LABORATORY REPORT

# Algorithm Laboratory (CS-39001)

## B.Tech Program in ECSc

Submitted By

**Name:-** SANNIDHI DEB

**Roll No:** 2330044

**Kalinga Institute of Industrial Technology**

**(Deemed to be University) Bhubaneswar, India**

Autumn, 2025

# Table of Contents

| | |
|---|---|
| **Experiment Number** | 3.1 |
| **Experiment Title** | Aim of the program: Write a menu program to sort list of array elements using **Merge Sort** technique and calculate the execution time only to sort the elements. Count the number of comparisons. |
| **Date of Experiment** | 14/08/2025 |
| **Date of Submission** | 20/08/2025 |

## 1. Algorithm:-

Sannidhi Deb, 2330044

```
3.1. Function MergeSort (arr, left, right).
  If left < right then
     mid ← (left + right) / 2
     call MergeSort (arr, left, mid).
     call MergeSort (arr, mid+1, right).
     call MergeSort (arr, left, mid, right)
End Function.

Function Merge (arr, left, mid, right).
   Create arrays L and R.
     n1 ← mid - left + 1
     n2 ← right - mid.
   For i ← 0 to n1-1 do
       L[i] ← arr[left + i].
   For j ← 0 to n2-1 do
       R[j] ← arr [mid + 1 + j].
   i ← 0, j ← 0, k ← left.
  while i < n1 and j < n2 do
     if L[i] ≤ R[j] then.
        arr[k] ← L[i]
          i ← i + 1.
     else
        arr[K] ← R[j].
          j ← j + 1
       k ← k + 1
    while i < n1 do
       arr[k] ← L[i]
       i ← i + 1.
       k ← k + 1
    while j < n2 do
       arr [k] ← R[j]
        j ← j + 1
        k ← k + 1
End Function
```

## 2. Code:-

```
#include <stdio.h>
#define MAX 500

void merge(int arr[], int l, int m, int r) {
   int L[MAX], R[MAX];
   int n1 = m - l + 1;
   int n2 = r - m;

   for (int i = 0; i < n1; i++)
      L[i] = arr[l + i];
   for (int j = 0; j < n2; j++)
      R[j] = arr[m + 1 + j];

   int i = 0, j = 0, k = l;

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

   while (i < n1)
      arr[k++] = L[i++];
   while (j < n2)
      arr[k++] = R[j++];
}

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);
   }
}
```

```
int main(void) {
  int arr[MAX], n;
  printf("\nSannidhi Deb\n 2330044\n\n");
  printf("Enter number of elements: \n");
  scanf("%d", &n);

  printf("\nEnter %d elements:\n", n);
  for (int i = 0; i < n; i++)
    scanf("%d", &arr[i]);

  mergeSort(arr, 0, n - 1);

  printf("\nSorted array:\n");
  for (int i = 0; i < n; i++)
    printf("%d ", arr[i]);
  printf("\n");

  return 0;
}
```

## 3.Results/Output:- Entire Screen Shot including Date & Time:-

Best Case :-

```
C:\Users\debsa\OneDrive\Desktop\AL_Lab_044>exp3_1

Sannidhi Deb
 2330044

Enter number of elements:
10

Enter 10 elements:
12 19 24 28 29 45 48 62 67 80

Sorted array:
12 19 24 28 29 45 48 62 67 80

C:\Users\debsa\OneDrive\Desktop\AL_Lab_044>
```

Average Case :-

```
C:\Users\debsa\OneDrive\Desktop\AL_Lab_044>gcc exp3_1.c -o exp3_1.exe

C:\Users\debsa\OneDrive\Desktop\AL_Lab_044>exp3_1

Sannidhi Deb
 2330044

Enter number of elements:
10

Enter 10 elements:
23 65 12 56 98 13 15 76 24 67

Sorted array:
12 13 15 23 24 56 65 67 76 98
```

Worst Case :-

```
C:\Users\debsa\OneDrive\Desktop\AL_Lab_044>gcc exp3_1.c -o exp3_1.exe

C:\Users\debsa\OneDrive\Desktop\AL_Lab_044>exp3_1

Sannidhi Deb
 2330044

Enter number of elements:
10

Enter 10 elements:
98 76 54 52 41 33 27 22 19 8

Sorted array:
8 19 22 27 33 41 52 54 76 98
```

## 4. Remarks:-

1. What type of algorithm is used?

Sannidhi Deb, 2330044

1. The Merge Sort algorithm uses divide and conquer approach. It divides the problem into subfunctions and solves them independently.
Merge Sort divides the array into halves, sorts them, and merges.

2. Analyze the complexity of your algorithm.

Sannidhi Deb, 2330044.

2. Merge Sort has $O(n \log n)$ as all three possible cases, i.e,
Best case: $O(n \log n)$
Average case: $O(n \log n)$
worst case: $O(n \log n)$

3. Any other observations?

Sannidhi Deb, 2330044.

3. Merge Sort always splits the array in half, regardless of the data/input. Temporary arrays are used during merging. It works well even for worst-case inputs and always has time complexity of $O(n \log n)$.

Sannidhi Deb
(2330044)

Signature of the FIC

Sannidhi Deb

(Name of the FIC)

| Experiment Number | 3.2 |
| --- | --- |
| Experiment Title | Aim of the program: Write a menu driven program to sort a list of elements in ascending order using **Quick Sort** technique. After sorting display the content of the output. Based on the partitioning position for each recursive call, conclude the input scenario is either best-case partitioning or worst-case partitioning. |
| Date of Experiment | 14/08/2025 |
| Date of Submission | 20/08/2025 |

## 1. Algorithm:-

Sannidhi Deb, 2330044

```
3.2. Function QuickSort (arr, low, high)
        if low < high then
           pivotIndex ← Partition(arr, low, high).
           QuickSort (arr, low, pivotIndex-1).
           QuickSort (arr, pivotIndex+1, high).
     end if
     end Function
     Function Partition (arr, low, high)
        pivot ← arr[high]
        i ← low-1.
        for j ← low to high-1 do
           if arr[j] ≤ pivot then
              i ← i+1.
              swap arr[i] with arr[j].
           end if
        end for
        swap arr[i+1] with arr[high]
        return i+1
     end Function.
```

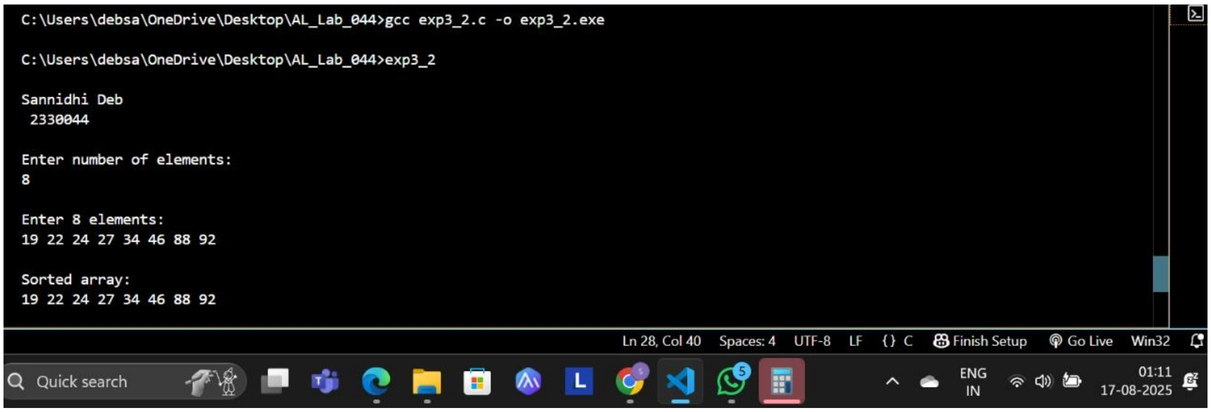
## 2. Code:-

```
#include <stdio.h>
#define MAX 500

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

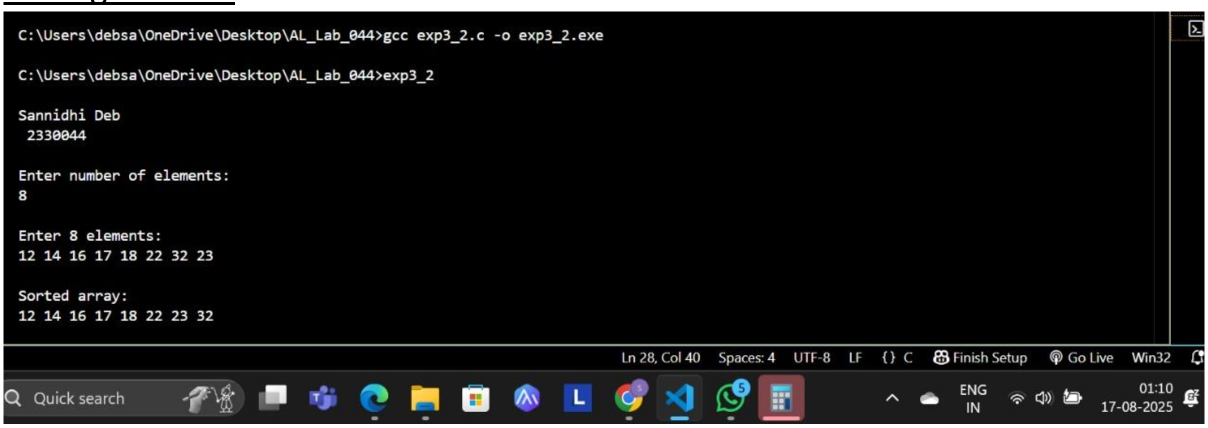
```c
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;

    for (int j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

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 main(void) {
    int arr[MAX], n;
    printf("\nSannidhi Deb\n 2330044\n\n");
    printf("Enter number of elements: \n");
    scanf("%d", &n);

    printf("\nEnter %d elements:\n", n);
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    quickSort(arr, 0, n - 1);

    printf("\nSorted array:\n");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");

    return 0;
}
```

## 3. Results/Output:- Entire Screen Shot including Date & Time:-

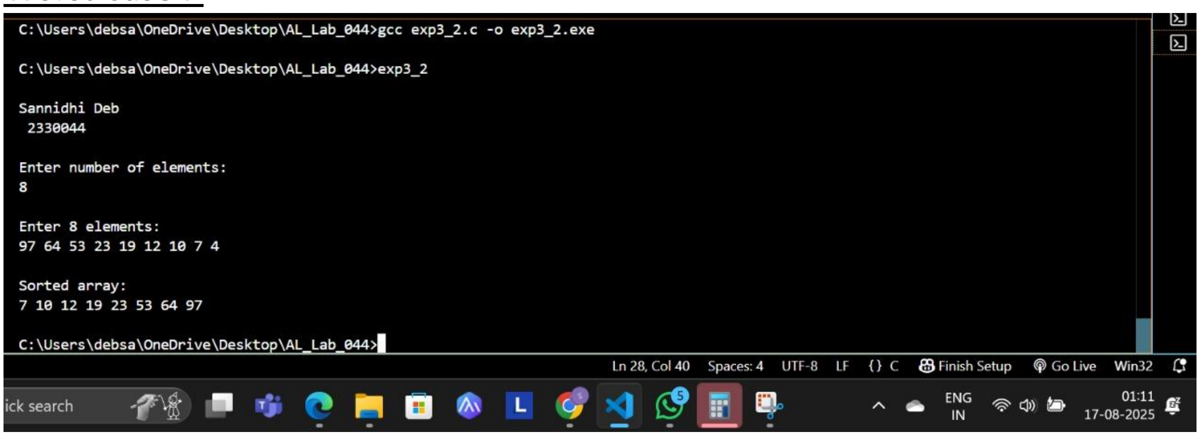
Best Case :-




Average Case :-




Worst Case :-



**4. Remarks:-**

1. What type of algorithm is used?

Sannidhi Deb, 2330044

1. The Quick Sort algorithm uses divide and conquer approach. It uses partition, the array around a pivot and sorts the partition.

2. Analyze the complexity of your algorithm.

Sannidhi Deb, 2330044.

2. The three cases of Quick sort complexity are

Best case : $O(n\log n)$

Average case : $O(n\log n)$

Worst case : $O(n^2)$

3.Any other observations?

Sannidhi Deb, 2330044.

3. Quick sort has a ~~good~~ pivot that leads to balanced partitions, and does not require extra arrays, just swaps within the original array. Random pivot selection avoids worst-case scenarios. Despite worst-case $O(n^2)$, it's often faster than Merge Sort.

Sannidhi Deb
(2330044)

Signature of the FIC

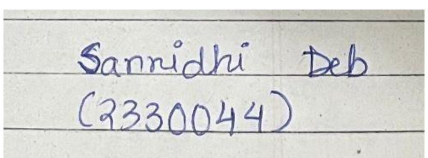
Sannidhi Deb

(Name of the FIC)

## 4. Conclusion:-

Merge Sort and Quick Sort are both fast sorting methods. Merge Sort is steady and works well with large data but uses more memory. Quick Sort is usually faster and uses less memory, but can slow down with bad input. Choosing between them depends on speed, memory, and data type.

Merge Sort and Quick Sort both use the divide and conquer method. They break the problem into smaller parts, solve them, and combine the results. Merge Sort is stable but uses more memory. Quick Sort is faster and uses less memory, but can slow down with bad pivot choices.




Signature of the FIC

Sannidhi Deb

(Name of the FIC)