Report on

“Design And Analysis Of Algorithm”

B.E. [Computer Engineering]

Submitted By

Shivani Jadhav	**41**
Vaishnavi Garghate	**42**
Siddhi Shimpi	**43**
Priti Aher	**44**

Under the guidance of

Prof. Priya Rakibe

Academic Year: 2022-2023

Department of Computer Engineering

[2022 - 2023]

Table of Contents

Sr. No.	Content	Page No.
1.	Problem Statement	1
2.	Introduction	1
3.	Merge Sort	1
4.	Merge Sort Using Multithreading	3
5.	Implementation	5
6.	Conclusion	9

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.

Introduction

Merge sort:

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 is similar to the quick sort algorithm as it uses the divide and conquer approach to sort the elements. It is one of the most popular and efficient sorting algorithms. It divides the given list into two equal halves, calls itself for the two halves and then merges the two sorted halves. We have to define the merge() function to perform the merging. The sub-lists are divided again and again into halves until the list cannot be divided further. Then we combine the pair of one element lists into two-element lists, sorting them in the process. The sorted two-element pairs is merged into the four-element lists, and so on until we get the sorted list.

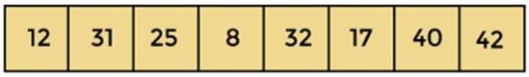
Algorithm Of Merge Sort:

MERGE_SORT(arr, beg, end)
if beg < end
set mid = (beg + end)/2
MERGE_SORT(arr, beg, mid)
MERGE_SORT(arr, mid + 1, end)
MERGE (arr, beg, mid, end)
end of **if**
END MERGE_SORT

Working of Merge Sort:

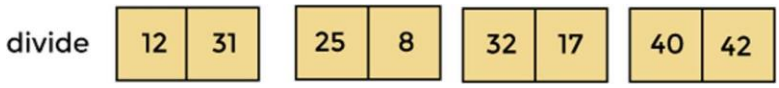
To understand the working of the merge sort algorithm, let's take an unsorted array. It will be easier to understand the merge sort via an example.

Let the elements of array are -



According to the merge sort, first divide the given array into two equal halves. Merge sort keeps dividing the list into equal parts until it cannot be further divided.As there are eight elements in the given array, so it is divided into two arrays of size 4.

Now, again divide these two arrays into halves. As they are of size 4, so divide them into new arrays of size 2.

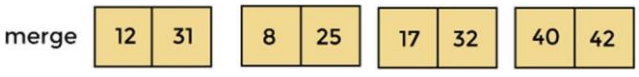


Now, again divide these arrays to get the atomic value that cannot be further divided.

Now, combine them in the same manner they were broken.

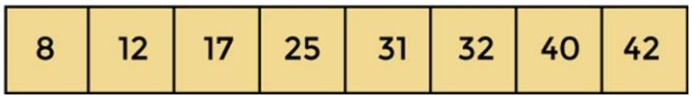
In combining, first compare the element of each array and then combine them into another array in sorted order.

So, first compare 12 and 31, both are in sorted positions. Then compare 25 and 8, and in the list of two values, put 8 first followed by 25. Then compare 32 and 17, sort them and put 17 first followed by 32. After that, compare 40 and 42, and place them sequentially.



In the next iteration of combining, now compare the arrays with two data values and merge them into an array of found values in sorted order.

Now, there is a final merging of the arrays. After the final merging of above arrays, the array will look like -



Now, the array is completely sorted.

Time and Space Complexity:

Time Complexity:

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

Space Complexity:

Space Complexity	O(n)
Stable	YES

Merge Sort Using Multi-Threading:

Multi-Threading:

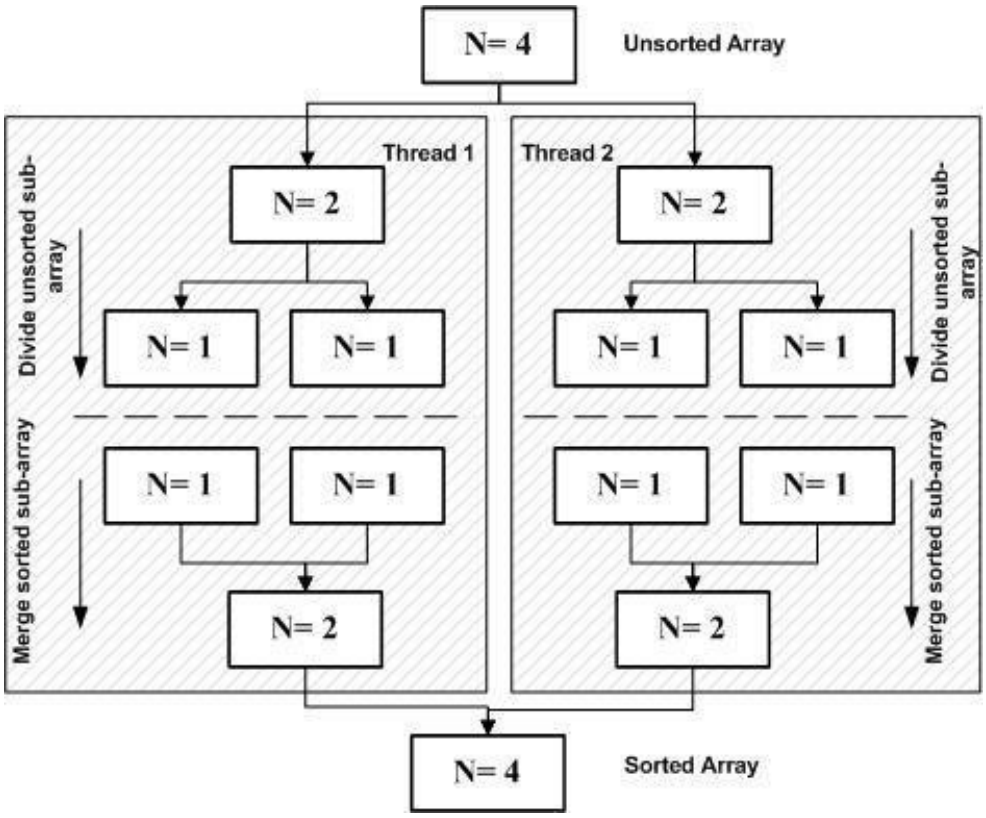
In the operating system, **Threads** are the lightweight process which is responsible for executing the part of a task. Threads share common resources to execute the task concurrently. **Multi-threading** is an implementation of multitasking where we can run multiple threads on a single processor to execute the tasks concurrently. It subdivides specific operations within a single application into individual threads. Each of the threads can run in parallel.

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.



Implementation

Merge Sort:

```
#include <iostream>
using namespace std;
void merge(int array[], int const left, int const mid,int const right)
{
        auto const subArrayOne = mid - left + 1;
        auto const subArrayTwo = right - mid;
        auto *leftArray = new int[subArrayOne],
                *rightArray = new int[subArrayTwo];
        for (auto i = 0; i < subArrayOne; i++)
                leftArray[i] = array[left + i];
        for (auto j = 0; j < subArrayTwo; j++)
                rightArray[j] = array[mid + 1 + j];
        auto indexOfSubArrayOne= 0, // Initial index of first sub-array
                indexOfSubArrayTwo= 0; // Initial index of second sub-array
        int indexOfMergedArray= left; // Initial index of merged array
        while (indexOfSubArrayOne < subArrayOne
                && indexOfSubArrayTwo < subArrayTwo) {
                if (leftArray[indexOfSubArrayOne]
                        <= rightArray[indexOfSubArrayTwo]) {
                        array[indexOfMergedArray]
                                = leftArray[indexOfSubArrayOne];
                        indexOfSubArrayOne++;
                }
                else {
                        array[indexOfMergedArray]=rightArray[indexOfSubArrayTwo];
                        indexOfSubArrayTwo++;
                }
                indexOfMergedArray++;
        }
        while (indexOfSubArrayOne < subArrayOne) {
```

```
                array[indexOfMergedArray]= leftArray[indexOfSubArrayOne];
                indexOfSubArrayOne++;
                indexOfMergedArray++;
        }
        while (indexOfSubArrayTwo < subArrayTwo) {
                array[indexOfMergedArray]= rightArray[indexOfSubArrayTwo];
                indexOfSubArrayTwo++;
                indexOfMergedArray++;
        }
        delete[] leftArray;
        delete[] rightArray;
}
void mergeSort(int array[], int const begin, int const end)
{
        if (begin >= end)
                return; // Returns recursively
        auto mid = begin + (end - begin) / 2;
        mergeSort(array, begin, mid);
        mergeSort(array, mid + 1, end);
        merge(array, begin, mid, end);
}
void printArray(int A[], int size)
{
        for (auto i = 0; i < size; i++)
                cout << A[i] << " ";
}
int main()
{
        int arr[] = { 12, 11, 13, 5, 6, 7 };
        auto arr_size = sizeof(arr) / sizeof(arr[0]);
        cout << "Given array is \n";
        printArray(arr, arr_size);
        mergeSort(arr, 0, arr_size - 1);
      cout << "\nSorted array is \n";
        printArray(arr, arr_size);
```

```
return 0;}
```

Output:

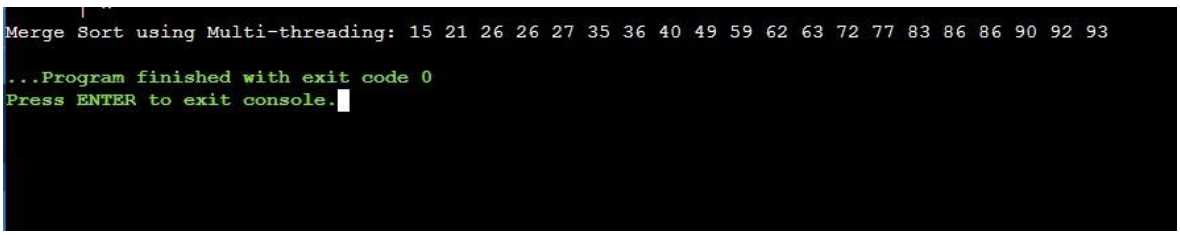
Merge Sort using Multi-Threading:

```
#include <iostream>
#include <pthread.h>
#include <time.h>
#define size 20
#define thread_size 4
using namespace std;
int arr[size];
int temp_val = 0;
void combine_array(int first, int mid_val, int end){
  int* start = new int[mid_val - first + 1];
  int* last = new int[end - mid_val];
  int temp_1 = mid_val - first + 1;
  int temp_2 = end - mid_val;
  int i, j;
  int k = first;
  for(i = 0; i < temp_1; i++){
    start[i] = arr[i + first];
  }
  for (i = 0; i < temp_2; i++){
    last[i] = arr[i + mid_val + 1];
  }
  i = j = 0;
```

```
  while(i < temp_1 && j < temp_2){
    if(start[i] <= last[j]){
      arr[k++] = start[i++];
    }
    else{
      arr[k++] = last[j++];
    }
  }
  while (i < temp_1){
    arr[k++] = start[i++];
  }
  while (j < temp_2){
    arr[k++] = last[j++];
  }
}
void Sorting_Threading(int first, int end){
  int mid_val = first + (end - first) / 2;
  if(first < end){
    Sorting_Threading(first, mid_val);
    Sorting_Threading(mid_val + 1, end);
    combine_array(first, mid_val, end);
  }
}
void* Sorting_Threading(void* arg){
  int set_val = temp_val++;
  int first = set_val * (size / 4);
  int end = (set_val + 1) * (size / 4) - 1;
  int mid_val = first  + (end  - first) /  2;
  if (first < end){
    Sorting_Threading(first, mid_val);
    Sorting_Threading(mid_val + 1, end);
    combine_array(first, mid_val, end);
  }
```

```
}
int main(){
   for(int i = 0; i < size; i++){
      arr[i] = rand() % 100;
   }
   pthread_t P_TH[thread_size];
   for(int i = 0; i < thread_size; i++){
      pthread_create(&P_TH[i], NULL, Sorting_Threading, (void*)NULL);
   }
   for(int i = 0; i < 4; i++){
      pthread_join(P_TH[i], NULL);
   }
   combine_array(0, (size / 2 - 1) / 2, size / 2 - 1);
   combine_array(size / 2, size/2 + (size-1-size/2)/2, size - 1);
   combine_array(0, (size - 1)/2, size - 1);
   cout<<"Merge Sort using Multi-threading: ";
   for (int i = 0; i < size; i++){
      cout << arr[i] << " ";
   }
   return 0;
}
```

Output:



Conclusion

Thus, we have successfully implemented merge sort and multithreaded merge sort and compared time required by both the algorithms.