ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 9| Issue: 12| December 2023|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2023: 8.224 || ISI Value: 1.188

CHECKING ARMSTRONG NUMBER USING FUNCTION *check_arm()* FUNCTION USING SEQUENTIAL SEARCH METHODOLOGY AND *check_rand_arm()* FUNCTION FURTHER COMPARISON BETWEEN BOTH THE FUNCTIONALITIES BY EVALUATING THE TIME COMPLEXITY OF THE FUNCTIONS - A CASE STUDY

6415 CDT Rohit Raj[1]
Class- XII 2023-24, Sainik School Amaravathinagar
Post: Amaravathinagar,Udumalpet Taluka,Tirupur Dt,Tamilnadu State

ABSTRACT

The time complexity of an algorithm is the amount of time taken by the algorithm to complete the execution of function of its input length, n. The time complexity of an algorithm is denoted by using asymptotic notations.

This manuscript specifically examines the efficiency of functions check_arm() and check_rand_arm(), these methodologies are used to find an Armstrong number in a vector. The efficiency of these two functions are calculated and assessed by using time complexity of functions. Further the space complexity also calculated, since both algorithm uses same space as per the input function. The endeavour of this paper is to express the better algorithm with respect to time.

KEYWORDS: ***check_arm(ca) check_rand_arm(cra), Random number (rn), Runtime Complexity (rc), Big O𝑂(n), Big Theta𝛩(n), Big Omega𝛺(n), Generalised approach (ga)***

1. INTRODUCTION

Armstrong number is the number in any given number base, which forms the total of the same number, when each of its digits is raised to the power of the number of digits in the number. It is of special interest to new programmers and those learning a new programming language because of the way the number behaves in a given number base. For example, using a simple number 153 and the decimal system, we see there are 3 digits in it. If we do a simple mathematical operation of raising each of its digits to the power of 3, and then totalling the sum obtained, we get 153. That is 1 to the power of 3, 5 to the power of 3, 3 to the power of three is 1 125 27 and sum is 153. This can also be represented as 1^3 5^3 3^3=153. The number 153 is an example of the Armstrong number which also has a unique property that one can use any number system .Thus if the number obtained totals to or equals the original number when each of the digits is raised to the power of the number of digits in the number and added to obtain a number, in any given number system, such a number is called an Armstrong number.

2. RELATED WORK

in number theory, an Armstrong number in a given number base b is a number that is the sum of its own digits each raised to the power of the number of digits. To put it simply, if I have a 3-digit number then each of the digits is raised to the power of three and added to obtain a number.

3. METHODOLOGY

There are different Python programs associated with finding an Armstrong number. You can check whether a given number is an Armstrong number or not. Alternatively, you can find all the Armstrong numbers within a specified range of numbers. We will go with both these approaches related to the identification of Armstrong numbers, in Python.

The methodology I have used to check Armstrong number is function check_arm() and check_random_arm().

The various programming languages has the capability to generate Armstrong number(ln) and store it in a file (permanently on secondary storage device). During the course of research, it was decided to use python programming language due to its large collection of library modules and the availability of online support.

ALGORITHM FOR CHECKING ARMSTRONG NUMBER

STEP 1: The number of digits in num is found out

STEP 2: The individual digits are obtained by performing num mod 10, where the mod is the remainder module.

STEP 3: The digit is raised to the power of the number of digits and stored.

ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 9| Issue: 12| December 2023|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2023: 8.224 || ISI Value: 1.188

STEP 4: Then the number is divided by 10 to obtain the second digit.

STEP 5: Steps 2, 3 and 4 are repeated until the value of num is greater than 0

STEP 6: Once num is less than 0, end the while loop

STEP 7: Check if the sum obtained is same as the original number

STEP 8: If yes, then the number is an Armstrong number.

PYTHON PROGRAM TO CHECK THE ARMSTRONG NUMBER AND STORING IN SECONDARY STORAGE DEVICE IN THE FORM OF TEXT FILE.

```
num = int(input("Enter a number: "))
sum = 0
n1 = len(str(num))
temp = num
while temp > 0:
  digit = temp % 10
  sum += digit ** n1
  temp //= 10
if num == sum:
  print(num,"is an Armstrong number")
else:
  print(num,"is not an Armstrong number")
```

ALGORITHM TO FIND A ARMSTRONG NUMBER IN A TEXT FILE

1.Convert the input number into a string usingstr(num).
2. Find the length of the string using len(num_str) and store it inn.
3. Initialize a variable sum to zero.
4. Iterate through each digit in the string using a for loop, and convert each digit back to an integer using int(digit).
5. Raise each digit to the power of n using int(digit)**n, and add the result to sum.
6. After the loop is complete, check whether sum is equal to num.
7. If sum is equal to num, return True (the input number is an Armstrong number).
8. If sum is not equal to num, return False (the input number is not an Armstrong number).

PYTHON PROGRAM TO CHECK ARNSTRONG NUMBER

```
def is_armstrong(num):
    num_str = str(num)
    n = len(num_str)
    sum = 0
    for digit in num_str:
        sum += int(digit)**n
    if sum == num:
        return True
    else:
        return False
num=153
print(is_armstrong(num))
```

4. COMPLEXITY OF ALGORITHM

In computer science, analysis of algorithms is a very crucial part. It is important to find the most efficient algorithm for solving a problem. It is possible to have many algorithms to solve a problem, but the challenge here is to choose the most efficient one.[2]

There are multiple ways to design an algorithm, or considering which one to implement in an application. When thinking through this, it's crucial to consider the algorithm's **time complexity** and **space complexity**.[3]

5. SPACE COMPLEXITY

The space complexity of an algorithm is the amount of space (or memory) taken by the algorithm to run as a function of its input length, n. Space complexity includes both auxiliary space and space used by the input.[3]

Auxiliary space is the temporary or extra space used by the algorithm while it is being executed. Space complexity of an algorithm is commonly expressed using **Big (O(n))** notation.[3]

The Space complexity is ignored in this research paper, since the space complexity of particular problem is not considered so important.

6. TIME COMPLEXITY

The time complexity of an algorithm is the amount of time taken by the algorithm to complete its process as a function of its input length, n. The time complexity of an algorithm is commonly expressed using asymptotic notations:[3]

Big O - O(n)
Big Theta - Θ(n)
Big Omega - Ω(n)

It's valuable for a programmer to learn how to compare performances of different algorithms and choose the best time-space complexity to solve a particular problem in the most efficient way possible.[3]

Big O notation is used in Computer Science to portrait the performance or complexity of an algorithm.

Big O specifically defines the worst-case scenario of an algorithm, and can be used to describe the execution time required or the space used (e.g. in memory or on disk) by an algorithm. here O stands for order of growth.

Big Theta(Θ) is used to represent the average case scenario of an algorithm and can be used to describe the execution time

ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 9| Issue: 12| December 2023|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2023: 8.224 || ISI Value: 1.188

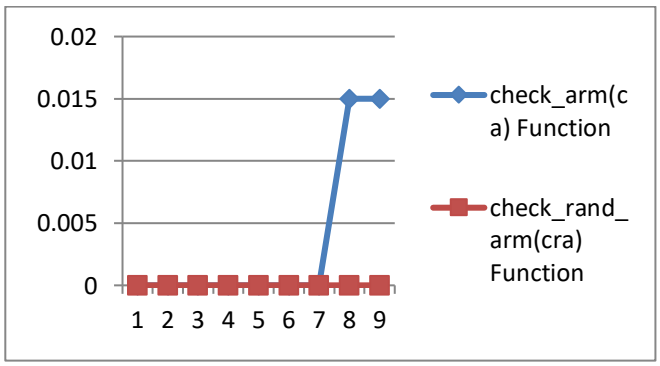
required or the space used (e.g. in memory or on disk) by an algorithm.

Big Omega (Ω)is used to represent the best case scenario of an algorithm and can be used to describe the execution time required or the space used (e.g. in memory or on disk) by an algorithm.

These three methods are the most common and very popular methods of design and analysis of an algorithm which are used for finding the efficiency of the program.

7. RUNTIME COMPLEXITY OF CHECKING ARMSTRONG NUMBER

Input (No of Digits)	check_arm(ca) Function	check_rand_arm(cra) Function
3	0.0	0.0
4	0.0	0.0
5	0.0	0.0
6	0.0	0.0
7	0.0	0.0
8	0.0	0.0
9	0.0	0.0
10	0.015	0.0
11	0.015	0.0



Graphical Representation of Runtime complexity of both the methods

8.GENERALISED APPROACH - *rc*

In the normal approach the program checks for the given number prime or not. The time complexity of the algorithm for worst case is denoted as:

Big (O(n))

9.LUCAS METHOD (LMM) - *rc*

The time complexity of the LUCAS Method is calculated as

Big (O(14))

10. CONCLUSION

The Lucas mathematical methodology has the greater efficiency for checking prime when comparing with general approach. Further it is also observed that generating prime series and storing in a file is one time process and it is time consuming but once the file is prepared the performance of the code is much higher than the normal approach. In addition to this it is also observed that the execution of expression also depends on the hardware configuration.

11.ACKNOWLEDGEMENT

Apart from the efforts of me, the success of any work or project depends largely on the encouragement and guidelines of many others. I take this opportunity to express my gratitude to the people who have been instrumental in the successful completion of this research paper.

I express deep sense of gratitude to almighty God for giving me strength for the successful completion of the research paper.

I express my heartfelt gratitude to my parents for constant encouragement while carrying out this research paper.

I express my deep sense of gratitude to the luminary **The Principal Capt. (IN) K Manikandan, Sainik School Amaravathinagar** who has been continuously motivating and extending their helping hand to us.

I express my sincere thanks to the Administrative officer LT COL K DEEPU, Sainik school Amaravathinagar

I express my sincere thanks to the academician The **Wg Cdr Deepti Upadhayay, Sainik School Amaravathinagar**, for constant encouragement and the guidance provided during this research.

My sincere thanks to **Mr.Praveen Kumar MurigeppaJigajinni**, Master In-charge, A guide, Mentor and great motivator,who critically reviewed my paper and helped in solving each and every problem, occurred during implementation of this research paper.

12. REFERENCES

1. *https://en.wikipedia.org/wiki/Lucas_number*
2. *https://www.freecodecamp.org/news/time-complexity-of-algorithms/*
3. *https://www.educative.io/edpresso/time-complexity-vs-space-complexity*