**Exp3**

**Name : Rais Saaim Ravish**

**Roll No : 24DDS03**

**Batch : 2**

**Aim:**Implement the Merge Sort

**Theory:**

Merge Sort is a Divide and Conquer sorting algorithm that recursively divides the array into two halves, sorts them, and then merges the sorted halves back together.

Working of Merge Sort:

1. Divide: Recursively split the array into two halves until each subarray contains a single element.
2. Conquer: Sort each subarray. Since a single element is already sorted, this step involves merging two sorted subarrays into a larger sorted array.
3. Merge: Compare elements from both subarrays and merge them in sorted order.
4. Repeat: This process continues until the entire array is merged into a sorted sequence.

Time Complexity Analysis:

* Best Case: O(nlog⁡n)O(n \log n)O(nlogn) (Even if the array is sorted, it still divides and merges)
* Average Case: O(nlog⁡n)O(n \log n)O(nlogn)
* Worst Case: O(nlog⁡n)O(n \log n)O(nlogn)

Space Complexity:

* O(n)O(n)O(n) (Uses extra space for temporary arrays during merging)
* Merge Sort is not an in-place sorting algorithm (unlike Quick Sort).

When to Use Merge Sort?

* When stability is required (preserves the order of duplicate elements).
* When sorting linked lists (avoids extra space needed for shifting elements).
* When dealing with large datasets where time complexity is crucial.

**Program:**

#include <stdio.h>

int count = 0; // Global variable to count comparisons

void Combine(int A[], int low, int mid, int high) {

int temp[high - low + 1]; // Temporary array

int i = low, j = mid + 1, k = 0;

while (i <= mid && j <= high) {

count++; // Count the comparison

if (A[i] < A[j]) {

temp[k++] = A[i++];

} else {

temp[k++] = A[j++];

}

}

while (i <= mid) {

temp[k++] = A[i++];

}

while (j <= high) {

temp[k++] = A[j++];

}

for (i = low, k = 0; i <= high; i++, k++) {

A[i] = temp[k]; // Copying sorted elements back

}

}

void mergesort(int A[], int low, int high) {

if (low < high) {

int mid = (low + high) / 2;

mergesort(A, low, mid);

mergesort(A, mid + 1, high);

Combine(A, low, mid, high);

}

}

int main() {

int n;

printf("Enter the number of elements: ");

scanf("%d", &n);

int A[n];

printf("Enter %d elements:\n", n);

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

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

}

mergesort(A, 0, n - 1);

printf("Sorted array:\n");

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

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

}

printf("\n");

printf("Number of comparisons: %d\n", count);

return 0;

}

**Output:**

Enter the number of elements: 5

Enter 5 elements:

5 4 3 2 1

Sorted array:

1 2 3 4 5

Number of comparisons: 7

1)Best Case:

Enter the number of elements: 5

Enter 5 elements:

1 2 3 4 5

Sorted array:

1 2 3 4 5

Number of comparisons: 7

2)Worst Case:

Enter the number of elements: 5

Enter 5 elements:

5 4 3 2 1

Sorted array:

1 2 3 4 5

Number of comparisons: 7

3)Average Case:

Enter the number of elements: 5

Enter 5 elements:

5 2 4 3 1

Sorted array:

1 2 3 4 5

Number of comparisons: 7

**Conclusion:**

Merge Sort is an efficient Divide and Conquer algorithm that consistently sorts an array in O(n log n) time complexity across best, worst, and average cases. It works by recursively dividing the array into smaller subarrays, sorting them, and then merging them back in a sorted manner.