Lecture 2 : 성능 분석과 알고리즘의 복잡도 (Complexity)

우리의 결과물은 결국 코드(code)의 형식으로 제공된다. 이 생산된 코드가 원하는 수준에 있는지의 여부는 다양한 test와 분석, 검증(verify)을 통하여 결정된다. 가장 보편적인 방법은 미리 준비한 test case data를 이용해서 올바르게 코딩되었는지를 살펴보는 것이라 할 수 있지만, 문제는 우리의 확신을 위해서 얼마나 다양한, 얼마나 많은 test case를 준비해야 하는가를 결정하는 것이라 하겠다. 물론 test case가 많으면 많을수록 좋겠지만 모든 가능한 경우를 가정한 수많은 test case를 준비하는 것은 현실적으로 불가능하다.

이를 위하여 컴퓨터 과학에서는 코드 자체에 내재한 복잡도를 수학적 함수를 이용하여 표현하는 방법을 사용하고 있다. 즉 데이터의 크기를 추상적인 크기 N으로 표시했을 때 제시한 알고리즘(코드)의 복잡한 정도를 N의 함수 $f(N)$으로 표현할 수 있다. 이 주제는 이후 알고리즘에서 본격적으로 다루겠지만 이번 자료구조에 기본적인 취지에 대하여 몇 가지 예를 사용하여 설명하고자 한다.

2.1 과학의 목적, 컴퓨터 과학의 목적, 컴퓨터 공학의 목적을 아래 기준으로 설명하시오.

- 예측
- 결과 획득
- 결과 분석
- 모형 설정

예) 100 mega byte의 영어 문서에서 관사 “a”나 “the”가 몇 번 나타나는지 조사하기

Q) 제일 좋은 자료구조는 어떤 자료구조인가? 같은 방법으로 아래를 설명하시오.

- 제일 좋은 지갑
- 제일 좋은 식당
- 제일 좋은 친구
- 제일 좋은 책상 정리 방법
- 제일 좋은 자료구조, 알고리즘

2.2 어떤 수열 $L = \{a_i\}$ = [23, 23, 5, 6, -10, -22, ...-9, 12, 45, 78] 이 있다. 그 길이는 N이다. 이 수열에서 i에서 j까지의 합을 $S_{i,j} = \sum_{i}^{j} a_i$라고 하자. 이 $S_{i,j}$를 빨리 구하는 방법과 이것을 위한 자료구조를 제시하시오.

- 방법1 - 2개의 for-loop으로 구하는 방법 (straightforward 한 방법)
- 방법2 - 미리 모든 경우에 대하여 답을 마련해주는 방법, 행렬 형식의 2차원 배열
- 방법3 - a[0]에서 a[i]까지의 누적 합을 S[i]에 저장해두는 방법

2.3 위 문제에서 정답을 구하는 알고리즘의 복잡도를 N의 함수로 제시하시오.

2.4 Standard Template Library란 무엇이며 어떤 의미가 있는지 설명하시오.

- Component-ware란 무엇인가?
- 여러분이 회사에 입사 후 <인터넷 쇼핑몰> 제작 명령을 받았다면 어떤 일부터 시작해야 할지 말해 보시오.
- 문제는 개발의 시작을 어떤 level부터 시작할 것인가를 결정하는 것이다.
  - Assembly level Or Machine Code
  - High level language
  - Public component ware = { STL, BOOST, JDSL, LEDA. }
  - Commercial Package
  - Buying Turn-Key System

2.6 성능 분석(이론적)과 측정(실제적)

- 왜 성능 분석을 해야하는가?
- 어떤 면을 분석해야 하는가?
  a) 시간 (Time)
  b) 공간 (Space)

- 측정을 어떻게 해야 할 것인가? 그 방법은 얼마나 안정적(stable)인가 ?

2.7 평가의 기준 { 최대, 최소, 평균}

4명의 이성친구 { A, B, C, D}에서 계속 사귈 사람을 선택하고자 한다.

남자	A	B	C	D
High **(기분이 아주 좋을 때)**	98	65	87	64
Low **(기분 아주 나쁠 때)**	12	62	55	28
Average **(평균적)**	73	63	80	49

2.8 프로그램: 측정 = 알고리즘 : 분석

- Time Complexity, Space Complexity
- Asymptotic Order, big O( ) notation

```
for(i=1 ; i <N ; i++)
    for(j=1; j<N; j++)
        if( i*i+(2*j+30)*3 % 10000 == 9999 ) exit ;
} // end of for
```

2.9 O( ) notation. 어떤 함수 $f(N)$의 closed form이 없을 경우, 차선책으로
그 order를 $O(g(N))$ 이라고 말한다. if and only if $\lim_{n\to\infty}\frac{f(n)}{g(n)}=c\geq 0$

예)

$$3n^3-4n^2+5=O(n^3),\quad 2n^2+5=O(n^3),\quad n^5-3n^2-1\neq O(n^4)$$

2.10 Time Complexity의 대표적인 두 가지 부류(Class)

- Polynomial Time Algorithm, n^3-2n^2+4
- Exponential Time Algorithm, 2^n+3n^2+1
- Exponential의 무서움, Logarithmic의 가벼움,

2.11 연봉으로 무려 < log 100억 원>을 받는 사나이의 최후는?

- 그가 열심히 노력해서 무려 200%의 인상을 받았다면
- 사람 A : 초봉 월 1,500만 원에서 매달 100만 원씩 더 받는 사람
- 사람 B : 매달 1.2배를 더 받는 사람의 차이 (시작은 50만 원)

2.12 자료구조, 알고리즘 전문가로서 할 수 있는 말

a. 이 프로그램은 D=15만 개의 데이터를 사용하면 대략 4시간 정도 걸릴 것이다.
30만 개라면 7시간 안쪽이 될 것이다.

b. 이 프로그램은 D= 10만 개의 데이터를 사용하면 일주일 이상 걸린 것이다.

2.13 코드의 실제 실행 시간 조사하기

- 단 windows에서 CPU 시간 측정은 믿을 것이 못 됨.
같은 코드를 돌려도 때와 장소(?)에 따라서 들쑥날쑥함.
왜 그런지 설명해 봅시다.
windows OS의 내재적 한계

```
#include <iostream>
#include <stdio.h>
#include <ctime>          // Time Check를 하기위해 필요한 헤더
#include <cmath>           Math Function들을 사용하기 위한 헤더
using namespace std;

int main() {
      clock_t     start;
      clock_t     end;
      time_tcurrentTime;
      structtm t;
      char* timeString;
      char  buffer[256];
      int         intX, counter;
      doublemydouble ;

      time(&currentTime);      // int 덧셈시간 측정.
       cout << "INT ADD Test\t";
      start = clock();
      for(int i=1; i < 10000; i++) {
            for(int j = 1; j < 10000; j++)
                  intX = i+j;
      }     // end of for i
      end = clock();            // 1. 실행시간 체크

      cout << ((double)(end - start))/(long)CLOCKS_PER_SEC \
       << " sec" << endl;
} // end of main( )
```

Python의 경우

```
# time.clock( )은 더이상 지원되지 않습니다. 2021년 3월 봄날에


import time

N = 1000000

start = time.process_time()
for i in range( N ) :  # 시간을 측정하고자 하는 code
    a = 109813.0*1023801.2*i

delta = time.process_time() - start     #
print("Ellasped =", delta, " milli." )

for i in range( N ) :
    a = 109813.0*1023801.2*i
    b = a*a*a

delta = time.process_time() - start
print("Ellasped =", delta, " milli." )
```

```
# Python List와 array의 속도 비교평가

import array as arr
import time

Ln = [10000, 20000, 40000, 80000, 160000, 320000]

print("\n Inserting back( ) in array) ")
for N in Ln :
    tbegin= time.process_time()
    da = arr.array('d', [])
    for w in range( N ) :
        da.append(w)
    tend = time.process_time()
    print( f'N={ N:7} elapsed= {tend-tbegin:10.7f} sec.')


print("\n Inserting front( ) in array) ")
for N in Ln :
    tbegin= time.process_time()
    da = arr.array('d', [])
    for w in range( N ) :
        da.insert(0,w)
    tend = time.process_time()
    print( f'N={ N:7} elapsed= {tend-tbegin:10.7f} sec.')


print("\n Inserting append( ) in list[]) ")
for N in Ln :
    tbegin= time.process_time()
    da = []
    for w in range( N ) :
        da.append(w)
    tend = time.process_time()
    print( f'N={ N:7} elapsed= {tend-tbegin:10.7f} sec.')


print("\n Inserting front( ) in list[]) ")
for N in Ln :
    tbegin= time.process_time()
    da = []
    for w in range( N ) :
        da.insert(0,w)
    tend = time.process_time()
    print( f'N={ N:7} elapsed= {tend-tbegin:10.7f}
```