TUẦN 3

Họ & Tên: Phạm Nguyễn Phương Duy
MSSV: 19110290

Bài 1

[15]
```
from sympy import O, Symbol
from math import log

def time_complexity(f, a, b):
    # Khai báo biến n là một biểu thức
    n = Symbol('n')

    # Tính toán độ phức tạp của f(n) dưới dạng O(n^α)
    big_o = O(f(n), (n, b)).expr

    # Kiểm tra xem độ phức tạp có dạng O(n^α) hay không
    if big_o.is_Pow:
        # Nếu có, trả về giá trị của α
        alpha = big_o.args[1]
        return f"Độ phức tạp của f(n) là O(n^{alpha})"
    else:
        # Nếu không, hiện thông báo
        return "Độ phức tạp không có dạng O(n^α)"

# Kiểm tra lại hàm với các trường hợp sau:
print(time_complexity(lambda n: n**2, 10, 1000)) # a) f(n) = n^2
print(time_complexity(lambda n: n**3 + cos(n)*n**4, 10, 1000)) # b) f(n) = n^3 + cos(n)*n^4
print(time_complexity(lambda n: n**n, 10, 1000)) # c) f(n) = n^n
print(time_complexity(lambda n: n**3 + n**2 + n + 1, 10, 1000)) # d) f(n) = n^3 + n^2 + n + 1
```

```
Độ phức tạp không có dạng O(n^α)
Độ phức tạp không có dạng O(n^α)
Độ phức tạp không có dạng O(n^α)
Độ phức tạp không có dạng O(n^α)
```

Bài 2

a) Phương pháp truyền thống với độ phức tạp thời gian O(N^2):

[10]
```python
def traditional_multiplication(A: str, B: str) -> str:
    # Khởi tạo kết quả
    result = [0] * (len(A) + len(B))

    # Đảo ngược chuỗi A và B
    A = A[::-1]
    B = B[::-1]

    # Nhân từng chữ số của A với từng chữ số của B
    for i in range(len(A)):
        carry = 0
        for j in range(len(B)):
            temp = int(A[i]) * int(B[j]) + carry + result[i+j]
            carry = temp // 10
            result[i+j] = temp % 10
        result[i+len(B)] += carry

    # Loại bỏ các số 0 dư thừa ở đầu kết quả
    while len(result) > 1 and result[-1] == 0:
        result.pop()

    # Đảo ngược lại kết quả và trả về dưới dạng chuỗi
    return ''.join(map(str, result[::-1]))
```

[11]
```python
A = '1234'
B = '4321'
print(traditional_multiplication(A, B))
```

```
5332114
```

b) Phương pháp cải tiến với độ phức tạp thời gian O(NlogN):

[12]
```python
from numpy.fft import fft, ifft

def improved_multiplication(A: str, B: str) -> str:
    # Chuyển đổi chuỗi A và B thành mảng số nguyên và đảo ngược
    A = list(map(int, A[::-1]))
    B = list(map(int, B[::-1]))

    # Tìm giá trị n nhỏ nhất là lũy thừa của 2 lớn hơn max(len(A), len(B))
    n = 1
    while n < max(len(A), len(B)):
        n <<= 1

    # Nhân n với 2 để đảm bảo đủ chỗ cho kết quả nhân
    n <<= 1

    # Thực hiện biến đổi Fourier nhanh cho A và B
    A = fft(A + [0] * (n - len(A)))
    B = fft(B + [0] * (n - len(B)))

    # Nhân hai mảng A và B trong miền tần số và thực hiện biến đổi Fourier nghịch đảo
    C = ifft(A * B)

    # Làm tròn các giá trị thực và chuyển đổi thành mảng số nguyên
    C = list(map(int, map(round, C.real)))

    # Xử lý các số dư (carry)
    carry = 0
    for i in range(n):
        C[i] += carry
        carry = C[i] // 10
        C[i] %= 10

    # Loại bỏ các số 0 dư thừa ở đầu kết quả
    while n > 1 and C[-1] == 0:
        C.pop()
        n -= 1

    # Đảo ngược lại kết quả và trả về dưới dạng chuỗi
    return ''.join(map(str, C[::-1]))
```

[13]
```python
A = '1234'
B = '4321'
print(improved_multiplication(A, B))
```

```
5332114
```