Bansilal Ramnath Agarwal Charitable Trust's

Vishwakarma Institute of Information Technology

Department of Artificial Intelligence and Data Science

Name: Siddhesh Dilip Khairnar

Class: TY

Division: B

Roll No: 372028

Semester: V

Academic Year: 2023-2024

Subject Name & Code: Design and Analysis of Algorithm: ADUA31202

Title of Assignment: Implementation the following algorithm using Divide & Conquer method.
(a)Merge sort
(b) Quick Sort
Also display execution time for different size of input and perform the analysis.

Date of Performance: 17-08-2023

Date of Submission: 24-08-2023

ASSIGNMENT NO. 2

Working of Merge Sort:

- **Divide:** The unsorted array is divided into two equal subarrays until each subarray consists of a single element.
- **Conquer:** The subarrays are recursively sorted using the merge sort algorithm.
- **Combine:** The sorted subarrays are merged back together to form a single sorted array.

The key idea behind merge sort is merging two sorted arrays into one sorted array. The merge operation compares the elements from the two subarrays and places them in the correct order in the merged array. This process is repeated until all the subarrays are merged and a fully sorted array is obtained.

Time Complexity Analysis of Merge Sort:

Best Case: O(n*log n)
Worst Case: O(n*log n)
Average Case: O(n*log n)
Merge sort has a consistent time complexity of O(n*log n) in all cases. This is because it always divides the array into two halves and takes linear time to merge the halves.

Working of Quick Sort:

- **Partition:** The array is partitioned into two subarrays based on a pivot element. Elements smaller than the pivot are placed to the left, and elements greater than the pivot are placed to the right.
- **Recursion:** The partitioned subarrays are recursively sorted using the quick sort algorithm.
- **Combine:** The sorted subarrays are combined to form a fully sorted array.

The key idea behind quick sort is selecting a pivot element and rearranging the array such that all elements smaller than the pivot are to its left, and all elements greater than the pivot are to its right. This partitioning step is performed recursively on the subarrays until the entire array is sorted.

Time Complexity Analysis of Quick Sort:

Best Case: O(n*log n)
Worst Case: O(n^2)
Average Case: O(n*log n)
The time complexity of quick sort varies depending on the choice of pivot and the input array. In the best case, when the pivot divides the array into two equal halves, the time complexity is O(nlog n). However, in the worst case, when the pivot is always the smallest or largest element, the time complexity can be O(n^2). On average, quick sort has a time complexity of O(nlog n).

Name : Siddhesh Dilip khairnar
Rollno : 372028

DAA Assignment 2

* Problem statement : Implementation of following algorithm using Divide & conquer method.

a) Merge sort   b) Quick sort.

Also display execution time for different size of input & perform the analysis.

* Objective :

i) To learn about divide & conquer method
ii) To learn about quick sort
iii) To learn about merge sort
iv) Analyze the quick sort algorithm.
v) Analyze merge sort algorithm.
vi) Implement quick sort & merge sort using divide & conquer method.

* Quick sort :

① Algorithm :

```
if (start < end) {
  (i) P = Partition (arr, start, end)
  (ii) Quicksort (arr, start, P-1)
  (iii) Quicksort (arr, P+1, end)
}
```

② Partition Algorithm :

```
i) Pivot = arr [high]
     i = low-1
ii) for (j = low to high-1) {
      if (arr [j] <= Pivot) {
        i++
      swap (i, j) }}
```

iii) swap(i+1, high)
iv) return i+1

③ example:

[7,1,3,5,2,6,4] , Pivot = 4

↓ Partition

[1,3,2, ④, 7,5,6]

left Pivot   right pivot

Pivot = 2 ↓   ↓ Pivot = 5

[1,2,3] 4 [5,6,7]

↓ Join left Pivot, Pivot & right Pivot

[1,2,3,4,5,6,7]

⊛ Merge sort:

① Algorithm:

```
i) If (j<=1) : return arr
ii) Set left = array containing element from 0 to j/2
~~iii)~~ set right = array containing element from j/2+1 to j
iii) left = mergesort(left)
iv) right = merge sort (right)
v) return merge(left, right)
```

② Example :

arr = [12,31, ~~35~~25, 8, 32, 17, 40, 42]

divide arr into two sub arrays ~~titt~~ till we reach individual element

[12,31,25,8,32,17,40,42]

[12,31,25,8]   [32,17,40,42]

[12,31] [25,8]   [32,17] [40,42]

[12] [31] [25] [8]   [32] [17] [40] [42]

Compare 12 & 31, 25 & 8, 32 & 17 and 40 & 42. if they are not sorted then place them in sorted order.

[12,31] [8,25] [17,32] [40,42]

[8,12,25,31]   [17,32,40,42]

[8,12,17,25,31,32,40,42]

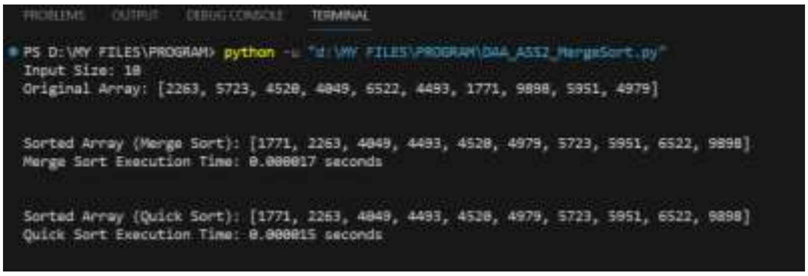
<u>Conclusion</u>: Thus, we have learned merge sort & quicksort algorithm & implement it using divide & conquer method.

CODE:

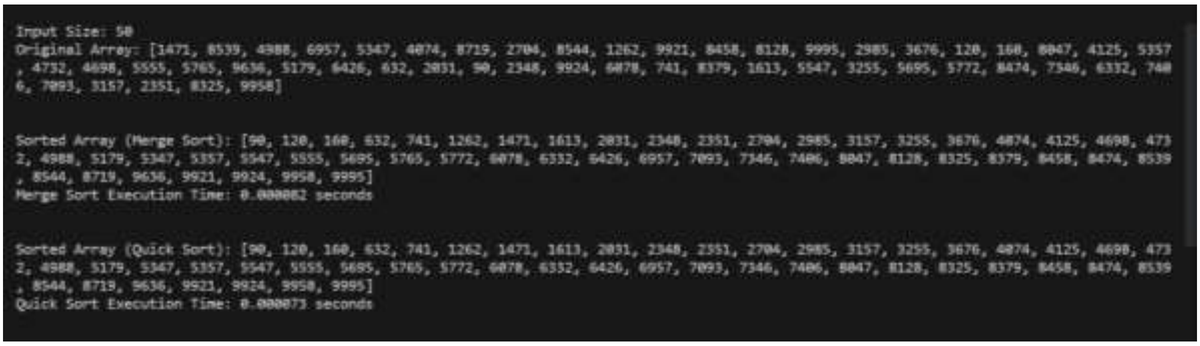
```
import random
import timeit

# Merge Sort implementation
def merge_sort(arr):
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left_half = arr[:mid]
    right_half = arr[mid:]

    left_half = merge_sort(left_half)
    right_half = merge_sort(right_half)

    return merge(left_half, right_half)

def merge(left, right):
    result = []
    left_idx, right_idx = 0, 0

    while left_idx < len(left) and right_idx < len(right):
        if left[left_idx] < right[right_idx]:
            result.append(left[left_idx])
            left_idx += 1
        else:
            result.append(right[right_idx])
            right_idx += 1

    result.extend(left[left_idx:])
    result.extend(right[right_idx:])

    return result

# Quick Sort implementation
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)

# Function to generate a random array of a given size
```

```
def generate_random_array(size):
    return [random.randint(1, 10000) for _ in range(size)]

# Function to measure execution time
def measure_execution_time(sort_function, arr):
    start_time = timeit.default_timer()
    sorted_array = sort_function(arr)
    end_time = timeit.default_timer()
    return sorted_array, end_time - start_time

# Test the algorithms for different input sizes
input_sizes = [10, 50, 100]
for size in input_sizes:
    arr = generate_random_array(size)

    # Measure execution time for Merge Sort
    sorted_arr_merge, merge_sort_time =
measure_execution_time(merge_sort, arr.copy())

    # Measure execution time for Quick Sort
    sorted_arr_quick, quick_sort_time =
measure_execution_time(quick_sort, arr.copy())

    print(f"Input Size: {size}")
    print(f"Original Array: {arr}")
    print("\n")
    print(f"Sorted Array (Merge Sort): {sorted_arr_merge}")
    print(f"Merge Sort Execution Time: {merge_sort_time:.6f}
seconds")
    print("\n")
    print(f"Sorted Array (Quick Sort): {sorted_arr_quick}")
    print(f"Quick Sort Execution Time: {quick_sort_time:.6f}
seconds")
    print("\n\n")
```

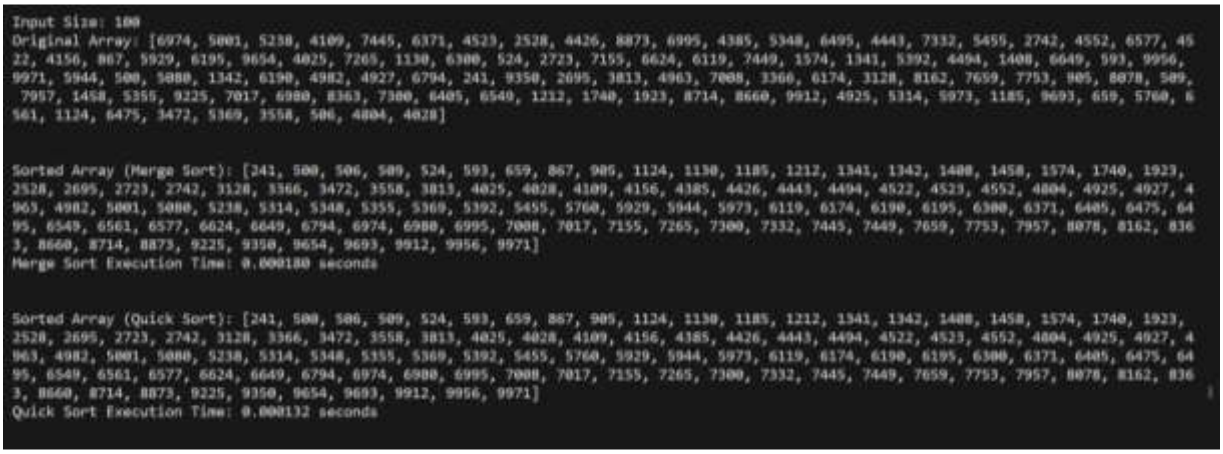
OUTPUT:



```
PS D:\MY FILES\PROGRAM> python -u "d:\MY FILES\PROGRAM\DAA_ASS2_MergeSort.py"
 Input Size: 10
 Original Array: [2263, 5723, 4520, 4049, 6522, 4493, 1771, 9898, 5951, 4979]


 Sorted Array (Merge Sort): [1771, 2263, 4049, 4493, 4520, 4979, 5723, 5951, 6522, 9898]
 Merge Sort Execution Time: 0.000017 seconds


 Sorted Array (Quick Sort): [1771, 2263, 4049, 4493, 4520, 4979, 5723, 5951, 6522, 9898]
 Quick Sort Execution Time: 0.000015 seconds
```

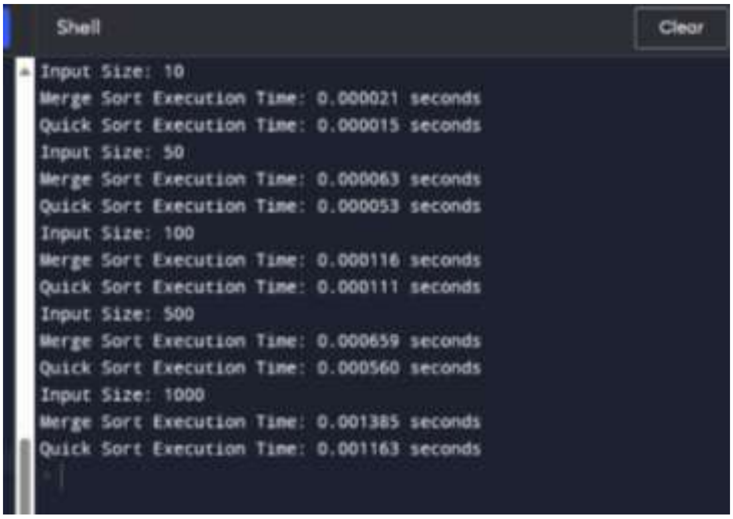


```
Input Size: 50
Original Array: [1471, 8539, 4988, 6957, 5347, 4074, 8719, 2704, 8544, 1262, 9921, 8458, 8128, 9995, 2985, 3676, 120, 160, 8047, 4125, 5357
, 4732, 4698, 5555, 5765, 9636, 5179, 6426, 632, 2031, 90, 2348, 9924, 6078, 741, 8379, 1613, 5547, 3255, 5695, 5772, 8474, 7346, 6332, 740
6, 7093, 3157, 2351, 8325, 9958]


Sorted Array (Merge Sort): [90, 120, 160, 632, 741, 1262, 1471, 1613, 2031, 2348, 2351, 2704, 2985, 3157, 3255, 3676, 4074, 4125, 4698, 473
2, 4988, 5179, 5347, 5357, 5547, 5555, 5695, 5765, 5772, 6078, 6332, 6426, 6957, 7093, 7346, 7406, 8047, 8128, 8325, 8379, 8458, 8474, 8539
, 8544, 8719, 9636, 9921, 9924, 9958, 9995]
Merge Sort Execution Time: 0.000082 seconds


Sorted Array (Quick Sort): [90, 120, 160, 632, 741, 1262, 1471, 1613, 2031, 2348, 2351, 2704, 2985, 3157, 3255, 3676, 4074, 4125, 4698, 473
2, 4988, 5179, 5347, 5357, 5547, 5555, 5695, 5765, 5772, 6078, 6332, 6426, 6957, 7093, 7346, 7406, 8047, 8128, 8325, 8379, 8458, 8474, 8539
, 8544, 8719, 9636, 9921, 9924, 9958, 9995]
Quick Sort Execution Time: 0.000073 seconds
```



```
Input Size: 100
Original Array: [6974, 5001, 5238, 4109, 7445, 6371, 4523, 2528, 4426, 8873, 6995, 4385, 5348, 6495, 4443, 7332, 5455, 2742, 4552, 6577, 45
22, 4156, 867, 5929, 6195, 9654, 4025, 7265, 1130, 6300, 524, 2723, 7155, 6624, 6119, 7449, 1574, 1341, 5392, 4494, 1400, 6649, 593, 9956,
9971, 5944, 500, 5080, 1342, 6190, 4982, 4927, 6794, 241, 9350, 2695, 3813, 4963, 7008, 3366, 6174, 3128, 8162, 7659, 7753, 905, 8078, 509,
 7957, 1458, 5355, 9225, 7017, 6980, 8363, 7300, 6405, 6549, 1212, 1740, 1923, 8714, 8660, 9912, 4925, 5314, 5973, 1185, 9693, 659, 5760, 6
561, 1124, 6475, 3472, 5369, 3558, 506, 4804, 4028]


Sorted Array (Merge Sort): [241, 500, 506, 509, 524, 593, 659, 867, 905, 1124, 1130, 1185, 1212, 1341, 1342, 1400, 1458, 1574, 1740, 1923,
2528, 2695, 2723, 2742, 3128, 3366, 3472, 3558, 3813, 4025, 4028, 4109, 4156, 4385, 4426, 4443, 4494, 4522, 4523, 4552, 4804, 4925, 4927, 4
963, 4982, 5001, 5080, 5238, 5314, 5348, 5355, 5369, 5392, 5455, 5760, 5929, 5944, 5973, 6119, 6174, 6190, 6195, 6300, 6371, 6405, 6475, 64
95, 6549, 6561, 6577, 6624, 6649, 6794, 6974, 6980, 6995, 7008, 7017, 7155, 7265, 7300, 7332, 7445, 7449, 7659, 7753, 7957, 8078, 8162, 836
3, 8660, 8714, 8873, 9225, 9350, 9654, 9693, 9912, 9956, 9971]
Merge Sort Execution Time: 0.000180 seconds


Sorted Array (Quick Sort): [241, 500, 506, 509, 524, 593, 659, 867, 905, 1124, 1130, 1185, 1212, 1341, 1342, 1400, 1458, 1574, 1740, 1923,
2528, 2695, 2723, 2742, 3128, 3366, 3472, 3558, 3813, 4025, 4028, 4109, 4156, 4385, 4426, 4443, 4494, 4522, 4523, 4552, 4804, 4925, 4927, 4
963, 4982, 5001, 5080, 5238, 5314, 5348, 5355, 5369, 5392, 5455, 5760, 5929, 5944, 5973, 6119, 6174, 6190, 6195, 6300, 6371, 6405, 6475, 64
95, 6549, 6561, 6577, 6624, 6649, 6794, 6974, 6980, 6995, 7008, 7017, 7155, 7265, 7300, 7332, 7445, 7449, 7659, 7753, 7957, 8078, 8162, 836
3, 8660, 8714, 8873, 9225, 9350, 9654, 9693, 9912, 9956, 9971]
Quick Sort Execution Time: 0.000132 seconds
```

Comparison Table:



```
Input Size: 10
Merge Sort Execution Time: 0.000021 seconds
Quick Sort Execution Time: 0.000015 seconds
Input Size: 50
Merge Sort Execution Time: 0.000063 seconds
Quick Sort Execution Time: 0.000053 seconds
Input Size: 100
Merge Sort Execution Time: 0.000116 seconds
Quick Sort Execution Time: 0.000111 seconds
Input Size: 500
Merge Sort Execution Time: 0.000659 seconds
Quick Sort Execution Time: 0.000560 seconds
Input Size: 1000
Merge Sort Execution Time: 0.001385 seconds
Quick Sort Execution Time: 0.001163 seconds
```