Mini Project Report on

"Implement merge sort and multithreaded merge sort. Compare time required by both the algorithms. Also analyse the performance of each algorithm for the best case and the worst case"

Submitted *By*

Mr. Atul Thete [60]

In partial fulfilment of the requirements for The

award of the degree of

Bachelor in

COMPUTER ENGINEERING

For Academic Year 2023 – 2024

DEPARTMENT OF COMPUTER ENGINEERING

MET's Institute of Engineering Bhujbal Knowledge City Adgaon,

Nashik – 422003.

THE MET LEAGUE OF COLLEGES
MET
AS SHARP AS YOU CAN GET

Bhujbal Knowledge City

Institute of Engineering

Department of Computer Engineering

Certificate

This is to Certify That

Mr. Atul Thete

Has completed the necessary
Mini Project Work and Prepared the Report on

"**Implement merge sort and multithreaded merge sort. Compare time required by both the algorithms. Also analyse the performance of each algorithm for the best case and the worst case**"

in Satisfactorily manner as a fulfilment of the requirement of the award of degree of the Bachelor in Computer Engineering in the Academic Year

2023 – 2024

Project Guide	H.O.D	Principal
Prof. V.P.Wani	Dr. M. U. Kharat.	Dr. V. P. Wani

Acknowledgements

Every work is source which requires support from many people and areas. It gives us proud privilege to complete the Design And Analysis of Algorithms Mini Project on "**Implement merge sort and multithreaded merge sort. Compare time required by both the algorithms. Also analyse the performance of each algorithm for the best case and the worst case**" under valuable guidance and encouragement of our guide **Prof. V.P.Wani.**

We are also extremely grateful to our respected **H.O.D. Dr. M. U. Kharat** for providing all facilities and every help for smooth progress of our Mini Project. At last we would like to thank all the staff members and our students who directly or indirectly supported me without which the Mini Project work would not have been completed successfully.

by

Mr. Atul Govind Thete

Contents

Introduction

The project titled "Design And Analysis of Algorithms: Implementing Merge Sort and Multithreaded Merge Sort" delves into the realm of sorting algorithms, specifically focusing on Merge Sort and its enhanced variant, Multithreaded Merge Sort. Sorting is a fundamental operation in computer science, and this project aims to explore the efficiency and performance of these sorting algorithms.

1. Implementation of Merge Sort and Multithreaded Merge Sort:

The project kicks off with the detailed implementation of two sorting algorithms: Merge Sort and Multithreaded Merge Sort. Merge Sort is known for its efficiency and stability, while Multithreaded Merge Sort enhances its performance by parallelizing the sorting process using multiple threads.

2. Comparative Analysis:

The project doesn't stop at mere implementation but delves into a thorough comparative analysis. It measures and contrasts the time required by the two sorting algorithms. This analysis is crucial in determining which algorithm performs better in different scenarios and can be the basis for algorithm selection in practical applications.

3. Performance Analysis for Best and Worst Cases:

Furthermore, the project goes a step further by delving into the performance analysis of both algorithms for the best and worst cases. This in-depth examination helps understand how these algorithms behave under various conditions and provides insights into their strengths and weaknesses.

Problem Statement

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

Objectives

1. Implementation of Merge Sort and Multithreaded Merge Sort: The primary objective is to implement the Merge Sort algorithm and its multithreaded variant, Multithreaded Merge Sort, to gain a deep understanding of these sorting techniques.

2. Time Comparison: Compare the execution time of the two sorting algorithms—Merge Sort and Multithreaded Merge Sort—under various scenarios to determine which one is more efficient and under what conditions.

3. Performance Analysis - Best Case: Analyze the performance of both algorithms in the best-case scenario to understand their capabilities when dealing with already sorted or nearly sorted data.

4. Performance Analysis - Worst Case: Examine the performance of the sorting algorithms in the worst-case scenario, where the input data is in reverse order, to identify their limitations and inefficiencies.

Working

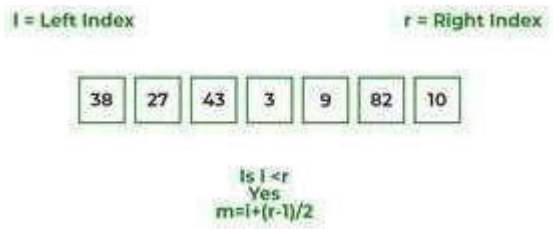
The Merge Sort algorithm is a sorting algorithm that is based on the Divide and Conquer paradigm. In this algorithm, the array is initially divided into two equal halves and then they are combined in a sorted manner.

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}

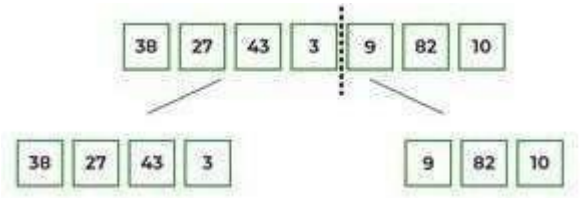
- At first, check if the left index of array is less than the right index, if yes then calculate its mid point



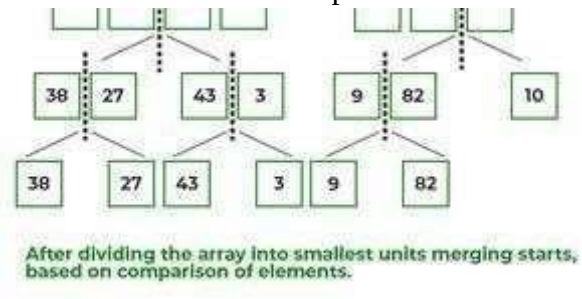
- Now, as we already know that merge sort first divides the whole array iteratively into equal halves, unless the atomic values are achieved.

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

- 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.

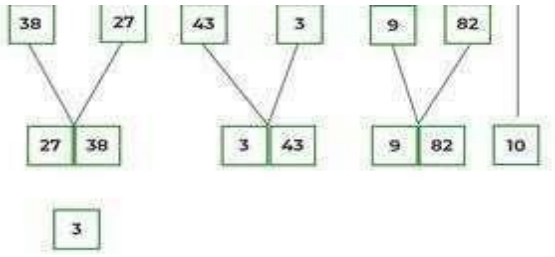


- Now, further divide these two arrays into further halves, until the atomic units of the array is reached and further division is not possible.

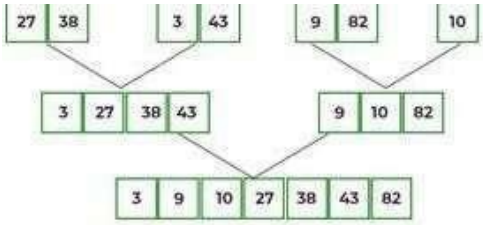


- After dividing the array into smallest units, start merging the elements again based on comparison of size of elements

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

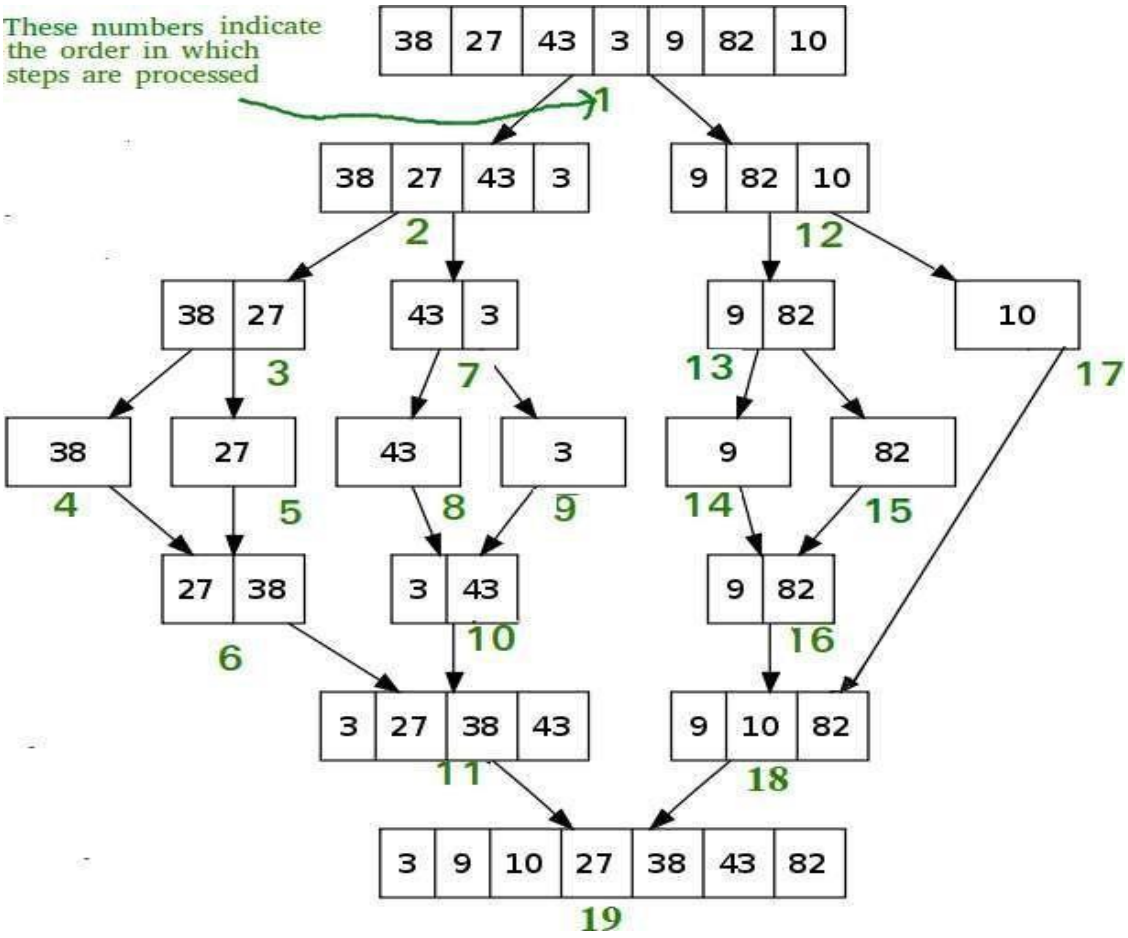


- After the final merging, the list looks like this:



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

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)

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 : 12, 11, 13, 5, 6, 7

Output : 5, 6, 7, 11, 12, 13.

Code

CPP Code for both Simple Merge Sort and Merge Sort Using Multi-Threading

```
#include <iostream>
#include <vector>
#include <thread>
#include <chrono>

void merge(std::vector<int>& arr, int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    std::vector<int> L(n1), R(n2);

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

    int 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(std::vector<int>& arr, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        merge(arr, left, mid, right);
    }
}

void mergeSortMultithread(std::vector<int>& arr, int left, int right, int depth) {
    if (depth == 0 || left >= right) {
        mergeSort(arr, left, right);
        return;
    }

    int mid = left + (right - left) / 2;

    std::thread leftThread(mergeSortMultithread, std::ref(arr), left, mid, depth - 1);
    std::thread rightThread(mergeSortMultithread, std::ref(arr), mid + 1, right, depth - 1);

    leftThread.join();
    rightThread.join();

    merge(arr, left, mid, right);
}

int main() {
    std::vector<int> arr = {83, 86, 77, 15, 93, 35, 86, 92, 49, 21, 62, 27, 90, 59, 63, 26, 40,
26, 72, 36};
    int arrSize = arr.size();

    auto startSimple = std::chrono::high_resolution_clock::now();
    mergeSort(arr, 0, arrSize - 1);
    auto endSimple = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> durationSimple = endSimple - startSimple;

    std::cout << "Sorted array (Simple Merge Sort): ";
```

```
    for (int num : arr)
        std::cout << num << " ";
    std::cout << std::endl;

    std::cout << "Time taken by Simple Merge Sort: " << durationSimple.count() << "
seconds" << std::endl;

    arr = {83, 86, 77, 15, 93, 35, 86, 92, 49, 21, 62, 27, 90, 59, 63, 26, 40, 26, 72, 36  };

    auto startMultithread = std::chrono::high_resolution_clock::now();
    mergeSortMultithread(arr, 0, arrSize - 1, 2); // Using 2 threads
    auto endMultithread = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> durationMultithread = endMultithread -
startMultithread;

    std::cout << "Sorted array (Multithreaded Merge Sort): ";
    for (int num : arr)
        std::cout << num << " ";
    std::cout << std::endl;

    std::cout << "Time taken by Multithreaded Merge Sort: " <<
durationMultithread.count() << " seconds" << std::endl;

    return 0;
}
```

Output

```
Sorted array (Simple Merge Sort): 15 21 26 26 27 35 36 40 49 59 62 63 72 77 83 86 86 90 92
93
Time taken by Simple Merge Sort: 1.3668e-05 seconds

Sorted array (Multithreaded Merge Sort): 15 21 26 26 27 35 36 40 49 59 62 63 72 77 83 86
86 90 92 93
Time taken by Multithreaded Merge Sort: 0.000288734 seconds
```

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.