



**BHARATIYA VIDYA BHAVAN'S**  
**SARDAR PATEL INSTITUTE OF TECHNOLOGY**  
MUNSHI NAGAR, ANDHERI (WEST), MUMBAI - 400 058.  
(Autonomous College Affiliated to University of Mumbai)  
**MASTER OF COMPUTER APPLICATIONS**

**Class : F.Y.MCA    Semester : II    Academic Year : 2024-25**

**Course Name : Design and Analysis of Algorithm MC507**

**Subject Incharge : Prof.Nikhita Mangaonkar**

**UCID: 2024510001    BATCH: A                      NAME: Atharva Vasant Angre**

**EXPERIMENT NO:    01**

**EXPERIMENT TITLE:** To implement Divide and conquer method

1.1 To Study different Time and space complexity tools

1.2 To Implement Quick sort algorithm and determine the Time and space complexity

1.3 To Implement Merge sort algorithm and determine the Time and space complexity

**Objective:**

1. To study various tools for time and space complexity.
2. To understand Big O,omega and Theta notations.
3. To implement the divide and conquer method and calculate the time and space complexity.



**BHARATIYA VIDYA BHAVAN'S**  
**SARDAR PATEL INSTITUTE OF TECHNOLOGY**  
MUNSHI NAGAR, ANDHERI (WEST), MUMBAI - 400 058.  
(Autonomous College Affiliated to University of Mumbai)  
**MASTER OF COMPUTER APPLICATIONS**

**Class : F.Y.MCA    Semester : II    Academic Year : 2024-25**

**Course Name : Design and Analysis of Algorithm MC507**

**Subject Incharge : Prof.Nikhita Mangaonkar**

**Program code: -**

**QuickSort**

```
1 import java.util.Arrays;
2 import java.util.Scanner;
3
4 public class QuickSort {
5     public static void main(String[] args) {
6         Scanner sc = new Scanner(System.in);
7
8         System.out.print("Enter the size of the array:");
9         int n = sc.nextInt();
10
11         int[] arr = new int[n];
12
13         System.out.print("Enter the data:");
14         for(int i = 0; i < n; i++)
15             System.out.print("Enter the data for index " + i + " : ");
16         arr[i] = sc.nextInt();
17     }
18
19     int low = 0;
20     int high = arr.length-1;
21     quickSort(arr, low, high);
22     System.out.println(Arrays.toString(arr));
23
24     static void quickSort(int[] arr, int low, int high) {
25         if (low < high)
26         {
27             int pivotIndex = partition(arr, low, high);
28
29             quickSort(arr, low, pivotIndex - 1);
30             quickSort(arr, pivotIndex + 1, high);
31         }
32     }
33
34     static int partition(int[] arr, int low, int high) {
35         int pivot = low;
36         int i = low + 1;
37         int j = high;
38
39         while (i <= j)
40         {
41             while (i <= j && arr[i] <= arr[pivot])
42                 i++;
43
44             while (j >= i && arr[j] > arr[pivot])
45                 j--;
46
47             if (i < j)
48             {
49                 int temp = arr[i];
50                 arr[i] = arr[j];
51                 arr[j] = temp;
52             }
53
54             int temp = arr[i];
55             arr[i] = arr[pivot];
56             arr[pivot] = temp;
57
58             return i;
59         }
60     }
61 }
```



**BHARATIYA VIDYA BHAVAN'S**  
**SARDAR PATEL INSTITUTE OF TECHNOLOGY**  
MUNSHI NAGAR, ANDHERI (WEST), MUMBAI - 400 058.  
(Autonomous College Affiliated to University of Mumbai)  
**MASTER OF COMPUTER APPLICATIONS**

**Class : F.Y.MCA    Semester : II    Academic Year : 2024-25**

**Course Name : Design and Analysis of Algorithm MC507**

**Subject Incharge : Prof.Nikhita Mangaonkar**

**MergeSort**

```
1 // MergeSort.java
2
3 import java.util.*;
4
5 public class MergeSort {
6     // MergeSort method
7     static void mergeSort(int arr[], int low, int mid, int high) {
8         // Base case
9         if (low < mid) {
10             // Recursion
11             mergeSort(arr, low, (low + mid) / 2, mid);
12             mergeSort(arr, mid, (mid + high) / 2, high);
13             // Merge
14             merge(arr, low, mid, high);
15         }
16     }
17
18     // Merge method
19     static void merge(int arr[], int low, int mid, int high) {
20         // Create temporary array
21         int temp[] = new int[high - low + 1];
22         // Merge left and right array
23         int i = low, j = mid + 1, k = low;
24         while (i <= mid && j <= high) {
25             if (arr[i] < arr[j]) {
26                 temp[k++] = arr[i++];
27             } else {
28                 temp[k++] = arr[j++];
29             }
30         }
31         // Copy remaining elements
32         while (i <= mid) {
33             temp[k++] = arr[i++];
34         }
35         while (j <= high) {
36             temp[k++] = arr[j++];
37         }
38         // Copy back to original array
39         for (i = low; i <= high; i++) {
40             arr[i] = temp[i - low];
41         }
42     }
43 }
```

```
1 // MergeSort.java
2
3 import java.util.*;
4
5 public class MergeSort {
6     // MergeSort method
7     static void mergeSort(int arr[], int low, int mid, int high) {
8         // Base case
9         if (low < mid) {
10             // Recursion
11             mergeSort(arr, low, (low + mid) / 2, mid);
12             mergeSort(arr, mid, (mid + high) / 2, high);
13             // Merge
14             merge(arr, low, mid, high);
15         }
16     }
17
18     // Merge method
19     static void merge(int arr[], int low, int mid, int high) {
20         // Create temporary array
21         int temp[] = new int[high - low + 1];
22         // Merge left and right array
23         int i = low, j = mid + 1, k = low;
24         while (i <= mid && j <= high) {
25             if (arr[i] < arr[j]) {
26                 temp[k++] = arr[i++];
27             } else {
28                 temp[k++] = arr[j++];
29             }
30         }
31         // Copy remaining elements
32         while (i <= mid) {
33             temp[k++] = arr[i++];
34         }
35         while (j <= high) {
36             temp[k++] = arr[j++];
37         }
38         // Copy back to original array
39         for (i = low; i <= high; i++) {
40             arr[i] = temp[i - low];
41         }
42     }
43 }
```

```
1 // MergeSort.java
2
3 import java.util.*;
4
5 public class MergeSort {
6     // MergeSort method
7     static void mergeSort(int arr[], int low, int mid, int high) {
8         // Base case
9         if (low < mid) {
10             // Recursion
11             mergeSort(arr, low, (low + mid) / 2, mid);
12             mergeSort(arr, mid, (mid + high) / 2, high);
13             // Merge
14             merge(arr, low, mid, high);
15         }
16     }
17
18     // Merge method
19     static void merge(int arr[], int low, int mid, int high) {
20         // Create temporary array
21         int temp[] = new int[high - low + 1];
22         // Merge left and right array
23         int i = low, j = mid + 1, k = low;
24         while (i <= mid && j <= high) {
25             if (arr[i] < arr[j]) {
26                 temp[k++] = arr[i++];
27             } else {
28                 temp[k++] = arr[j++];
29             }
30         }
31         // Copy remaining elements
32         while (i <= mid) {
33             temp[k++] = arr[i++];
34         }
35         while (j <= high) {
36             temp[k++] = arr[j++];
37         }
38         // Copy back to original array
39         for (i = low; i <= high; i++) {
40             arr[i] = temp[i - low];
41         }
42     }
43 }
```



**BHARATIYA VIDYA BHAVAN'S**  
**SARDAR PATEL INSTITUTE OF TECHNOLOGY**  
MUNSHI NAGAR, ANDHERI (WEST), MUMBAI - 400 058.  
(Autonomous College Affiliated to University of Mumbai)  
**MASTER OF COMPUTER APPLICATIONS**

**Class : F.Y.MCA    Semester : II    Academic Year : 2024-25**

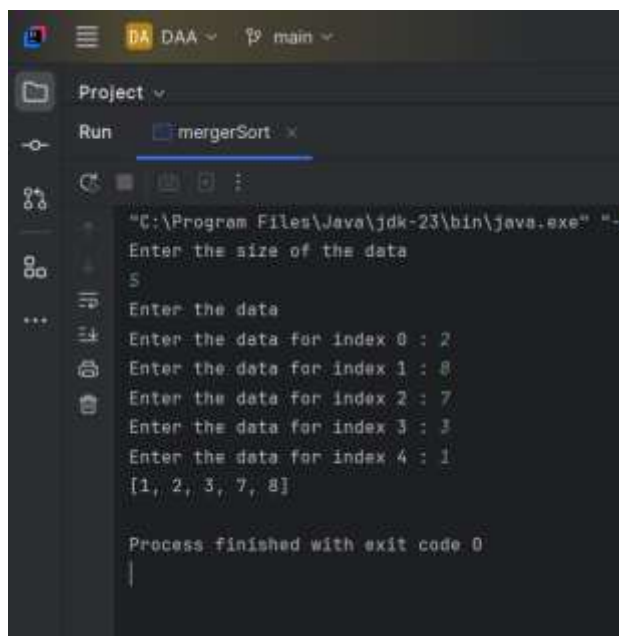
**Course Name : Design and Analysis of Algorithm MC507**

**Subject Incharge : Prof.Nikhita Mangaonkar**

**Output:**

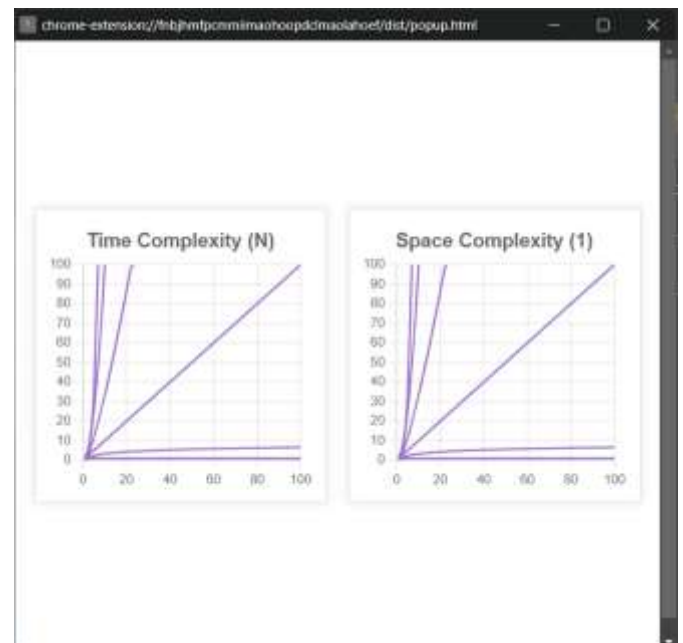
After executing the Merge Sort and Quick Sort implementations with different input cases, the results display a sorted array. Below is the time complexity analysis for each sorting algorithm:

**Merge Sort:**



```
"C:\Program Files\Java\jdk-23\bin\java.exe" "-Xms1024m" "-Xmx1024m"
Enter the size of the data
5
Enter the data
Enter the data for index 0 : 2
Enter the data for index 1 : 8
Enter the data for index 2 : 7
Enter the data for index 3 : 3
Enter the data for index 4 : 1
[1, 2, 3, 7, 8]

Process finished with exit code 0
```



- Best Case Complexity:  $O(n \log n)$  (Occurs when the array is already sorted)
- Worst Case Complexity:  $O(n \log n)$  (Occurs for any random order of elements since Merge Sort always divides the array into two halves)
- Average Case Complexity:  $O(n \log n)$

Time and Space complexity of merge sort using the extension Time and Space complexity.



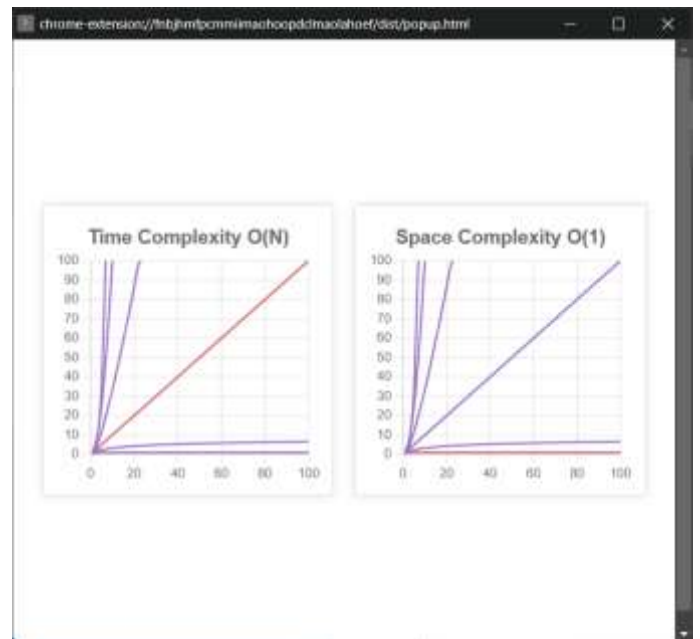
**Class : F.Y.MCA    Semester : II    Academic Year : 2024-25**

**Course Name : Design and Analysis of Algorithm MC507**

**Subject Incharge : Prof.Nikhita Mangaonkar**

**Quick Sort:**

```
"C:\Program Files\Java\jdk-23\bin\java.  
Enter the size of the data  
5  
Enter the data  
Enter the data for index 0 : 9  
Enter the data for index 1 : 1  
Enter the data for index 2 : 7  
Enter the data for index 3 : 3  
Enter the data for index 4 : 8  
[1, 3, 7, 8, 9]  
  
Process finished with exit code 0
```



- Best Case Complexity:  $O(n \log n)$  (Occurs when the pivot divides the array into nearly equal halves)
- Worst Case Complexity:  $O(n^2)$  (Occurs when the smallest or largest element is always chosen as the pivot, resulting in an unbalanced partition)
- Average Case Complexity:  $O(n \log n)$

Time and Space complexity of merge sort using the extension Time and Space complexity.





**Class : F.Y.MCA    Semester : II    Academic Year : 2024-25**

**Course Name : Design and Analysis of Algorithm MC507**

**Subject Incharge : Prof.Nikhita Mangaonkar**

**Result:**

**Merge Sort:**

- Best Case: Even when the array is sorted, Merge Sort still recursively divides the array and performs merge operations, leading to  $O(n \log n)$ .
- Worst Case: It performs the same number of comparisons and merge steps regardless of the order of input, so the worst case remains  $O(n \log n)$ .
- Average Case: The division and merging operations make the complexity  $O(n \log n)$  in all cases.

**Quick Sort:**

- Best Case: Achieved when the pivot splits the array into two nearly equal halves, making the complexity  $O(n \log n)$ .
- Worst Case: If the pivot is always the smallest or largest element, it results in  $O(n^2)$  due to highly unbalanced partitions.
- Average Case: The expected time complexity remains  $O(n \log n)$  assuming the pivot is chosen randomly or efficiently.

**Conclusion:**

This experiment helped in understanding:

1. Divide and Conquer Approach – Both Merge Sort and Quick Sort divide the problem into smaller subproblems, solve them recursively, and combine the results.
2. Time Complexity Notations – Best, worst, and average case complexities were analyzed using Big (O) notations.
3. Comparison of Sorting Algorithms – Merge Sort provides stable sorting and consistent performance, while Quick Sort is faster in practice but suffers from poor worst-case scenarios if not optimized.

Applications of These Sorting Techniques:

1. Merge Sort Applications:
  - Sorting linked lists (as merging is easy in linked lists)
  - External sorting (large datasets that don't fit into memory)
  - Data processing applications (e.g., log file analysis)
2. Quick Sort Applications:
  - Database sorting (due to fast in-memory sorting)
  - Used in standard libraries like Java's `Arrays.sort()`