Marwadi University Marwadi Chandarana Group NAAC A+	**Marwadi University** **Faculty of Engineering and Technology** **Department of Information and Communication Technology**	
Subject: DAA (01CT0512)	**AIM: Implementing Sorting Algorithm using D&C approach**	
Experiment No: 4	**Date:**	**Enrolment No: 92301733046**

Theory:

This method sorts an array by repeatedly dividing it into two halves until each subarray has one element, then merges the subarrays in a sorted manner to produce the final sorted array.

- Divide and Conquer
- Efficient for Large Data

Programming Language: Python

1) **Merge sort:**
   Code:

```
def merge(arr):
    if(len(arr)<=1):
        return arr
    mid=len(arr)//2
    left=merge(arr[:mid])
    # merge(left)
    right=merge(arr[mid:])
    # merge(right)
    return merge_sort(left,right)


def merge_sort(left, right):
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
```

Marwadi University Marwadi Chandarana Group	Marwadi University Faculty of Engineering and Technology Department of Information and Communication Technology	
Subject: DAA (01CT0512)	AIM: Implementing Sorting Algorithm using D&C approach	
Experiment No: 4	Date:	Enrolment No: 92301733046

```
    if left[i]<right[j]:
        result.append(left[i])
        i += 1
    else:
        result.append(right[j])
        j += 1
  result.extend(left[i:])
  result.extend(right[j:])
  return result

arr=[1,2,9,6,23]
print(merge(arr))
```

Output:

```
PS D:\DAA\Lab and Lecture Codes> python -u "d:\DAA\Lab and Lecture Codes\merge_sort_using_Divide_and_conquer_approach.py"
[1, 2, 6, 9, 23]
```

Space complexity: O(n)

Justification: During merging, new temporary arrays are created to store results, requiring space proportional to the size of the input, i.e., **O(n)**. Additionally, recursion uses **O(log n)** stack space, but this is negligible compared to O(n). Hence, the overall space complexity is **O(n)**

Time complexity:

Best case time complexity: O(n)

Justification: In merge sort, even if the array is already sorted, the algorithm does not take advantage of it. The array is still split into halves recursively until

each subarray has a single element, and then all elements are merged back. The merging step compares and processes every element at each level of recursion. Since there are **log n levels** and each level takes **O(n)** work

Worst case time complexity: O(nlogn)

Justification: If the array is in reverse order or completely unsorted, merge sort behaves the same way. It still divides the array into halves and performs full merging by comparing every element. The amount of work at each level does not increase beyond O(n), and the recursion depth remains O(log n). Thus, the worst-case time complexity is also **O(n log n)**.

2) Quick Sort:

Theory:
This method sorts an array by selecting a pivot element, partitioning the array into two subarrays (elements less than pivot and elements greater than pivot), and then recursively applying the same process to each subarray until the whole array is sorted.

- Divide and Conquer
- Efficient for Large Data

Code

```
import math
def partition(arr,low,high):
    pivot=arr[high]
    pIndex=low
    for j in range(low,high):
        if arr[j] <= pivot:
            arr[j], arr[pIndex]=arr[pIndex],arr[j]
            pIndex += 1
```

```
    arr[high], arr[pIndex]=arr[pIndex], arr[high]
    return pIndex

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

arr=[1,8,9,2,3]
QuickSort(arr,0,len(arr)-1)
print(arr)
```

Space complexity: O(n)

Justification: Partitioning only swaps elements within the original array.The only extra memory comes from recursive function calls stored in the call stack

Time complexity: O(nlogn)

Best case time complexity: O(nlogn)

Justification: The best case happens when the pivot divides the array into two equal (or nearly equal) halves at every step.

Worst case time complexity: O(nlogn)

Justification: The worst case occurs when the pivot chosen is always the smallest or largest element (for example, when the array is already sorted and we always pick the first element as pivot).

Marwadi University Marwadi Chandarana Group NAAC A+	**Marwadi University** **Faculty of Engineering and Technology** **Department of Information and Communication Technology**	
Subject: DAA (01CT0512)	**AIM: Implementing Sorting Algorithm using D&C approach**	
Experiment No: 4	**Date:**	**Enrolment No: 92301733046**