Assignment 7 - To Store the given n files in given sequential storage device such that the mean retrieval time is minimized.

Parag Parihar
Roll No:- IIT2016095
iit2016095@iiita.ac.in

Rakshit Sai
Roll No:- IIT2016126
iit2016126@iiita.ac.in

Adarsh Agrawal
Roll No:- IIT2016516
iit2016516@iiita.ac.in

Nilotpal Pramanik
Roll No:- IRM2016501
irm2016501@iiita.ac.in

***Abstract*—For a given sequential storage device, a set of n files $f_1, f_2, , f_n$ with corresponding lengths $l_1, l_2, , l_n$ and frequency of retrieval $r_1, r_2, , r_n$ respectively. Store the files such that the mean retrieval time is minimized.**

I. Introduction and Literature Survey

In the given problem we will have a storage device which stores files. Let us say it stores n files $f_1, f_2, ...f_n$ and each file is of length $l_1, l_2, ...l_n$ and their retrieval times is $r_1, r_2, ...r_n$ respectively. We have to store the given files in such a way that the retrieval time is least. So, this is a problem where we have think of the way or the order in which we store the files in the storage device. The problem seems like we need to have futuristic approach on how we solve it for a given set of files. The task is to write an algorithm which satisfies the above condition and then do proper analysis of the algorithm, calculate the time complexity and then plot the graph of time vs no. of files.

The report contains :-
2. Algorithm Design
3. Analysis and Discussion
4. Experimental Analysis
5. Conclusion

II. Algorithm Design

To solve the given problem we are using Greedy Algorithms.

This example gives us our first greedy algorithm. To minimize the total expected cost of accessing the files, we put the file that is cheapest to access first, and then write everything else; no backtracking, no dynamic programming, just make the best local choice and blindly plow ahead. If we use an efficient sorting as heap sort algorithm, the running time is clearly $O(nlog(n))$, plus the time required to actually write the files.To prove the greedy algorithm is actually correct, we simply prove that the output of any other algorithm can be improved by some sort of swap.

Let us generalize this idea further. Suppose we are also given an array $r[1, ..., n]$ of retrieval frequencies for each file; file i will be accessed exactly $r[i]$ times over the lifetime of the sequential storage device. Now the total cost of accessing all the files on the device is

$$\sum cost(\pi) = \sum_{k=1}^{n} \left(r(\pi(k)) * \sum_{i=1}^{k} l(\pi(i)) \right)$$

or

$$\sum cost(\pi) = \sum_{k=1}^{n} \sum_{i=1}^{n} (r(\pi(k)) * l(\pi(i)))$$

where $\pi(i)$ denote the index of the file stored at position i on the device.
Now what order should store the files if we want to minimize the total cost? There may be three conditions:-

1. **when frequency of retrieval is same**- We can say that if all the frequencies are equal, then we should sort the files by increasing size.

2. **when length of files are same**- If the frequencies are all different but the file lengths are all equal, then intuitively, we should sort the files by decreasing access frequency, with the most-accessed file first.

3.**when both are different**- if the sizes and the frequencies are both different? In this case, we should sort the files by the ratio $\frac{L}{r}$.

In our algorithm, we have calculated **ratio** array which is ratio of **l** and **r** array. Now, we will sort our **ratio** array in increasing order and according to ratio array we will update our **l**(length array) and **r**(frequency array) using **heap sort**.Now using discussed formula we can calculate minimized cost to retrieve all the files given no. of times.

Algorithm 1 Heapify function

INPUT: ratio array, n(length of array), i
OUTPUT: Max Heapify our heap
$largest \leftarrow i$
$l \leftarrow 2 * i + 1$
$r \leftarrow 2 * i + 2$
if l $<$ n and ratio[l] $>$ ratio[largest] **then**
    $largest \leftarrow l$
end if
if r $<$ n and ratio[r] $>$ ratio[largest] **then**
    $largest \leftarrow r$
end if
if $largest \neq i$ **then**
    swap(i, largest)
    Heapify(n, largest)
end if

Algorithm 2 HeapSort function

INPUT: ratio array, n(length of array)
OUTPUT: Order the l(length) array and f(frequency) array according to ratio array(i.e, ratio array should be in increasing order)
$flag \leftarrow 1$
for $i = n/2 - 1$ *to* 0 **do**
    Heapify(ratio, n, i)
end for
for $i = n - 1$ *to* 0 **do**
    swap(0, i)
    Heapify(ratio, n, i)
end for

Algorithm 3 Swap function

INPUT: a,b
swap ratio[a] and ratio[b]
swap l[a] and l[b]
swap f[a] and f[b]

Algorithm 4 Main Function

Scan the value of n(no. of files) and length array(l) and frequency array(f)
$flag \leftarrow 1$
$min\_cost \leftarrow 0$
for $i = 0$ *to* n **do**
    $ratio[i] \leftarrow \frac{l[i]}{f[i]}$
end for
HeapSort(ratio, n)
$pre\_compute[0] \leftarrow l[0]$
for $i = 1$ *to* n **do**
    $pre\_compute[i] \leftarrow l[i] + pre\_compute[i-1]$
end for
for $i = 0$ *to* n **do**
    $cost \leftarrow cost + f[i] * pre\_compute[i]$
end for
print **min_cost**

III. Analysis and Discussion

In this section we have analysed all the cases including time complexity and graphs and notations.

A. *Time-Complexity*——-

*1) **Best-Case**:* For the best case, we are assuming that length array and frequency array will be in such a way that generated ratio array will be in ascending order in that case, in this case our heap sort time complexity will be proportional to n. So our best case time complexity will be $O(n)$.
Time complexity for best case is $O(n)$

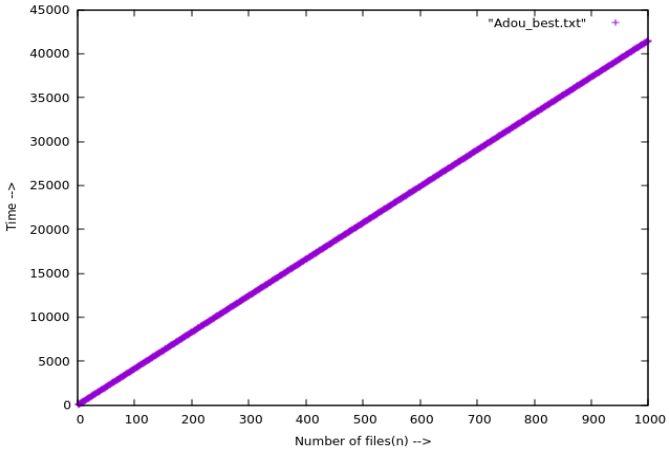

Fig.1 Number of files(n) vs Time

*2) **Worst-Case**:* As far as worst case is concerned, we are assuming that length array and frequency array will be in such a way that in generated ratio array, all elements will be distinct.And now we will sort ratio array in ascending order, So for worst case heap sort will take time $O(nlogn)$ which is obvious worst case time complexity of heap sort. Overall time complexity of worst case is:- $O(nlogn)$.

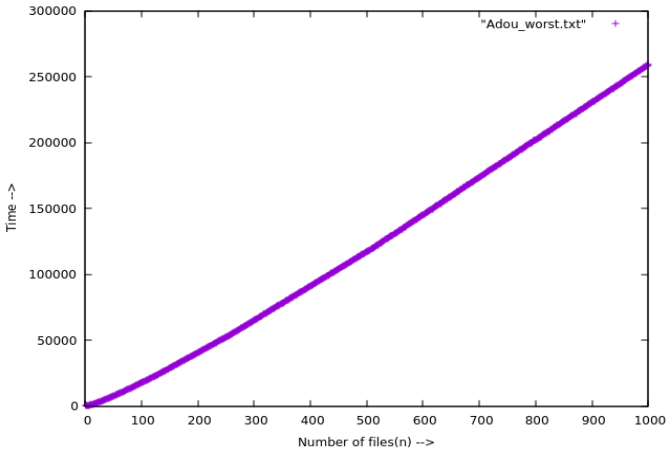

Fig.2 Number of files(n) vs Time

*3) **Average-Case**:* For average case, we can say that we are assuming that length array and frequency array will be in such a way that in generated ratio array some elements may be same or not, Then time

complexity will lie between best case and worst case.

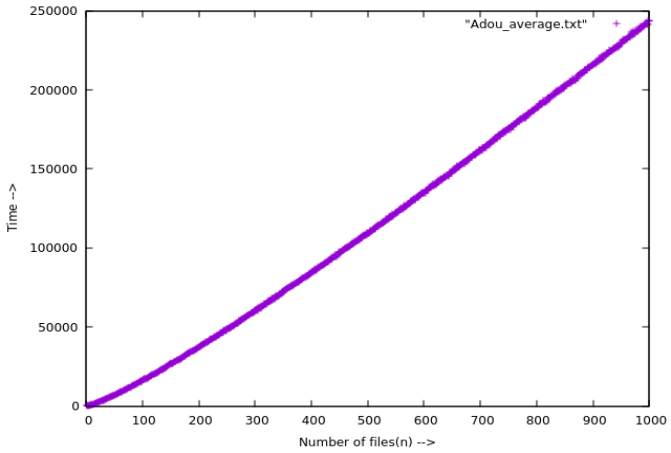

Fig.3 Number of files(n) vs Time

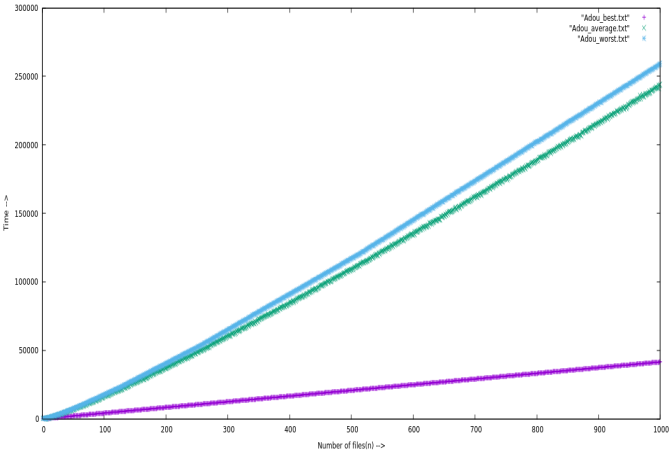

Fig.4 Number of files(n) vs Time

VIOLET CURVE:- **best case**
GREEN CURVE:- **average case**
BLUE CURVE:- **worst case**

IV. Experimental Analysis

We revised the commands and basic functions of **GNUPlot** to plot the time complexity analysis graphs and other relevant analysis related to our algorithm.
While making this report we also learnt the basics of making reports using LATEX in IEEE format using **IEEEtran class**.

For the analysis of the Complexity of the algorithm we had to generate 3 kinds of test cases. For each of these 3 kinds our code was run on **1000** test cases of each type and The algorithm was run on these three files separately and graphs were plotted using **GNU-Plot**.The graphs have been vividly explained in the previous section. In each of these test cases the length of **array** was randomly chosen greater then 1.

For the **Best case**, we generated the test cases where ratio array is in ascending order as we know we have to sort length array and frequency array in such a way that $\frac{l}{f}$ will be in increasing order.

For the **Worst case**, we generated test cases where ratio array's every element should be distinct such that heap sort will take time complexity of $O(nlogn)$.

And for the **average case**, we generated test cases where ratio array was filled with random number. So it may be possible that there may be some element which will be identical in that case heap sort will some less amount of time in comparison to worst case.

TABLE :- Comparison Of time taken in Best Case, Average Case and Worst Case

n(no. of files)	time$_{bestcase}$	time$_{averagecase}$	time$_{worstcase}$
5	203	328	353
10	421	771	946
20	836	2186	2361
50	2081	6831	7606
75	3108	11558	12508
100	4156	16431	17681

V. Conclusion

According to the Assignment Problem we have to Store the given n files $f_1, f_2, , f_n$ with corresponding lengths $l_1, l_2, , l_n$ and frequency of retrieval $r_1, r_2, , r_n$ respectively. Store the files such that the mean retrieval time is minimized.
By generating the test cases where ratio array is in ascending order as we know we have to sort length array and frequency array in such a way that $\frac{l}{f}$ will be in increasing order.And this provide the **Best Case** with time complexity $O(n)$.
As well as for the **Worst case**, we generated test cases where ratio array's every element should be distinct such that heap sort will take time complexity of $O(nlogn)$.
Accordingly in case of the **Average case**, we generated test cases where ratio array was filled with random number. So it may be possible that there may be some element which will be identical in that case heap sort will some less amount of time in comparison to worst case.And the corresponding time complexity will lie in between the best and worst case.

References

[1] Introduction To Algorithms (Second Edition) by Thomas H.Cormen, Charles E.Leiserson , R.L.Rivest and Clifford Stein.
[2] The C Programming Language by Brian Kernighan and Dennis Ritchie.
[3] https://www.geeksforgeeks.org/
[4] https://www.stackoverflow.com/
[5] http://cs.slu.edu/ echambe5/spring09/cs314/schedule/x01-greedy.pdf