Number Theory Assignment-1

Group members:

- Riddhi Chatterjee (IMT2020094)
- Asmita Zjigyasu (IMT2020507)

Instructions to run:

1. Use the command "make assignment1" to create the executable file called "assignment1".
2. Execute the executable file using the command "./assignment1".
3. For internal computations, the base is 2^m. We are now told to enter the value of m. The base is stored in a variable of 'int' data type. Thus we have to enter a value of m such that (2^(2m)) (i.e. the value of base^2) doesn't exceed the maximum value that a 'int' data type variable can store.
4. Next, we are told to choose between two options:
   [1] sqrt(n) -> Used to calculate the value of "square root of n" upto certain number of digits of precision. On choosing this option (by entering 1 to indicate this choice), we are asked to enter the value of n. We can enter any value of n (Eg: 256 or 12345 or 1234.56 etc), after which we are asked to enter the number of digits of precision required in the final answer.
   [2] PI -> Used to calculate the value of PI upto certain number of digits of precision. On choosing this option (by entering 2 to indicate this choice), we are asked to enter the number of digits of precision required in the final answer.

Algorithms used:

1. **Addition**: The standard algorithm shown in class has been used. Slight modifications have been done to handle arbitrary precision real numbers.
2. **Subtraction**: The standard algorithm shown in class has been used. Slight modifications have been done to handle arbitrary precision real numbers. This algorithm is very similar to our addition algorithm.
3. **Division**: The standard algorithm shown in class has been used. Slight modifications have been done to handle arbitrary precision real numbers.
4. **Multiplication**: We have implemented Karatsuba's algorithm (as shown in class) for performing multiplication of large numbers. Slight modifications have been done to handle arbitrary precision real numbers.
   Note: Since the Karatsuba's algorithm is a recursive algorithm, it is taking a lot of time when the required precision is very large (for example 10000 digits of precision). This is most probably because of the large number of recursive calls that are being made and the large amount of memory that is being used as a result of this.
5. **Computation of square root**: We have used the algorithm mentioned in the assignment pdf, which uses Newton's approximation to find √R. The algorithm is as follows:

Algorithm 2 Newton's Approximation to find $\sqrt{R}$

0: Initialize x_0 Randomly (of course closer it is to $\sqrt{R}$ the better)
0: **while** (Precision not obtained) **do**
0: $x_{n+1} = \frac{1}{2}\left(x_n + \frac{R}{x_n}\right)$
0: **end while**=0

The x_n in this algorithm will give closer and closer approximations to √R as n increases.

6. **Computation of PI**: We have used the algorithm mentioned in the assignment pdf (i.e Borwein's exp-2 Algorithm). The algorithm is as follows:

Algorithm 1 Borwein's *exp-2* Algorithm

0: $a_0 = \sqrt{2}, b_0 = 0, p_0 = 2 + \sqrt{2}$

0: **while** (Precision not obtained) **do**

0: $a_{n+1} = \frac{\sqrt{a_n} + \frac{1}{\sqrt{a_n}}}{2}$

0: $b_{n+1} = \frac{(1+b_n)\sqrt{a_n}}{a_n + b_n}$

0: $p_{n+1} = \frac{(1+a_{n+1})p_n b_{n+1}}{1+b_{n+1}}$

0: **end while**$=0$

The p_n in this algorithm will give closer and closer approximations to π as n increases.