MAT333E – Data Processing Algorithms

Term Project

Assist. Prof. Dr. İzzet GÖKSEL

Karatsuba Multiplication

Yiğit Berk SARIBOĞA

Zeynep İNCE

Deniz AKIŞ

Deniz Zakir EROĞLU

Bican Çağrı GÖKSEL

Objective of the Project

The objective of the Karatsuba multiplication project is to implement an efficient algorithm for multiplying large integers. The Karatsuba algorithm is a fast multiplication method that can be used to multiply two n-digit numbers in fewer than n^2 single-digit multiplications, which is the complexity of the naive multiplication algorithm. This can be particularly useful for very large integers, as the naive algorithm can be very slow for these cases.

The Karatsuba algorithm was developed by Anatolii Alekseevich Karatsuba in the 1960s, and it has since become a popular method for efficiently multiplying large integers. It works by dividing the two numbers being multiplied into two smaller numbers, and then using these smaller numbers to compute the product of the larger numbers using a series of recursive multiplications and additions. [1]

Overall, the goal of the Karatsuba multiplication project is to implement this algorithm in Python, and to evaluate its performance relative to other multiplication algorithms.

Problem Definition

The problem definition might be stated as given two n-digit integers X and Y, implement an efficient algorithm for multiplying X and Y using the Karatsuba multiplication method. The algorithm should be able to handle large integers (i.e., n may be very large), and should be implemented in Python programming language.

To clarify, the problem being addressed in this project is the need to efficiently multiply large integers. The Karatsuba algorithm is one solution to this problem, and the objective of the project is to implement and evaluate this solution. In terms of specific requirements, the project might specify that the Karatsuba algorithm should be implemented as a function that takes two integers as input and returns their product. The function should be able to handle very large integers (i.e., n may be very large) and should be efficient in terms of time and space complexity. [2]

Programming Language and Environment

Python is a popular and versatile programming language that is widely used in a variety of applications, including scientific computing, data analysis, and web development. It is known for its simplicity, readability, and strong support for object-oriented programming.

There are several reasons why Python might be a good choice for implementing the Karatsuba algorithm:

* Python has built-in support for large integers, which means that you can work with very large numbers without having to worry about overflow or other numerical issues.

* Python has a large standard library and a wide range of third-party libraries that can be used for tasks such as testing, debugging, and profiling. This can make it easier to develop and test your implementation of the Karatsuba algorithm.

* Python has a large and active community of users, which means that you can find a lot of resources and support online if you have questions or need help with your project.

As for the environment, it is important to make sure that you have a suitable development environment set up for working on your project. This might include installing the necessary software, such as a text editor or integrated development environment (IDE), as well as configuring any necessary dependencies or libraries.

By choosing Python and setting up a suitable development environment, you will have the tools and resources you need to implement and test the Karatsuba algorithm efficiently and effectively. [3]

Karatsuba Algorithm

The Karatsuba algorithm works by dividing each of the two numbers being multiplied into two smaller numbers, and then using these smaller numbers to compute the product of the larger numbers using a series of recursive multiplications and additions. This approach allows the algorithm to compute the product of two large integers in fewer steps than the naive algorithm, which makes it more efficient for large input sizes.

To use the Karatsuba algorithm to multiply two integers, you first need to split each integer into two parts, such that the left part has about half the digits of the original integer and the right part has the remaining digits. You can then use the following steps to calculate the product:

1 - Recursively compute the products of the left and right parts of each integer using the Karatsuba algorithm.

2 - Compute the sum of the left parts and the sum of the right parts.

3 - Recursively compute the product of the sum of the left parts and the sum of the right parts using the Karatsuba algorithm.

4 - Subtract the products of the left and right parts from the product of the sums to obtain the final result.

By following these steps, the Karatsuba algorithm is able to efficiently multiply two large integers by breaking down the problem into smaller subproblems that can be solved more quickly. This approach allows the algorithm to achieve a time complexity of O(n^1.585), which is significantly faster than the traditional grade school algorithm, which has a time complexity of O(n^2).

Overall, the Karatsuba algorithm is an efficient method for multiplying two large integers and can be a useful tool in scenarios where the traditional grade school algorithm may be too slow. [4]

Analysis of Karatsuba Algorithm

The time complexity of an algorithm refers to the amount of time it takes to execute as a function of the size of the input. In the case of the Karatsuba algorithm, the input size is determined by the number of digits in the two integers being multiplied (i.e., n).

One way to analyze the time complexity of the Karatsuba algorithm is to determine the number of recursive multiplications and additions that are performed as a function of n. For example, if we let T(n) represent the time complexity of the Karatsuba algorithm for multiplying two n-digit integers, we can write the following recurrence relationship:

$T(n) = 3T(n/2) + O(n)$

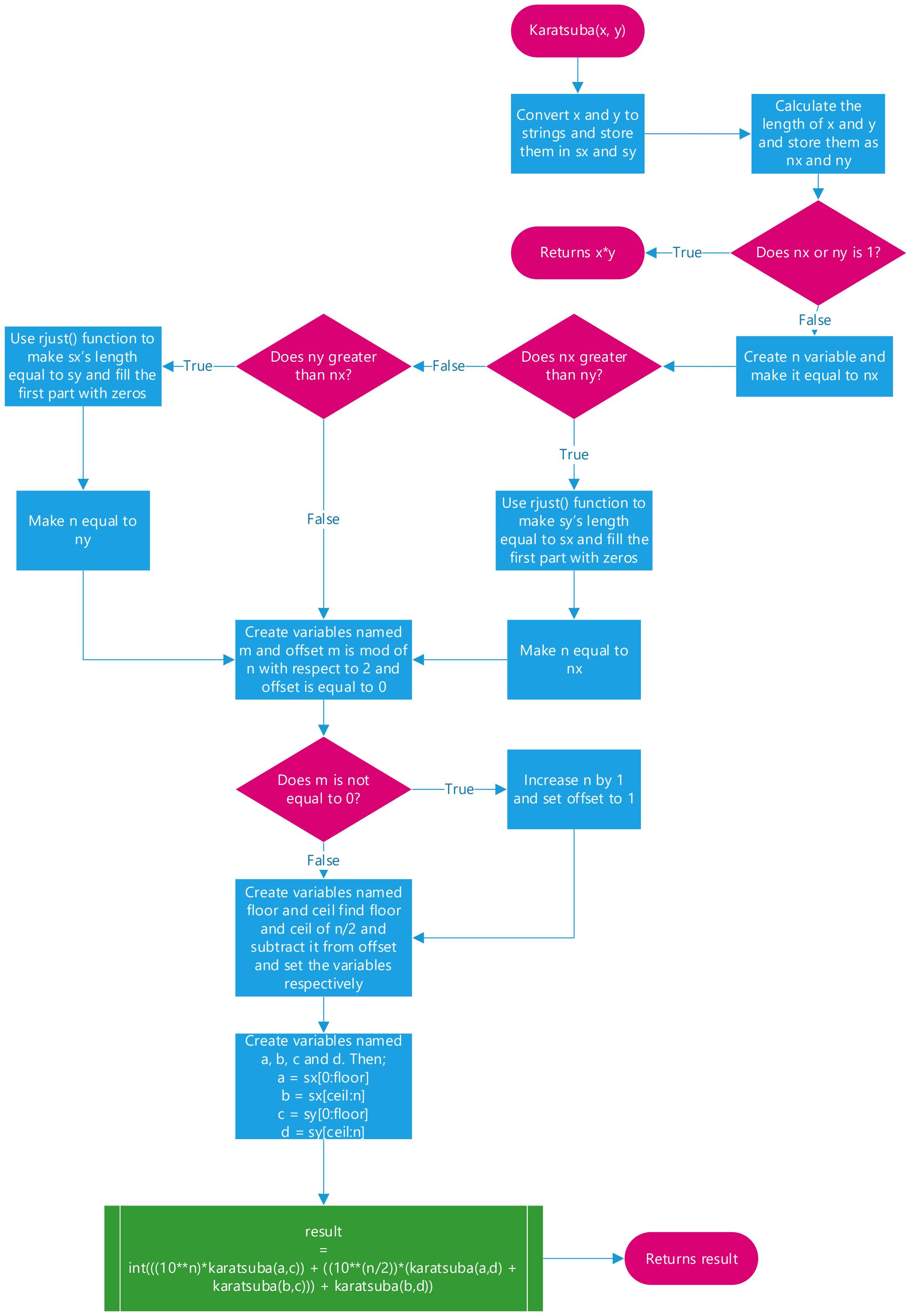
This relationship indicates that the time complexity of the Karatsuba algorithm is determined by the number of recursive multiplications and additions performed, which is $3T(n/2)$ for the two n/2-digit numbers being multiplied, plus $O(n)$ for the additional multiplications and additions needed to compute the final product.

Using this relationship, we can determine the time complexity of the Karatsuba algorithm as a function of n. For example, if we assume that the time complexity of the Karatsuba algorithm is dominated by the recursive multiplications (i.e., $T(n) \approx 3T(n/2)$), we can use the recurrence relationship to derive the following asymptotic complexity:

T(n) ≈ O(n^(log_23)) ≈ O(n^1.585) This result indicates that the time complexity of the Karatsuba algorithm is approximately 1.585 times the size of the input, which is significantly better than the naive algorithm, which has a time complexity of O(n^2).

Overall, the analysis of the time complexity of the Karatsuba algorithm can help to determine how efficient the algorithm is for different input sizes, and can be used to compare the performance of the Karatsuba algorithm to other multiplication methods. [5]

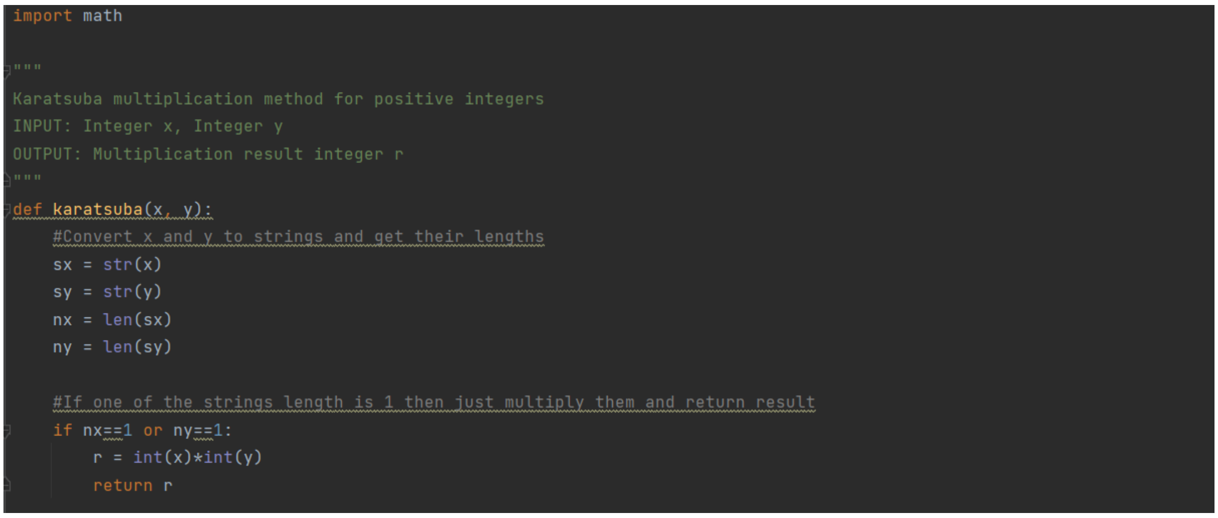
Flow Diagram of the Karatsuba Algorithm



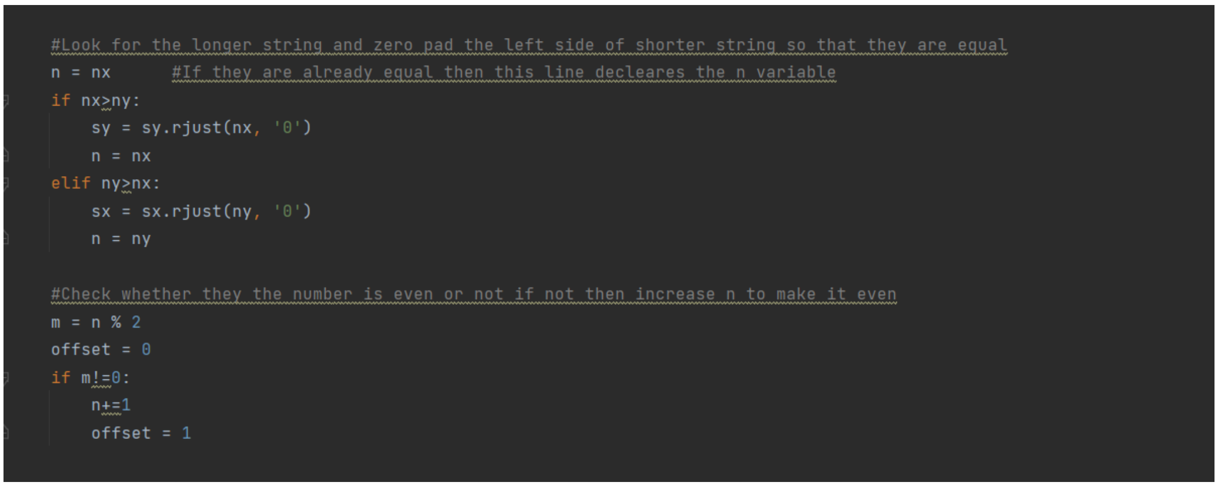
Karatsuba(x, y)
Convert x and y to strings and store them in sx and sy
Calculate the length of x and y and store them as nx and ny
Does nx or ny is 1?
True
Returns x*y
False
Create n variable and make it equal to nx
Does nx greater than ny?
True
Use rjust() function to make sy's length equal to sx and fill the first part with zeros
Make n equal to nx
False
Does ny greater than nx?
True
Use rjust() function to make sx's length equal to sy and fill the first part with zeros
Make n equal to ny
False
Create variables named m and offset m is mod of n with respect to 2 and offset is equal to 0
Does m is not equal to 0?
True
Increase n by 1 and set offset to 1
False
Create variables named floor and ceil find floor and ceil of n/2 and subtract it from offset and set the variables respectively
Create variables named a, b, c and d. Then;
a = sx[0:floor]
b = sx[ceil:n]
c = sy[0:floor]
d = sy[ceil:n]
result
=
int(((10**n)*karatsuba(a,c)) + ((10**(n/2))*(karatsuba(a,d) + karatsuba(b,c))) + karatsuba(b,d))
Returns result

Explanations of Key Parts of the Code

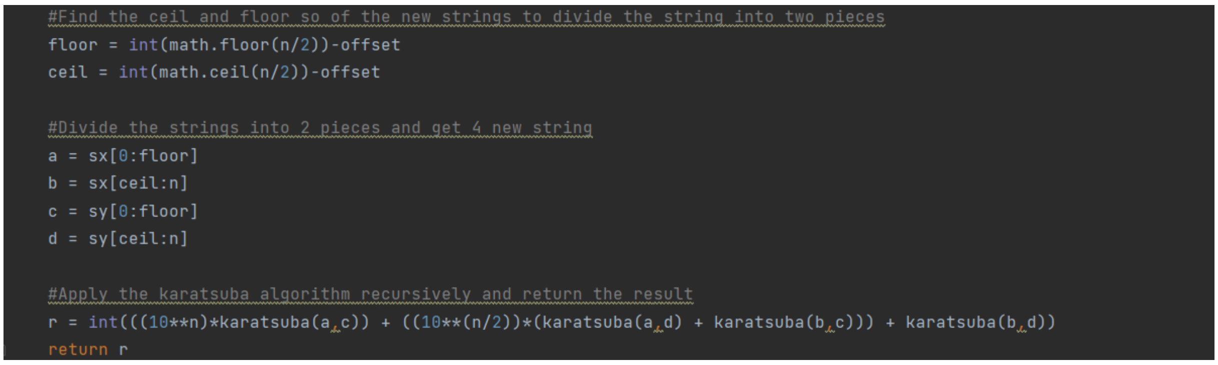
Comments are given with Python code.



```python
import math

"""
Karatsuba multiplication method for positive integers
INPUT: Integer x, Integer y
OUTPUT: Multiplication result integer r
"""
def karatsuba(x, y):
    #Convert x and y to strings and get their lengths
    sx = str(x)
    sy = str(y)
    nx = len(sx)
    ny = len(sy)

    #If one of the strings length is 1 then just multiply them and return result
    if nx==1 or ny==1:
        r = int(x)*int(y)
        return r
```

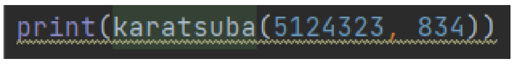


```python
    #Look for the longer string and zero pad the left side of shorter string so that they are equal
    n = nx      #If they are already equal then this line decleares the n variable
    if nx>ny:
        sy = sy.rjust(nx, '0')
        n = nx
    elif ny>nx:
        sx = sx.rjust(ny, '0')
        n = ny

    #Check whether they the number is even or not if not then increase n to make it even
    m = n % 2
    offset = 0
    if m!=0:
        n+=1
        offset = 1
```



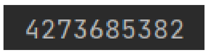
```python
    #Find the ceil and floor so of the new strings to divide the string into two pieces
    floor = int(math.floor(n/2))-offset
    ceil = int(math.ceil(n/2))-offset

    #Divide the strings into 2 pieces and get 4 new string
    a = sx[0:floor]
    b = sx[ceil:n]
    c = sy[0:floor]
    d = sy[ceil:n]

    #Apply the karatsuba algorithm recursively and return the result
    r = int(((10**n)*karatsuba(a,c)) + ((10**(n/2))*(karatsuba(a,d) + karatsuba(b,c))) + karatsuba(b,d))
    return r
```

Example:



```
print(karatsuba(5124323, 834))
```

Output:



```
4273685382
```

Result

The results of the Karatsuba multiplication project will depend on the specific goals and objectives of the project, as well as the methodologies and techniques used to test and evaluate the algorithm. Some possible results that might be obtained include:

- The implementation of the Karatsuba algorithm as a function in a programming language (e.g., Python). This function might take two integers as input and return their product using the Karatsuba algorithm.
- Timing results for the Karatsuba algorithm for different input sizes, compared to other multiplication methods. These results might be presented in the form of tables or graphs, and could show the relative efficiency of the Karatsuba algorithm for different input sizes.
- An analysis of the time and space complexity of the Karatsuba algorithm, using techniques such as asymptotic analysis or empirical testing. This analysis could provide insight into the performance of the algorithm for different input sizes, and could be used to compare the Karatsuba algorithm to other multiplication methods.
- Conclusions and recommendations based on the results obtained. These conclusions might include a summary of the performance of the Karatsuba algorithm, and may suggest further improvements or refinements that could be made to the algorithm.

Overall, the results of the Karatsuba multiplication project will provide valuable information on the effectiveness and efficiency of the Karatsuba algorithm for multiplying large integers, and could be used to inform the development of other algorithms or systems that rely on fast multiplication methods.

Recommendations

There are many potential directions that could be taken to further improve or extend the Karatsuba multiplication algorithm, or to address other related problems. Some possible areas for future work might include:

Optimizing the implementation of the Karatsuba algorithm for specific programming languages or hardware architectures. For example, techniques such as loop unrolling or hardware-specific optimizations could be used to improve the performance of the algorithm.

Investigating alternative multiplication algorithms that might be more efficient than the Karatsuba algorithm for certain input sizes or types of inputs. For example, there are several other fast multiplication methods that have been developed, such as the Toom-Cook algorithm or the Schönhage-Strassen algorithm, which might be more efficient than the Karatsuba algorithm in certain cases. [6]

Applying the Karatsuba algorithm to other mathematical problems that involve large integer multiplication, such as polynomial multiplication or matrix multiplication.

Developing hybrid algorithms that combine the Karatsuba algorithm with other methods to achieve even better performance. For example, the Karatsuba algorithm could be combined with the naive algorithm to create a hybrid method that is efficient for both small and large input sizes.

Overall, the development and improvement of fast multiplication algorithms is an active area of research, and there are many possibilities for further work in this area.

Our Video

https://www.youtube.com/watch?v=Yy8W_TapCGA&ab_channel=Yi%C4%9Fit

References

[1]: https://iq.opengenus.org/karatsuba-algorithm/

[2]: https://courses.csail.mit.edu/6.006/spring11/exams/notes3-karatsuba

[3]: https://techvidvan.com/tutorials/python-advantages-and-disadvantages/

[4]:https://math.mit.edu/research/highschool/primes/circle/documents/2022/Zoe%20&%20Palak.pdf

[5]: https://python.plainenglish.io/karatsuba-multiplication-65a2efcccfd9

[6]:

https://brilliant.org/wiki/karatsuba-algorithm/#:~:text=The%20Karatsuba%20algorithm%20is%20a,approach%20to%20multiply%20two%20numbers.

[7]: https://github.com/ray-dino/karatsuba-multiplier/blob/master/karatsuba.py

Appendix [7]

```
import math

"""
Karatsuba multiplication method for positive integers
INPUT: Integer x, Integer y
OUTPUT: Multiplication result integer r
"""
def karatsuba(x, y):
  #Convert x and y to strings and get their lengths
  sx = str(x)
  sy = str(y)
  nx = len(sx)
  ny = len(sy)

  #If one of the strings length is 1 then just multiply them and return result
  if nx==1 or ny==1:
    r = int(x)*int(y)
    return r

  #Look for the longer string and zero pad the left side of shorter string so that they are equal
  n = nx   #If they are already equal then this line decleares the n variable
  if nx>ny:
    sy = sy.rjust(nx, '0')
    n = nx
  elif ny>nx:
    sx = sx.rjust(ny, '0')
    n = ny

  #Check whether they the number is even or not if not then increase n to make it even
  m = n % 2
  offset = 0
  if m!=0:
    n+=1
    offset = 1

  #Find the ceil and floor so of the new strings to divide the string into two pieces
  floor = int(math.floor(n/2))-offset
  ceil = int(math.ceil(n/2))-offset

  #Divide the strings into 2 pieces and get 4 new string
  a = sx[0:floor]
  b = sx[ceil:n]
  c = sy[0:floor]
  d = sy[ceil:n]

  #Apply the karatsuba algorithm recursively and return the result
  r = int(((10**n)*karatsuba(a,c)) + ((10**(n/2))*(karatsuba(a,d) + karatsuba(b,c))) +
karatsuba(b,d))
  return r
```