A01G14Q14 : INSPECTION OF ARMSTRONG NUMBER

IV Semester B.Tech, Department of Studies in Information Technology,

Indian Institute of Technology Allahabad, Prayagraj, India

Abstract: In this paper, we have designed an algorithm that checks whether the given no. is Armstrong no. or not.
We have discussed the time and memory complexity of the algorithm by both Apriori and Apostriori analysis.

I. INTRODUCTION

Armstrong no. is a no. whose sum of digits raised to the power of no. of digits is equal to the given no. assume N is that no to be checked. The constraints of the number N is, it can be a 2 or 3 digit no. from (9<N<999), the given no. must be positive.

So our algorithm is designed in a way that the no. should be checked first as it is input by user, that no. is valid or not, if not print invalid input.

In our algorithm we have adopted two while loops first is used to calculate no. of digits(exponent) and second loop is benefited in calculating the exponential powers of the digits.

II. ALGORITHM DESCRIPTION

This algorithm simulates inspection of an Armstrong no. The algorithm takes N as input, N should be a positive integer ranging from (9<N<999). The algorithm inspects an Armstrong no. in two stages:

Stage 1: In stage one we input a no. and check its validity range, if the no. is not valid print invalid. Then N is transferred to **form** function in step: 4 to count the no. of digits in N.

Stage2: In this stage the value that we receive **from** is used by **pow** variable in step: to calculate sum of the digits raised to the power of no. of digits in N that we received from operation **form**.

After the final sum if the sum is equal to the given no. N then we print **Armstrong no.** else print **not Armstrong**.

III. ALGORITHM AND ANALYSIS

Algorithm: Inspection
Input **N;**
Output result: **Armstrong no., not Armstrong no., invalid input.**

Method:

```
Step1  : if(N<9 || N>1000)then
Step2  :         result← "invalid input", exit
Step3  : temp ← N
Step4  : while(temp!= 0) //form function
         {
Step5  :     no_of_digits++;
Step6  :     temp← temp/10;
         }
Step7  : temp ← N
Step8  : while(temp!=0)
         {
Step9  :         d ← temp%10;
Step10:          pow ← 1;
Step11:          for i to no_of_digits do
Step12:                  pow ← pow *d;
Step13:          sum ← sum + pow;
Step14:          temp ← temp/10;
         }
Step15: if(sum== N)
         {
Step16:     result ← "Armstrong Number"
         }
         else
         {
Step17:     result ← "Not an Armstrong Number"
         }
```

AlgEnds

III (a). Apriori Analysis

Table1: time complexity for Apriori analysis

	Time	FreqBest	Frequency
Step1	3	1	1
Step2	1	0	1
Step 3	1	1	1
Step 4	1	3	$[\text{Log}_{10}N+1]$
Step 5	2	2	$[\text{Log}_{10}N]$

Step 6	2	2	[Log$_{10}$N]
Step 7	1	1	1
Step 8	1	3	[Log$_{10}$N+1]
Step 9	2	2	[Log$_{10}$N]
Step 10	1	2	[Log$_{10}$N]
Step 11	3	4	[Log$_{10}$N]
Step 12	2	4	[Log$_{10}$N]
Step 13	2	2	[Log$_{10}$N]
Step 14	2	2	[Log$_{10}$N]
Step 15	1	1	1
Step 16	1	1	1
Step 17	1	1	1

T_Ω α 3*1+1*0+1*1+1*3+2*2+2*2+1*1+1*3+ 2*2+1*2+3*4+2*4+2*2+2*2+1*1+1*1+1*1.

T_Ω α 56

Ω(1)

For N is two digit no., the algorithm takes minimum time for execution.

To α 3*1+1*1+1*1+1* [Log$_{10}$N+1]+2*[Log$_{10}$N]+2*[Log$_{10}$N]+1*1+1*[Log$_{10}$N+1]+2*[Log$_{10}$N]+1*[Log$_{10}$N]+ 3*[Log$_{10}$N]+2*[Log$_{10}$N]+2*[Log$_{10}$N]+2*[Log$_{10}$N]+1*1+1*1+1*1.

To α 9+ 16Log$_{10}$N +2[Log$_{10}$N+1]

O(Log$_{10}$N)

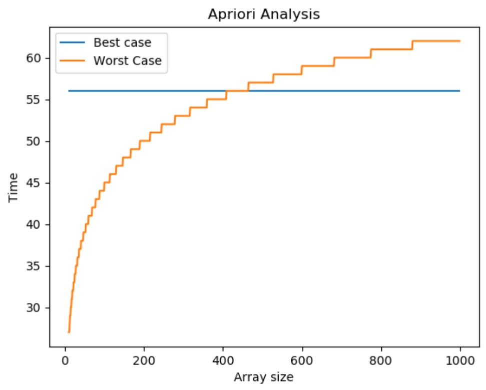

Figure1:Time complexity graph for apriori analysis.

IV. EXPERIMENTAL ANALYSIS AND PROFILING

Table2:Time complexity for Aposteriori analysis

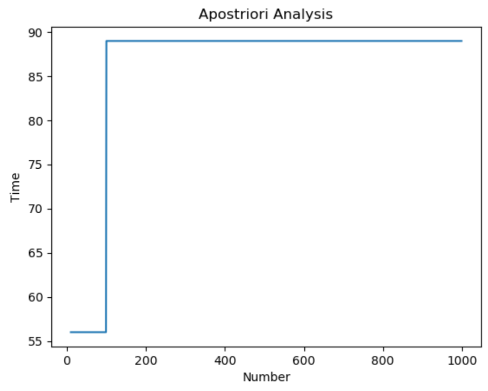

Figure2: Time complexity graph for aposteriori analysis.

In the above graph, the exponent 'N' is considered along x-axis and time along y-axis. The graph represents Time v/s N. The above graph shows that the Apostriori analysis is consistent with Apriori analysis.

CONCLUSION

From this paper we can conclude that the algorithm is an Ω(1)-O(log$_{10}$N) algorithm, the algorithm takes minimum time when the exponent N is two digit no. and the algorithm gives logarithmic function for time .

REFERENCES

[1] R.G Dromey, How to solve it by Computer, New Delhi: Prentice –Hall of India, 2004.

[2]https://en.wikipedia.org/wiki/Talk%3AArmstrong_number.