ASSIGNMENT 3 OF DAA

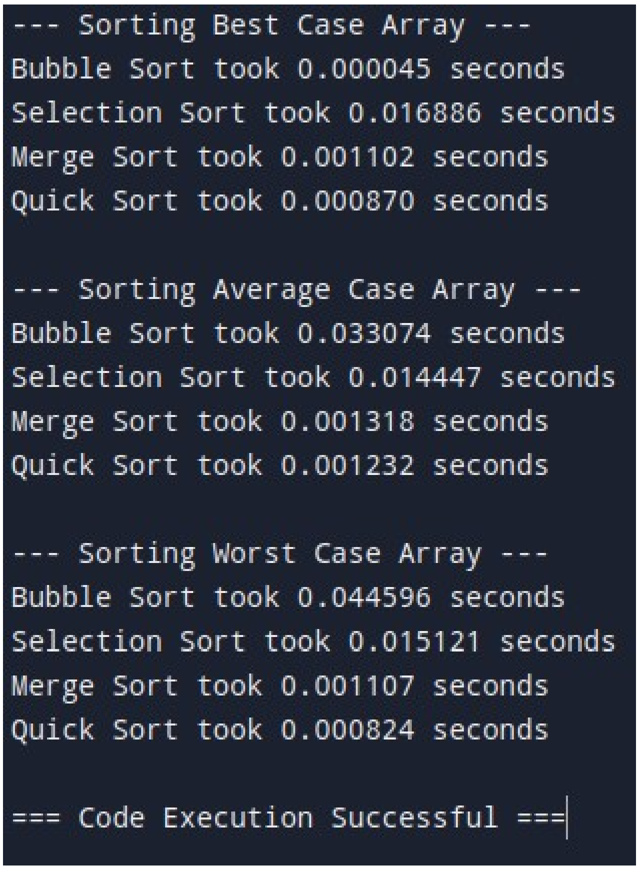
NAME: SHUMAIL ABDUL REHMAN.
SAP: 47891.

Github link: https://github.com/ShumailCodes?tab=repositories

Python_Code:

```
import time
import random

# Helper function to calculate execution time
def measure_time(func, arr):
    start = time.time()
    func(arr)
    end = time.time()
    return end - start

# Bubble Sort
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        swapped = False
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
                swapped = True
        if not swapped:
            break

# Selection Sort
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i+1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]

# Merge Sort
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        L = arr[:mid]
        R = arr[mid:]

        merge_sort(L)
        merge_sort(R)

        i = j = k = 0
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1

        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1

        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1

# Quick Sort
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)

# Main execution and timing
def run_sorts_and_measure(arr, arr_name):
    print(f"\n--- Sorting {arr_name} ---")
    algorithms = [("Bubble Sort", bubble_sort),
                  ("Selection Sort", selection_sort),
                  ("Merge Sort", merge_sort),
                  ("Quick Sort", lambda x: quick_sort(x))]
    for name, algo in algorithms:
        arr_copy = arr.copy()  # Make a copy to avoid in-place sorting effects
        time_taken = measure_time(algo, arr_copy)
        print(f"{name} took {time_taken:.6f} seconds")

# Define arrays for best-case, average-case, and worst-case scenarios
size = 1000
array_best = list(range(size))              # Best-case: Sorted array
array_avg = random.sample(range(size), size) # Average-case: Random order array
array_worst = list(range(size, 0, -1))     # Worst-case: Reversed array

# Run and time the sorts
run_sorts_and_measure(array_best, "Best Case Array")
run_sorts_and_measure(array_avg, "Average Case Array")
run_sorts_and_measure(array_worst, "Worst Case Array")
```

OUTPUT:



```
--- Sorting Best Case Array ---
Bubble Sort took 0.000045 seconds
Selection Sort took 0.016886 seconds
Merge Sort took 0.001102 seconds
Quick Sort took 0.000870 seconds

--- Sorting Average Case Array ---
Bubble Sort took 0.033074 seconds
Selection Sort took 0.014447 seconds
Merge Sort took 0.001318 seconds
Quick Sort took 0.001232 seconds

--- Sorting Worst Case Array ---
Bubble Sort took 0.044596 seconds
Selection Sort took 0.015121 seconds
Merge Sort took 0.001107 seconds
Quick Sort took 0.000824 seconds

=== Code Execution Successful ===
```