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

| | |
|---|---|
| **Batch: A1** | **Roll No.: 16010123012** |
| **Experiment No. 3** | |
| **Grade: AA / AB / BB / BC / CC / CD /DD** | |
| **Signature of the Staff In-charge with date** | |

## Title: Study, Implementation, and Comparative Analysis of Merge Sort and Quick Sort.

**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 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 $p \leq q \leq$ 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)
{
   $n_1 \leftarrow q - p + 1$
   $n_2 \leftarrow r - q$
   Create arrays L[1 . . $n_1$ + 1] and R[1 . . $n_2$ + 1]
   FOR $i \leftarrow 1$ TO $n_1$
      DO L[$i$] ← A[$p + i - 1$]
   FOR $j \leftarrow 1$ TO $n_2$
      DO R[$j$] ← A[$q + j$ ]
   L[$n_1$ + 1] ← ∞
   R[$n_2$ + 1] ← ∞
  $i \leftarrow 1$
  $j \leftarrow 1$
  FOR $k \leftarrow p$ TO $r$
     DO IF L[$i$ ] ≤ R[ $j$]
          THEN A[$k$] ← L[$i$]
               $i \leftarrow i + 1$
          ELSE A[k] ← R[j]
               $j \leftarrow j + 1$

}
```

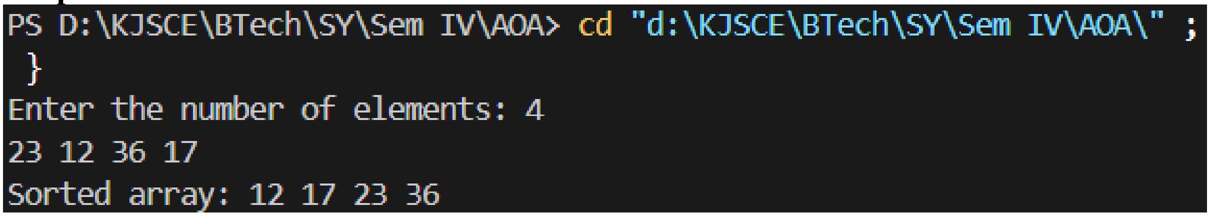
**The space complexity of Merge sort: O(n)**

**The time complexity of Merge sort: O(nlogn)**

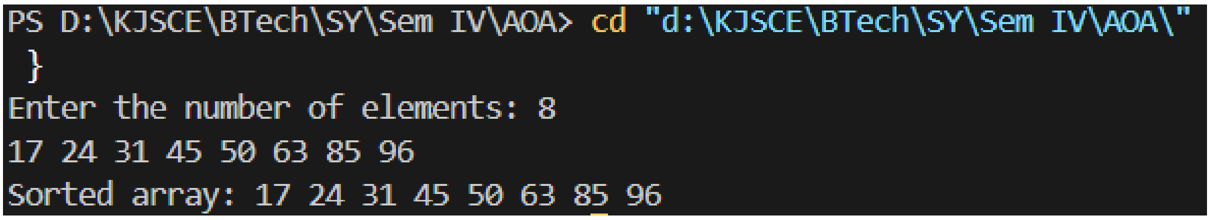
**Code:**

```
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

void merge(vector<int> &a, int l, int m, int r)
{
  int n1 = m - l + 1;
  int n2 = r - m;
  vector<int> L(n1), R(n2);
  for (int i = 0; i < n1; i++)
  {
    L[i] = a[l + i];
  }
  for (int j = 0; j < n2; j++)
  {
    R[j] = a[m + 1 + j];
  }
  int i = 0, j = 0, k = l;

  for (; i < n1 && j < n2; k++)
  {
    if (L[i] <= R[j])
    {
      a[k] = L[i];
      i++;
    }
    else
    {
      a[k] = R[j];
      j++;
    }
  }

  for (; i < n1; i++, k++)
  {
    a[k] = L[i];
  }
  for (; j < n2; j++, k++)
  {
    a[k] = R[j];
  }
}

void mergeSort(vector<int> &a, int l, int r)
{
  if (l >= r)
  {
   return;
  }
  int m = l + (r - l) / 2;
  mergeSort(a, l, m);
  mergeSort(a, m + 1, r);
  merge(a, l, m, r);
}

int main()
{
  int n;
  cout << "Enter the number of elements: ";
  cin >> n;
  vector<int> v(n);
  for (int i = 0; i < n; i++)
  {
   cin >> v[i];
  }
  mergeSort(v, 0, n - 1);
  cout << "Sorted array: ";
  for (int i = 0; i < n; i++)
  {
   cout << v[i] << " ";
  }
  cout << endl;
}
```

**Output:**

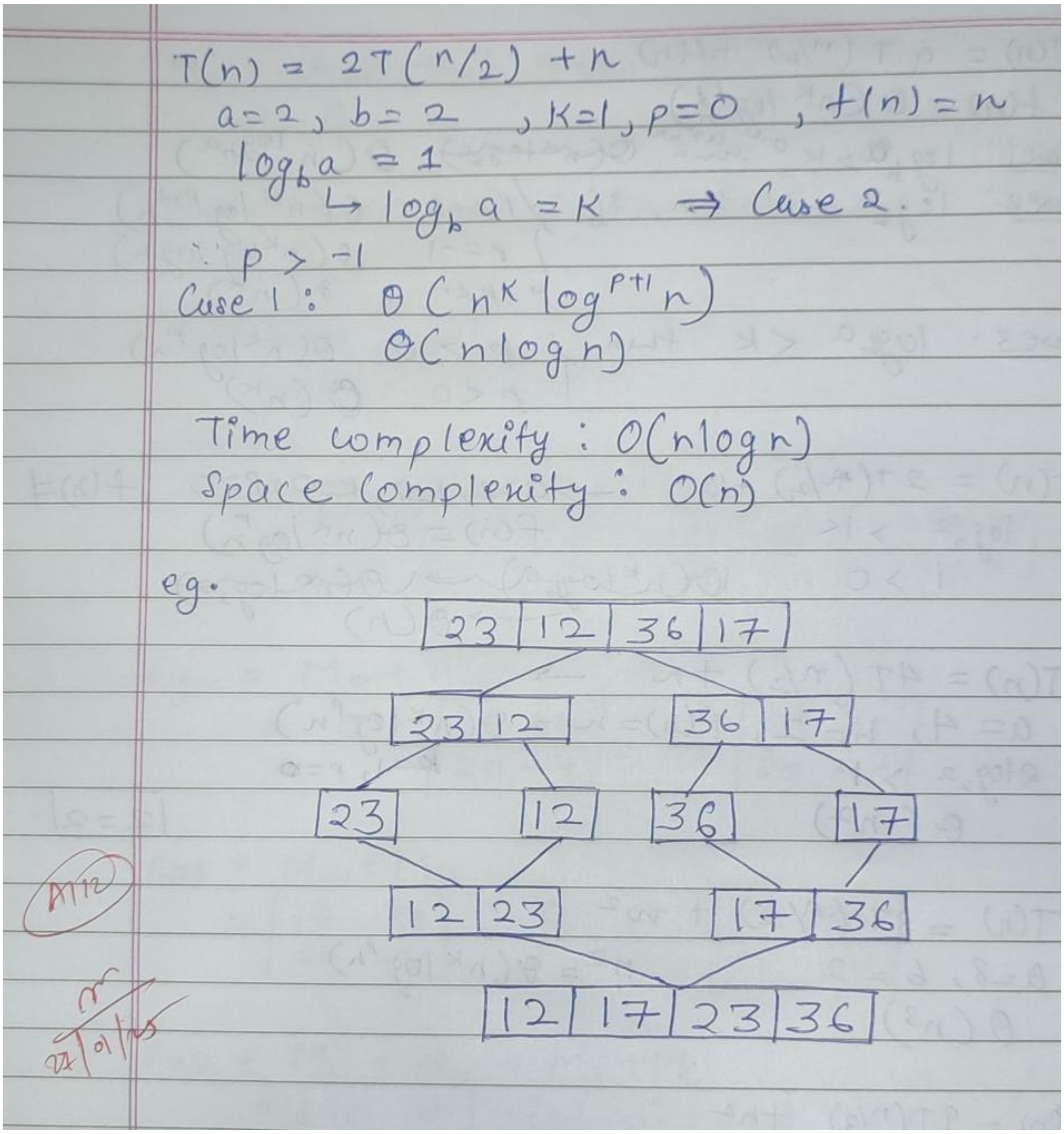


```
PS D:\KJSCE\BTech\SY\Sem IV\AOA> cd "d:\KJSCE\BTech\SY\Sem IV\AOA\" ;
 }
Enter the number of elements: 4
23 12 36 17
Sorted array: 12 17 23 36
```



```
PS D:\KJSCE\BTech\SY\Sem IV\AOA> cd "d:\KJSCE\BTech\SY\Sem IV\AOA\"
 }
Enter the number of elements: 8
17 24 31 45 50 63 85 96
Sorted array: 17 24 31 45 50 63 85 96
```

**Derivation of best case and worst-case time complexity (Merge Sort):**



$T(n) = 2T(n/2) + n$

$a=2,\ b=2,\ K=1,\ p=0,\ f(n)=n$

$\log_b a = 1$

$\hookrightarrow \log_b a = K \Rightarrow$ Case 2.

$\because p > -1$

Case 1 : $\Theta(n^K \log^{p+1} n)$

$\Theta(n \log n)$

Time complexity : O(nlogn)

Space complexity : O(n)

eg.

| 23 | 12 | 36 | 17 |

| 23 | 12 |  | 36 | 17 |

| 23 |  | 12 |  | 36 |  | 17 |

| 12 | 23 |  | 17 | 36 |

| 12 | 17 | 23 | 36 |

A/12

22/01/25

**CONCLUSION:**

I have successfully implemented the Merge Sort algorithm using the divide and conquer technique. Merge Sort consistently achieves a time complexity of O(nlogn) for the best, average and worst cases making it a highly efficient algorithm for sorting large datasets. The space complexity is O(n) due to the additional memory required for temporary arrays during the merge process. Its predictable performance and stability make Merge Sort an excellent choice for applications where consistent sorting time is critical, even for large inputs.