DESIGN AND ANALYSIS OF ALGORITHMS

LAB WORKBOOK WEEK – 4

NAME: Konda Bhavani

ROLL NUMBER: CH.SC.U4CSE24120

CLASS: CSE-B

Sorting Techniques

Merge Sort

Code:

```c
#include <stdio.h>
void merge(int a[], int low, int mid, int high) {
    int i = low, j = mid + 1, k = 0;
    int temp[100];

    while (i <= mid && j <= high) {
        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];
}

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);
        merge(a, low, mid, high);
    }
}
```

```c
int main() {
    int a[100], n, i;

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

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

    mergeSort(a, 0, n - 1);

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

    return 0;
}
```

Output:

```
Enter number of elements: 12
Enter elements:
157 110 147 122 111 149 151 141 123 112 117 133
Sorted array:
110 111 112 117 122 123 133 141 147 149 151 157
```

Time Complexity:
Outer division runs log n times.
Merging process runs n times at each level.
Total operations = n × log n.
Time Complexity = O(n log n)

Space Complexity:
Auxiliary array of size n is used.
Extra space increases with input size.
Space Complexity = O(n)

Quick Sort:

Code:

```c
#include <stdio.h>

void quickSort(int a[], int low, int high) {
    int i = low, j = high;
    int pivot = a[(low + high) / 2];

    while (i <= j) {
        while (a[i] < pivot)
            i++;
        while (a[j] > pivot)
            j--;

        if (i <= j) {
            int temp = a[i];
            a[i] = a[j];
            a[j] = temp;
            i++;
            j--;
        }
    }

    if (low < j)
        quickSort(a, low, j);
    if (i < high)
        quickSort(a, i, high);
}
```

```c
int main() {
    int n;

    printf("Enter 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]);
    }

    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 12 elements:
157 110 147 122 111 149 151 141 123 112 117 133
Sorted array:
110 111 112 117 122 123 133 141 147 149 151 157
```

Time Complexity:

Outer recursive calls depend on pivot selection.
Partitioning runs n times in each call.
Worst case operations = n × n.
Time Complexity = $O(n^2)$

Space Complexity:

Recursive call stack is used.
Space depends on recursion depth.
Space Complexity = O(log n)