AMRITA VISHWA VIDYAPEETHAM

Second Year B.Tech

(Computer Science and Engineering)

Design and Analysis of Algorithms

TASK – 3

Name: Shaik Kolimi Dada Hussain

College Name: Amrita vishwa Vidyapeetham

Roll No : ch.sc.u4cse24144

Department: CSE-B

Academic Year : 2024-2028

1) Write a c program to sort the given numbers Using Merge Sort .

[157 110 147 122 111 149 151 141 123 112 117 133]

CODE:

```c
//CH.SC.U4CSE24144
#include <stdio.h>
void merge(int a[], int l, int m, int r)
{
    int i = l, j = m + 1, k = l;
    int b[100];
    while (i <= m && j <= r)
    {
        if (a[i] < a[j])
            b[k++] = a[i++];
        else
            b[k++] = a[j++];
    }
    while (i <= m)
        b[k++] = a[i++];
    while (j <= r)
        b[k++] = a[j++];

    for (i = l; i <= r; i++)
        a[i] = b[i];
}
void mergeSort(int a[], int l, int r)
{
    if (l < r)
    {
        int m = (l + r) / 2;
        mergeSort(a, l, m);
        mergeSort(a, m + 1, r);
        merge(a, l, m, r);
    }
}
```

```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]);
    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 and Space Complexity with Justification:

Time Complexity

Best Case: O(n log n)

Average Case: O(n log n)

Worst Case: O(n log n)

Justification:

The array is always divided into two halves using recursion.

Each division takes log n levels.

At every level, all n elements are merged.

So total work = n × log n

Therefore, time complexity is O(n log n) for all cases

Space Complexity: O(n)

Justification:

An extra array b[100] (temporary array) is used to store merged elements.

This array requires n extra memory.

Hence, Merge Sort needs additional memory.

So, space complexity is O(n).

2) Write a c program to sort the given numbers Using Quick Sort .

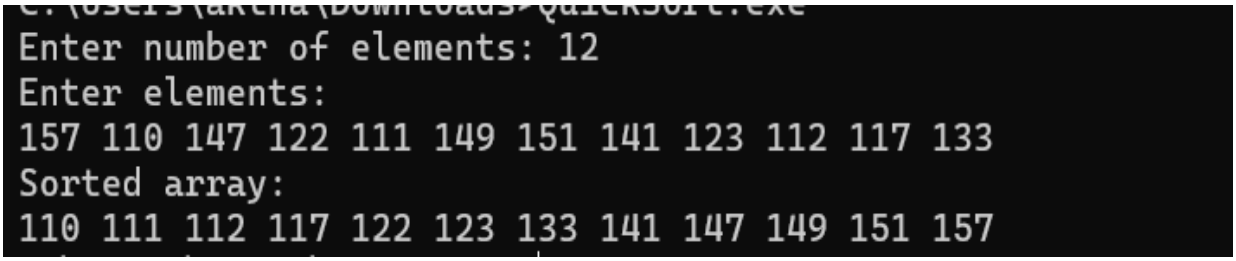
[157 110 147 122 111 149 151 141 123 112 117 133]

CODE:

```c
//CH.SC.U4CSE24144
#include <stdio.h>
int partition(int a[], int low, int high)
{
    int pivot = a[low];
    int i = low + 1;
    int j = high;
    int 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)
{
    if (low < high)
    {
        int pos = partition(a, low, high);

        quickSort(a, low, pos - 1);
        quickSort(a, pos + 1, 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]);

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

Output:



Time Complexity and Space Complexity with Justification:

Time Complexity

Best Case: O(n log n)

Average Case: O(n log n)

Worst Case: $O(n^2)$

Justification:

In best and average case, the pivot divides the array into two nearly equal parts.

The array is divided into log n levels.

At each level, n elements are compared.

So time = n × log n = O(n log n).

In the worst case (when array is already sorted or reverse sorted), the pivot creates very unbalanced partitions.

So time becomes $O(n^2)$.

Space Complexity

Space Complexity: O(log n)

Justification:

- Quick Sort does not use extra arrays.
- It uses recursive function calls.
- The recursion stack needs log n space in average case.

So space complexity is O(log n).