Lý thuyết 5: Thử và lỗi

Exercise 1.

a) Design a brute-force algorithm for computing the value of a polynomial

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$$

at a given point x_0 and determine its worst-case efficiency class.

b) If the algorithm you designed is in $\Theta(n^2)$, design a linear algorithm for this problem.

Gộp ý a và b, cách giải $O(n)$ là như sau:

```
def polynomial(a, x):
    n = len(a)
    p = a[n-1]
    for i in range(n-2, -1, -1):
        p = p*x + a[i]
    return p
```

c) Is it possible to design an algorithm with a better-than-linear efficiency for this problem?
Có thể là Fast fourier transform

Exercise 2.

Sort the list S, O, R, T , I, N, G in alphabetical order by selection sort.

Selection sort là lần lượt tìm phần tử nhỏ nhất rồi đưa lên vị trí hiện tại, sau đó tăng biến đấy lên 1, đồng thời loại bỏ phần tử đó ra khỏi danh sách.

Khởi tạo: Danh sách cần sắp xếp là `S, O, R, T, I, N, G`

Khởi tạo `ans = []` là mảng rỗng.

Ban đầu duyệt qua 7 phần tử, `G` là bé nhất:

⇒ `ans = [G]`, danh sách còn lại là `S, O, R, T, I, N`

Tiếp tục, trong 6 phần tử còn lại, `I` bé nhất

⇒ `ans = [G, I]`, danh sách còn lại là `S, O, R, T, N`

Trong 5 phần tử, `N` bé nhất

⇒ `ans = [G, I, N]`, danh sách còn lại là `S, O, R, T`

Trong 4 phần tử, `O` bé nhất

⇒ `ans = [G, I, N, O]`, danh sách còn lại là `S, R, T`

Trong 3 phần tử, `R` bé nhất

⇒ `ans = [G, I, N, O, R]`, danh sách còn lại là `S, T`

Trong 2 phần tử, `T` bé nhất

⇒ `ans = [G, I, N, O, R, T]`, danh sách còn lại là `S`

Còn lại là `S`, thêm vào `ans = [G, I, N, O, R, T, S]`

Lúc này ta có mảng đã được sắp xếp.

Exercise 3.

Sort the list S, O, R, T , I, N, G in alphabetical order by bubble sort.
Phương pháp nổi bọt: Duyệt 2 vòng for, vòng for thứ 1 đảm bảo cho những phần tử ở trước nó đã được sắp xếp, vòng for thứ 2 tìm kiếm những phần tử đang bé hơn phần tử hiện tại để đổi chỗ.

Việc biến đổi sẽ được thực hiện trực tiếp trên 1 mảng, ở đây là copy từ mảng gốc.

`ans = [S, O, R, T, I, N, G]`

```
for i in range(0, len(ans)-2):
  for j in range(i+1, len(ans)-1):
    if (ans[i] > ans[j]):
      ans[i], ans[j] = ans[j], ans[i] # swap.
```

i= 0 j= 1 => ['O', 'S', 'R', 'T', 'I', 'N', 'G']
i= 0 j= 4 => ['I', 'S', 'R', 'T', 'O', 'N', 'G']

i= 0 j= 6 => ['G', 'S', 'R', 'T', 'O', 'N', 'I']
i= 1 j= 2 => ['G', 'R', 'S', 'T', 'O', 'N', 'I']
i= 1 j= 4 => ['G', 'O', 'S', 'T', 'R', 'N', 'I']
i= 1 j= 5 => ['G', 'N', 'S', 'T', 'R', 'O', 'I']
i= 1 j= 6 => ['G', 'I', 'S', 'T', 'R', 'O', 'N']
i= 2 j= 4 => ['G', 'I', 'R', 'T', 'S', 'O', 'N']
i= 2 j= 5 => ['G', 'I', 'O', 'T', 'S', 'R', 'N']
i= 2 j= 6 => ['G', 'I', 'N', 'T', 'S', 'R', 'O']
i= 3 j= 4 => ['G', 'I', 'N', 'S', 'T', 'R', 'O']
i= 3 j= 5 => ['G', 'I', 'N', 'R', 'T', 'S', 'O']
i= 3 j= 6 => ['G', 'I', 'N', 'O', 'T', 'S', 'R']
i= 4 j= 5 => ['G', 'I', 'N', 'O', 'S', 'T', 'R']
i= 4 j= 6 => ['G', 'I', 'N', 'O', 'R', 'T', 'S']
i= 5 j= 6 => ['G', 'I', 'N', 'O', 'R', 'S', 'T']

Exercise 4.

The average number of key comparisons made by sequential search (without a sentinel, under standard assumptions about its inputs) is given by the formula

$$C(n) = p(n+1)/2 + n(1-p)$$

where p is the probability of a successful search. Determine, for a fixed n, the values of p $(0 \leq p \leq 1)$ for which this formula yields the maximum value of $C(n)$ and the minimum value of $C(n)$.

Code:

```
def ex4(p, n):
    formula = p*(n+1)/2 + n*(1-p)
    return formula

n = 100
step = 100
mn = ex4(0, n)
mx = ex4(1, n)
for i in range(1, step):
    mn = min(mn, ex4(1/i, n))
    mx = max(mx, ex4(1/i, n))
```

```
print(mn, mx)
```

Vét cạn ở chỗ chúng ta sẽ chia nhỏ p ra để tìm kết quả tốt nhất. Số lần chia nhỏ là biến `step`. `step` càng lớn thì kết quả càng chính xác.