



SCHOOL OF
COMPUTING

DESIGN AND ANALYSIS OF ALGORITHMS
LAB WORKBOOK
WEEK - 4

NAME : Kurra Venkata Gokul
ROLL NUMBER : CH.SC.U4CSE24121
CLASS : CSE-B

Question 1: Write a C program that reads N integers from the user and sorts it using Merge Sort.

CODE:

```
#include <stdio.h>

void merge(int arr[], int left, int mid, int right) {

    int i, j, k;

    int n1 = mid - left + 1;
    int n2 = right - mid;

    int L[50], R[50];
    for (i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];

    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++;
}
while (j < n2) {
    arr[k] = R[j];
    j++;
    k++;
}
void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}
int main() {
    int arr[50], n;
    printf("CH.SC.U4CSE24142 - SANTHOSH A\n");
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter the 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:

```
Microsoft Windows [Version 10.0.26200.7462]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>gcc -o m.c merge.c  
  
C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>  
C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>.\m.c  
ch.sc.u4cse24121  
Enter number of elements: 5  
Enter the elements:  
15  
98  
65  
2  
236  
Sorted array:  
2 15 65 98 236  
C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>
```

Space Complexity: $O(N)$

Time Complexity: $O(N \log N)$

Justification:

- The array is divided into two equal halves at each step, and the merge operation compares all n elements. Since this division continues for $\log n$ levels, the overall **Time Complexity** becomes $O(N \log N)$.
- An extra array is used during the merging process to temporarily store elements. This additional storage requires space proportional to the number of elements, resulting in **$O(n)$ Space Complexity**.

Question 2: Create a C program that implements Quick Sort to sort an array of integers.

CODE:

```
#include <stdio.h>
```

```
void swap(int *a, int *b) {
```

```
    int temp;
```

```
    temp = *a;
```

```
    *a = *b;
```

```
    *b = temp;
```

```
}
```

```
int partition(int arr[], int low, int high) {
```

```
    int pivot;
```

```
    int i, j;
```

```
    pivot = arr[high];
```

```
    i = low - 1;
```

```
    for (j = low; j < high; 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) {  
    int pi;  
    if (low < high) {  
        pi = partition(arr, low, high);  
        quickSort(arr, low, pi - 1);  
        quickSort(arr, pi + 1, high);  
    }  
}
```

```
int main() {  
    printf("ch.sc.u4cse24121\n");  
  
    int arr[50];  
    int n, i;  
  
    printf("Enter number of elements: ");  
    scanf("%d", &n);  
  
    printf("Enter the elements:\n");  
    for (i = 0; i < n; i++) {  
        scanf("%d", &arr[i]);  
    }  
  
    quickSort(arr, 0, n - 1);  
  
    printf("Sorted array:\n");  
    for (i = 0; i < n; i++) {  
        printf("%d ", arr[i]);  
    }
```

```
    }  
  
    return 0;  
}
```

OUTPUT:

```
C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>gcc -o q.c quick.c  
C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>.\q.c  
ch.sc.u4cse24121  
Enter number of elements: 6  
Enter the elements:  
10  
20  
58  
1  
98  
654  
Sorted array:  
1 10 20 58 98 654  
C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>
```

Space Complexity: $O(N)$

Time Complexity: $O(N^2)$

Justification:

- In the worst case, Quick Sort makes highly unbalanced recursive calls, reducing the problem size by only one element at each step.
As a result, the recursion depth becomes n , and the recursion stack stores up to n function calls, leading to **$O(N)$ Space Complexity**.
- This occurs when the pivot divides the array into highly unbalanced parts, such as in already sorted or reverse sorted array. Hence the **Time Complexity** is **$O(N^2)$** .