



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

Batch: B2 Roll No.: 110

Experiment No. ____ 3 ____

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. Rajasekaran,” Fundamentals of computer algorithm”, University Press
2. T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein,” Introduction to algorithms”,2nd Edition ,MIT press/McGraw Hill,2001
3. <http://en.wikipedia.org/wiki/Quicksort>
4. <https://www.cs.auckland.ac.nz/~jmor159/PLDS210/qsrt.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 algorithms in which division is dynamically carried out. They are one the most efficient sorting algorithms.



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

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 rearranges 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;
}
```

The space complexity of Quick Sort:

O(1) as in place sorting is present, we are not utilising any external space



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

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

Algorithm Merge Sort

MERGE-SORT (A, p, r)

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

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

```
IF  $p < r$                                 // Check for base case
    THEN  $q = \text{FLOOR} [(p + r)/2]$         // 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)

```
{
     $n_1 \leftarrow q - p + 1$ 
     $n_2 \leftarrow r - q$ 
    Create arrays  $L[1 \dots n_1 + 1]$  and  $R[1 \dots n_2 + 1]$ 
    FOR  $i \leftarrow 1$  TO  $n_1$ 
        DO  $L[i] \leftarrow A[p + i - 1]$ 
    FOR  $j \leftarrow 1$  TO  $n_2$ 
        DO  $R[j] \leftarrow A[q + j]$ 
     $L[n_1 + 1] \leftarrow \infty$ 
     $R[n_2 + 1] \leftarrow \infty$ 
     $i \leftarrow 1$ 
     $j \leftarrow 1$ 
    FOR  $k \leftarrow p$  TO  $r$ 
        DO IF  $L[i] \leq R[j]$ 
            THEN  $A[k] \leftarrow L[i]$ 
                 $i \leftarrow i + 1$ 
            ELSE  $A[k] \leftarrow R[j]$ 
                 $j \leftarrow j + 1$ 
}
```

Code links <https://github.com/Aatmaj-Zephyr/AOA/tree/main/Practical/Expt%203>

(Code too long, hence attached file on GitHub)



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

The space complexity of Merge sort:

$$n \cdot \log(n)$$

$$O(n \log n)$$

Each time we are splitting array into two parts. Total length in each division is n . We split the array $\log n$ times, hence total space complexity is $n \log n$

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

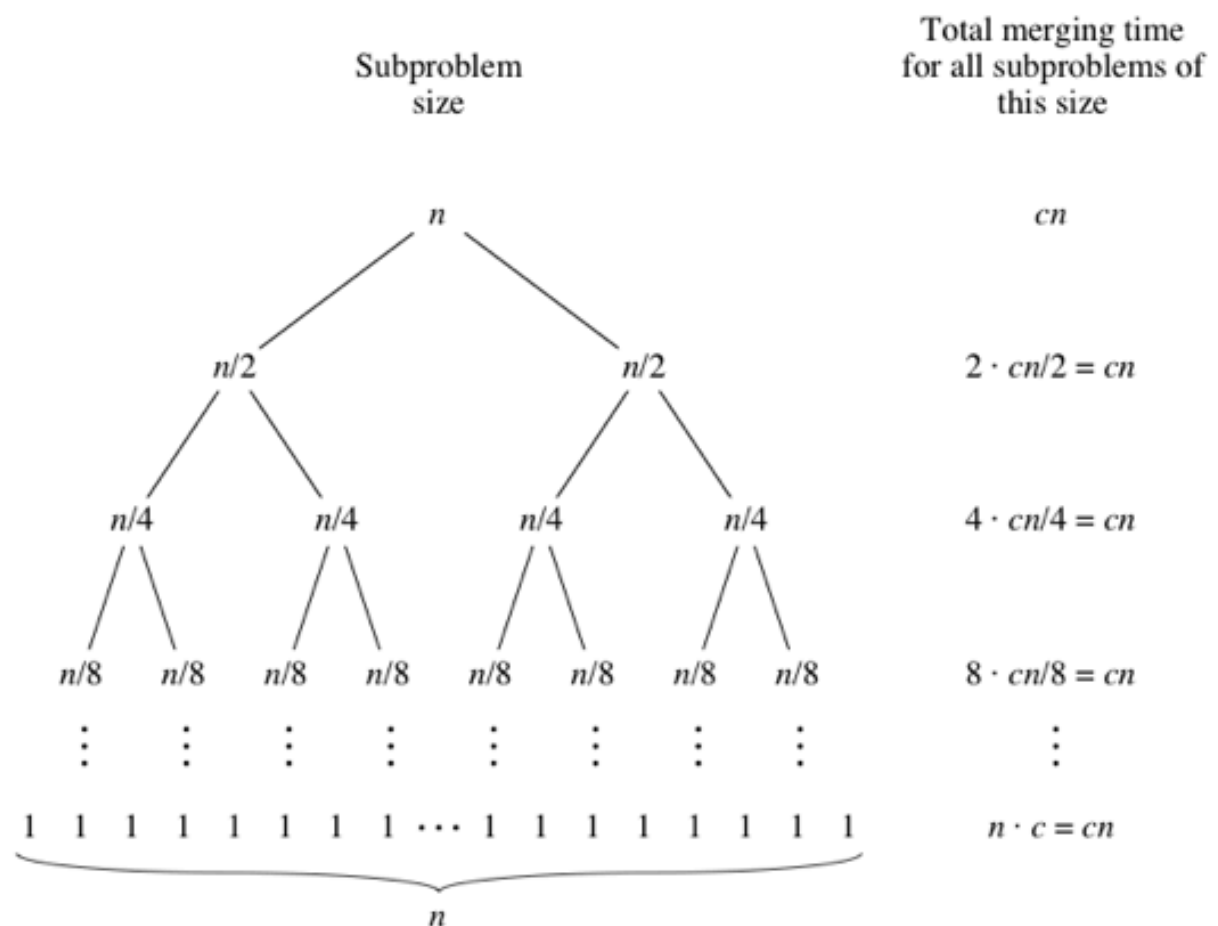
Best case time is when the pivot is exactly the middle element. Hence it is $n \log n$

Worst time complexity is when pivot is the first element, in that case $n \cdot n = O(n^2)$ is time complexity

Time complexity for merge sort

Time complexity for merge sort is always $n \log n$ as always the array is split into two parts and the total time for merging is n

Example for quicksort/Merge tree for merge sort:





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

CONCLUSION:

Thus we have implemented merge sort and quick sort. We understood how divide and conquer techniques work for sorting arrays. We also measured the time complexity required for the techniques. Merge sort in general has the time complexity of $n \log n$ while quick sort has $n \log n$ time complexity for best and average cases and n^2 time complexity in the worst case scenario.

Divide and conquer is a very effective technique for problem solving as it can be implemented recursively.