FINDING WORDS IN ASCENDING OR DESCENDING ORDER

Indian Institute of Information technology, Allahabad

DAA ASSIGNMENT-2 , GROUP 8

Swaraj Bhosle IIT2019024

Ritesh Raj IIT2019025

Utkarsh Garg IIT2019026

Abstract: In this Paper we have devised an algorithm to find the number of valid English words which have ascending(A to Z) or descending(Z to A) ordered characters.

I. INTRODUCTION

In computer science, the process of searching a string in a given set of strings is known as String Searching. We have created a text file containing at least 1000 valid English words. Basically ascending characters in a string means all the characters should be in a alphabetical order i.e from A to Z and the descending order means all the characters should be in a reverse alphabetical order i.e from Z to A. For creating a new algorithm from scratch, we look into the many crucial aspects of algorithm designing and analysis.The algorithm created is tested for different sample test cases for time complexity, space complexity and accuracy. In our case
This report further contains -

- Algorithm Description.
- Algorithm And Analysis.
- Time Complexity Analysis
- Conclusion

II. ALGORITHM DESIGN

We are using a file system in our algorithm as our input is stored in the file. Our algorithm will first of all read the input from the txt file. We have created the *input.txt* file and all the input strings are inside this file.

Step1 : First thing we will open the *input.txt* file and will read the input for the programme.
To read the information from a file into your programme we have to use the extraction operator $>>$ just as we use that operator to input information from the keyboard. The only difference is that we use an if stream or of stream object instead of the cin object.

Step2 : check whether the file is opened or not. i.e the open fails if the file doesn't exist, or if it can't be read. a failure can be detected with the code like !(logical not) operator

Step3 : create the character array of MAX_SIZE of 1000. We will store the input string from the file into this character array

Step4 : Traverse through each input string from .txt file.

Step5 : create two different character arrays and store the input string in these two character arrays differently.

Step6 : Sort one of the above taken arrays in alphabetical order (A to Z) and sort the other array in reverse alphabetical order (Z to A)

Step7 : compare the original string with these two created ascending and descending strings. If the original string matches one of the two strings then count it.

Step8 : print the count number.

Step9 : close the opened file

Advantages of using merge sort:- It is quicker for larger lists because unlike insertion and bubble sort it doesn't go through the whole list several times. It has a consistent running time, carries out different bits with similar times in a stage.

III. ALGORITHM AND ANALYSIS

Algorithm 1: Algorithm

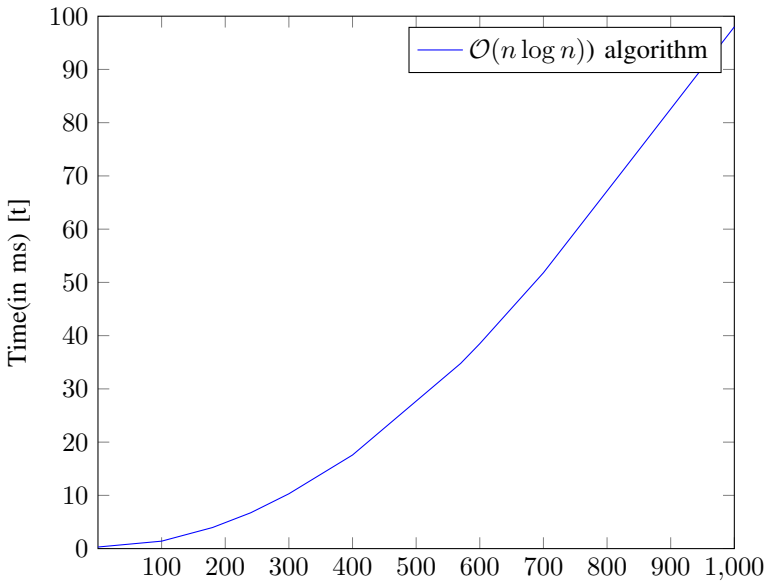
Input: string from the text document.

Output: The number of English words that have ascending or descending characters

```
1  Function Main():
2      Declare an input file steam
3      ifstream inFile
4      open the file stream
5      inFile.open("input.txt");
6      Check that the file was opened
7      if ( file can't open ) then
8          Exit
9      create the char array of MAX_SIZE
10     char str[MAX_SIZE]
11     assign count equal to zero
12      count = 0
13     while inFile >> str do
14         Store the length of string in l
15          n = strlen(str)
16         create two char array of MAX_SIZE
17         char asc[MAX_SIZE]
18         char dsc[MAX_SIZE]
19         store str string in both asc and dsc array
20         for i ← 0  to  n do
21             asc[i] = str[i]
22             dsc[i] = str[i]
23         Sort asc array in ascending order
24         merge_sort(asc, asc + l)
25         Sort the dsc array in descending order
26         merge_sort(dsc,dsc+l,greater<char> ());
27         compare the str string with dsc and asc
            array
28         if (str == asc or str == dsc)
29         count++;
30         if it matches any one of them then count
           _it.
31     print count
```

IV. TIME CALCULATION

Complexity analysis for Naive $\mathcal{O}(n \log n))$ algorithm



No. of Words	Execution Time(ms)
100	1.4
250	4
500	34
800	70
1000	98

V. TIME COMPLEXITY

Let the average size of all strings be N. Then we merge sort the string in ascending and descending order and compare the obtained strings with the original string, and if the string matches with any of the sorted strings then we increment the answer.

The time complexity will be $\mathcal{O}(n \log n))$ due to the merge sort algorithm in a best, average and worst case. The time complexity will be affected due to the merge sort in sorting the 1D array and the total time complexity will become $\mathcal{O}(n \log n))$ in all the cases i.e. best case, worst case and average.

Best Case :

$$\mathcal{O}(n) + \mathcal{O}(n \log n) + \mathcal{O} => \mathcal{O}(n \log n)$$

Average Case :

$$\mathcal{O}(n) + \mathcal{O}(n \log n) + \mathcal{O} => \mathcal{O}(n \log n)$$

Worst Case :

$$\mathcal{O}(n) + \mathcal{O}(n \log n) + \mathcal{O} => \mathcal{O}(n \log n)$$

BEST	AVERAGE	WORST CASE
$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$

VI. CONCLUSION

Merge sort is one of the most efficient sorting algorithms. It works on the principle of Divide and Conquer. Merge sort repeatedly breaks down a list into several sub lists until each sub list consists of a single element and merging those sub lists in a manner that results into a sorted list in logarithmic time.

This algorithm works as the string gets passed through the sorting algorithm and sorted in ascending and descending order which helps in comparing with the initial string of the text document and calculating the number of english words that are in ascending or descending characters.

VII. REFERENCES

1) http://www.fredosaurus.com/notes-cpp/io/readtextfile.html
2) https://www.tutorialspoint.com/cplusplus/cpp_files_streams.html
3) https://www.geeksforgeeks.org/program-sort-string-descending-order