Experiment 3

AA Experiment DATE:

Kreena Shah
60004210243
C'32

Aim : To implement randomized quick sort (randomized algorithms)

Theory :

Quick Sort is a popular sorting algorithm that chooses a pivot element & sorts the input list around that pivot element Randomized quick sort is designed to decrease the chances of the algorithm being executed in the worst case time complexity of $O(n^2)$

The worst case time complexity of quick sort arises when the input given is an already sorted list, leading to $n(n-1)$ comparisons

There are two ways to randomize quick sort :

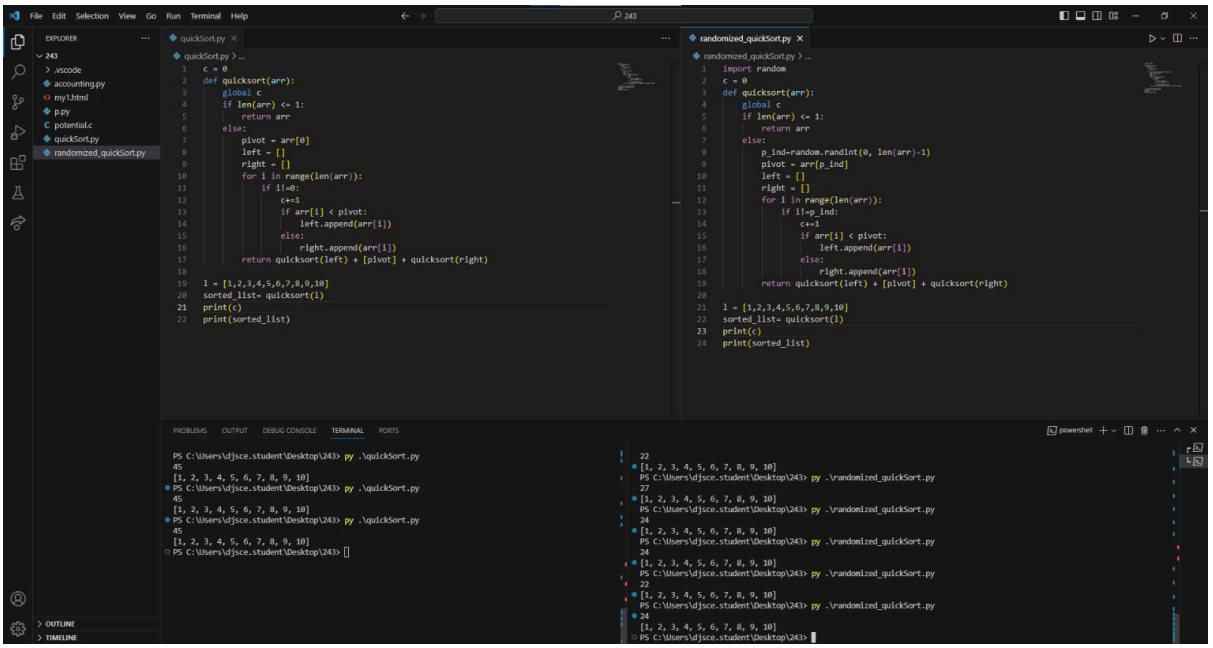
(1) Randomly Shuffling the input

(2) Randomly choosing the pivot element

Algorithm :

```
QuickSortRandomized (A, l, r) {
  if (l < r) {
      x ← an element selected randomly
      i ← Partition (A, l, r, x)
      QuickSortRandomized (A, l, l-1)
      QuickSortRandomized (A, l+1, r)
  }
}
```

Code & Output :

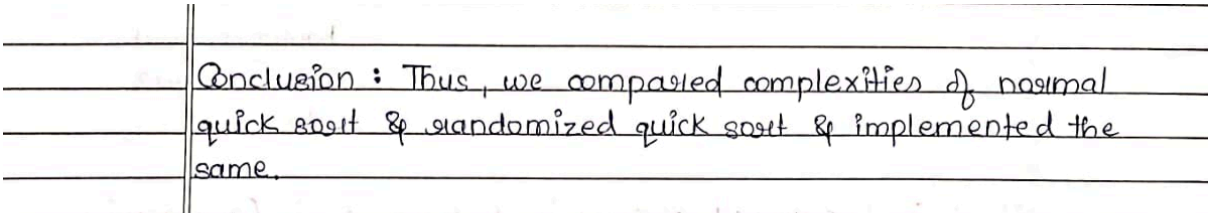


```
c = 0
def quicksort(arr):
    global c
    if len(arr) <= 1:
        return arr
    else:
        pivot = arr[0]
        left = []
        right = []
        for i in range(len(arr)):
            if i!=0:
                c+=1
                if arr[i] < pivot:
                    left.append(arr[i])
                else:
                    right.append(arr[i])
        return quicksort(left) + [pivot] + quicksort(right)

l = [1,2,3,4,5,6,7,8,9,10]
sorted_list= quicksort(l)
print(c)
print(sorted_list)
```

```
PS C:\Users\djsce.student\Desktop\243> py .\quickSort.py
45
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
PS C:\Users\djsce.student\Desktop\243> py .\quickSort.py
45
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
PS C:\Users\djsce.student\Desktop\243> py .\quickSort.py
45
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
PS C:\Users\djsce.student\Desktop\243>
```

```
import random
c = 0
def quicksort(arr):
    global c
    if len(arr) <= 1:
        return arr
    else:
        p_ind=random.randint(0, len(arr)-1)
        pivot = arr[p_ind]
        left = []
        right = []
        for i in range(len(arr)):
            if i!=p_ind:
                c+=1
                if arr[i] < pivot:
                    left.append(arr[i])
                else:
                    right.append(arr[i])
        return quicksort(left) + [pivot] + quicksort(right)

l = [1,2,3,4,5,6,7,8,9,10]
sorted_list= quicksort(l)
print(c)
print(sorted_list)
```

```
22
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
PS C:\Users\djsce.student\Desktop\243> py .\randomized_quickSort.py
27
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
PS C:\Users\djsce.student\Desktop\243> py .\randomized_quickSort.py
24
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
PS C:\Users\djsce.student\Desktop\243> py .\randomized_quickSort.py
24
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
PS C:\Users\djsce.student\Desktop\243> py .\randomized_quickSort.py
22
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
PS C:\Users\djsce.student\Desktop\243> py .\randomized_quickSort.py
24
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
PS C:\Users\djsce.student\Desktop\243>
```



Conclusion : Thus, we compared complexities of normal quick sort & randomized quick sort & implemented the same.