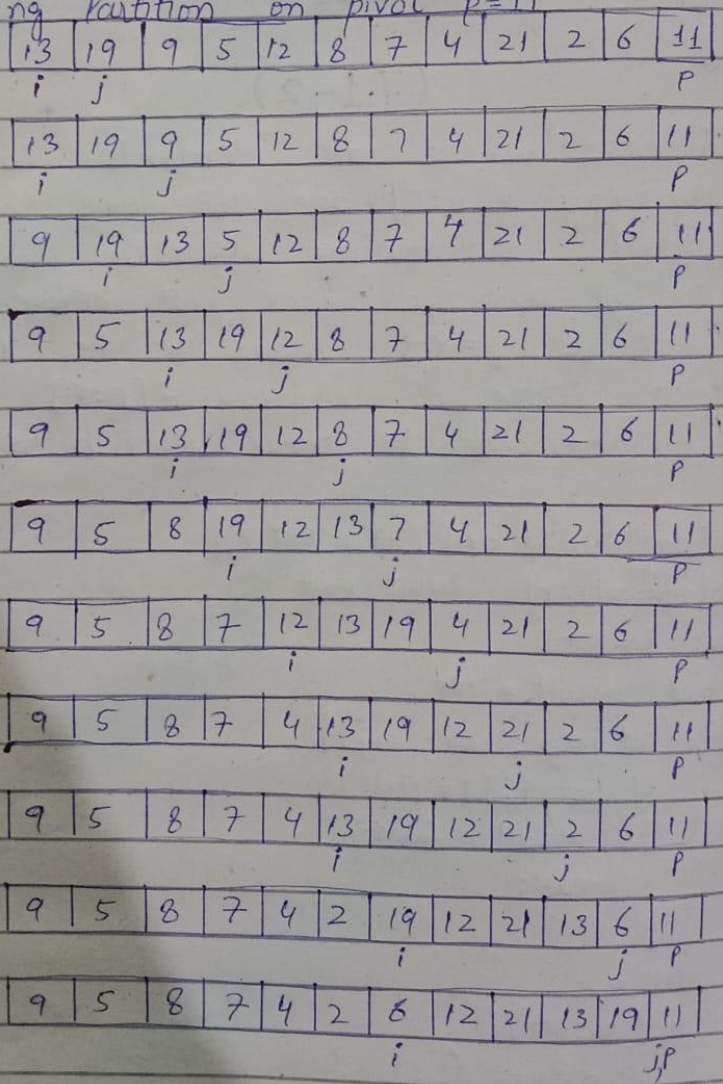
Assignment #02

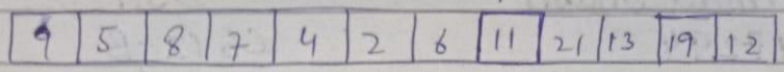
Sarmad Shabir
417294
DSA
CS 12-B
Submitted To:- Dr. Sohail Iqbal

(7.1-1)

A = {13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11}

Applying Partition on pivot p = 11





Partitioned array would be;

$A = \{9,5,8,7,4,2,6,11,21,13,19,12\}$.

Pivot = 11

(7.1-2)

In case where all elements in the subarray A[p:r] are the same, the typical partitioning algorith will set "q" as the index just before the pivot i.e $q = r-1$ where "r" is the last index of subarray A[p:r]
The c++ code for above algorith is as;

```
int partition(int arr[], int p, int r){
 int pivot = arr[r]   //select pivot
 int i = p-1
 bool result = true;
 for(int j= p ; j<r ; ++j){
     if (arr[j] != pivot) {
          result = false;
        break;
 }}
 int q;
 if (result) {
    q = (p+r)/2;
 }
```

```
else {
  q = p-1
  for (int j=p ; j<r ; ++j){
   if (arr[j] <= pivot){
        q =q+1;
    swap(arr[q], arr[j]);
   }}
   q=q+1;
  swap(arr[q], arr[r]);
   return q;
}
```

(7.1-3)

The partition subroutine operates in $\Theta(n)$ time on a subarray of size n because it requires linear processing during the rearrangement of elements around the pivot. This linear complexity is directly proportional to the size of the input array, observed in both best and average cases. Despite the possibility of a higher complexity in the worst-case scenario, its occurrences is highly unlikely.

(7.1-4)

Code in C++ :-

```
int partition(int arr[], int l, int h){
 int pivot = arr[h] // pivot = last element
 int i = l-1;
 for(int j= l; j<h ; ++j){
   if(arr[j] >= pivot{
     i++;
    swap(arr[i], arr[j];
 }}
    swap(arr[i+1], arr[h]);
    return i+1;
}
void QuickSort(int arr[], int l, int h){
    if (l < h){
  int pivot = partition(arr, l, h);
  QuickSort(arr, l, pivot-1);
  QuickSort (arr, pivot+1, h);
 }}
 void swap( int &x, int &y){
      int   temp = x
          x = y;
        y = temp;
 }
 void decreaseSort(int arr[], int n){
  QuickSort (arr, 0, n-1);
 }
```