Name:- Mitali Vaghela
ID NO.:- 1002143562

GitHub Link:-.https://github.com/mitalivaghela1515/Hands-On-6.git

Q-1. Implement both versions of quicksort (random and non-random choice for the pivot) and share the GitHub repository with your source code.

Ans: - versions of quicksort random choice for the pivot.

```
import random

def partition(arr, low, high):
    pivot = arr[high]
    i = low - 1
    for j in range(low, high):
        if arr[j] <= pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]
    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    return i + 1

def randomized_partition(arr, low, high):
    random_index = random.randint(low, high)
    arr[high], arr[random_index] = arr[random_index], arr[high]
    return partition(arr, low, high)

def randomized_quicksort(arr, low, high):
    if low < high:
        pi = randomized_partition(arr, low, high)
        randomized_quicksort(arr, low, pi - 1)
        randomized_quicksort(arr, pi + 1, high)

# Example usage:
arr = [10, 7, 8, 9, 1, 5]
n = len(arr)
print("Original array:", arr)
```

```
# Shuffle the array
random.shuffle(arr)
print("Shuffled array:", arr)

# Sort using random pivot quicksort
randomized_quicksort(arr, 0, n - 1)
print("Sorted array (random pivot):", arr)
```

Ans: - versions of quicksort non-random choice for the pivot.

```
def partition(arr, low, high):
    pivot = arr[high]  # Selecting the last element as the pivot
    i = low - 1

    for j in range(low, high):
        if arr[j] <= pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]

    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    return i + 1

def quicksort(arr, low, high):
    if low < high:
        pi = partition(arr, low, high)
        quicksort(arr, low, pi - 1)
        quicksort(arr, pi + 1, high)

# Example usage:
arr = [10, 7, 8, 9, 1, 5]
n = len(arr)
print("Original array:", arr)

# Sort using non-random pivot quicksort
quicksort(arr, 0, n - 1)
print("Sorted array (non-random pivot):", arr)
```

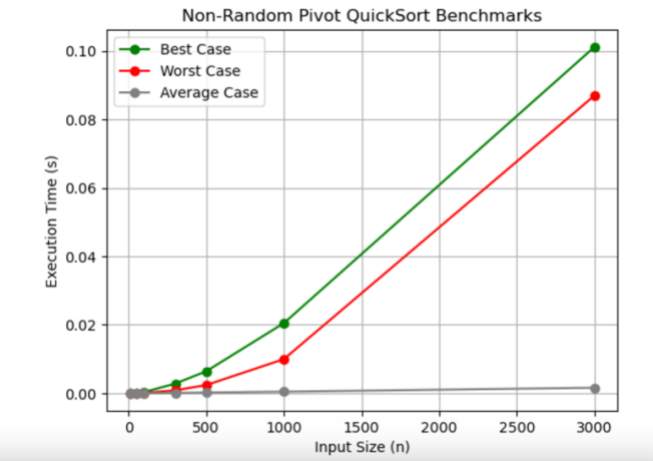
Q-2. For the non-random pivot version of quicksort show the following benchmarks on the same graph:

2a) best case (generate a set of inputs that will always be the best case, repeat for multiple array input sizes "n").

2b) worst case (generate a set of inputs that will always be the worst case, repeat for multiple array input sizes "n").

2c) average case (generate a set of inputs from a uniform distribution, repeat for multiple array input sizes "n").

Ans: -



Q-3. Mathematically derive the average runtime complexity of the non-random pivot version of quicksort.

Ans: -

Let's look at the recurrence relation that characterizes the behavior of the algorithm to determine the average runtime complexity of the non-random pivot version of quicksort.

The array is divided using a selected pivot element in the non-random pivot variant of quicksort. The following represents the recurrence relation for the worst-case temporal complexity:

$T(n)=T(n-1) +T(0) +0(n)$

In this case, the temporal complexity for an array of size n is represented by T(n). According to the recurrence relation, the time complexity of an array of size n is equal to the product of the time complexities of the subarrays of size n−1 and 0, as well as the O(n) partitioning step.

Let's now calculate the average-case temporal complexity. We assume that the pivot is selected evenly at random in the average situation. Let X X be the random variable that, following partitioning, represents the size of the smaller subarray. The following is an expression for the average-case time complexity:

$$T(n) = \frac{1}{n}\sum_{i=0}^{n-1}(T(i) + T\ (n - i - 1) + O\ (n)$$

The chance of selecting any element as the pivot is represented as $\frac{1}{n}$ In this case, and the sum considers all possible pivot positions between 0 and n − 1.

We can solve the recurrence relation using the Master Theorem or other techniques to reduce this statement. Because the subproblems in this situation are not of the same magnitude, the Master Theorem cannot be applied directly; rather, it can be applied to specific words inside the total.

The non-random pivot version of quicksort typically has an average-case time complexity of O (n log n) but depending on the assumptions made about the randomness of pivot selection, the precise mathematical derivation may require probabilistic methods and require a more thorough analysis.