[과제 #1] O(n) 시간 vs. O(n^2) 시간 체험하기

$O(n)$ 코드 구현의 경우 재귀함수와 일반 For Loop문 두 개들 모두 구현 하였습니다.

실행 코드 (재귀함수)

```
# -*- coding:utf-8 -*-
import time, random, sys
sys.setrecursionlimit(10000) # recusion depth limit을 10000으로 설정

def evaluate_n2(A, x, *args):
    fx = 0
    tmp = x
    for i, a in enumerate(A):
        if i == 0:
            x =1
        for _ in range(i - 1):
            x *= tmp
        fx += a * x
        x = tmp
    return fx


def evaluate_n(A, x, n):
    if n == 1:
        return A[-1]
    return A[-n] + x * evaluate_n(A, x, n - 1)


random.seed()  # random 함수 초기화
n = int(input())
A = [random.randint(-999, 999) for _ in range(n)]
x = random.randint(-99, 99)


# evaluate_n(A,x, len(A))
# evaluate_n2(A,x)


def time_measure(eval_func, A, x, n=None):
    before = time.process_time()
    eval_func(A, x, n)
    after = time.process_time()
    return print(after - before)


time_measure(evaluate_n, A, x, len(A))
time_measure(evaluate_n2, A, x)
```

실행코드(일반 For문)

```
# -*- coding:utf-8 -*-
import time, random
import matplotlib.pyplot as plt

def evaluate_n2(A, x, *args):
    fx = 0
    tmp = x
    for i, a in enumerate(A):
        if i == 0:
            x =1
        for _ in range(i - 1):
            x *= tmp
        fx += a * x
        x = tmp
    return fx


def evaluate_n(A, x):
    fx = A[-1]
    for i in range(1,len(A)):
        if i-1 == len(A):
            x = 1
        fx = fx* x + A[-i-1]
    return fx

random.seed()  # random 함수 초기화
n = int(input())
A = [random.randint(-999, 999) for _ in range(n)]
x = random.randint(-99, 99)


def time_measure(eval_func, A, x):
    before = time.process_time()
    eval_func(A, x)
    after = time.process_time()
    return (after - before)


time_measure(evaluate_n, A, x)
time_measure(evaluate_n2, A, x)
```

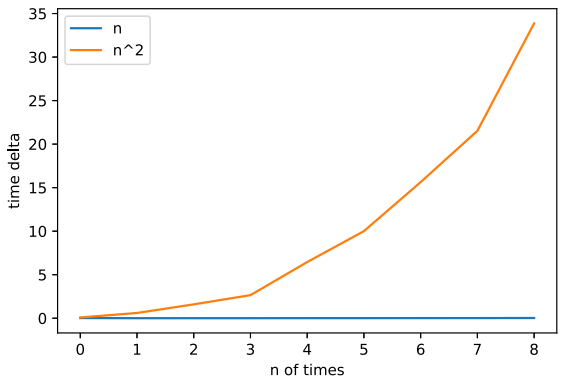
실행 시간 비교

실행 시간 비교를 위해 다음과 같이 코드를 추가함.

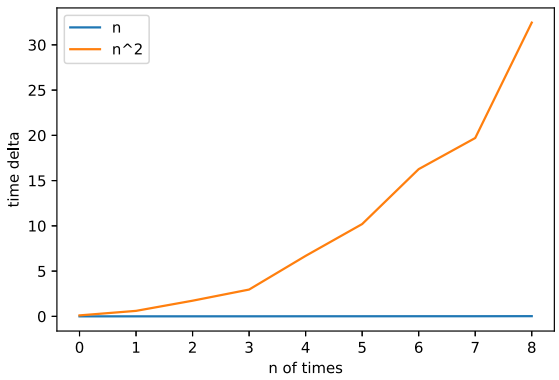
```
times = range(500, 3000, 50)
n_time = []
n2_time = []
for n in times:
    A = [random.randint(-999, 999) for _ in range(n)]
    x = random.randint(-99, 99)
    n_time.append(time_measure(evaluate_n, A, x, len(A)))
    n2_time.append(time_measure(evaluate_n2, A, x))

# time_measure(evaluate_n,A,x)
# time_measure(evaluate_n2,A,x)

plt.plot(n_time, label="n")
plt.plot(n2_time, label="n^2")
plt.ylim(top=2)
plt.legend()
plt.xlabel("n of times")
plt.ylabel("time delta")
plt.show()
```



재귀 함수를 통해 O(n) 구현



일반 For문을 통한 O(n) 구현

위 그래프를 보면 n의 실행 횟수에 비례하여 $O(n^2)$ 의 그래프는 $O(n)$ 그래프에 비해 기울기가 지수 형태로 매우 가파르게 상승하는 것을 알 수 있다. 또한 재귀 함수로 실행 했을 때와 재귀 호출 문을 사용했을 때 $O(n)$그래프가 비슷하나 나왔다.

느낀점

5일동 안 재귀 알고리즘 아이디어가 떠오르지 않아 고생을 많이 했다. 과제 진행하는데 10시간 이상 앉아서 고민 한적은 처음 이였다. 하지만, 이번 계기로 재귀 알고리즘 개념을 완벽히 이해 하여 자신감이 생겼다.

자료구조 교재나 numpy 교재를 첫 부분에 보면 항상 실행 시간을 비교하는 챕터가 있었는데 해당 교재나 인터넷을 참고하지 않고 밑바닥부터 차근차근 과제를 수행 하였다. (과제 수행시 최대한 혼자 수행 해보라는 선배의 조언이 있었기 때문) 짧지만 코드 한 줄 한 줄 기억을 끄집어 내며 그래프까지 완성 시켜 보람찼다.

더불어, 자료구조는 C언어로 배웠는데 iteration 횟수가 1000 이상이 넘어 갈 수록 C언어에 비해 느린 것을 체감 할 수 있었다. 실제 프로그래밍에서 왜 numpy 같은 외부 라이브러리를 사용하는지도 체감 할 수 있었다.

첨부

알고리즘 구현 과정

$n=0$ . .

$f(n) = f(k) +$

$f(n) = a_0 \cdot n^0 + a_1 \cdot n^1 + a_2 \cdot n^2 + a_3 \cdot n^3$

$k^0 + k^1 + k^2 + k^2 + k^3 + \cdots k^{n-1}$

$Sum(n) = Sum(n-1) + k^n +$

$Sum(4) = k^0 + k^1 + k^2 + k^3$

$= 1 + k + k \cdot k + k \cdot k + k =$

$1 + k(1 + k + k \cdot k) = 1 + k \times Sum(3)$

$1 + k(1+k)$

$(1+k) \times (Sum$

$Sum(4)$