

Design And Analysis Of Algorithms

Practical

Objective:- Implement and analyze the complexity of Quick And Merge Sort.

Code:-

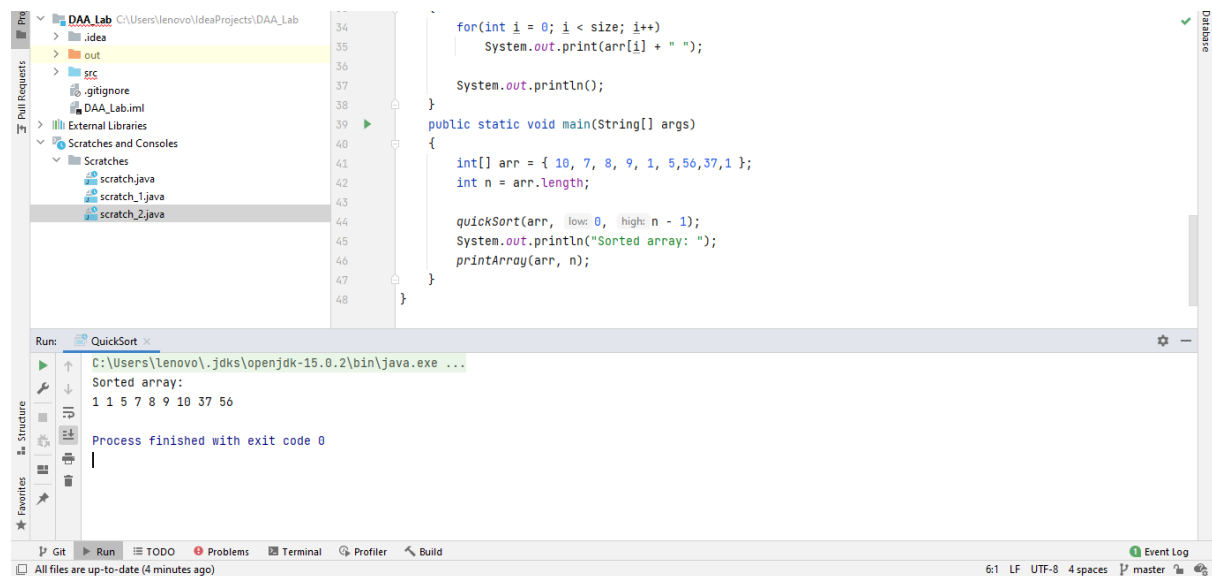
```
class QuickSort{
    static void swap(int[] arr, int i, int j)
    {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
    static int partition(int[] arr, int low, int high)
    {
        int pivot = arr[high];
        int i = (low - 1);
        for(int j = low; j <= high - 1; j++)
        {
            if (arr[j] < pivot)
            {
                i++;
                swap(arr, i, j);
            }
        }
        swap(arr, i + 1, high);
        return (i + 1);
    }
    static 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);
        }
    }
    static void printArray(int[] arr, int size)
    {
        for(int i = 0; i < size; i++)
            System.out.print(arr[i] + " ");

        System.out.println();
    }
    public static void main(String[] args)
    {
        int[] arr = { 10, 7, 8, 9, 1, 5, 56, 37, 1 };
        int n = arr.length;

        quickSort(arr, 0, n - 1);
        System.out.println("Sorted array: ");
        printArray(arr, n);
    }
}
```

```
}
```

Output:-



The screenshot shows an IDE with a project named 'DAA_Lab'. The 'src' folder contains 'scratch_1.java' and 'scratch_2.java'. The 'out' folder is highlighted. The code editor shows a Java program for QuickSort. The 'Run' window displays the output: 'Sorted array: 1 1 5 7 8 9 10 37 56' and 'Process finished with exit code 0'.

```
for(int i = 0; i < size; i++)
    System.out.print(arr[i] + " ");

System.out.println();

public static void main(String[] args)
{
    int[] arr = { 10, 7, 8, 9, 1, 5, 56, 37, 1 };
    int n = arr.length;

    quickSort(arr, low: 0, high: n - 1);
    System.out.println("Sorted array: ");
    printArray(arr, n);
}
```

Run: QuickSort
C:\Users\lenovo\.jdk\openjdk-15.0.2\bin\java.exe ...
Sorted array:
1 1 5 7 8 9 10 37 56
Process finished with exit code 0

Analyze:-Best Time Complexity : $O(n \log n)$ Average Time Complexity : $O(n \log n)$
Worst Time Complexity : $O(n^2)$

Objective:- Merge Sort

Code:-

```
class MergeSort
{
    void merge(int arr[], int l, int m, int r)
    {
        int n1 = m - l + 1;
        int n2 = r - m;
        int L[] = new int[n1];
        int R[] = new int[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;
        int k = l;
        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 sort(int arr[], int l, int r)
{
    if (l < r) {
        int m = l + (r-l)/2;
        sort(arr, l, m);
        sort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

static void printArray(int arr[])
{
    int n = arr.length;
    for (int i = 0; i < n; ++i)
        System.out.print(arr[i] + " ");
    System.out.println();
}

public static void main(String args[])
{
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    System.out.println("Given Array");
    printArray(arr);
    MergeSort ob = new MergeSort();
    ob.sort(arr, 0, arr.length - 1);
    System.out.println("\nSorted array");
    printArray(arr);
}
}

```

Output:-

The screenshot shows an IDE with a project named 'DAA_Lab'. The code in the editor is as follows:

```
47 {
48     int n = arr.length;
49     for (int i = 0; i < n; ++i)
50         System.out.print(arr[i] + " ");
51     System.out.println();
52 }
53 public static void main(String args[])
54 {
55     int arr[] = { 12, 11, 13, 16, 77, 5, 6, 7 };
56     System.out.println("Given Array");
57     printArray(arr);
58     MergeSort ob = new MergeSort();
59     ob.sort(arr, 0, arr.length - 1);
60     System.out.println("\nSorted array");
61     printArray(arr);
62 }
```

The Run window shows the following output:

```
Run: MergeSort
C:\Users\lenovo\.jdk\openjdk-15.0.2\bin\java.exe ...
Given Array
12 11 13 16 77 5 6 7

Sorted array
5 6 7 11 12 13 16 77

Process finished with exit code 0
```

Analyze:-

The time complexity of MergeSort is **$O(n \cdot \log n)$** in all the 3 cases (worst, average and best) as the mergesort always divides the array into two halves and takes linear time to merge two halves.