Name Chirag Rana
SE COMPS
2018130043

Experiment 2

AIM: Experiment based on divide and Conquer approach (**Quicksort).**

Theory:

SORTING:

Sorting is nothing but arranging the data in ascending or descending order. The term sorting came into picture, as humans realised the importance searching quickly.

There are so many things in our real life that we need to search for, like a particular record in database, roll numbers in merit list, a particular telephone number in telephone directory, a particular page in a book etc. All this would have been a mess if the data was kept unordered and unsorted, but fortunately the concept of sorting came into existence, making it easier for everyone to arrange data in an order, hence making it easier to search.

Sorting arranges data in a sequence which makes searching easier.

SORTING ALGORITHMS:

There are many different techniques available for sorting, differentiated by their efficiency and space requirements. Following are some sorting techniques which we will be covering in next few tutorials.

1. Bubble Sort
2. Insertion Sort
3. Selection Sort
4. Quick Sort
5. Merge Sort
6. Heap Sort

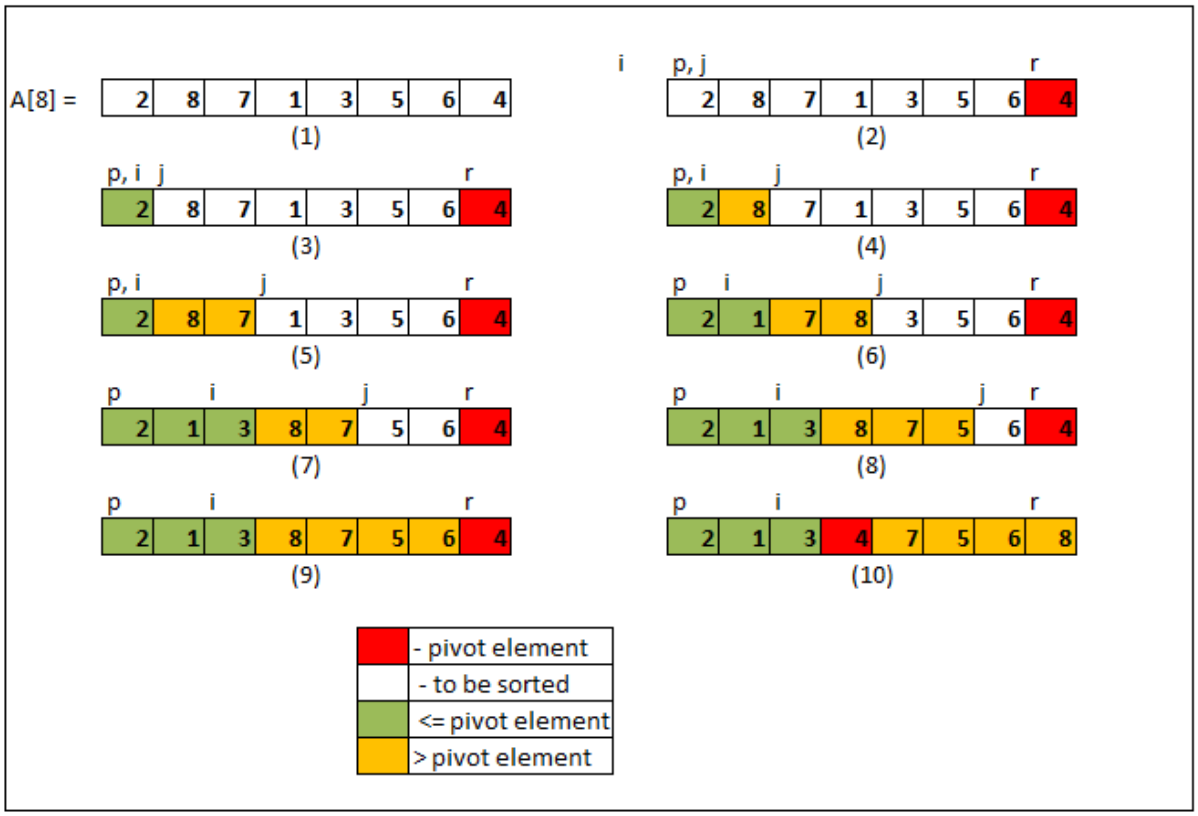
QUICK SORT:

Quick sort is a highly efficient sorting algorithm and is based on partitioning of array of data into smaller arrays. A large array is partitioned into two arrays one of which holds values smaller than the specified value, say pivot, based on which the partition is made and another array holds values greater than the pivot value.

Quicksort partitions an array and then calls itself recursively twice to sort the two resulting subarrays. This algorithm is quite efficient for large-sized data sets as its average and worst-case complexity are O(nLogn) and (n2), respectively.

The pivot value divides the list into two parts. And recursively, we find the pivot for each sub-lists until all lists contains only one element.

Here we find the proper position of the pivot element by rearranging the array using partition function. Then we divide the array into two halves left side of the pivot (elements less than pivot element) and right side of the pivot (elements greater than pivot element) and apply the same step recursively.



Algorithm:

QuckSort(Arr,start,end){

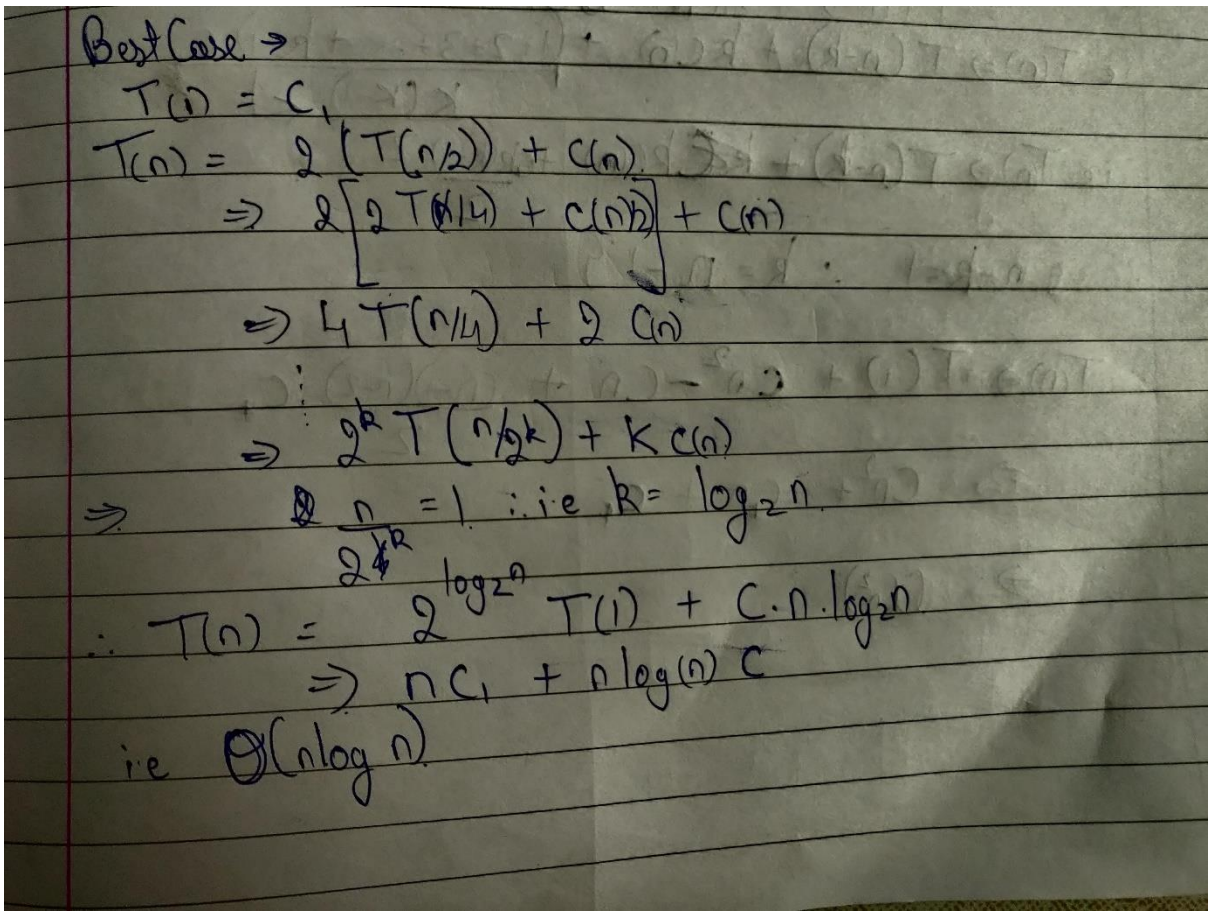
```
    If (start< end){
        pIndex ← partion(Arr,start,end)
        QuickSort(arr,start,pIndex-1)
        QuickSort(arr,pIndex+1,end)
    }
}
```

Partition(Array,start,end){

```
        Pivot← A[start]
        pIndex← start
        for j← start to end-1{
                if (A[j] <= pivot){
                        swap(A[j], A[pIndex+1])
                        pIndex← pIndex +1
                }
        }
        swap(A[pIndex,pivot])
        return pIndex
}
```

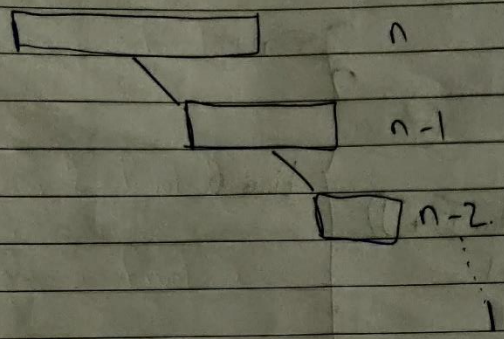
Time Complexity:

Best Case



Worst Case

Worst Case. [Totally unbalanced partitioning]
Sorted array. i.e.
There wont be any left segment



$T(1) = C_1$

$T(n) = T(n-1) + C.n$

i.e ~~$T(n) = T\,n\,2$~~

$T(n) = T(n-2) + C(n-1) + C(n)$

$\Rightarrow T(n-2) + 2Cn - C.$

$\Rightarrow T(n-3) + 3Cn - 3C.$

$= T(n-4) + 4Cn - 6C.$

i.e $T(n) \Rightarrow T(n-k) + kC(n) + \underbrace{(1+2+3+\ldots+k-1)}_{\frac{K(K-1)}{2}} C_2$

i.e $T(n) \Rightarrow T(n-k) + kC(n) + \frac{k(k-1)}{2}$

$\therefore \{n-k=1 \quad \therefore k = n-1\}$

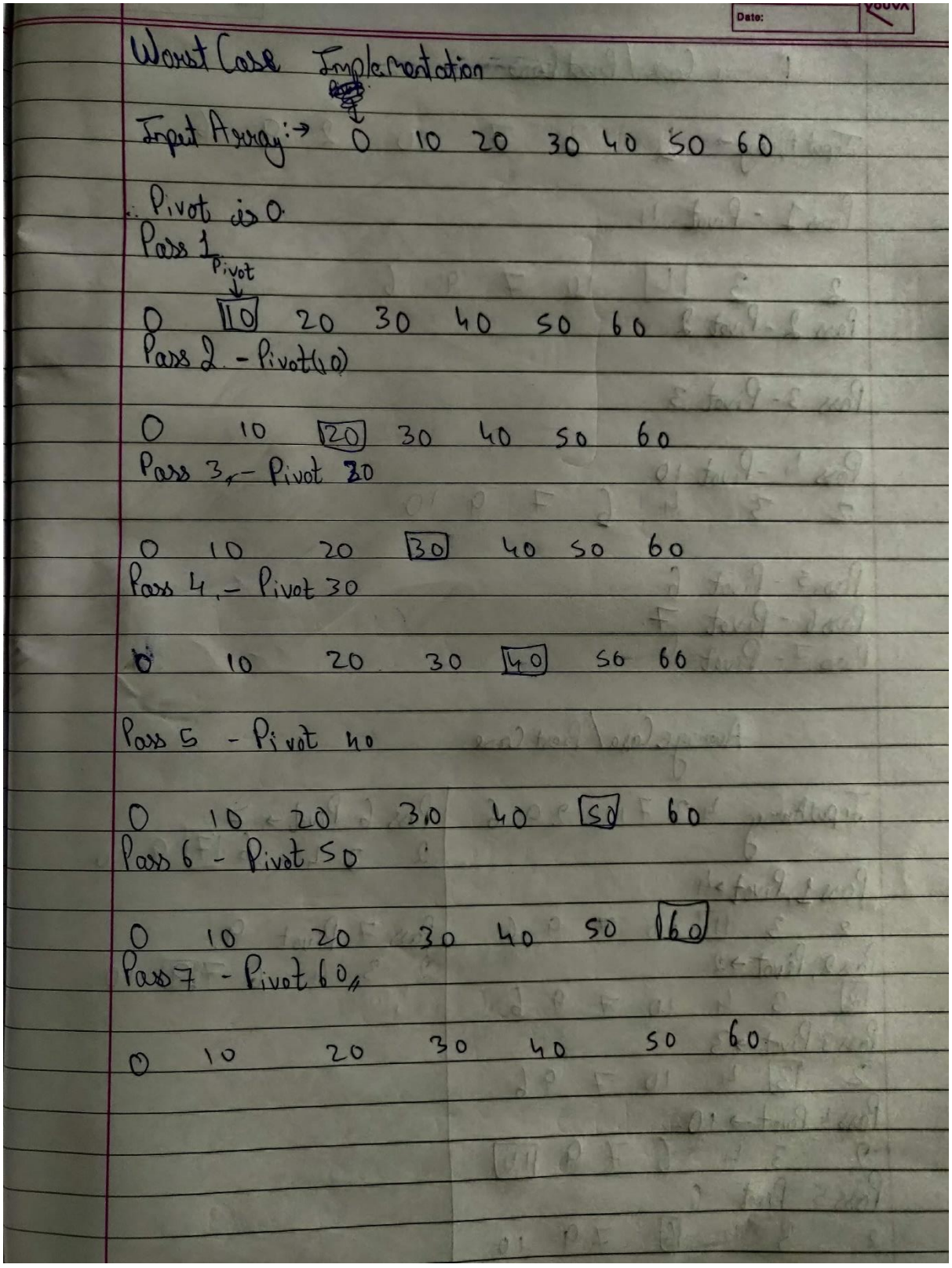
$\therefore T(n) \Rightarrow T(1) + cn^2 - Cn + \frac{(n-1)(n-2)}{2} C_2$

$\Rightarrow Cn^2 + C_3 n + C_4$

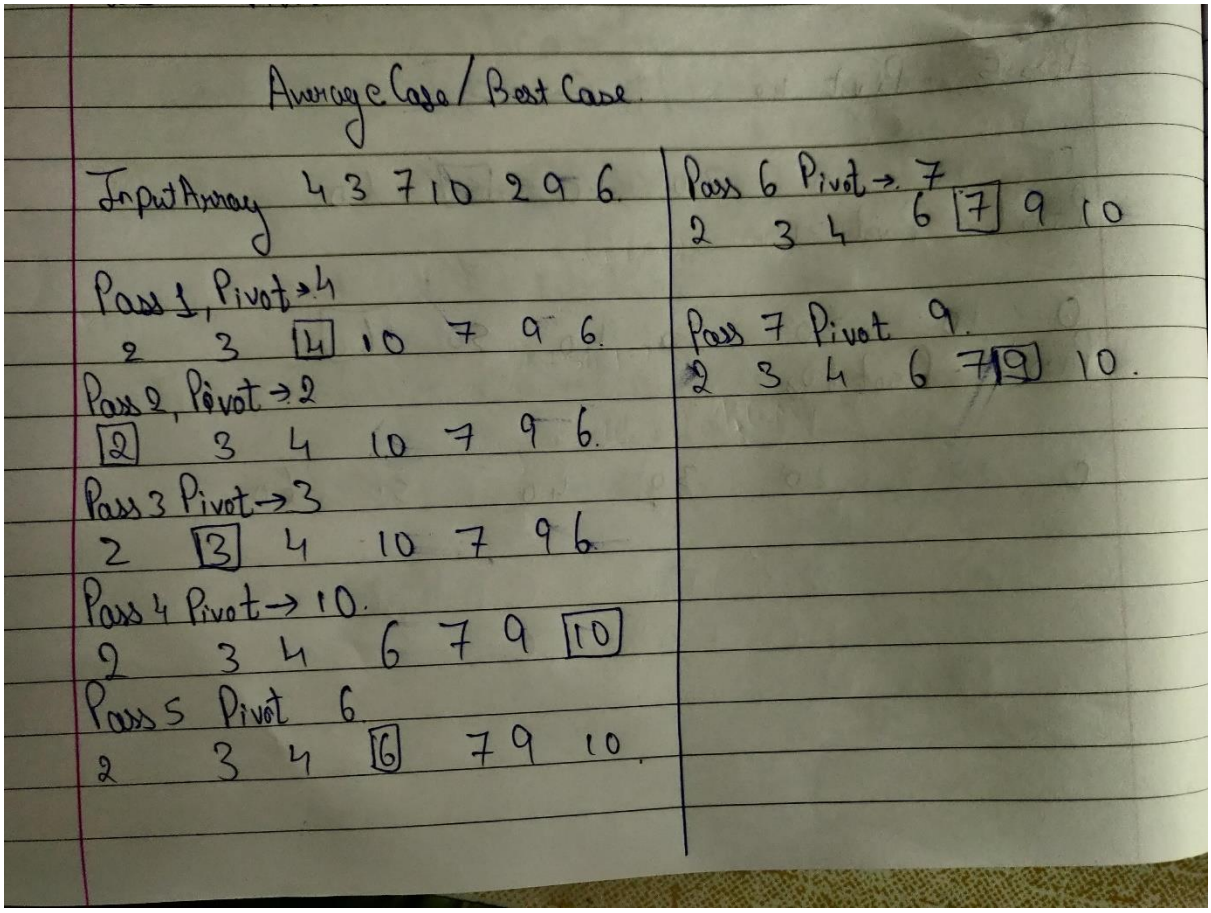
$\Rightarrow O(n^2)$

Design And Implementation:

Worst Case



Best/Average case:



Program Code:

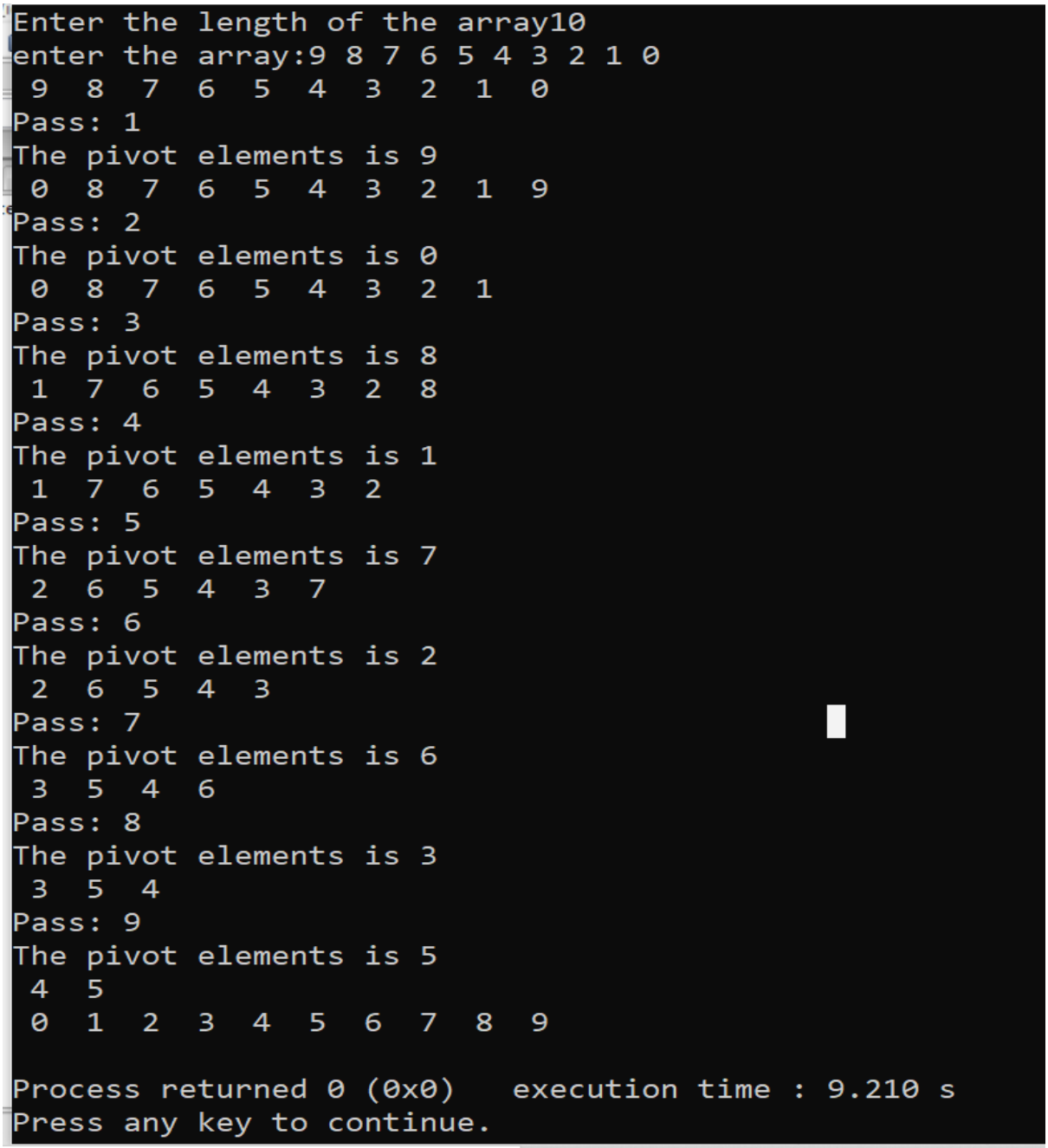
```
#include<stdio.h>
#include<stdlib.h>

void display(int array[],int start,int len);
void QuickSort(int arr[],int start,int end);
int pass = 1;
int main(){
   int len;
        int* array;
        printf("Enter the length of the array");
        scanf("%d",&len);
        array = (int*)malloc(len * sizeof(int));
        printf("enter the array:");
```

```
        for (int i = 0; i< len ; i++){
                scanf("%d",&array[i]);
    }
    display(array,0,len-1);
        QuickSort(array,0,len-1);
    display(array,0,len-1);
    return 0;
}
void QuickSort(int array[], int start, int end){
    int i= start;
    int j = start;
    int temp;
    if(start < end){
    printf("Pass: %d\n",pass++);
    printf("The pivot elements is %d\n",array[i]);
        while(j <= end){
            while(j <= end && array[++j] > array[start]);
                if(j <= end){
                    temp = array[i+1];
                    array[++i] = array[j];
                    array[j] = temp;
                }
        }
        temp = array[i];
        array[i] = array[start];
        array[start] = temp;
        display(array,start,end);
        QuickSort(array,start,i-1);
        QuickSort(array,i+1,end);
    }
}
```

```
void display(int array[],int start, int end){
        for(int i = start; i <= end ; i++){
                printf(" %d ",array[i]);
        }
        printf("\n");
return ;
}
```

Output:

Average/ Best case :



```
Enter the length of the array10
enter the array:1 2 3 4 5 6 7 8 9 10
 1  2  3  4  5  6  7  8  9  10
Pass: 1
The pivot elements is 1
 1  2  3  4  5  6  7  8  9  10
Pass: 2
The pivot elements is 2
 2  3  4  5  6  7  8  9  10
Pass: 3
The pivot elements is 3
 3  4  5  6  7  8  9  10
Pass: 4
The pivot elements is 4
 4  5  6  7  8  9  10
Pass: 5
The pivot elements is 5
 5  6  7  8  9  10
Pass: 6
The pivot elements is 6
 6  7  8  9  10
Pass: 7
The pivot elements is 7
 7  8  9  10
Pass: 8
The pivot elements is 8
 8  9  10
Pass: 9
The pivot elements is 9
 9  10
 1  2  3  4  5  6  7  8  9  10

Process returned 0 (0x0)   execution time : 14.223 s
Press any key to continue.
```

Conclusion:

1] I have Implemented QuickSort using a single function combining the partition function with the Quicksort function

2] I have learned how the sorted array can also be a Worst case in Quicksort.