5)b)Implement Merge sort and observe the execution time for various input sizes(Average, Worst,Best).

AIM: To Implement Merge sort and observe the execution time for various input sizes(Average, Worst, Best).

ALGORITHM:

Merge Sort Function:

- Input: An array A, and two indices low & high.
- Output: The array A is sorted between indices low & high.

1. If low < high:
   - Calculate the mid point :mid (low + high)/2
   - Recursively sort the left half : mergesort(A, low, mid)
   - Recursively sort the right half : mergesort(A, mid+1,high)
   - Merge the sorted halves : merge(A, low, mid, high)

2. Merge Function(Merge):
   - Input : An array A, and indices low, mid and high.
   - Output : The portion of the array A from index low to high is merged and sorted.
   - Create temporary array B to hold the merged values.
   - Initialize three indices:
     i = low(start of the left half)
     j = mid+1 (start of the right half)
     k=0 (index for temporary array B)

3. Merge process:
   - While i<=mid && j>=high
     Compare A[i] and A[j]
   - Copy the smaller value to B[k], increment the respective index (i or j) and increment k,
   - After one half is executed, copy the remaining elements from the other half to B.

4. Copy the merged elements from B back to the original array A from index low to high.

- Divide : Recursively split the array into halves until you reach single element array.
- Merge : Combine the halves back together in sorted order using the merge function.

SOURCE CODE:

```
// Program to implement Merge Sort

#include <stdio.h>

void printArray(int A[ ], int n)

{

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

   {

      printf("%d ", A[i]);
```

```
    }
    printf("\n");
}
void merge(int A[], int mid, int low, int high)
{
    int i, j, k, B[100];
    i = low;
    j = mid + 1;
    k = low;
    while (i <= mid && j <= high)
    {
        if (A[i] < A[j])
        {
            B[k] = A[i];
            i++;
            k++;
        }
        else
        {
            B[k] = A[j];
            j++;
            k++;
        }
    }
    while (i <= mid)
    {
        B[k] = A[i];
        k++;
        i++;
    }
    while (j <= high)
    {
```

```
        B[k] = A[j];
        k++;
        j++;
    }
    for (int i = low; i <= high; i++)
    {
        A[i] = B[i];
    }
}
void mergeSort(int A[], int low, int high){
    int mid;
    if(low<high){
        mid = (low + high) /2;
        mergeSort(A, low, mid);
        mergeSort(A, mid+1, high);
        merge(A, mid, low, high);
    }
}
int main()
{
    int A[30],n,i;
    printf("enter no of elements:");
    scanf("%d",&n);
    printf("\nenter %d elements:\n",n);
    for(i=0;i<n;i++)
    scanf("%d",&A[i]);
    printf("\nelements before sorting:\n");
    printArray(A,n);
    mergeSort(A, 0, n-1);
    printf("\nelements after sorting:\n);
    printArray(A, n);
    return 0;
```

```
}
```

OUTPUT:

```
enter no of elements:5
enter 5 elements:
5
4
3
2
1
elements before sorting:
5 4 3 2 1
elements after sorting:
1 2 3 4 5
```

CONCLUSION: The above program is executed successfully to Implement Merge sort and observe the execution time for various input sizes(Average, Worst, Best).