DESIGN AND ANALYSIS OF ALGORITHMS

LAB WORKBOOK WEEK – 4

NAME: CHANTATI SAI PURNISHA MAHI

ROLL NUMBER: CH.SC.U4CSE24156

CLASS: CSE-B

Sorting Techniques

Merge Sort

Code:

```c
#include <stdio.h>

void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;
    int L[n1], R[n2];

    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int i = 0; i < n2; i++)
        R[i] = arr[mid + 1 + i];

    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
```

```c
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

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

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

    printf("Given array: ");
    printArray(arr, n);

    mergeSort(arr, 0, n - 1);

    printf("\nSorted array: ");
    printArray(arr, n);
    printArray(arr, n);

    return 0;
}
```

Output:

```
root@ubuntu:/home/purnisha# nano Merge.c
root@ubuntu:/home/purnisha# gcc -o Merge Merge.c
root@ubuntu:/home/purnisha# ./Merge
Enter the number of elements: 12
Enter the elements:
157 110 147 122 111 149 151 141 123 112 117 133
Given array: 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

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:

GNU nano 7.2 Quick.c

```c
#include <stdio.h>

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

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

    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;

    return (i + 1);
}

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
```

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

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

    printf("Given array: ");
    printArray(arr, n);

    quickSort(arr, 0, n - 1);

    printf("\nSorted array: ");
    printArray(arr, n);

    return 0;
}
```

Output:

```
root@ubuntu:/home/purnisha# nano Quick.c
root@ubuntu:/home/purnisha# gcc -o Quick Quick.c
root@ubuntu:/home/purnisha# ./Quick
Enter the number of elements: 12
Enter the elements:
157 110 147 122 111 149 151 141 123 112 117 133
Given array: 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

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).