Merge Sort

Code:

```
#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)
{
    int mid;
    if (low < high)
    {
        mid = (low + high) / 2;
        mergeSort(a, low, mid);
        mergeSort(a, mid + 1, high);
        merge(a, low, mid, high);
    }
}
int main()
{
    int a[10], 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]);
    return 0;
}
```

Output :

```
vivek@Vivek:/mnt/c/Users/user/Documents/Haskell$ gcc mergesort.c -o mergesort
vivek@Vivek:/mnt/c/Users/user/Documents/Haskell$ ./mergesort
Enter number of elements: 6
Enter elements:
2
9
4
8
2
6
Sorted array:
2 2 4 6 8 9 vivek@Vivek:/mnt/c/Users/user/Documents/Haskell$
vivek@Vivek:/mnt/c/Users/user/Documents/Haskell$
```

Complexity Analysis of Merge Sort

- **Best Case:** O(n log n)
- **Average Case:** O(n log n)
- **Worst Case:** O(n log n)

Quick Sort

Code:

```c
#include <stdio.h>
int partition(int a[], int low, int high)
{
    int pivot = a[low];
    int i = low + 1, j = high, temp;

    while (i <= j)
    {
        while (a[i] <= pivot)
            i++;
        while (a[j] > pivot)
            j--;
        if (i < j)
        {
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }
    temp = a[low];
    a[low] = a[j];
    a[j] = temp;
    return j;
}
void quickSort(int a[], int low, int high)
{
    int p;
    if (low < high)
    {
        p = partition(a, low, high);
        quickSort(a, low, p - 1);
        quickSort(a, p + 1, high);
    }
}
int main()
{
    int a[10], n, i;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    printf("Enter elements:\n");
    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);
    quickSort(a, 0, n - 1);
    printf("Sorted array:\n");
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Output:

```
vivek@Vivek:/mnt/c/Users/user/Documents/Haskell$
vivek@Vivek:/mnt/c/Users/user/Documents/Haskell$ gcc quicksort.c -o quicksort
vivek@Vivek:/mnt/c/Users/user/Documents/Haskell$ ./quicksort
Enter number of elements: 6
Enter elements:
2
9
4
8
2
6
Sorted array:
2 2 4 6 8 9 vivek@Vivek:/mnt/c/Users/user/Documents/Haskell$
vivek@Vivek:/mnt/c/Users/user/Documents/Haskell$
```

Complexity Analysis of Quick Sort

- **Best Case:** O(n log n)
- **Average Case:** O(n log n)
- **Worst Case:** $O(n^2)$