K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch:- B-2	**Roll No:-** 16010122151

Experiment No.03

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Implementation of Quick sort/Merge sort algorithm

Objective: To learn the divide and conquer strategy of solving the problems of different types

---

CO to be achieved:

CO 2 Describe various algorithm design strategies to solve different problems and analyze Complexity.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://en.wikipedia.org/wiki/Quicksort**
4. **https://www.cs.auckland.ac.nz/~jmor159/PLDS210/qsort.html**
5. **http://www.cs.rochester.edu/~gildea/csc282/slides/C07-quicksort.pdf**
6. **http://www.sorting-algorithms.com/quick-sort**
7. **http://www.cse.ust.hk/~dekai/271/notes/L01a/quickSort.pdf**
8. **http://en.wikipedia.org/wiki/Merge_sort**
9. **http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/mergeSort.htm**
10. **http://www.sorting-algorithms.com/merge-sort**
11. **http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Merge_sort.html**

Pre Lab/ Prior Concepts:
Data structures, various sorting techniques

Historical Profile:
Quicksort and merge sort are divide-and-conquer sorting algorithm in which division is dynamically carried out. They are one the most efficient sorting algorithms.

New Concepts to be learned:
Number of comparisons, Application of algorithmic design strategy to any problem, Classical problem solving vs Divide-and-Conquer problem solving.

Algorithm Recursive Quick Sort:

void quicksort(Integer A[ ], Integer left, Integer right)
//sorts A[left.. right] by using partition() to partition A[left.. right], and then //calling itself // twice to sort the two subarrays.
{ **IF** (left < right) then
{ q = partition(A, left, right);
quicksort(A, left, q–1);
quicksort(A, q+1, right);
}
}

Integer ***partition(integer*** **AT[], Integer** ***left*****, Integer** ***right*****)**
//This function rarranges *A*[*left*..*right*] and finds and returns an integer *q*, such that *A*[*left*], ..., //*A*[*q*–1] <~ *pivot*, *A*[*q*] = *pivot*, *A*[*q*+1], ..., *A*[*right*] > *pivot*, where *pivot* is the first element of //a[left…right], before partitioning.
{
pivot = A[left]; lo = left+1; hi = right;
WHILE (lo ≤ hi)
{ **WHILE** (A[hi] > pivot) hi = hi – 1;
WHILE (lo ≤ hi and A[lo] <~pivot) lo = lo + 1;
IF (lo ≤ hi) then swap(A[lo], A[hi]);
}
swap(pivot, A[hi]);
RETURN hi;
}

CODE:-

```
def quicksort(arr):
    if len(arr) <= 1:
        return arr

    pivot = arr[0]
    left = [x for x in arr[1:] if x < pivot]
    right = [x for x in arr[1:] if x >= pivot]

    return quicksort(left) + [pivot] + quicksort(right)

# Take input as a list separated by spaces
user_input = input("Enter the array elements :- ")
arr = [int(x) for x in user_input.split()]
sorted_arr = quicksort(arr)
```

```
print("Sorted array:", sorted_arr)
```

OUTPUT:-

```
Enter the array elements :-  5 8 2 0 6 4 8
Sorted array: [0, 2, 4, 5, 6, 8, 8]

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

The Time and space complexity of Quick Sort:

Quick Sort 16010122139

$T(n) = aT(n-1) + f(n)$

$a = 1$
$b = 1$
$f(n) = n$

$\therefore \Rightarrow O(f(n) \cdot n)$
$\Rightarrow O(n^2) \Rightarrow$ Time complexity

Space Complexity :- $O(1)$

09.02.24

Derivation of best case and worst-case time complexity (Quick Sort)Algorithm Merge Sort

MERGE-SORT (*A*, *p*, *r*)

// To sort the entire sequence A[1 .. n], make the initial call to the procedure MERGE-SORT (*A*, //1, *n*). Array *A* and indices *p*, *q*, *r* such that $p \le q \le r$ and sub array *A*[*p* .. *q*] is sorted and sub array //*A*[*q* + 1 .. *r*] is sorted. By restrictions on *p*, *q*, *r*, neither sub array is empty.

//OUTPUT: The two sub arrays are merged into a single sorted sub array in *A*[*p* .. *r*].

```
IF p < r                              // Check for base case
   THEN q = FLOOR [(p + r)/2]         // Divide step
         MERGE (A, p, q)              // Conquer step.
         MERGE (A, q + 1, r)          // Conquer step.
         MERGE (A, p, q, r)           // Conquer step.
```

```
MERGE (A, p, q, r)
{
    n1 ← q − p + 1
    n2 ← r − q

  Create arrays L[1 . . n1 + 1] and R[1 . . n2 + 1]
    FOR i ← 1 TO n1
        DO L[i] ← A[p + i − 1]
     FOR j ← 1 TO n2
        DO R[j] ← A[q + j ]
    L[n1 + 1] ← ∞
    R[n2 + 1] ← ∞
   i ← 1
   j ← 1
   FOR k ← p TO r
       DO IF L[i ] ≤ R[ j]
            THEN A[k] ← L[i]
                 i ← i + 1
            ELSE A[k] ← R[j]
                j ← j + 1

}
```

CODE:-

```
def merge_sort(arr):
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
```

```python
    left_half = arr[:mid]
    right_half = arr[mid:]
    return merge(merge_sort(left_half), merge_sort(right_half))

def merge(left, right):
    merged = []
    left_index = 0
    right_index = 0

    while left_index < len(left) and right_index < len(right):
        if left[left_index] <= right[right_index]:
            merged.append(left[left_index])
            left_index += 1
        else:
            merged.append(right[right_index])
            right_index += 1

    while left_index < len(left):
        merged.append(left[left_index])
        left_index += 1

    while right_index < len(right):
        merged.append(right[right_index])
        right_index += 1

    return merged

# Example usage:
arr = list(map(int, input("Enter a list of numbers:-  ").split()))
sorted_arr = merge_sort(arr)
print("Sorted array:", sorted_arr)
```

OUTPUT:-

Output

```
Enter a list of numbers :-  1 7 9 87 56 72 79
Sorted array: [1, 7, 9, 56, 72, 79, 87]

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

The space complexity of Merge sort:

Merge Sort

$T(n) = 2T\left(\frac{n}{2}\right) + n$

$a=2 \quad b=2 \quad k=1 \quad p=0$

$\log_b a = k \qquad \log_2 2 = 1$

$p > -1$

$O(n^k \log^{p+1} n)$

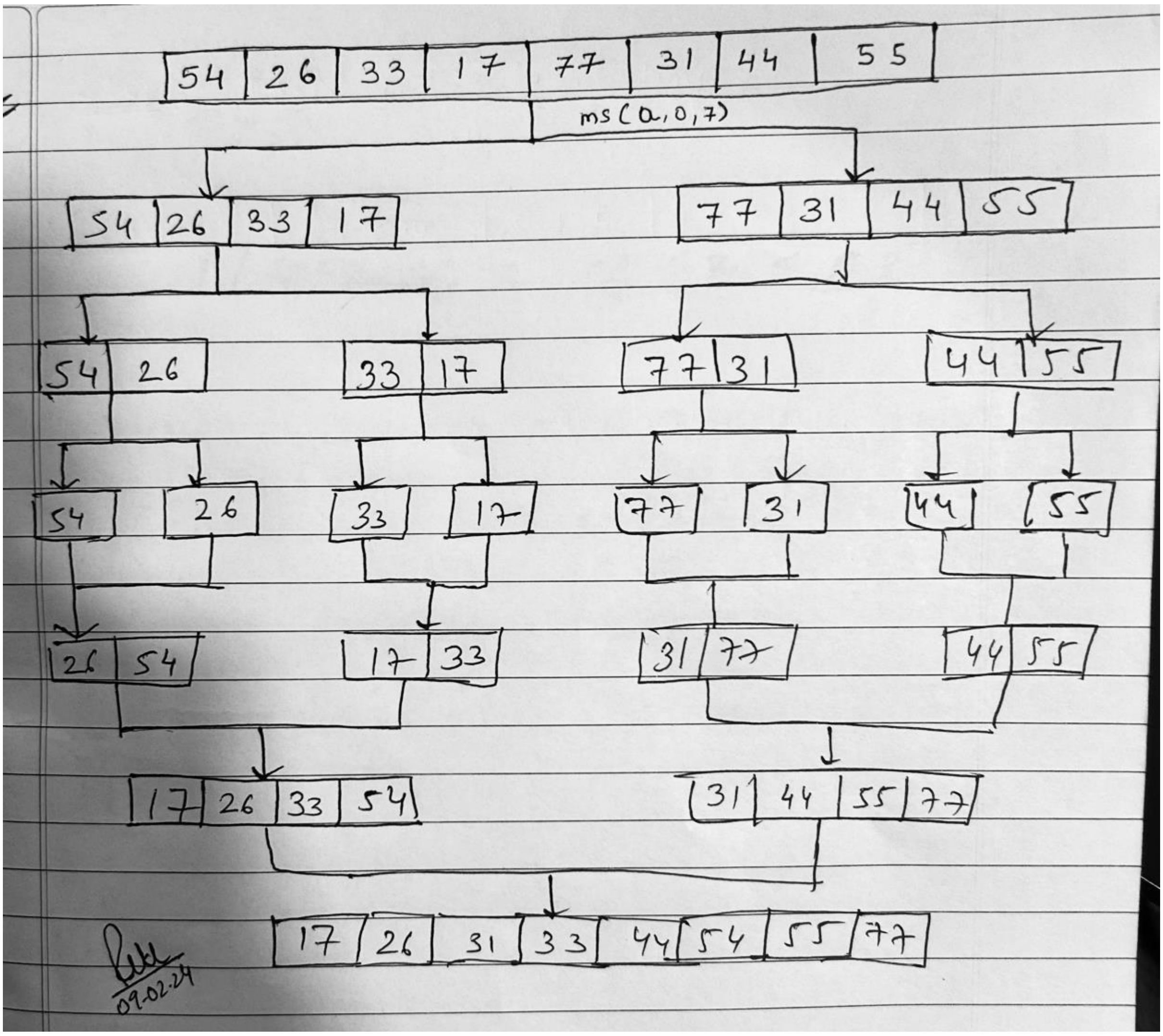
$O(n^1 \log^{0+1} n)$

$\Rightarrow O(n \log n)$

09.02.24

Space complexity = $O(n)$

Example for Merge tree for merge sort;



CONCLUSION:

From this experiment we have learnt the divide and conquer strategy of solving the problems using merge and quick sort and analysed their complexities.