K. K. Wagh Institute of Engineering Education and Research, Nashik.
Department of Computer Engineering
Academic Year 2022-23

Course Name: Laboratory Practice-III **Course Code:** 410246
Class: BE **Div:** A

Name of Students:	Tanuja Baliram Kaklij	(65)
	Gauri Kailas Bankar	(66)
	Srushti Bhausaheb Hire	(67)
	Digvijay Bhaupatil Wagh	(68)

Name of Faculty: Prof. K.P.Birla

Mini-Project Report

Title of Mini-Project: - Implement merge sort and multithreaded merge sort. Compare time required by both the algorithms. Also analyze the performance of each algorithm for the best case and the worst case.

Problem Statement: Implement merge sort and multithreaded merge sort. Compare time required by both the algorithms. Also analyze the performance of each algorithm for the best case and the worst case.

1. Implement merge sort and multithreaded merge sort.
2. Compare time required by merge sort and multithreaded merge sort. Also analyze the performance of each algorithm for the best case and the worst case.

Objective:

1. Learn how to implement algorithms that follow algorithm design strategy divide and conquer.

Introduction of Mini-Project:

The Merge Sort algorithm is a sorting algorithm that is based on the Divide and Conquer paradigm. In this algorithm, the array is initially divided into two equal halves and then they are combined in a sorted manner.

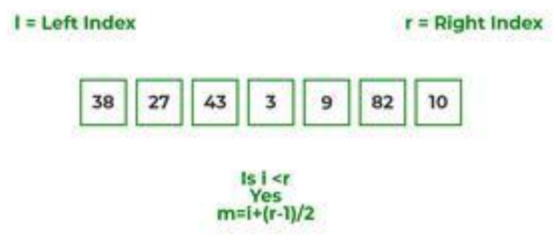
Merge Sort Working Process:

Think of it as a recursive algorithm continuously splits the array in half until it cannot be further divided. This means that if the array becomes empty or has only one element left, the dividing will stop, i.e. it is the base case to stop the recursion. If the array has multiple elements, split the array into halves and recursively invoke the merge sort on each of the halves. Finally, when both halves are sorted, the merge operation is applied. Merge operation is the process of taking two smaller sorted arrays and combining them to eventually make a larger one.

Illustration:

To know the functioning of merge sort, lets consider an array arr[] = {38, 27, 43, 3, 9, 82, 10}

- At first, check if the left index of array is less than the right index, if yes then calculate its mid point

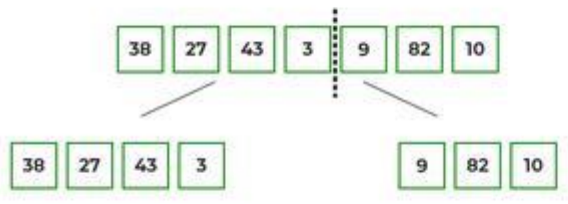


- Now, as we already know that merge sort first divides the whole array iteratively into equal halves, unless the atomic values are achieved.

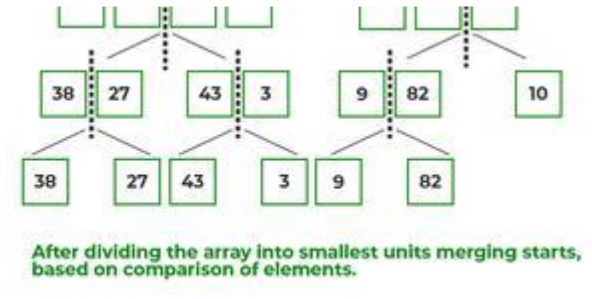
- Here, we see that an array of 7 items is divided into two arrays of size 4 and 3 respectively.

l = Left Index
r = Right Index
38
27
43
3
9
82
10

- Now, again find that is left index is less than the right index for both arrays, if found yes, then again calculate mid points for both the arrays.

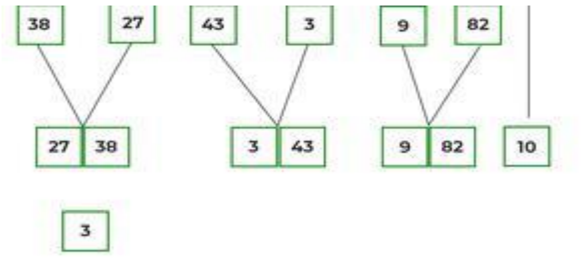


- Now, further divide these two arrays into further halves, until the atomic units of the array is reached and further division is not possible.

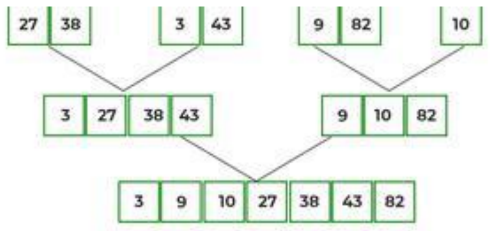

38
27
43
3
9
82
10
38
27
43
3
9
82
After dividing the array into smallest units merging starts, based on comparison of elements.

- After dividing the array into smallest units, start merging the elements again based on comparison of size of elements

- Firstly, compare the element for each list and then combine them into another list in a sorted manner.

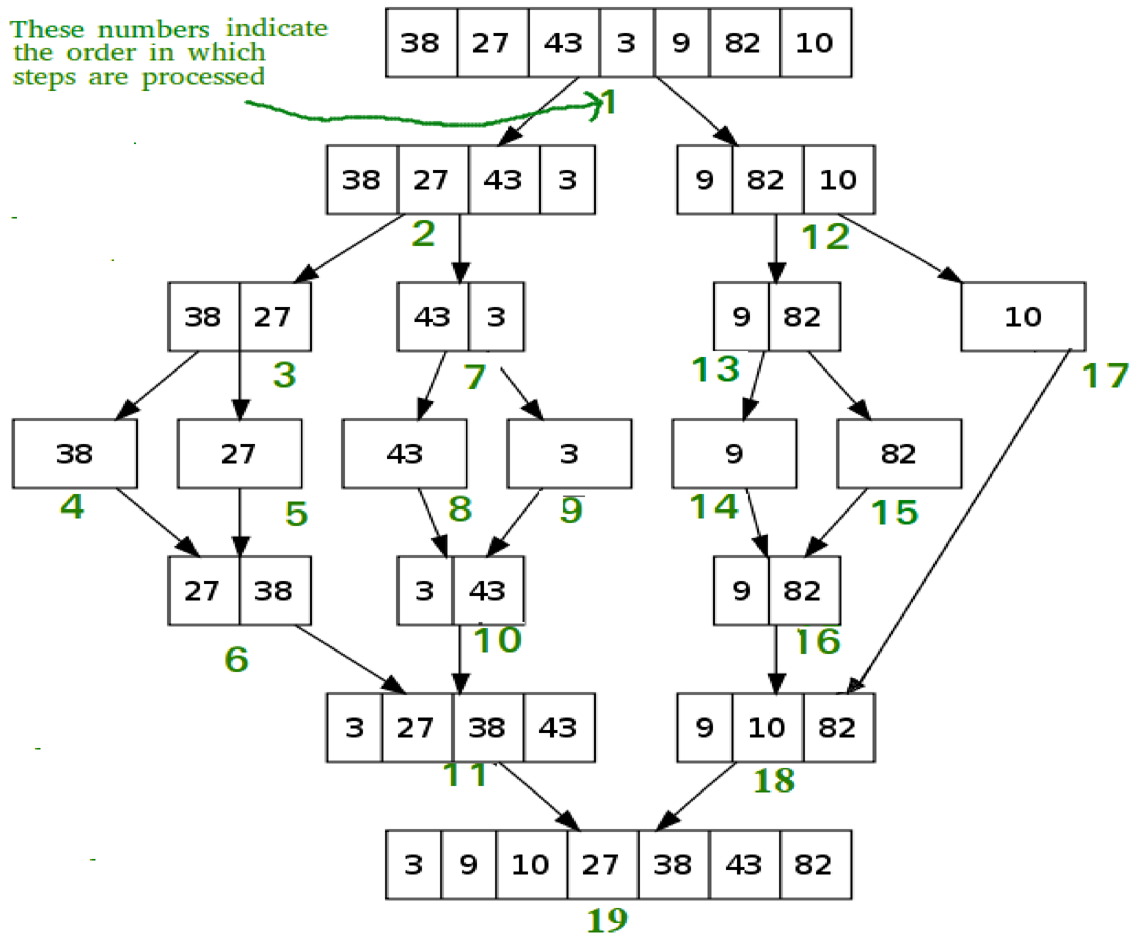


- After the final merging, the list looks like this:



The following diagram shows the complete merge sort process for an example array {38, 27, 43, 3, 9, 82, 10}.

If we take a closer look at the diagram, we can see that the array is recursively divided into two halves till the size becomes 1. Once the size becomes 1, the merge processes come into action and start merging arrays back till the complete array is merged.



Algorithm:

```
step 1: start
step 2: declare array and left, right, mid variable
step 3: perform merge function.
    if left > right
        return
    mid= (left+right)/2
    mergesort(array, left, mid)
    mergesort(array, mid+1, right)
    merge(array, left, mid, right)
step 4: Stop
```

Follow the steps below the solve the problem:

MergeSort(arr[], l, r)

If r > l

- Find the middle point to divide the array into two halves:
- middle m = l + (r – l)/2
- Call mergeSort for first half:
- Call mergeSort(arr, l, m)
- Call mergeSort for second half:
- Call mergeSort(arr, m + 1, r)
- Merge the two halves sorted in steps 2 and 3:
- Call merge(arr, l, m, r)

Java program for Merge Sort

```
class MergeSort1 {
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);

        MergeSort1 ob = new MergeSort1();

        ob.sort(arr, 0, arr.length - 1);

        System.out.println("\nSorted array");

        printArray(arr);

}

}
```

Output

```
Given array is

12 11 13 5 6 7

Sorted array is

5 6 7 11 12 13
```

Time Complexity: O(N log(N)), Sorting arrays on different machines. Merge Sort is a recursive algorithm and time complexity can be expressed as following recurrence relation.

$T(n) = 2T(n/2) + \theta(n)$

Merge Sort using Multi-threading

Merge Sort is a popular sorting technique which divides an array or list into two halves and then start merging them when sufficient depth is reached. Time complexity of merge sort is O(nlogn).

Threads are lightweight processes and threads shares with other threads their code section, data section and OS resources like open files and signals. But, like process, a thread has its own program counter (PC), a register set, and a stack space.

Multi-threading is way to improve parallelism by running the threads simultaneously in different cores of your processor. In this program, we'll use 4 threads but you may change it according to the number of cores your processor has.

Examples:

```
Input :  83, 86, 77, 15, 93, 35, 86, 92, 49, 21,

         62, 27, 90, 59, 63, 26, 40, 26, 72, 36

Output : 15, 21, 26, 26, 27, 35, 36, 40, 49, 59,

         62, 63, 72, 77, 83, 86, 86, 90, 92, 93

Input :  6, 5, 4, 3, 2, 1

Output : 1, 2, 3, 4, 5, 6
```

Java Program to implement merge sort using multi-threading

```
import java.lang.System;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Random;

class MergeSort{

        private static final int MAX_THREADS = 4;

        private static class SortThreads extends Thread{

        SortThreads(Integer[] array, int begin, int end){

                super(()->{

                        MergeSort.mergeSort(array, begin, end);

                });

                this.start();

        }

}
```

```
public static void threadedSort(Integer[] array){
long time = System.currentTimeMillis();
final int length = array.length;
boolean exact = length%MAX_THREADS == 0;
int maxlim = exact? length/MAX_THREADS: length/(MAX_THREADS-1);
maxlim = maxlim < MAX_THREADS? MAX_THREADS : maxlim;
final ArrayList<SortThreads> threads = new ArrayList<>();
for(int i=0; i < length; i+=maxlim){
	int beg = i;
	int remain = (length)-i;
	int end = remain < maxlim? i+(remain-1): i+(maxlim-1);
	final SortThreads t = new SortThreads(array, beg, end);
	threads.add(t);
}
for(Thread t: threads){
	try{
		t.join();
	} catch(InterruptedException ignored){}
}
for(int i=0; i < length; i+=maxlim){
	int mid = i == 0? 0 : i-1;
	int remain = (length)-i;
	int end = remain < maxlim? i+(remain-1): i+(maxlim-1);
	merge(array, 0, mid, end);
}
time = System.currentTimeMillis() - time;
System.out.println("Time spent for custom multi-threaded recursive merge_sort(): "+
time+ "ms");
```

```
}
public static void mergeSort(Integer[] array, int begin, int end){
        if (begin<end){
                int mid = (begin+end)/2;
                mergeSort(array, begin, mid);
                mergeSort(array, mid+1, end);
                merge(array, begin, mid, end);
        }
}
        public static void merge(Integer[] array, int begin, int mid, int end){
        Integer[] temp = new Integer[(end-begin)+1];
        int i = begin, j = mid+1;
        int k = 0;
        while(i<=mid && j<=end){
                if (array[i] <= array[j]){
                        temp[k] = array[i];
                        i+=1;
                }else{
                        temp[k] = array[j];
                        j+=1;
                }
                k+=1;
        }
        while(i<=mid){
                temp[k] = array[i];
                i+=1; k+=1;
        }
        while(j<=end){
                temp[k] = array[j];
                j+=1; k+=1;
        }
        for(i=begin, k=0; i<=end; i++,k++){
                array[i] = temp[k];
        }
}
}
class Driver{
private static Random random = new Random();
private static final int size = random.nextInt(100);
private static final Integer list[] = new Integer[size];
static {
for(int i=0; i<size; i++)
        list[i] = random.nextInt(size+(size-1))-(size-1);
}
}
public static void main(String[] args){
System.out.print("Input = [");
for (Integer each: list)
        System.out.print(each+", ");
System.out.print("] \n" +"Input.length = " + list.length + '\n');
Integer[] arr1 = Arrays.copyOf(list, list.length);
long t = System.currentTimeMillis();
Arrays.sort(arr1, (a,b)->a>b? 1: a==b? 0: -1);
t = System.currentTimeMillis() - t;
```

```
System.out.println("Time spent for system based Arrays.sort(): " + t + "ms");

Integer[] arr2 = Arrays.copyOf(list, list.length);

t = System.currentTimeMillis();

MergeSort.mergeSort(arr2, 0, arr2.length-1);

t = System.currentTimeMillis() - t;

System.out.println("Time spent for custom single threaded recursive merge_sort(): " + t + "ms");

Integer[] arr = Arrays.copyOf(list, list.length);

MergeSort.threadedSort(arr);

System.out.print("Output = [");

for (Integer each: arr)

        System.out.print(each+", ");

System.out.print("]\n");

}

}
```

Output:

```
Sorted array: 15 21 26 26 27 35 36 40 49 59 62 63 72 77 83 86 86 90 92 93

Time taken: 0.001023
```

Conclusion

Hence, Here we Implement merge sort and multithreaded merge sort. By Comparing time required by both the algorithms merge sort and multithreaded merge sort. We analyze the performance of each algorithm for the best case and the worst case.