DESIGN AND ANALYSIS OF ALGORITHMS

LAB WORKBOOK WEEK – 4

NAME: GADIRAJU TEJA PRATHAP VARMA

ROLL NUMBER: CH.SC.U4CSE24117

CLASS: CSE-B

Sorting Techniques

Merge Sort

Code:

```c
#include <stdio.h>

void merge(int arr[], int l, int m, int r) {
    int i = l, j = m + 1, k = 0;
    int temp[r - l + 1];

    while (i <= m && j <= r) {
        if (arr[i] <= arr[j])
            temp[k++] = arr[i++];
        else
            temp[k++] = arr[j++];
    }

    while (i <= m)
        temp[k++] = arr[i++];
    while (j <= r)
        temp[k++] = arr[j++];

    for (i = l, k = 0; i <= r; i++, k++)
        arr[i] = temp[k];
}
```

```c
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = (l + r) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);

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

    mergeSort(arr, 0, n - 1);

    printf("Sorted array:\n");
```

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

    return 0;
}
```

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

The array is divided log n times.
At each level, merging takes n steps.
So, the total work is n × log n.
Hence, the time complexity is O(n log n).

Space Complexity

An extra array of size n is required.
The space used grows with the input size.
Therefore, the space complexity is O(n).

Quick Sort:

Code:

```c
#include <stdio.h>

int partition(int a[], int low, int high) {
    int pivot = a[high];
    int i = low - 1, temp;

    for (int j = low; j < high; j++) {
        if (a[j] <= pivot) {
            i++;
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }
    temp = a[i + 1];
    a[i + 1] = a[high];
    a[high] = temp;

    return i + 1;
}
```

```c
void quickSort(int a[], int low, int high) {
    if (low < high) {
        int p = partition(a, low, high);
        quickSort(a, low, p - 1);
        quickSort(a, p + 1, high);
    }
}

int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);

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

    quickSort(a, 0, n - 1);

    printf("Sorted array:\n");
    for (int i = 0; i < n; i++)
```

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

    return 0;
}
```

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

Recursive calls vary based on pivot selection.
Each call processes all n elements during partition.
Worst case operations result in n^2 work.
So, the time complexity is $O(n^2)$.

Space Complexity

Additional space is used for recursion.
The required space depends on recursion depth.
Thus, the space complexity is O(log n).