

**Experiment No.: 2**

**Title:** Divide and Conquer Strategy

# Batch: B2 Roll No.: 1914078 Experiment No.: 2

**Aim:** To implement and analyze time complexity of Quick-sort and Merge sort and compare both.

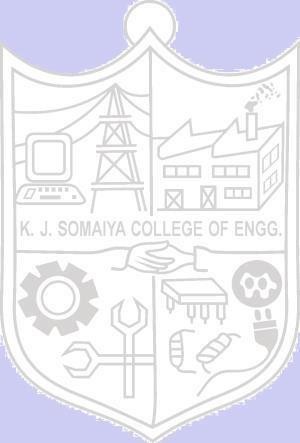
\_

**Algorithm:**

**Quick Sort:**

1. start
2. initialize n
3. accept n as number of elements from user
4. initialize array arr [n]
5. accept the unsorted array values
6. call the quick sort function and pass the parameters arr [], low=0, high=n-1
7. print sorted array arr []
8. stop

* quick sort function:

1. start
2. if low < high
   1. initialize variable pi and assign its integer value returned from the function partition
   2. call the quick sort function and pass the parameters arr [], low=low, high=pi-1
   3. call the quick sort function and pass the parameters arr [], low=pi+1, high=high
3. stop

* partition function:

1. start
2. initialize variable pivot and assign its integer value arr [high]
3. initialize variable i and assign its integer value (low -1)
4. for loop from j = low to (high-1)
   1. if arr [j] < pivot
      1. increase i by 1
      2. swap arr [i] & arr [j]
5. swap arr [i+1] & arr [high]
6. return (i + 1)
7. stop

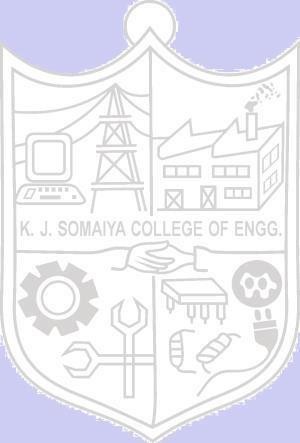
**Merge Sort:**

1. start
2. initialize n
3. accept n as number of elements from user
4. initialize array arr [n]
5. accept the unsorted array values
6. call the merge sort function and pass the parameters arr [], l=0, r=n-1
7. print sorted array arr []
8. stop

* merge sort function:

1. start
2. if l < r
   1. initialize variable m and assign its value l + (r-1)/2
   2. call the merge sort function and pass the parameters arr [], l = l, r = m
   3. call the merge sort function and pass the parameters arr [], l = m+1, r = r
   4. call the merge function and pass the parameters arr [], l = 1, r = r, m = m
3. stop

* merge function:

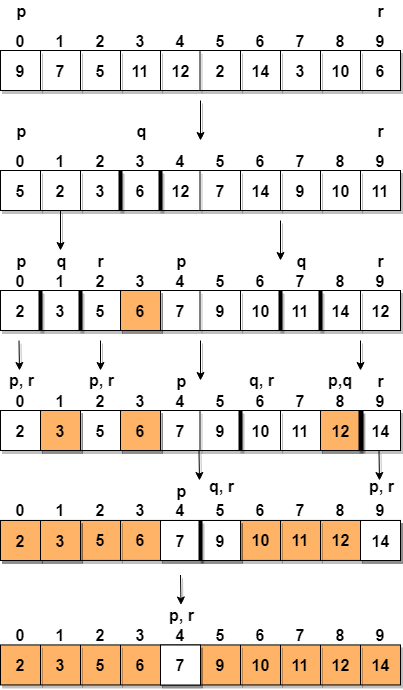
1. start
2. initialize variable int i, j, k, n1 = m - l + 1, n2 = r – m, L[n1], R[n2]
3. for loop from i = 0 to n1
   1. L[i] = arr [l + i]
4. for loop from j = 0 to n2
   1. R[j] = arr [m + l + j]
5. set variables i=0, j=0, k=l
6. while i < n1 & j < n2
   1. if L[i] <= R[j]
      1. arr [k] = L[i]
      2. increase i by 1
   2. else
      1. arr [k] = R[j]
      2. increase j by 1
   3. increase k by 1
7. while i < n1
   1. arr[k] = L[i];
   2. increase i by 1
   3. increase k by 1
8. while j < n2
   1. arr[k] = R[j];
   2. increase j by 1
   3. increase k by 1
9. stop

# Sample (Step by Step Execution):

## Quick Sort:

Let's consider an array with values {9, 7, 5, 11, 12, 2, 14, 3, 10, 6}

Below, we have a pictorial representation of how quick sort will sort the given array.



In step 1, we select the last element as the **pivot**, which is 6 in this case, and call for partitioning, hence re-arranging the array in such a way that 6 will be placed in its final position and to its left will be all the elements less than it and to its right, we will have all the elements greater than it.

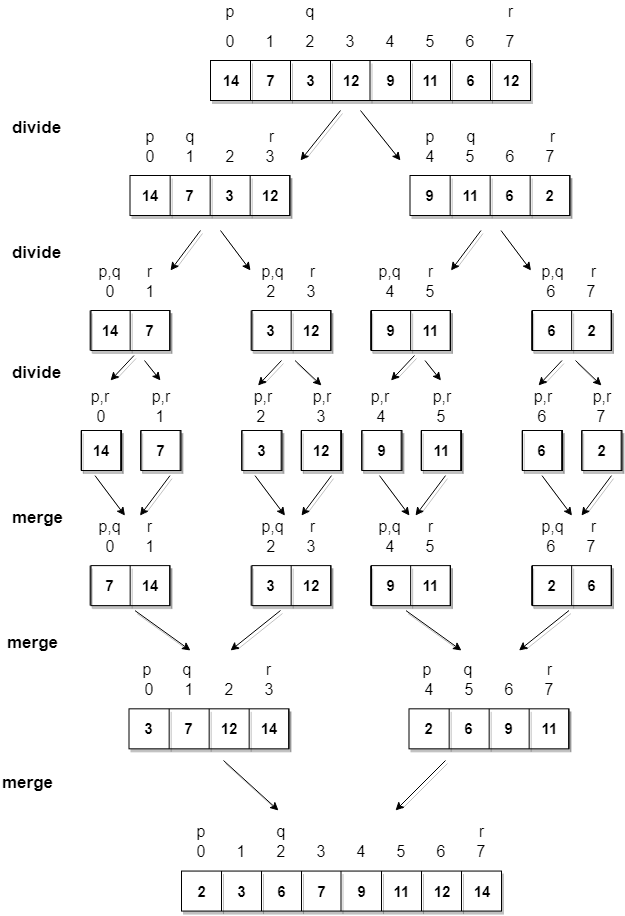
Then we pick the subarray on the left and the subarray on the right and select a **pivot** for them, in the above diagram, we chose 3 as pivot for the left subarray and 11 as pivot for the right subarray.

And we again call for partitioning.

## Merge Sort:

Let's consider an array with values {14, 7, 3, 12, 9, 11, 6, 12}

Below, we have a pictorial representation of how merge sort will sort the given array.



# Derivation of Analysis:

## Quick Sort:

### Worst Case Analysis:

Solving the Recurrence Relation:

T(N) = N + T(N-1)

T(N-1) = (N-1) + T(N-2)

T(N-2) = (N-2) + T(N-3)

...

T (3) = 3 + T (2)

T (2) = 2 + T (1)

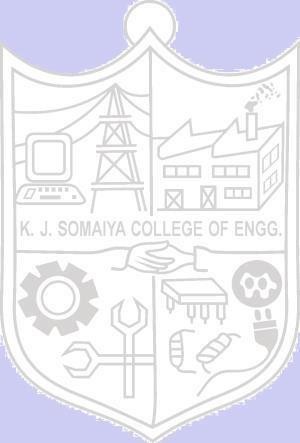
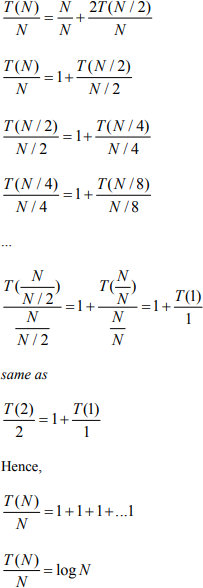
T (1) = 0

Hence,

T(N) = N + (N-1) + (N-2) ... + 3 + 2 ≈ N2 / 2 T(N) = O(N2)

### Best Case Analysis:

Solving the Recurrence Relation by dividing both the sides by N:



T(N) = O (N log N)

**Merge Sort:**

### Worst Case Analysis:

Solving the Recurrence Relation: T(N) = 2T(N/2) + c N

T(N/2) = 4T(N/4) + 2 c N T(N/4) = 8T(N/8) + 3 c N

…

T(N) = 2k T(N/2k) + k c N

T(N) = 2logN T(1) + c N logN T(N) = c’ N + c N log N T(N) = O (N log N)

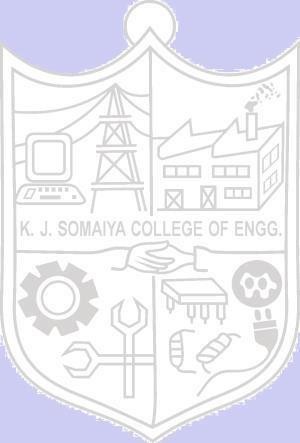
### Best Case Analysis:

Solving the Recurrence Relation: T(N) = 2T(N/2) + c N

T(N/2) = 4T(N/4) + 2 c N T(N/4) = 8T(N/8) + 3 c N

…

T(N) = 2k T(N/2k) + k c N

T(N) = 2logN T(1) + c N logN T(N) = c’ N + c N log N T(N) = O (N log N)

# Program:

## Quick Sort:

### Code:

#include <stdio.h>

void Quicksort(int[], int, int); int partition(int[], int, int); void swap(int \*, int \*);

void print\_output(int[], int);

int main()

{

int n;

printf("Enter size of array: "); scanf("%d", &n);

int arr[n];

printf("Enter values:\n");

for (int i = 0; i < n; i++)

{

scanf("%d", &arr[i]);

}

Quicksort(arr, 0, n - 1); printf("Sorted array: "); print\_output(arr, n);

return 0;

}

void Quicksort(int arr[], int low, int high)

{

if (low < high)

{

int pi = partition(arr, low, high); Quicksort(arr, low, pi - 1); Quicksort(arr, pi + 1, high);

}

}

int partition(int arr[], int low, int high)

{

int pivot = arr[high]; int i = (low - 1);

for (int j = low; j <= high - 1; j++)

{

if (arr[j] < pivot)

{

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

void swap(int \*a, int \*b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void print\_output(int arr[], int size)

{

for (int i = 0; i < size; i++) printf("%d ", arr[i]);

**Output:**

### Text Description automatically generated

### Merge sort:

### Code:

#include <stdio.h>

void mergeSort(int[], int, int); void merge(int[], int, int, int); void print\_output(int[], int);

void main()

{

int n;

printf("Enter size of array: "); scanf("%d", &n);

int arr[n];

printf("Enter values:\n");

for (int i = 0; i < n; i++)

{

scanf("%d", &arr[i]);

}

mergeSort(arr, 0, n - 1);

printf("Sorted array: ");

print\_output(arr, n);

}

void mergeSort(int arr[], int l, int r)

{

if (l < r)

{

int m = l + (r - l) / 2; mergeSort(arr, l, m); mergeSort(arr, m + 1, r); merge(arr, l, m, r);

}

}

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1; int n2 = r - m;

int L[n1], R[n2];

for (i = 0; i < n1; i++) L[i] = arr[l + i];

for (j = 0; j < n2; j++) R[j] = arr[m + 1 + j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i]; i++;

}

else

{

arr[k] = R[j]; j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i]; i++;

k++;

}

while (j < n2)

{

arr[k] = R[j]; j++;

k++;

}

}

void print\_output(int arr[], int size)

{

for (int i = 0; i < size; i++) printf("%d ", arr[i]);

}

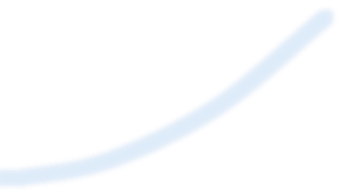
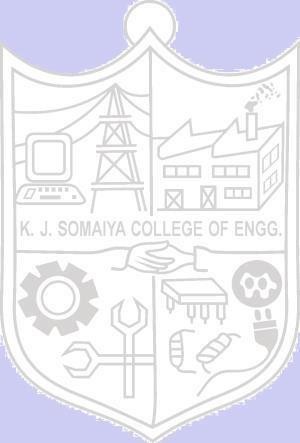
### Output:

### Text Description automatically generated

**Result:**

**Quick Sort:**

**Time Complexity of Quick sort: O(N2) [Worst Case] & O(N Log N) [Best Case] Worst Case Analysis:**



2500000

2000000

1500000

1000000

500000

0

0

200 400 600 800 1000 1200 1400 1600

-500000

**Input Size**

**GRAPH:**

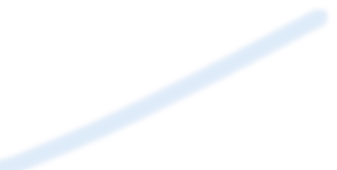
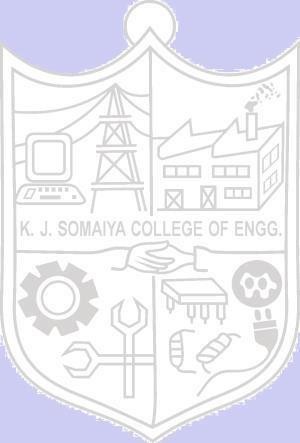
|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Input size** | **No of steps from Algorithm analysis** | **No of steps from Theoretical analysis** |
| 1 | 5 | 25 | 25 |
| 2 | 10 | 100 | 100 |
| 3 | 50 | 2500 | 2500 |
| 4 | 100 | 10000 | 10000 |
| 5 | 500 | 250000 | 250000 |
| 6 | 1000 | 1000000 | 1000000 |
| 7 | 1500 | 2250000 | 2250000 |

**No of Steps from Analysis**

**Best Case Analysis:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Input size** | **No of steps from Algorithm analysis** | **No of steps from Theoretical analysis** |
| 1 | 5 | 4 | 4 |
| 2 | 10 | 10 | 10 |
| 3 | 50 | 85 | 85 |
| 4 | 100 | 200 | 200 |
| 5 | 500 | 1350 | 1350 |
| 6 | 1000 | 3000 | 3000 |
| 7 | 1500 | 4765 | 4765 |

**GRAPH:**



6000

5000

4000

3000

2000

1000

0

0

200 400

600 800

**Input Size**

1000 1200 1400 1600

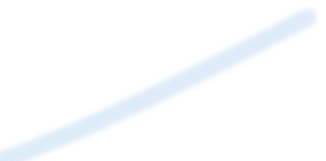
**Merge Sort:**

**Time Complexity of Merge sort: O(N Log N) Worst Case Analysis:**

**No of Steps from Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Input size** | **No of steps from Algorithm analysis** | **No of steps from Theoretical analysis** |
| 1 | 5 | 4 | 4 |
| 2 | 10 | 10 | 10 |
| 3 | 50 | 85 | 85 |
| 4 | 100 | 200 | 200 |
| 5 | 500 | 1350 | 1350 |
| 6 | 1000 | 3000 | 3000 |
| 7 | 1500 | 4765 | 4765 |

**GRAPH:**



6000

5000

4000

3000

2000

1000

0

0

200 400 600

800

**Input Size**

1000 1200 1400 1600

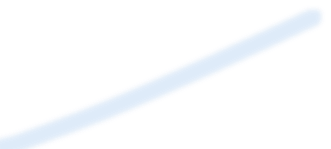
**No of Steps from Analysis**

Whiteboard

Description automatically generated with medium confidence**Best Case Analysis:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Input size** | **No of steps from Algorithm analysis** | **No of steps from Theoretical analysis** |
| 1 | 5 | 4 | 4 |
| 2 | 10 | 10 | 10 |
| 3 | 50 | 85 | 85 |
| 4 | 100 | 200 | 200 |
| 5 | 500 | 1350 | 1350 |
| 6 | 1000 | 3000 | 3000 |
| 7 | 1500 | 4765 | 4765 |

**GRAPH:**



6000

5000

4000

3000

2000

1000

0

0

200 400 600

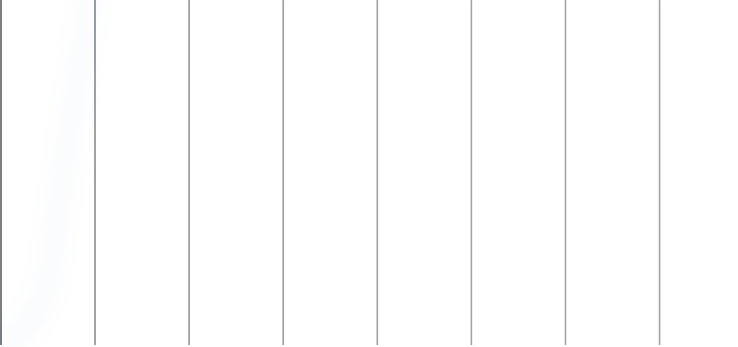
800

**Input Size**

1000 1200 1400 1600

**No of Steps from Analysis**

**Graph Comparison of Quick and Merge sort time analysis for all cases:**



Quick Sort Worst Case

Quick Sort Best Case

Merge Sort Worst Case Merge Sort Best Case

50000

45000

40000

35000

30000

25000

20000

15000

10000

5000

0

0

200

400

600

800 1000

**Input Size**

1200 1400 1600

**No of Steps from Analysis**

Whiteboard

Description automatically generated with medium confidenceAs we can observe the comparison of the graphs of Quick Sort and Merge Sort where the time complexity of worst case of quick sort is O(N2) and for best case is O(N Log N) and for merge sort in both the cases it is O(N Log N) we can conclude that Merge sort is overall a more stable sorting algorithm when compared to quick sort.

# Outcome:

Implement Divide and Conquer algorithms with recurrence.

# Conclusion:

We learnt Quick Sort and Merge Sort and analyzed them. These algorithms are based on Divide and Conquer strategy and were studied by writing a working program studying the algorithm using step by step examples.

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

**Signature of faculty in-charge with date**

**References:**

1. Richard E. Neapolitan, " Foundation of Algorithms ", 5th Edition 2016, Jones & Bartlett Students Edition

T.H. Coreman ,C.E. Leiserson,R.L. Rivest, and C. Stein, " Introduction to algorithms", 3rd Edition 2009, Prentice Hall India Publication.