DAA Assignment 1

Ashutosh Kaushik (ICM2017002), Pratik Kedia(ICM2017004),Sagar Lakhani(ICM2017003),
Animesh kumar jha(ICM2017001)

1 INTRODUCTION

1.1 Question

Check if **n** is an exact square number (without using sqrt function).
Where ***10^4 < n < 10^6***

1.2 Definition

In mathematics, a **square number** or perfect square is an non-integer that is the square of an integer,in other words it is the product of some integer with itself.

1.3 Aim

Our aim is to design an Algorithm which can test the given number and tell us that it is a perfect square or not . we want our Algorithm to be optimal in both ways (Time and Space). But we have a restriction over here that we cannot use inbuilt sqrt function provided in C language to find the square root of a non-negative number.

1.4 Idea

The main idea of the Algorithm is based on the observation : If there is an integer number x such that square of this number is greater than the given number n then all the numbers greater or equal to x cannot be the square root of n. So, our Algorithm exploits this fact and this property allows us to use binary search over here. Binary search is a search Algorithm which compares the target value with middle value of the range and in each iteration it discards half of the search domain. Another fact is that if the number is perfect square then it always ends with 1,4,9,6,5,0.

2 . Algorithm design

2.1 Part a : CHECK ( )

Parameters:

x and n :
x is a number to be checked whether its square is equal to n or not.

Variables :

temp: temporary variable to store the square of x

Pseudo code:

```
1.CHECK(x,n)
2..     temp ← x*x
3.      If temp< n
4.              Return true
5.      If temp == n
6.              PRINT (N is perfect square )
7.              EXIT
8.      Return false
```

2.2 Part b : PERFECT_SQUARE_CHECKER ( )

Parameters:

hi : highest possible value of square root of given number

lo : lowest possible value of square root of given number

n : number which is needed to be checked as perfect square

Variables:

mid : temporary variable to store the middle value of the current domain.

lst : variable to store the last digit of the given number

Pseudo code:

```
1.  PERFECT_SQUARE_CHECKER( hi , lo , n)
2.        lst ← n mod 10
3.        If lst == 2 OR lst == 3 OR lst == 7 OR lst == 8
4.                  PRINT( N is not the perfect square )
5.                  EXIT
6.       while( lo <= hi)
7.                mid ← (lo) + (hi - lo) / 2
8.                If CHECK(mid,n)
9.                          lo ← mid + 1;
10.               else
11.                         hi ← mid - 1;
12.       PRINT(N is not the perfect square )
```

3. Analysis And Discussion

3.1 Time complexity

Best case:

Best case of this **Algorithm** takes 6 computation.

$T_{best} = 6$
$T_{best} = \Omega(1)$

Worst case:

Number of instructions executed in worst case is $6 + 11*\log_2 n$ as there are 11 computation in the while loop which is executed $\log_2 n$ times.

At each step n is divided into 2 halves and one of the half is discarded until the lower value of range is less than or equal to upper value of range, this implies

f(n) = n + n/2 + n/4 + n/8 + n/16 … + n/(2^k)

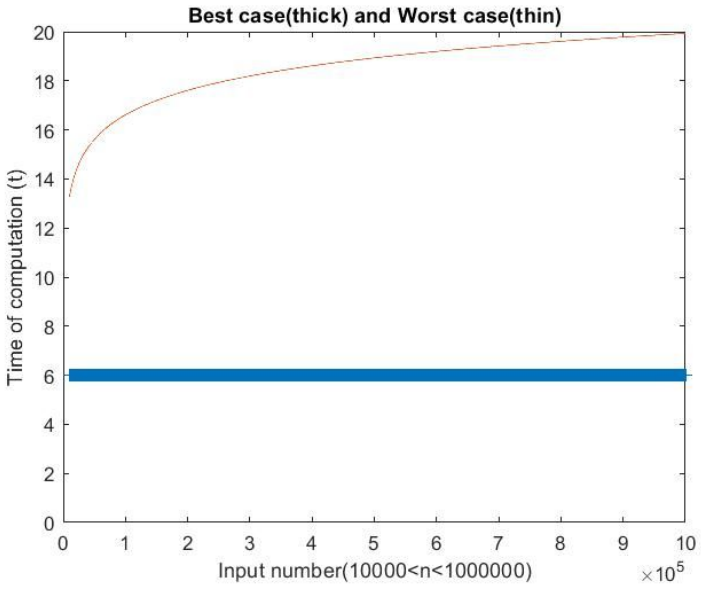
We will calculate the number of terms in this function because these many times iterations are performed.

Such that $2^k = n$
Taking log both side $\Rightarrow k = \log_2 n$

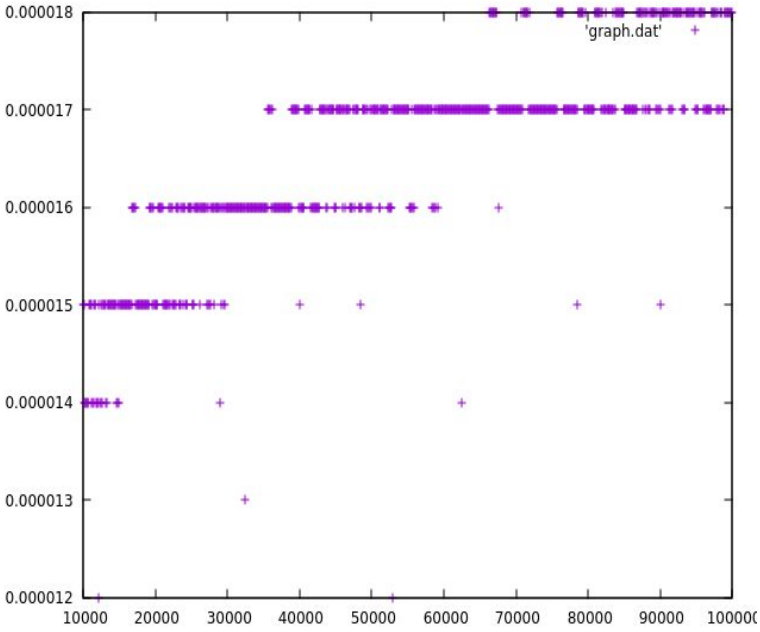
This means number of terms in f(n) is k which is $\log_2 n$ and our algorithm operates for these many steps and in each step it executes 11 computations.

$T_{worst} \propto 6 + 11*\log_2 n$
$T_{worst} = O(\log_2 n)$

⇒ **Plot of worst case and best case complexity :**



⇒ **Plot of average case complexity :**



gnuplot

3.2 Space Complexity

Space complexity of our Algorithm is negligible as we are just using few temporary variable.

Space complexity = O(1)

4. Experimental Study

Time shown in the table below is of some random values of n (10^4 < n < 10^6) taken by our algorithm to test those values.

n	**10000**	**12100**	**14800**	**44800**	**99000**
t(μs)	**15**	**12**	**14**	**16**	**18**

5.Conclusion

Algorithm discussed in this paper is to check whether the given number is perfect square or not . We made few observations and used some facts to make our algorithm efficient. This algorithm takes maximum of log2n operations for testing a number, basically it uses the idea of binary search.

6. References

Gnuplot :- http://www.gnuplot.info/help.html
Google Docs: https://docs.google.com/document