Sanghavi College Of Engineering,Nashik

Shree Mahavir Education Society's

Department of Computer Engineering

(2023- 2024)

"Implement Merge Sort and Multithreaded Merge Sort."

Submitted to the

Savitribai Phule Pune University

In partial fulfillment for the award of the Degree of

Bachelor of Engineering

in

Computer Engineering

By

1) Asjad Samdani	**72159955K**
2) Anjali Kumawat	**72221939D**
3) Sakshi Kathe	**72221943B**

Under the guidance of

Prof. Swati Hinge

<u>CONTENTS</u>

Introduction

Merge Sort

Merge sort is a sorting technique which is based on divide and conquer technique where we divide the array into equal halves and then combine them in a sorted manner.

Algorithm to implement merge sort is
- check if there is one element in the list then return the element.
- Else, Divide the data recursively into two halves until it can't be divided further.
- Finally, merge the smaller lists into new lists in sorted order.

In merge sort, the problem is divided into two subproblems in every iteration. Hence efficiency is increased drastically.

It follows the divide and conquer approach

     Divide break the problem into 2 subproblem which continues until the problem set is left with one element only

     Conquer basically merges the 2 sorted arrays into the original array

Multi-Threading

In the operating system, **Threads** are the lightweight process which is responsible for executing the part of a task. Threads share common resources to execute the task concurrently.

Multi-threading is an implementation of multitasking where we can run multiple threads on a single processor to execute the tasks concurrently. It subdivides specific operations within a single application into individual threads. Each of the threads can run in parallel.

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!!

<h1 align="center"><u>Problem Statement</u></h1>

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.

<h1 align="center"><u>Objective</u></h1>

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

<h1 align="center"><u>Theory</u></h1>

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.
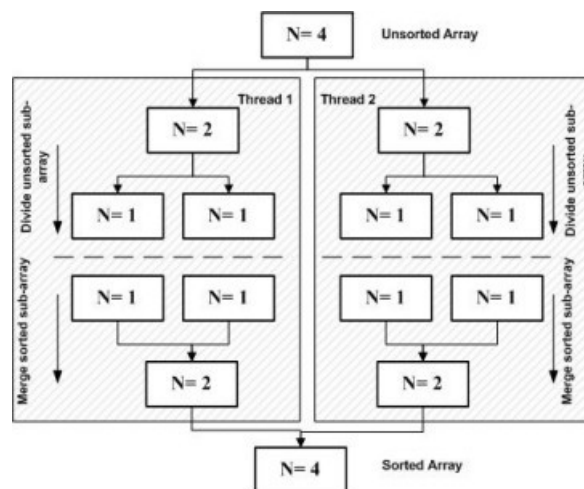

Fig. . Multithread Merge Sort

CODE :

Merge Sort

```java
public        class
Merge {
   public static void print Array(int array[]) {
       for (int i = 0; i < array.length; i++) {
          System.out.print(array[i] + " ");
   }
      System.out.println();
 }

  public static void mergeSort(int array[], int si, int ei) {
   if (si >= ei) {
   return;

}
   int mid = si + (ei - si) / 2;
    mergeSort(array, si, mid);
   mergeSort(array, mid + 1, ei);
    merge(array, si, mid, ei);
}

 public static void merge(int array[], int si, int mid, int ei) {
 int temp[] = new int[ei - si + 1];
 int i = si;
 int j = mid + 1;
 int k = 0;
 while (i <= mid && j <= ei) {
 if (array[i] < array[j]) {
 temp[k] = array[i];
     i++;
   } else {
        temp[k] = array[j];
              j++;
}
  k++;
```

```java
    }
      while (i <= mid) {
        temp[k++] = array[i++];

    }
    while (j <= ei) {
      temp[k++] = array[j++];

}
      for (k = 0, i = si; k < temp.length; k++, i++) {
        array[i] = temp[k];
    }

}

      public static void main(String[] args) {
        int array[] = { 4, 9, 5, 1, 6, 10, 8, 3, 11 };
        mergeSort(array, 0, array.length-1);
        printArray(array);

    }

    }
```

```
/tmp/lIn8LlSXbh.o
enter array elements -
10
52
7
9
3
15
Sorted array:
3 7 9 10 15 52
```

Multi-threaded Merge Sort

```java
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<>();
```

```java
            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;
            }
```

```java
                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");

        // Test custom single-threaded merge sort (recursive merge) implementation
        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   :

```
/tmp/zcFhlrT2Jt.o
Multi-threading merge sort -
enter array elements
40
29
50
39
27
18
Sorted array:
18 27 29 39 40 50
```

Time Complexity and Performance

Merge Sort

1. Time Complexity

Case	Time Complexity
Best Case	O(n*logn)
Average Case	O(n*logn)
Worst Case	O(n*logn)

- **Best Case Complexity** - It occurs when there is no sorting required, i.e. the array is already sorted. The best-case time complexity of merge sort is **O(n*logn)**.

- **Average Case Complexity** - It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. The average case time complexity of merge sort is **O(n*logn)**.

- **Worst Case Complexity** - It occurs when the array elements are required to be sorted in reverse order. That means suppose you have to sort the array elements in ascending order, but its elements are in descending order. The worst-case time complexity of merge sort is **O(n*logn)**.

2. Space Complexity

Space Complexity	O(n)
Stable	YES

- The space complexity of merge sort is O(n). It is because, in merge sort, an extra variable is required for swapping.

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): $T(n) = \Theta \log(n) + \Theta(n) = \Theta(n)$

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-cplus plus

- 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