```
[13] from random import randint
     import math
     import numpy as np
     import matplotlib.pyplot as plt
     import time

     N = range(1024, 2049, 56)

     A = list()
     B = list()
     for n in N:
       a = randint(10 ** (n - 1), 10 ** n - 1)
       b = randint(10 ** (n - 1), 10 ** n - 1)
       A.append(a)
       B.append(b)
```

Tạo danh sách N chứa các số mà ta sẽ lấy ra để tạo ngẫu nhiên A, B nguyên có N chữ số (sử dụng randint). Một điểm khác so với đề bài là N là *range(1024, 2049, 56)* thay vì là 2^k với k trong khoảng [10, 32] do khi N càng lớn, tức càng có nhiều chữ số thì thuật toán nhân sẽ chạy càng lâu (do độ phức tạp là hàm mũ, thuật toán nhân truyền thống bắt đầu tốn nhiều hơn 1 giờ khi k > 12).

```
[9]  def traditional_multiply(A, B):
       result = 0
       for i in range(len(A)):
         num1 = A[i]
         for j in range(len(B)):
           num2 = B[j]
           result = result + int(num1) * int(num2) * 10 ** ((len(A) - 1 - i) + (len(B) - 1 - j))
       return result

[11] time_traditional = list()
     result_traditional = list()
     for a, b in zip(A_2, B_2):
       start_time = time.time()
       result_traditional.append(traditional_multiply(str(a), str(b)))
       time_traditional.append(time.time() - start_time)
```

Thuật toán nhân truyền thống, ta lần lượt lấy từng chữ số của A nhân với từng chữ số B, sau đó nhân 10^x với x là tổng vị trí của hai chữ số được lấy ra nhân. Ví dụ, ta lấy 2 trong 121 (vị trí 1 từ phải qua, ứng với hàng chục), nhân với 1 trong 1000 (vị trí thứ 3 từ phải qua, ứng với hàng nghìn), ta sẽ có kết quả là $2 * 1 * 10^{1+3} = 20000$. Kết quả có được trong mỗi lần nhân sẽ được cộng lại với nhau để cho ra kết quả cuối cùng.

Ta bắt đầu chạy với các cặp số A, B. Thời gian chạy cho mỗi phép nhân sẽ được tính sẽ dụng module time, sau đó lưu vào mảng. Mục tiêu chính của chúng ta là kiểm tra độ phức tạp thuật toán, do đó ta sẽ không in kết quả phép nhân ra, thay vào đó ta sẽ lưu vào một mảng kết quả.

```
[16] def karatsuba(A, B):
       if (int(A) < 10 or int(B) < 10):
         return int(A) * int(B)

       split = math.ceil(min(len(A), len(B)) / 2)
       high1, low1 = A[:-split], A[-split:]
       high2, low2 = B[:-split], B[-split:]

       z0 = karatsuba(low1, low2)
       z1 = karatsuba(str(int(high1) + int(low1)), str(int(high2) + int(low2)))
       z2 = karatsuba(high1, high2)

       return (z2 * 10 ** (split * 2)) + ((z1 - z2 - z0) * 10 ** split) + z0
```

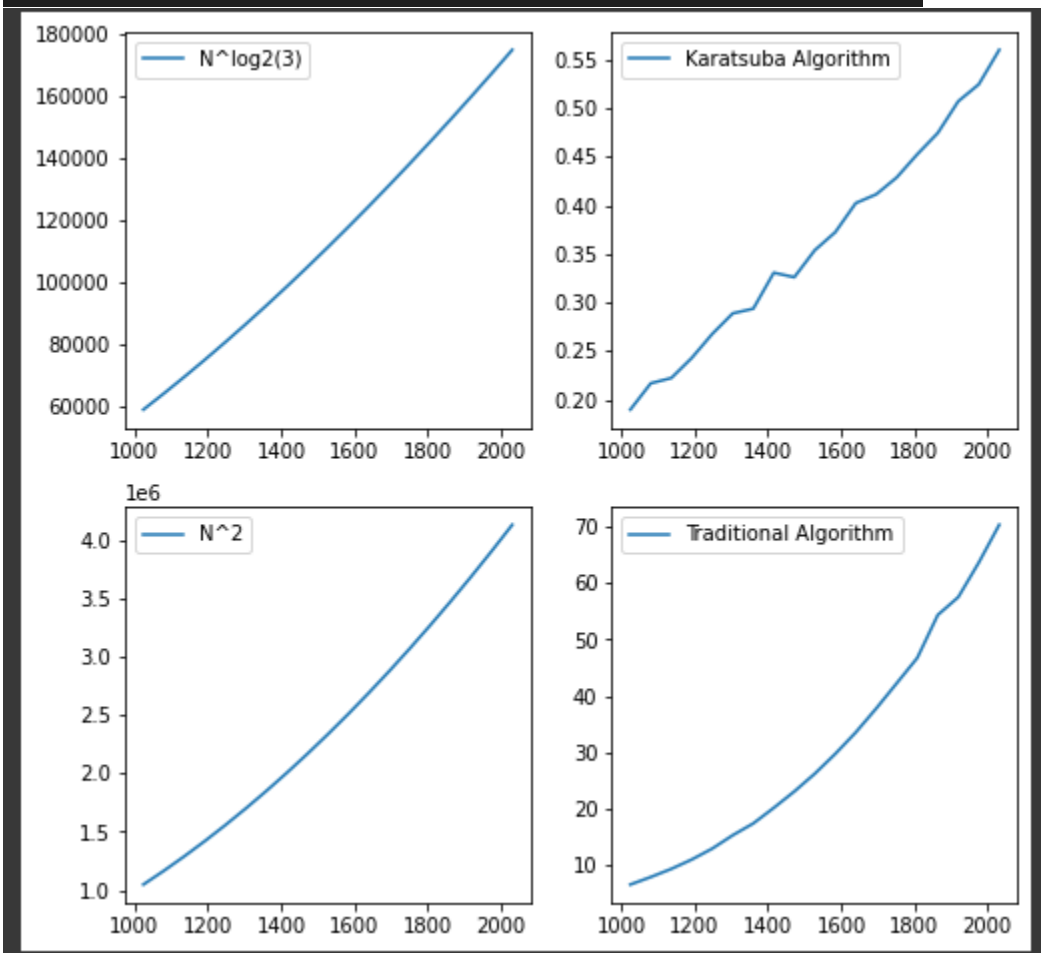
```
time_karatsuba = list()
result_karatsuba = list()
i=0
for a, b in zip(A, B):
  i+=1
  print(i)
  start_time = time.time()
  result_karatsuba.append(karatsuba(str(a), str(b)))
  time_karatsuba.append(time.time() - start_time)
```

Với thuật toán Karatsuba, ta tiến hành chọn điểm cắt 2 số truyền vào, lấy độ dài của số có ít chữ số nhỏ nhất trong hai chữ số chia 2 (trong bài này thì hai số có số chữ số bằng nhau), nếu số chữ số là lẻ thì ta có thể làm tròn lên hoặc xuống tùy ý (ở đây làm tròn lên bằng method ceil của module math). Với số vừa tính được, ta sẽ đi cắt từ bên phải của hai số A, B bấy nhiêu chữ số. Ví dụ, ta tính được 1, và A là 1234, khi đó ta sẽ cắt thành “123” và “4”.

Tiếp đến, ta gọi đệ quy để tính các số z_0, z_1, z_2 theo thuật toán Karatsuba, điều kiện dừng là khi có ít nhất một số truyền vào bé hơn 1, khi đó ta sẽ nhân bằng phép nhân truyền thống. Lưu ý, thuật toán Karatsuba được viết ở đây nhận tham số truyền vào là chuỗi, do đó ở z_1, sau khi chuyển kiểu dữ liệu từ string sang int cho high và low để cộng

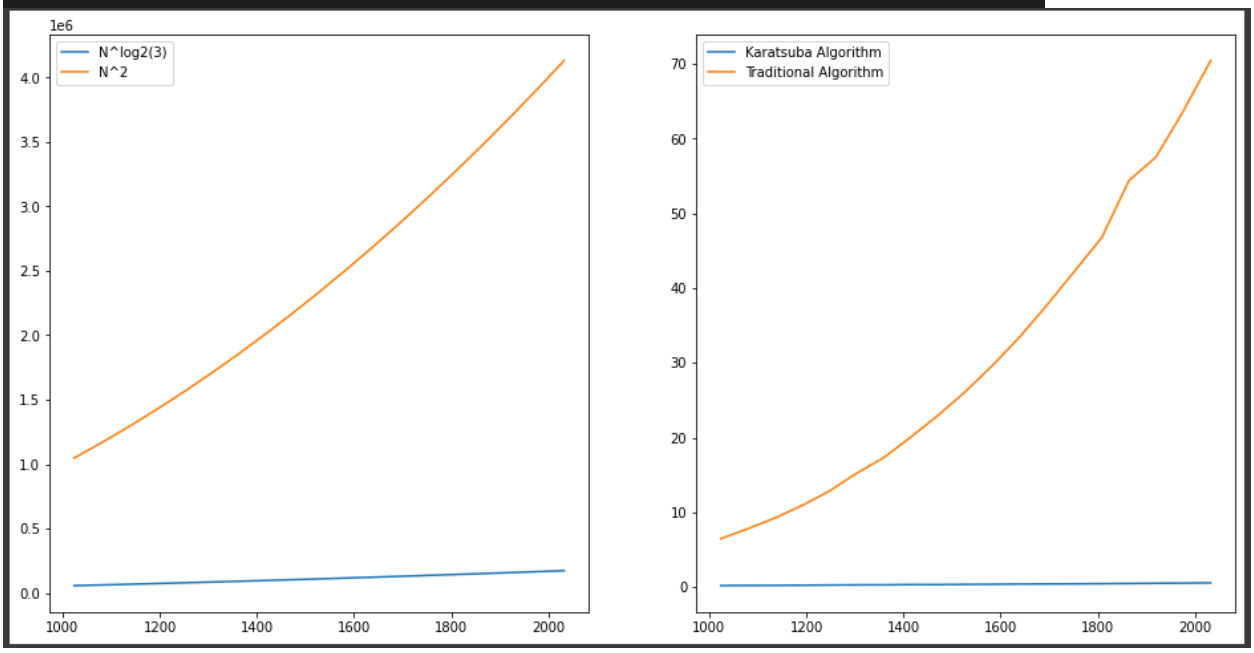
hai số lại với nhau, ta cần chuyển kết quả sang string để có thể truyền vào thuật toán Karatsuba.

```
[38] plt.figure(figsize=(8, 8))
     plt.subplot(221)
     plt.plot(N, [i ** math.log2(3) for i in N], label = "N^log2(3)")
     plt.legend()
     plt.subplot(222)
     plt.plot(N, time_karatsuba, label = "Karatsuba Algorithm")
     plt.legend()
     plt.subplot(223)
     plt.plot(N, [i ** 2 for i in N], label = "N^2")
     plt.legend()
     plt.subplot(224)
     plt.plot(N, time_traditional, label = "Traditional Algorithm")
     plt.legend()
     plt.show()
```



Khi vẽ riêng từng biểu đồ, ta thấy rõ được hình dạng nhưng không thấy rõ được sự chênh lệch giữa hai thuật toán.

```
plt.figure(figsize=(16, 8))
plt.subplot(121)
plt.plot(N, [i ** math.log2(3) for i in N], label = "N^log2(3)")
plt.plot(N, [i ** 2 for i in N], label = "N^2")
plt.legend()
plt.subplot(122)
plt.plot(N, time_karatsuba, label = "Karatsuba Algorithm")
plt.plot(N, time_traditional, label = "Traditional Algorithm")
plt.legend()
plt.show()
```



Khi vẽ chung các biểu đồ, ta có thể thấy rõ thuật toán truyền thống chạy lâu hơn Karatsuba rất nhiều, đồng thời có thể khẳng định đúng thật là thuật toán truyền thống có độ phức tạp $O(N^2)$, thuật toán Karatsuba có độ phức tạp $O(N^{log2(3)})$.