OPTIMAL MERGE PATTERN

C2_ASSIGNMENT_6

GROUP MEMBERS :

PRITIK SRIVASTAVA (IIT2019192)

CHETAN PATIDAR (IIT2019193)

RAHUL ROY (IIT2019194)

Abstract

Given n number of sorted files, the task is to find the minimum computations done to reach Optimal Merge Pattern. When two or more sorted files are to be merged all together to form a single file, the minimum computations done to reach this file are known as **Optimal Merge Pattern.**

The project aims at :Solving the **Optimal Merge Pattern** problem ,implementing the approach by maximum time and space utillisation and analysing it practically.

Contents

INTRODUCTION

We are given n sorted files and we need to find the total number of minimum computations required to reach Optimal Merge Pattern. If more than 2 files need to be merged then it can be done in pairs. For example, if need to merge 4 files A, B, C, D. First Merge A with B to get X1, merge X1 with C to get X2, merge X2 with D to get X3 as the output file.

We will solve the given problem using greedy approach

This report further contains -

II. Algorithm Design

III. Algorithm Analysis

IV. Experimental Study

V. Conclusion

ALGORITHM DESIGN

- If we have two files of sizes m and n, the total computation time will be m+n. Here, we use greedy strategy by merging two smallest size files among all the files present.

- Firstly we add all the nodes in a priority queue (Min Heap) where node.weight = file size and given nodes are greater than 2
  Then we initiate the process by initializing count = 0 which is the variable to store file computations.

- Following the algorithm:

- An array of files is created storing the size of each file in a sorted manner.

- The priority queue is created to handle each iteration over each file.
- In each iteration we find the two files with minimum sizes and merge them into one and store it.
- Then in every next iteration we merge the current file into the previous merged file
- In this way we compute the total number of iterations required to reach optimal merge pattern.
- **For example-**
- **Consider input array as follows:-**
- **Input:** n = 3, size = {2, 3, 4}
- Firstly we merge 2 and 3 into one file of size 5. Then the next is 4. So we merge file with size 4 with the previously merged file of size 5 to get 4+5=9.
- Hence total cost = 14

ALGORITHM DESIGN

- we create a new node such that :
- new node = pq.poll().weight+pq.poll().weight;

where pq denotes priority queue, remove 1st smallest and 2nd smallest element and add their weights to get a new node.

- count = count + node.weight
- add this new node to priority queue;

PROGRAM

- **Algorithm** minimumComputation
- **Passed** vector<int> files

```
• while(!q.empty())
•     {
•         ll x=-1*q.top();

•         q.pop();

•         if(q.empty())
•               break;

            ll y=-1*q.top();

•           q.pop();

•           ans+=x+y;
•
•           q.push(  -1*(x+y) );
•

•   }
•



•
```

ALGORITHM ANALYSIS

A. Time complexity

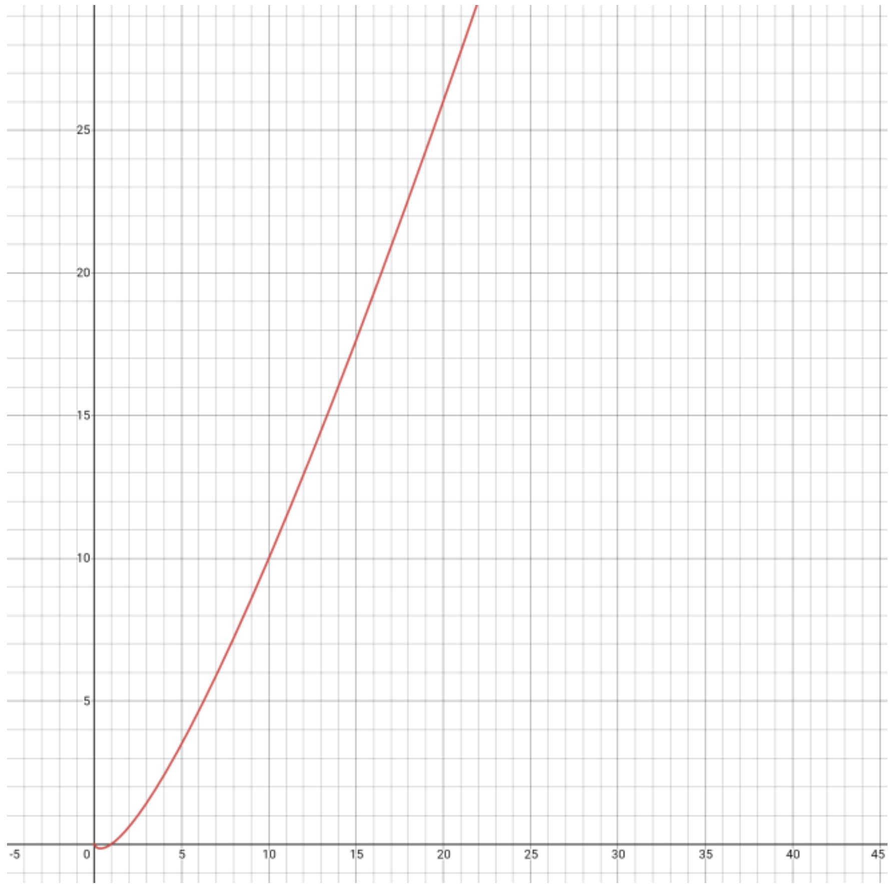
- The **A**lgorithm uses minheap to provide a greedy solution.
- The algorithm takes into account the two minimum elements or file sizes present in the array or vector in O(logn) and then iterates for n-1 elements to find this which takes O(n).
- So overall time complexity of the algorithm becomes O(nlogn).

B. Space complexity

- The space complexity of the program is **O(n)** as we store the size of n files.

EXPERIMENTAL STUDY

From the graph plotted between the n(no. of rows :x-axis), m(no. of columns:y-axis) and time(z-axis) we can conclude the **O(nlogn)** behaviour of the algorithm



CONCLUSION

- In this presentation we proposed an algorithm to find the, minimum computations done to reach Optimal Merge Pattern given n number of sorted files. The experimental study showed that the time complexity of the algorithm using greedy programming is **O(n*log(n))** which depends on the number of the sorted files
-

Reference

[1] https://www.geeksforgeeks.org

[2] Introduction to Algorithms by Thomas H.cormen