AMRITA SCHOOL OF COMPUTING

DESIGN AND ANALYSIS OF ALGORITHMS

(23CSE211)

Name: P.Jeeven Sandeep

Roll No.: CH.SC.U4CSE24134

Class: BTech (CSE-B)

School: Amrita School of Computing,

Chennai Campus.

LAB-4

1) MERGE SORT:

Code:

```c
#include <stdio.h>
void merge(int arr[], int low, int mid, int high) {
    int temp[50], i=low, j=mid+1, k=low;

    while(i<=mid && j<=high)
        temp[k++] = (arr[i]<arr[j]) ? arr[i++] : arr[j++];
    while(i<=mid) temp[k++] = arr[i++];
    while(j<=high) temp[k++] = arr[j++];

    for(i=low;i<=high;i++) arr[i]=temp[i];
}
void mergeSort(int arr[], int low, int high) {
    if(low<high){
        int mid=(low+high)/2;
        mergeSort(arr,low,mid);
        mergeSort(arr,mid+1,high);
        merge(arr,low,mid,high);
    }
}
int main(){
    int n, arr[50];
    printf("Enter n: ");
    scanf("%d",&n);

    for(int i=0;i<n;i++) scanf("%d",&arr[i]);
    mergeSort(arr,0,n-1);

    for(int i=0;i<n;i++) printf("%d ",arr[i]);
    return 0;
}
```

Output:

```
Enter n: 12
157 110 147 122 111 149 151 141 143 112 117 133
110 111 112 117 122 133 141 143 147 149 151 157
--------------------------------
```

Time Complexity:

O(n log n) because the array is always divided into two halves (log n levels), and each merge operation processes all *n* elements once.

Space Complexity:

O(n) because an auxiliary array temp[] is used during merging, plus O(log n) recursion stack space.

2) Quick Sort:

Code:

```c
#include <stdio.h>
int arrange(int list[], int left, int right) {
    int pivotValue = list[right];
    int boundary = left - 1;
    int tempValue;
    for (int current = left; current < right; current++) {
        if (list[current] <= pivotValue) {
            boundary++;
            tempValue = list[boundary];
            list[boundary] = list[current];
            list[current] = tempValue;
        }
    }
    tempValue = list[boundary + 1];
    list[boundary + 1] = list[right];
    list[right] = tempValue;
    return boundary + 1;
}
void quickArrange(int list[], int left, int right) {
    if (left < right) {
        int splitPoint = arrange(list, left, right);
        quickArrange(list, left, splitPoint - 1);
        ickArrange(list, splitPoint + 1, right);
    }
}
int main() {
    int count, values[50], index;
    printf("Enter number of elements: ");
    scanf("%d", &count);
    printf("Enter elements:\n");
    for (index = 0; index < count; index++)
        scanf("%d", &values[index]);
    quickArrange(values, 0, count - 1);
    printf("Sorted array:\n");
    for (index = 0; index < count; index++)
        printf("%d ", values[index]);
    return 0;
}
```

ELF-3-2-1.exe

Output:

```
Enter number of elements: 12
Enter elements:
157 110 147 122 111 149 151 141 143 112 117 133
Sorted array:
110 111 112 117 122 133 141 143 147 149 151 157
```

Time Complexity:

O(n log n) (for average case) because the array is repeatedly partitioned into two nearly equal halves, and each partition step scans all *n* elements once. $O(n^2)$ (for worst case) because choosing the last element as pivot causes highly unbalanced partitions (e.g., already sorted array), leading to *n* recursive calls each doing *O(n)* work.

Space Complexity:

O(log n) (for average case) due to recursion stack when partitions are balanced, O(n) (for worst case) when recursion becomes skewed.