AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY UNDER SECTION 3 OF UGC ACT, 1956
SCHOOL OF COMPUTING

Devadharshan.S

CH.SC.U4CSE24113

Week – 3

Date - 03/01/2026

Design and Analysis of Algorithm(23CSE211)

1. Merge Sort

Code:

```
#include <stdio.h>
void merge(int arr[], int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];
    for (int i = 0; i < n1; i++) L[i] = arr[l + i];
    for (int j = 0; j < n2; j++) R[j] = arr[m + 1 + j];
    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) arr[k++] = L[i++];
        else arr[k++] = R[j++];
    }
    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
}
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}
int main() {
    int n;
    printf("CH.SC.U4CSE24113\n");
    printf("Enter number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter the elements:");
    for (int i = 0; i < n; i++) scanf("%d", &arr[i]);
    mergeSort(arr, 0, n - 1);
    printf("Sorted array: ");
    for (int i = 0; i < n; i++) printf("%d ", arr[i]);
    printf("\n");
    return 0;
}
```

Output:

```
devadharshan@devadharshan-HP-Pavilion-Laptop-15-...
devadharshan@devadharshan-HP-Pavilion-Laptop-15-eg3xxx:/media/devad
harshan/New Volume/DAA/Week-3/C-Files$ gcc MergeSort.c -o MergeSort
devadharshan@devadharshan-HP-Pavilion-Laptop-15-eg3xxx:/media/devad
harshan/New Volume/DAA/Week-3/C-Files$ ./MergeSort
CH.SC.U4CSE24113
Enter number of elements: 10
Enter the elements:5 6 7 8 9 1 2 3 4 0
Sorted array: 0 1 2 3 4 5 6 7 8 9
devadharshan@devadharshan-HP-Pavilion-Laptop-15-eg3xxx:/media/devad
harshan/New Volume/DAA/Week-3/C-Files$
```

Space Complexity: O(n log n)
Justification: The algorithm recursively divides the array into halves, which takes log n levels of division. At each level, it performs a linear merge process requiring n operations, resulting in a consistent n x log n performance.

Time Complexity: O(n)
Justification: Merge Sort requires an auxiliary array to temporarily hold elements during the merge process. This additional memory is proportional to the size of the input array.

2. Quick Sort

Code:

```
#include <stdio.h>
void swap(int* a, int* b) {
    int t = *a;
    *a = *b;
    *b = t;
}
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++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    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);
    }
}
int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("CH.SC.U4CSE24113\n");
    int arr[n];
    printf("Enter the elements:");
    for (int i = 0; i < n; i++) scanf("%d", &arr[i]);
    quickSort(arr, 0, n - 1);
    printf("Sorted array: ");
    for (int i = 0; i < n; i++) printf("%d ", arr[i]);
    printf("\n");
    return 0;
}
```

Output:

```
devadharshan@devadharshan-HP-Pavilion-Laptop-15-...
devadharshan@devadharshan-HP-Pavilion-Laptop-15-eg3xxx:/media/devadharshan/New Volume/DAA/Week-3/C-Files$ gcc QuickSort.c -o QuickSort
devadharshan@devadharshan-HP-Pavilion-Laptop-15-eg3xxx:/media/devadharshan/New Volume/DAA/Week-3/C-Files$ ./QuickSort
Enter number of elements: 10
CH.SC.U4CSE24113
Enter the elements:21 31 64 95 32 12 74 12 30 71
Sorted array: 12 12 21 30 31 32 64 71 74 95
devadharshan@devadharshan-HP-Pavilion-Laptop-15-eg3xxx:/media/devadharshan/New Volume/DAA/Week-3/C-Files$
```

Space Complexity: $O(n^2)$
Justification: This occurs when the pivot consistently picks the smallest or largest element (e.g., on a sorted array). This results in highly unbalanced partitions where one side has 0 elements and the other has n-1, leading to n recursive levels with O(n) work each.

Time Complexity: O(n)
Justification: In the worst-case scenario of an unbalanced partition, the recursion stack depth increases from the ideal log n to n. Each recursive call stays on the stack until it finishes, requiring memory proportional to the number of elements.