**Pune Institute of Computer Technology**

Shape

Description automatically generated with medium confidence

**Department of Computer Engineering**

(2022- 2023)

# **“Implement merge sort and multithreaded merge sort.”**

**Submitted to the**

**Savitribai Phule Pune University**

**In partial fulfilment for the award of the Degree of**

**Bachelor of Engineering**

**in**

**Computer Engineering**

**By**

1. **Tushar Patil 41354**
2. **Saurabh Sahare 41364**
3. **Samyak Samdariya 41365**

Under the guidance of

**Prof. Priyanka Savdekar**

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **TITLE** | **Page no** |
| **1.** | Introduction | **3** |
| **2.** | Problem Statement | **4** |
| **3.** | Objectives | **4** |
| **4.** | Theory | **4** |
| **5.** | Conclusion | **15** |
| **6.** | References | **15** |

**Introduction**

## **Merge Sort**

## Merge Sort is one of the most popular sorting algorithms that is based on the principle of Divide and Conquer Algorithm. . It divides the given list into two equal halves, calls itself for the two halves and then merges the two sorted halves.

**Steps in merge sort:**

Suppose we had to sort an array A. A subproblem would be to sort a sub-section of this array starting at index p and ending at index r, denoted as A[p..r].

Divide:

If q is the half-way point between p and r, then we can split the subarray A[p..r] into two arrays A[p..q] and A[q+1, r].

Conquer:

In the conquer step, we try to sort both the subarrays A[p..q] and A[q+1, r]. If we haven't yet reached the base case, we again divide both these subarrays and try to sort them.

Combine:

When the conquer step reaches the base step and we get two sorted subarrays A[p..q] and A[q+1, r] for array A[p..r], we combine the results by creating a sorted array A[p..r] from two sorted subarrays A[p..q] and A[q+1, r]

## **Multi-Threading**

A thread is a single sequential flow of execution of tasks of a process so it is also known as thread of execution or thread of control.

**Multithreading is the ability of a central processing unit (CPU) (or a single core in a multi-core processor) to provide multiple threads of execution concurrently, supported by the operating system.**

[Merge sort](https://www.prowaretech.com/articles/current/c-plus-plus/algorithms/merge-sort) is a good design for multi-threaded sorting because it allocates sub-arrays during the merge procedure thereby avoiding data collisions. This implementation breaks the array up into separate ranges and then runs its algorithm on each of them, but the data must be merged (sorted) in the end by the main thread. The more threads there are, the more unsorted the second to last array is thereby causing the final merge to take longer!!

**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.

**Objective**

Implement merge sort and multi-threaded merge sort. Compare their time complexities and analyze performance.

**Theory**

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.

## 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*

*Multi-threaded Merge sort*

## *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.*

## **For Example-:**

**In**−int arr[] = {3, 2, 1, 10, 8, 5, 7, 9, 4}

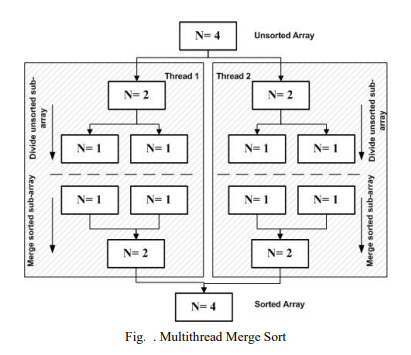
**Out**−Sorted array is: 1, 2, 3, 4, 5, 7, 8, 9, 10

**Explanation**−we are given an unsorted array with integer values. Now we will sort the array using merge sort with multithreading.

**In**−int arr[] = {5, 3, 1, 45, 32, 21, 50}

**Out**−Sorted array is: 1, 3, 5, 21, 32, 45, 50

**Explanation**−we are given an unsorted array with integer values. Now we will sort the array using merge sort with multithreading.



**CODE :**

**Merge Sort**

#include <iostream>

using namespace std;

void merge(int a[], int beg, int mid, int end)

{

int i, j, k;

int n1 = mid - beg + 1;

int n2 = end - mid;

int LeftArray[n1], RightArray[n2];

for (int i = 0; i < n1; i++)

LeftArray[i] = a[beg + i];

for (int j = 0; j < n2; j++)

RightArray[j] = a[mid + 1 + j];

i = 0;

j = 0;

k = beg;

while (i < n1 && j < n2)

{

if(LeftArray[i] <= RightArray[j])

{

a[k] = LeftArray[i];

i++;

}

else

{

a[k] = RightArray[j];

j++;

}

k++;

}

while (i<n1)

{

a[k] = LeftArray[i];

i++;

k++;

}

while (j<n2)

{

a[k] = RightArray[j];

j++;

k++;

}

}

void mergeSort(int a[], int beg, int end)

{

if (beg < end)

{

int mid = (beg + end) / 2;

mergeSort(a, beg, mid);

mergeSort(a, mid + 1, end);

merge(a, beg, mid, end);

}

}

void printArray(int a[], int n)

{

int i;

for (i = 0; i < n; i++)

cout<<a[i]<<" ";

}

int main()

{

int a[] = { 11, 30, 24, 7, 31, 16, 39, 41 };

int n = sizeof(a) / sizeof(a[0]);

cout<<"Before sorting array elements are - \n";

printArray(a, n);

mergeSort(a, 0, n - 1);

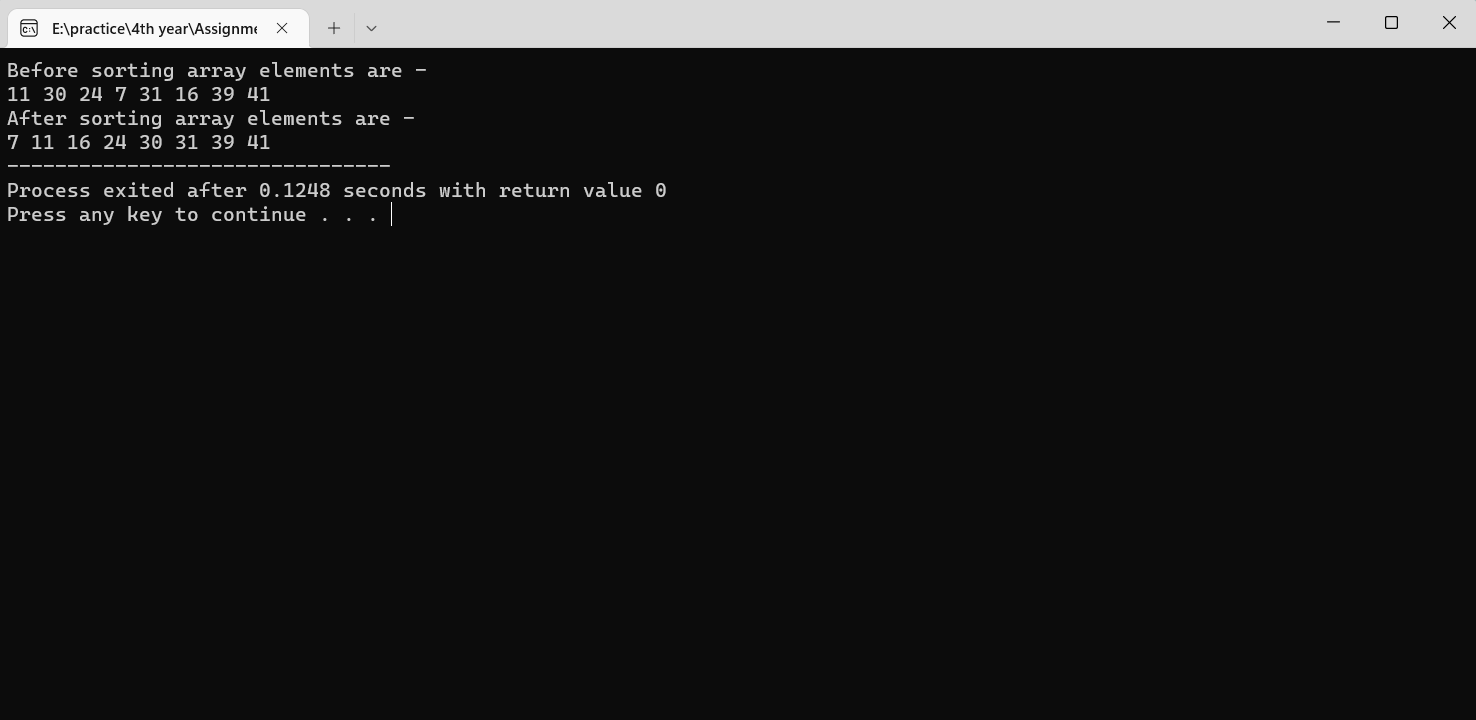
cout<<"\nAfter sorting array elements are - \n";

printArray(a, n);

return 0;

}

OUTPUT :



**Multi-threaded Merge Sort**

#include <iostream>

#include <pthread.h>

#include <time.h>

#include<bits/stdc++.h>

#define MAX 20

#define THREAD\_MAX 4

using namespace std;

int a[MAX];

int part = 0;

// merge function for merging two parts

void merge(int low, int mid, int high)

{

int\* left = new int[mid - low + 1];

int\* right = new int[high - mid];

// n1 is size of left part and n2 is size

// of right part

int n1 = mid - low + 1, n2 = high - mid, i, j;

// storing values in left part

for (i = 0; i < n1; i++)

left[i] = a[i + low];

// storing values in right part

for (i = 0; i < n2; i++)

right[i] = a[i + mid + 1];

int k = low;

i = j = 0;

// merge left and right in ascending order

while (i < n1 && j < n2) {

if (left[i] <= right[j])

a[k++] = left[i++];

else

a[k++] = right[j++];

}

// insert remaining values from left

while (i < n1) {

a[k++] = left[i++];

}

// insert remaining values from right

while (j < n2) {

a[k++] = right[j++];

}

}

// merge sort function

void merge\_sort(int low, int high)

{

// calculating mid point of array

int mid = low + (high - low) / 2;

if (low < high) {

// calling first half

merge\_sort(low, mid);

// calling second half

merge\_sort(mid + 1, high);

// merging the two halves

merge(low, mid, high);

}

}

// thread function for multi-threading

void\* merge\_sort(void\* arg)

{

// which part out of 4 parts

int thread\_part = part++;

// calculating low and high

int low = thread\_part \* (MAX / 4);

int high = (thread\_part + 1) \* (MAX / 4) - 1;

// evaluating mid point

int mid = low + (high - low) / 2;

if (low < high) {

merge\_sort(low, mid);

merge\_sort(mid + 1, high);

merge(low, mid, high);

}

}

// Driver Code

int main()

{

// generating random values in array

for (int i = 0; i < MAX; i++)

a[i] = rand() % 100;

// t1 and t2 for calculating time for

// merge sort

clock\_t t1, t2;

t1 = clock();

pthread\_t threads[THREAD\_MAX];

// creating 4 threads

for (int i = 0; i < THREAD\_MAX; i++)

pthread\_create(&threads[i], NULL, merge\_sort,

(void\*)NULL);

// joining all 4 threads

for (int i = 0; i < 4; i++)

pthread\_join(threads[i], NULL);

// merging the final 4 parts

merge(0, (MAX / 2 - 1) / 2, MAX / 2 - 1);

merge(MAX / 2, MAX/2 + (MAX-1-MAX/2)/2, MAX - 1);

merge(0, (MAX - 1)/2, MAX - 1);

t2 = clock();

// displaying sorted array

cout << "Sorted array: ";

for (int i = 0; i < MAX; i++)

cout << a[i] << " ";

// time taken by merge sort in seconds

cout << "Time taken: " << (t2 - t1) /

(double)CLOCKS\_PER\_SEC << endl;

return 0;

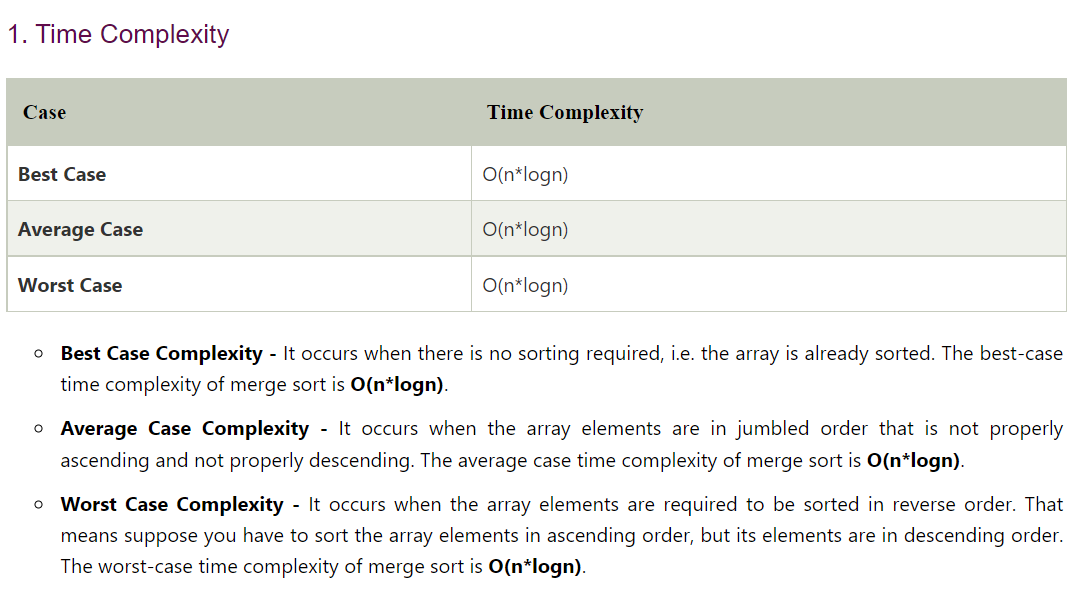
}

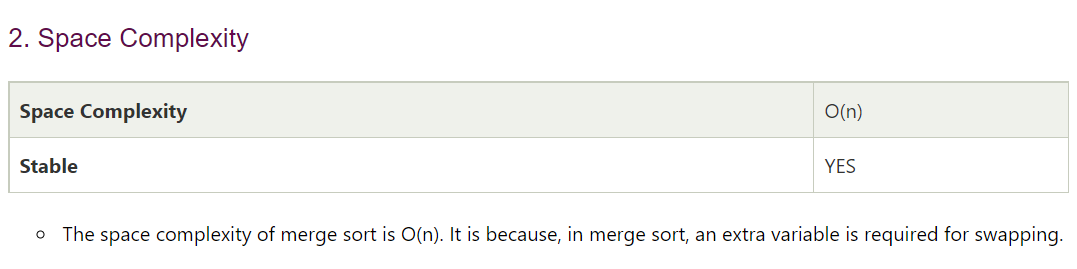
OUTPUT :



**Time Complexity and Performance**

**Merge Sort**





**Multi-threaded Merge Sort**

Multithread merge sort, creates thread recursively, and stops work when it reaches a certain size, with each thread locally sorting its data. Then threads merge their data by joining threads into a sorted main list. The multithread merge sort that have array of 4 elements to be sorted. Merge sort in multithread is based on the fact that the recursive calls run in parallel, so there is only one n/2 term with the time complexity (2): T(n) = Θ log(n) + Θ(n) = Θ(n)

**Conclusion**

Thus, We have implemented and compared time complexity and analysed performance of the Merge Sort and Multi-threaded Merge Sort.

**References**

* <https://www.tutorialspoint.com/merge-sort-using-multithreading-in-cplusplus>
* https://www.geeksforgeeks.org/merge-sort-using-multi-threading/
* <https://www.geeksforgeeks.org/merge-sort/>
* <https://en.wikipedia.org/wiki/Merge_sort>
* https://www.javatpoint.com/merge-sort