Report lab 3

Nguyễn Quốc Bảo - 18110053

Câu 1 Để $f(n)$ có độ phức tạp là $O(n^{\alpha})$

```
def fix_valid(equation):
    equation = equation.replace(" ", "")
    equation = equation.replace("^", "**")
    equation = equation.replace("-","+ (-1)*")
    return equation

def eval_polynomial(equation, x_value):
    fixed_equation = fix_valid(equation.strip())
    parts = fixed_equation.split("+")
    x_str_value = str(x_value)
    parts_with_values = (part.replace("n", x_str_value) for part in parts )
    partial_values = (eval(part) for part in parts_with_values)
    return sum(partial_values)
```

Hàm **fix-valid** dùng để chuyển đổi các ký tự cho phù hợp cho việc tính toán với các tham số.

Hàm **eval-polynomial** dùng để tính từng phép toán cho từng toán tử với giá trị nằm trong khoảng a,b cho trước.

```
def compute_exponent(equation,N):
    temp = 0
    for n in N:
        F = eval_polynomial(equation,n)
        alpha = log(abs(F))/log(n)
        M = max(alpha, temp)
        temp = alpha
    return int(M)
```

Hàm **compute-expoenet** để tính xấp sỉ α sao cho $f(n) = O(n^{\alpha})$, khi đó $f(n) \sim n^{\alpha}$ nên ta lấy log 2 vế ta có thể tính được α.

```
def find_maximum_degree(equation):
    fixed_equation = fix_valid(equation.strip())
    parts = fixed_equation.split("+")
    degree = []
    for part in parts:
        idx = part.find('**')
        while (part[idx].isdigit() != True):
            if (part[idx] == 'n' ):
                return 'n'
            idx += 1
        degree.append(part[idx])
    return max(degree)
```

Hàm **find-maximum-degree** để tìm bậc của đa thức, cũng là bậc cao nhất của đa thức.
Ta có kết quả như sau:

```
-----------Exercise 1------------
check with the following cases:
a = 10 and b = 1000
1 ) f(n) =  n^2
when f(n) = n^2 then f(n) = O(n^2)
2 ) f(n) =  n^3 + cos(n)*n^4
when f(n) = n^3 + cos(n)*n^4 then f(n) = O(n^4)
3 ) f(n) =  n^n
when f(n) = n^n then f(n) = O(n^1000)
4 ) f(n) =  n^3 + n^2 + n + 1
f(n) has no form O(n^3)

>> input f(n): n^5 + 5*n^6 + cos(n)*n^4
>> Input number a: 10
>> Input number b: 1000
when f(n) = n^5 + 5*n^6 + cos(n)*n^4  then f(n) = O(n^6)
                 END EX1
```

Ta thử kiểm tra với các trường hợp cụ thể

$$1) f(n) = n^2$$
$$2) f(n) = n^3 + cos(n) * n^4$$
$$3) f(n) = n^n$$
$$4) f(n) = n^3 + n^2 + n + 1$$

Câu 2 Chương trình nhân 2 số nguyên lớn A, B có N chữ số

Phương pháp cổ điển

```python
# Phương pháp cổ điển
def multiply(A,B):
    global counter_assign
    global counter_compare
    #--- Nhân từng phần tử từ phải sang trái ---
    result = []
    for i in range(len(B)-1,-1,-1):
        memory = 0
        ls = [0]*(len(B)-i-1)
        counter_assign += 2
        for j in range(len(A)-1,-1,-1):
            r = int(B[i])*int(A[j])
            if (memory != 0):
                r = r + memory
                memory = 0
            temp = r
            if (temp >= 10):
                k = temp%10
                memory = int(temp/10)
            else:
                k = r
            ls.append(k)
            counter_compare += 2
            counter_assign += 6
```

Ta dễ dàng thấy số lần lặp của 2 vòng lặp có chiều dài n phần tử, là n^2.Và các vòng lặp còn lại đều bằng n nên, ta có thể xem hàm **multiply** có độ phức tạp là $O(n^2)$.

```python
        while (len(ls) != (len(A)+len(B))):
            ls.append(0)
        ls.reverse()
        result.append(ls)
        # print(ls)
    result = np.array(result)


    #---Cộng các giá trị vừa nhân theo hàng dọc---
    Sum = list(sum(result))
    memory = 0
    counter_assign += 2
    for i in range(len(Sum)-1,-1,-1):
        temp = Sum[i]
        if (memory != 0):
            temp = temp + memory
            memory = 0
        if (temp >= 10):
            k = temp%10
            memory = int(temp/10)
        else:
            k = temp
        Sum[i] = k
        counter_compare += 2
        counter_assign += 4
    while (Sum[0] == 0):
        Sum.pop(0)

    return Sum
```

Với phương pháp cổ điển này, ta thực hiện 2 bước:
- Bước 1: nhân từng phần tử của B cho từng phần tử của , kết quả được dịch sang trái 1 vị trí sau mỗi lần nhân.
- Bước 2: cộng các giá trị đã nhân được theo hàng dọc.

Phương pháp nhân nhanh của Karatsuba

$$A = A_1 * 10^{n/2} + A_2$$
$$B = B_1 * 10^{n/2} + B_2$$

Đặt

$$C = A_1 * B_1$$
$$D = A_2 * B_2$$
$$E = (A_1 + A_2)(B_1 + B_2) - C - D$$

Khi đó

$$A * B = C * 10^n + E * 10^{n/2} + D$$

```
def karatsuba(X, Y):
    global counter_assign
    global counter_compare

    counter_compare += 1
    # -- Base case --
    if X < 10 and Y < 10:
        return X * Y

    counter_assign += 1
    # --- determine the size of X and Y ---
    size = max(len(str(X)), len(str(Y)))

    # --- Split X and Y ---
    n = ceil(size/2)
    p = 10 ** n
    a = floor(X // p)
    b = X % p
    c = floor(Y // p)
    d = Y % p
    counter_assign += 6

    # --- Recur until base case ---
    ac = karatsuba(a, c)
    bd = karatsuba(b, d)
    e = karatsuba(a + b, c + d) - ac - bd

    # --- return the equation ---
    return int(10 ** (2 * n) * ac + (10 ** n) * e + bd)
```

Nếu $n = 2^k$ với k bất kỳ, thì thuật toán sẽ lặp lại ba lần trên $\frac{n}{2}$. Và có O(n) phép cộng và trừ cần thiết cho thuật toán. Do đó, sự lặp lại tổng thể cho thuật toán Karatsuba là

$$\begin{cases} T(n) = 3T([\frac{n}{2}]) + O(n) & n > 1 \\ T(1) = 1 & n = 1 \end{cases}$$

Bằng quy nạp toán học

$$\begin{aligned} &T(n) = 3T([\frac{n}{2}]) + O(n) \\ \Longrightarrow &T(n) = 3^{\log(n)} + O(n) \end{aligned}$$

Đặt

$$\begin{aligned} x = 3^{\log(n)} &\Longrightarrow \log_3 x = \log n \\ &\Longrightarrow \log_3 x = \frac{\log x}{\log 3} \\ &\Longrightarrow logx = \log 3 \log_3 x = \log 3 \log n = \log n^{\log 3} \\ &\Longrightarrow x = n^{\log 3} \end{aligned}$$

Vậy $T(n) = O(n^{\log 3})$

Ta chạy thử với 2 phương pháp có kết quả:

```
-----------Exercise 2------------
>> input number A: 1234
>> input number B: 1324
>> input number of digits N: 4
By classic method
C = A.B = 1633816
count assign: 138 step and count compare: 48 step
------------------------------------------------------------------------------
By Karatsuba method
C = A.B = 1633816
count assign: 35 step and count compare: 16 step
                END EX2
```

kiểm tra chương trình lại với $N = 2^k$, k =10,11,....,32. Ta có kết quả trong file **result-ex2.txt**