Lecture 1 : 알고리즘(Algorithm)과 계산과학

21세기는 계산과학(computational science)의 전성시대라고 할 수 있다. 현재 수많은 계산과학이 있다. 계산물리학, 계산화학, 계산 생물학은 물론, 최근에는 계산 법학, 계산 윤리학까지 나타나고 있다. 컴퓨터 과학(computer science)은 그 대표적인 학문의 하나이다. 계산과학의 원류는 순수 수학의 구성주의(constructivism)와 닿아있다. 구성주의란 어떤 결과를 우리가 구체적으로 찾을 수 없는 명제에 대해서는 가치를 부여하지 않는 것이다. 즉 전통적으로 우리가 믿고있는 배타성의 원리, 즉 어떤 명제는 "참이든지 혹은 거짓이다" 이런 주장은 구성주의적 관점에서 본다면 의미가 없다고 할 수 있다. 예를 들어 원주율(π)에는 "9가 연속되어 10번 나타나는 구간이 존재한다"라면 명제는 true나 false가 될 수 없는 제 3의 명제가 될 수 있다는 것이 이들의 주장이다. 알고리즘도 이와 같이 우리가 어떤 답을 구할 수 있는 "인간적인 과정"[1]이 존재하지 않음이 증명된다면 이러한 문제는 논외가 되는 것이다. 우리는 당면의 목적인 좋은 알고리즘 개발에 앞서 이 역사적 맥락을 이해하고 있어야 한다.

1.1 과학을 공부하는 목적은 무엇인지, 특히 Mathematical Science의 궁극적인 목적은 무엇인지 자신의 생각을 자유롭게 말해 봅시다. (인문학, 공학, 과학의 차이점)

- 인문학과 자연과학의 차이
- 공학과 과학의 차이, 일반과학과 계산과학의 차이

1.2 수학과 컴퓨터 과학의 차이는 어디에 있을까 ?

- "존재하는 것"과 "구할 수" 있는 것의 차이점
- 세상에서 가장 무책임한 속담

$\forall$ Z $\rightarrow$ There is Z' for each Z, where mate(Z, Z')=yes

1.3 알고리즘(ALGORITHM)의 어원[2]

알고리즘(algorithm)은 특정한 계산을 위한 규칙이라는 의미로 쓰인다. 알고리즘의 어원은 9세기 페르시아엔 아부자파 모하메드 이븐 무사 알 콰리지미(Abu Ja'far Mohammed ibn Musa al Khowarizimi) 이라는 학자의 이름에서 기원한다. 콰리지미는 당대 최고의 수학, 천문학, 지리학전문가였다. '알 콰리지미 (al Khowarizimi)' 는 '콰라즘 마을 출신' 이라는 뜻이 있다. 알 콰라지미는 인도에 기원이 있는 산수(arithmetic)와 대수(algebra)의 기초를 고안하였다고 한다. 우리가 알고 있는 산술(arithmetics)에 관학 책을 집필하여 아랍인과 유럽인에게 소개하였다. 당시 유럽에서는 알 콰라지미가 전파한 대수를 '아라비아 수' 또는 '콰라즘에서 온 사람이 가르쳐 준 기술이라는 뜻이라는 '알 콰라즘'이라고 불리게 되고 '알고리즘(algorithm)'으로 정착되었다.

1) 컴퓨터 과학에서는 현존하는 컴퓨터보다 1000배나 더 빠른 컴퓨터를 가정해도 100년안에 풀 수 없는 계산문제는 "해결불가능(intractable)"한 문제로 분류된다.

2) https://en.wikipedia.org/wiki/Algorithm

1.4 계산적(compuational) 알고리즘이란 무엇인가 ?

- 3가지 특성 = { 유한성(finite), 결정성(deterministic), 효율성(efficiency)}

Q) "**꺼진 불도 다시 보자**"라는 표어는 왜 위험한 <표어>인지 알고리즘 적 관점으로(algorithmic) 설명하시오.

Q) 3x+1 Problem: 모든 양수 x에 대하여 종료되는가?[3)]

예를 들어보자. x= 11라고 한다면 그 궤적(orbit)은 다음과 같다.

34 17 52 26 13 40 20 10 5 16 8 4 2 1

```
n= 10
while(n >1 ):
    if n%2 == 0 : n = n // 2
    else :          n = 3*n + 1
    print(n)
```

모든 양의 정수에 대하여 위 과정이 반드시 종료됨. 즉 위 코드가 알고리즘이 됨을 아무도 증명하지 못하고 있다.

1.5 알고리즘을 잘 이해하면, 또는 명확하게 이해하지 못하면 어떤 일이 벌어지나 ?

- 이 프로그램을 한번 돌려봐야 그 수행시간을 알 것 같습니다.
- 데이터가 만 개일 때 1시간 걸렸으니 2만개이면 2시간 걸릴 것이다.
- 2만개로 늘어나면 글쎄,,, 수행해봐야 알 것 같은데
- 오 예.. 2만개로 돌렸더니 더 빨라졌네! 아주 좋은 시스템이야

1.6 Computer Algorithm과 "일반(general) 알고리즘"의 차이는 무엇인가 ?

1.7 이 세상의 문제를 해결하는 두 가지 기본입장

i) 알고리즘 접근법(Algorithmic approach) – 지혜와 압축
ii) 데이터 접근법(Data Approach) - 경험과 기억
iii) 뚜 방식을 적절히 조합하여 활용하는 방법

Q) 3456×5366를 빨리 계산하는 방법

- 곱셈 알고리즘
- Table pre-computation 방법 – time $\propto$ RANDOM access time

---

3) Collatz problem 혹은 hailstone problem. 이라고 불리는 매우 유명한 문제이다. 비슷한 문제로는 3x-1 문제도 있다. See this http://oeis.org/wiki/3x-1_problem

Big Data의 패러다임은 결국 Data Approach의 변형일 뿐이다.
지금의 상황은 이미 오래전에 나타난 상황이다. 지금 상황의 미래는 어디엔가 기록이 되어 있다.

- 한의학의 원리
- 도사(guru)의 예언 원리

1.8 의사(pseudo)-알고리즘으로서의 레시피 (예. 배추 겉절이) See

재료 인분 : 4
배추 (중간크기) 1개 , 물 2컵 , 물 1컵 , 고춧가루 3큰술 (또는 2큰술)
깨소금 1작은술 , 부추 (작은크기) 200g (한단) , 소금 3큰술
찹쌀가루 2큰술 , 젓갈 3큰술 , 다진마늘 2큰술 , 참기름 1작은술

요리법
준비시간: 30분 › 조리시간: 10분 › 추가시간 : 5시간 절임 › 요리시간:5시간40분

1. 배추를 세로로 4 쪽으로 머리 부분을 잘라낸 후 찬물에 씻어서 헹구어 줍니다.
2. 소금을 배춧잎 사이사이에 골고루 뿌린 후 물을 붓고 4-5 시간 절입니다.
3. 소금에 절인 배추를 3-4번 찬물에 씻어서 소쿠리에 받쳐서30분 놓아두어 물기를 뺍니다.
4. 작은 냄비에 물과 찹쌀가루를 넣고 반투명해질 때까지 풀을 쑤어주세요. 큰 그릇에 담아서 식혀줍니다.
5. 식힌 찹쌀풀에 젓갈, 고춧가루, 다진 마늘, 깨소금, 참기름을 골고루 섞어주세요.
6. 그리고 소금에 절인 배추를 넣고 ,부추는 2cm 길이로 잘라서 같이 섞어줍니다.
7. 접시에 놓을 때는 겉절이를 차곡차곡 예쁘게 쌓아주세요.

돼지 껍질 감자선 요리법

준비시간: 15분 › 조리시간: 8분 › 요리시간:23분

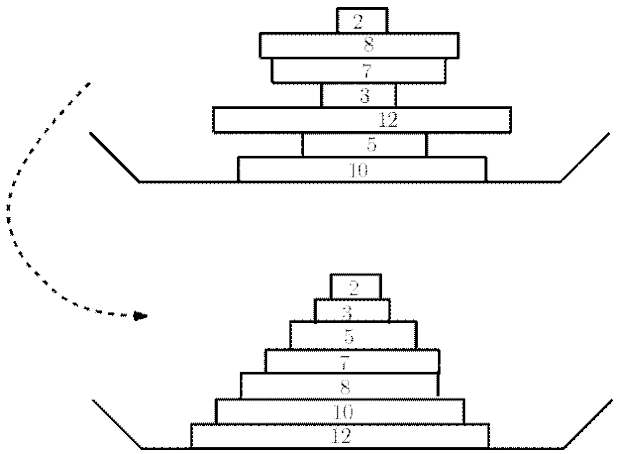
1. 으깬 감자 (매쉬드 포테이토)에 버터와 계란을 넣고 잘 섞어주세요.
2. 돼지껍데기는 작게 썰어서 바삭하게 구워 밀가루와 섞어준 뒤,
   다진 양파를 넣고 (1)의 감자와 섞어주세요. 소금과 후추로 간을 하세요.
3. 반죽을 작은 경단 모양으로 빚고 납작하게 눌러준 뒤,
   큰 프라이 팬에 카놀라유를 두르고 노릇하게 앞뒤로 부쳐주세요.

1.9 알고리즘 문제 - 팬케이크 정렬 (Pancake sorting)

프라이팬에 아래 그림과 같이 지름이 각각 다른 펜케이크가 다음과 같이 쌓여있다.

- 허용되는 동작 : "뒤집게" 도구를 중간에 찔러 넣어서 그 위에 있는 모든 팬케이크를 뒤집어 배치함.

Q) 어떤 경우라도 여러분은 쌓여있는 pancake를 크기대로 정렬할 수 있는가 ?



질문) 어떤 경우라도 pancake를 정렬할 수 있는가 ?
있다면 그것을 증명하시오. 12에서 뒤집고 그런식으로…
가장 큰수를 제일 위에 올리고… 이런식으로 2(N-1)정도 됨

질문) 쌓인 펜케이크를 정렬할 수 있는 방법 중에서 가장 좋은 방법은 ?
즉 가장 적은 횟수의 뒤집게를 사용하여 정렬하는 방법을 제시하시오.

Lecture 2 : 알고리즘의 효율, 분석, 차수(order)

이 장에서는 알고리즘 분석의 가장 기초적인 도구인 차수 분석에 대하여 살펴본다. 알고리즘을 연구하는 목적은 이론을 통하여 성능의 미래를 예측하는 것이다. 즉 실제 알고리즘을 코드로 작성하여 구체적인 컴퓨터에서 수행해보고 그 성능을 살펴보는 것이 아니라 알고리즘의 동작 원리가 준비되고 동작을 위한 자료구조가 마련된 후, 크기가 확정된 데이터를 입력으로 사용하면 개략적으로 얼마나 많은 시간이 소요되는지를 알고리즘 분석 기법으로 추측할 수 있어야 한다. 이것이 알고리즘을 공부하는 가장 주된 목표이다. 예를 들어 우리가 화성 탐사를 위하여 실험으로 우주선을 발사하는 것이 아니라 우리가 발사한 우주선이 화성의 환경에 어떻게 적응하고 어떤 동선을 따라서 움직여야 하는가를 가상의 공간에서 이론으로 예측을 하는 것과 같다.

그런데 우리가 사용하는 알고리즘의 차수(order)를 이용한 성능 예측은 구체적으로 수행 시간이 몇 분이다, 몇 시간인지를 결정해주는 작업은 아니다. 그 시간은 사용되는 컴퓨터의 속도나 환경(메모리의 크기)에 따라서 달라질 수 있다. 일반적인 PC에서 1시간 걸리는 작업을 슈퍼컴퓨터를 이용하면 몇 초 안에 끝낼 수도 있기 때문이다. 우리가 살펴볼 차수(order)는 이렇게 활용된다. 만일 어떤 알고리즘의 수행시간 Oder가 O(N)[4]라는 의미는 그 갯수가 늘어나는 것과 비례하여 수행시간도 늘어난다는 것을 의미한다.

즉 데이터 갯수가 만 개일때 수행시간은 10분이라면 그 갯수가 3배로 늘어갈 경우 수행시간은 10분의 3배인 30분이 될 것이라는 것을 암시하는 것이다. 만일 그 oder가 $O(N^2)$이라고 한다면 데이터의 크기가 3배 늘어날 때 시간은 그 제곱인 9배가 늘어남을 의미하는 것이다. 여러분은 앞으로 배울 알고리즘의 차수 표현에서 바로 위와 같은 입장을 이해하고 유지해야 한다.

2.1 점근적 차수(Asymptotic order) 표현법

- 존재하는 모든 것이 항상 명시적(explicitly) 표현되지는 않는다.

  $x=3$, x는 $f(x)=x^3-4x^2+7x-1$의 해 중 하나이다.

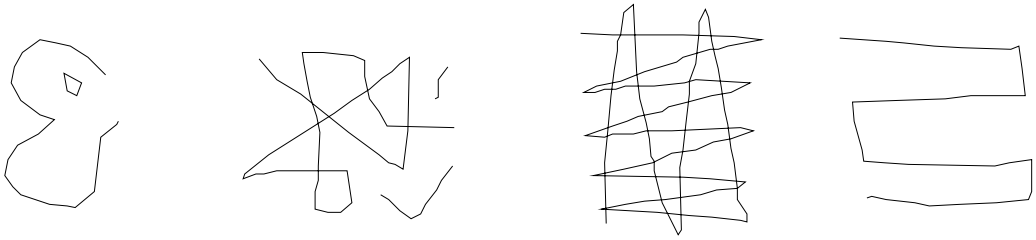
- 왜 유독 알고리즘에서만 차수 표현법을 사용할까 ?

  배열 A[n]에서 음수가 존재하는지를 검사하는 알고리즘을 생각해보자.

```
for w in A :
        if w < 0 : show "flag" ;
```

2.2 다음 4개의 그림을 자신의 기준으로 “복잡한” 순서대로 나열해보시오.

4) N은 입력 데이터의 크기(주로 갯수라고 생각하면 된다)



(a) (b) (c) (d)

기준 1: 교차된 점의 수 (number of intersections)

기준 2: 꺽어진 각도의 수

기준 3: 서로 분리된 선분(connected component)의 수

Note: 평가에는 반드시 구체적인 기준이 있어야 한다. e.g., 술의 평가기준
"세상은 기준에 대한 싸움이다."

2.3 함수의 증가 정도, 즉 증가 차수를 표현하는 3가지 방법

- $O(f(n))$, Otherwise, a better one is - $o(f(n))$: 확실히 작음.
- $\Theta(f(n))$
- $\Omega(f(n))$

어떤 알고리즘 복잡도 함수 $f(n)$의 점근적 order는 $O(n^2)$: 의미

A) 다음 문장이 말하는 의미를 일상 언어적, 경험적 관점으로 설명해보시오.

- 그 사람의 주량은 O(소주 3병) 이다.
  // 최대한 마셔봐야 3병, 3병이상 마시는 것을 본 적이 거의 없다.
- 그 사람의 주량은 Ω(3병) 이다.
  // 시작하면 최소 3병을 깔고 시작한다.
- 그 사람의 주량은 Θ(3병) 이다.
  / 시작하면 3병, 그러나 3병 이상 마시지는 않는다.

Q) 어떤 사람이 가장 안정적인 술친구 일까요 ?

- 벤처기업 A사의 연봉은 O(5000만원) 이다.
- 공기업 B사의 연봉은 Ω(2300만원) 이다.

B) 전형적인 Order 계산에 관한 문제

Q) $n^2+10n \in O(n^2)$ 임을 증명하라.

Q) $n^2+10n+10000 \in O(n^3)$ 임을 증명하라.

우리는 $g(n) \le c \cdot f(n)$ 을 항상 만족하는 c와 "$n > N_c$" 의 N_c을 찾으면 된다.

즉 여러분은 두 가지 값을 제시해야 한다. c와 N_c

예) c = 100, $N_c = 50$ 이라고 하면

$n^2 + 10n \leq 10n^2,\ n \geq N_c = 50$

C) 전형적인 function Order 문제

Q) $n^2 + 10n \in \Omega(n^2)$ 임을 증명하라.

Q) $n^2 + 10n + 10000 \in \Omega(n)$ 임을 증명하라.

우리는 $g(n) \geq c \cdot f(n)$ 을 항상 만족하는 c와 $n > N_0$ 을 만족하는 N_0을 찾으면 된다. 즉 일정 범위 이상에서 동호가 성립하게 하는 c와 N_0

D) 전형적인 Order에 관한 문제

Q) $30n^2 - 10n \in \Theta(n^2)$ 임을 증명하라.

Q) $5n + 10000 \in \Omega(n)$ 임을 증명하라.

우리는 $c \cdot f(n) \leq g(n) \leq d \cdot f(n)$ 을 만족하는 상수 c, d와 N을 찾으면 된다.

Hint : 가장 간편한 방법은 로피탈의 정리를 이용하는 것이다.

2.4 여러분에게 같은 작업을 하는 서로 다른 종류의 알고리즘 3개 A, B, C가 주어져 있다면 과연 여러분은 어떤 평가기준으로 이중 하나를 결정해야 할 것인가

- 시간복잡도 (Time Complexity)
- 공간복잡도 (Space Complexity),
- 이해복잡도? (Human Complexity)

예를 들어 어떤 집합 S에서 질의 원소 q가 있는지의 여부를 질문.

자료구조: unordered linear list를 사용한 경우

ordered array를 사용한 경우

hash table을 사용한 경우

2.5 모든 평가 기준에는 3개의 관점, 즉 최악(Worst), 최선(Best), 평균(Average)의 경우가 있다. 부산에서 기차를 타고 서울로 올라가 삼송기업에 최종 면접을 두고 있다고 가정하여, 그 면접장에 도착하기 위하는 가는 방법으로 이 문제를 설명해 보자.

2.6 다음은 어떤 스마트폰에 대한 여러 사람들의 평가다. 여러분은 과연 어떤 폰을 구입할 것인가 ? 질문: 현재 고려하고 있는 휴대폰 다른 3종 중에 하나 추천바랍니다. 이것을 위해서 인터넷에 있는 여러 사용기를 참고하여 표를 만들었다.

심송 갤녹스 S	LQ 안드로메다-13	애플, Yphone-3
그럭저럭 쓸 만 해요. 참을 만 합니다.	이것도 폰이냐.. 헐.... 절대 비추. 고장나면 완전 짱남. AS 최악. 반품 요구하니 미친 넘이라 욕함. 확- 불질러 버리고 싶음	가격대비 좋음. 잠시 쓰기엔 짱 그런데 발열이 장난이 아님, 손난로 대용
좋습니다. 빠른 배송은 아님	짱 좋아요.. 최고.. 최고	별 불편한지 모르겠음. 좋음 아저씨들에게 좋음. 큰 자판
여친이 좋아하네요.	써본 것 중 최고. AS 센터 아가씨 정말 친절. 고장 날 염려가 없음. 3 만원 쿠폰 제공^^	값이 조금 비싼 것 외에는, 만족. 3년 무료 As맘에 듬
다 좋은데 다자인 구려요. 스크레치가 좀 잘 남. 그 외 좋음. 끼워주는 것이 없음.	강추. 자난 달에만 8개를 또 새로 샀음. 여러분 꼭 사세요.	밧데리 오래감. 그러나 80년대 평범한 디자인. 그러나 오래 보면 정이 감. ㅋㅋㅋㅋ

[정의] **알고리즘의 복잡도**는 우리가 정한 기본동작의 수행횟수를 데이터의 크기 **N의 함수로 표현한 수식**이다.

변형) 데이터의 크기가 N, M으로 나타날 수도 있다. (행렬)
변형) 가장 좋은 경우, 나쁜 경우, 평균의 경우로 나눌 수 있다.

2.8 몸 풀기(warming up) 문제

a. n개의 원소를 가진 리스트 L에서 가장 큰 수 찾기
b. L에서 두 번째로 작은 수 찾기
c. 가장 큰 수와 작은 수를 동시에 찾아내기

2.9 아래 코드를 보고 기본동작의 횟수를 점근적(asymptotic order) 차수로 표현

```
for(i=1; i< N ; i++) {
  for(j=1; j< N ; j++) {
     solve_basic_one( ) ;
     if ( X ) Y ;
   }
}
```

X : ((i+j)^675 +90*j)^4 % (i*j+100) == 0 // 아주 복잡한 조건

Y : break 정답) $O(N^2)$, 최대 N^2번

Y : `solve_basic_one( )` ; $\Theta(N^2)$ 적어도 N^2번, 많아도 N^2번

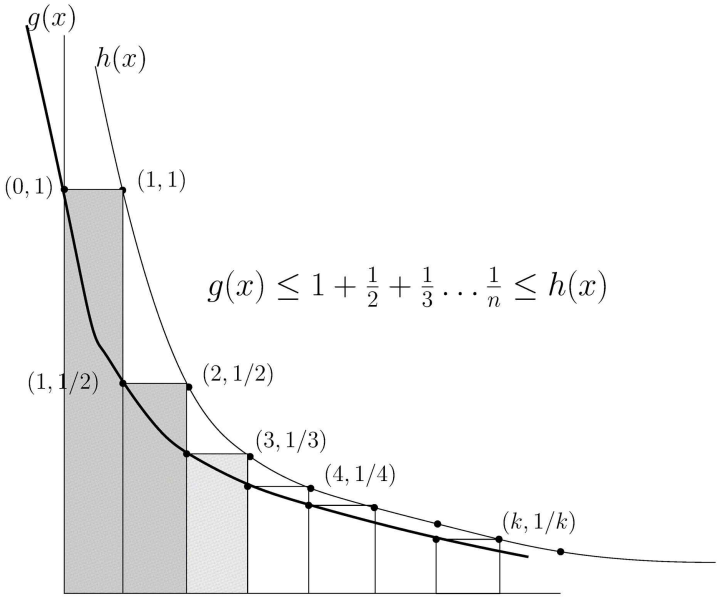
2.10 다음 함수의 점근적(asymptotic) Order를 구해보시오.

a. $S(n) = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} .... \frac{1}{n-1} ... + \frac{1}{n}$

b. $P(n) = 1 + \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} .... + \frac{1}{2^n}$

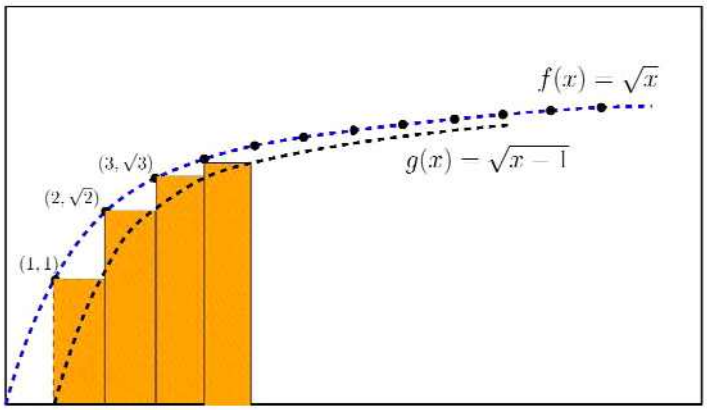
c. $Q(n) = \sin(1) + \sin(2) ... + \sin(n)$

d. $W(n) = 1 + \sqrt{2} + \sqrt{3} + \cdots \quad \sqrt{n}$



$$S(n) = 1 + \sqrt{2} + \sqrt{3} + \quad \cdots \quad + \sqrt{n-1} + \sqrt{n}$$

$$\int_1^n g(x)\,dx \;<\; S(n) \;<\; \int_0^{n+1} g(x)$$



$$\int_0^n \sqrt{x}\;\; dx = \frac{2x^{3/2}}{3}(x = 0, n) = O(n\sqrt{n})$$

2.12 다음 식에서 f(n) = A(g(n)) 일 때 A가 Ω, Θ, O 중에서 가장 정확한 것으로 하나 선택하시오. (각 order를 크기대로 나열해봅시다.)

	1	2	3	4	5	6
f(n)	$10n+\log n$	$3\log n$	$n^2/\log n$	$(\log n)^3$	$\sqrt{n}$	$n2^n$
g(n)	n	$\log n$	$n(\log n)^2$	$n/\log n$	$(\log n)^5$	3^n

풀이 : $\lim_n \frac{n2^n}{3^n} = n\left(\frac{2}{3}\right)^n = \frac{n}{\left(\frac{3}{2}\right)^n} \Rightarrow \frac{1}{c^n \log c} \Rightarrow 0,\ c=1.5,$

또는 ratio 방법을 활용할 수도 있다.

$R(n)=n(2/3)^n,\ R(n-1)=(n-1)(2/3)^{n-1}$

$R(n)/R(n-1) = n/(n-1)2/3 = 3n/(2n-2)$ > 1.0 따라서 무한발산한다.

2.13 점근적 표현이 필요한 또 다른 이유는 함수가 재귀적으로 정의되어 있을 때 이다. 어떤 함수는 다음과 같이 재귀적(recursive)으로만 정의되어 있다. 어떤 함수가 더 빨리 증가하는지 계산해 보시오. 만일 두 함수의 closed form을 구할 수 있다면 구해보시오. (아마 힘들걸...)

$f(n)=0.25\,f(n/3) + f(n/10)+\log n\ ,\ f(1)=1$

$g(n)=n+\ \log g(n-1)+1$

2.14 재귀적으로 표현된 식의 계산, closed form or asymptotic form

주의 사항: 일단 충분히 전개해서 그 안에 숨어있는 규칙을 찾아야 한다.

$T(1)=1,\quad T(n)=T(n/2)+n$

$S(1)=1,\quad S(n)=S(n-3)+(n-1)$

$T(1)=1,\quad T(n)=2\cdot T(n/2)+n-1$

2.15 접근적 복잡도 구하기를 위한 실습 문제(약간 어려움)

Q) $D(n)=D(n/2)+\log_2 n$ 일 때 $D(n)$의 접근적 차수(order)를 구하시오.

= O(log(n) + log(n/2) + log(n/4) + log(n/8) + ... + log(2))
= O(log(n * n/2 * n/4 * n/8 * ... * 2)) using log multiplication rule
= O(log(n * n * n * ... * n / 2 / 4 / 8 / ... / n))
= O(log($n^{\log n}$ / 2 / 4 / 8 / ... / n))
= O(log($n^{\log n}$ / (1 * 2 * 4 * 8 * ... * n/4 * n/2 * n))
= O(log($n^{\log n}$ / ((1 * n) * (2 * n/2) * (4 * n/4) * ...) group first and last terms

= O(log($n^{\log n}$ / (n * n * n * ...)) since we grouped terms,
= O(log($n^{\log n}$ / $n^{(\log n)/2}$) we halved the number of terms
= O(log($n^{\log n}$) - log($n^{(\log n)/2}$)) log division rule
= O(log n.log n) - ((log n)/2).log n) log power rule * 2
= O(log n.log n) - (log n.log n)/2)
= O(log n.log n)/2)
= O($(\log n)^2$/2)
= O($(\log n)^2$)

2.16 점근적 복잡도를 계산할 때 필요한 고급 기술

$A(n) = 9 \cdot A(n/3) + n - 2$

- 계산이 복잡할 때에는 그보다 더 간단한 bound를 잡아서 O( )로 표시한다.

$A(n) > R(n) = 9R(n) + n$

$B(n) = B(2n/3) + 1$

2.17 다음 수식의 Asymptotic Order를 구하시오. Base n=1에서의 값은 모두 1. 필요한 경우 n을 특정한 일반 값, 예를 들면 $n = 2^k$ 등으로 가정해야만 제대로 풀 수 있다. 그 값이 아니라면 $2^{k-1} \le N \le 2^k$ 를 이용한 "가위치기"[5] 방법으로 그 order를 점근적으로 구할 수 있다.

a) $T(n) = 2T(n/3) + 1$

b) $$\begin{aligned} R(n) &= R(n/2) + n^2 \\ &= R(n/4) + (n/2)^2 + n^2 \\ &= R(n/8) + (n/4)^2 + (n/2)^2 + n^2 \\ &= R(n/8) + n^2\left(\frac{1}{2^4} + \frac{1}{2^2} + \frac{1}{1}\right) \\ &= R(n/2^k) + n^2\left(\frac{1}{2^{2(k-1)}} + \frac{1}{2^{2(k-2)}} + \cdots \frac{1}{2^2} + \frac{1}{1}\right) \end{aligned}$$

1+(1/2)+(1/4) + (1/16) + (1/256) + ... < 2 이므로

$R(n) = R(1) + C \cdot n^2 < R(1) + 2 \cdot n^2 = \Theta(n^2)$

c) $W(n) = W(\sqrt{n}) + 1$, Assume $n_b = k^2 \le n \le n_a = (k+1)^2$

배운 내용 요약정리 6)

---

5) 다른 이름으로 "목조르기"라고도 할 수 있다. 미지의 함수 $f(n)$의 order를 잘 모를 경우 그 보다 큰 함수, 작은 함수 중에서 bound를 알 수 있는 것을 찾아서 양쪽에서 목을 조른다. 예를 들어 $a(n) \le f(n) \le b(n)$이고 $a(n) = b(n) = n^2$ 이라면 당연히 $f(n) = O(n^2)$이 될 수 밖에 없다.

1. 어떤 알고리즘 A가 주어져 있다. 이 알고리즘의 복잡도(Complexity)를 계산하시오.

Step 1) 입력 데이터의 크기를 변수로 결정한다. $|D| = n$

Step 2) 알고리즘의 수행 성능을 결정할 기본 동작 (Basic operation)을 선택한다.

기본동작은 하나 이상도 가능하다. **`if ( x[i] == x[j] )`**

2. - $O(f(n))$: Big-Oh of f(n), 즉 최대 속도가 $f(n)$이거나 그를 넘지 않는 함수

$n_0 \le n$이상에서 즉 $g(n) \le c \cdot f(n)$인 $f(n)$집합,

$O(n^3) = \{ 3n^2,\ n^3/10,\ n^2/2,\ \log n \},\quad \notin 10n^4$

- $o(f(n))$: small Oh of f(n), 확실히 작음. $\Theta(\ )$는 포함되지 않음

$o(n^3) = \{ 2n^2 \log n,\ n^{2.5},\ n \log n, ... \},\quad \notin 3n^3, 10n^4$

확실히 작음.

- $\Theta(f(n))$: $\Theta(n^2) = \{ 3n^2,\ n^2+10n-8,\ n^2/42, ... \},\quad \notin 10n, n^3+2n+1$
- $\Omega(f(n))$: $\Omega(n^2) = \{ 3n^2,\ n^3+3n-1,\ 2^n+n^3, ... \},\quad \notin 5n+2,\ \log_2 4n+3$

$\lim_{n\to\infty} \frac{g(n)}{f(n)}$ 의 결과 { c, ∞, 0 }에 따라서 판단

3. 시간 복잡도 (time complexity)에 왜 Asymptotic Order가 필요할까 ?

a) 최선: Closed form을 구할 수 있으면 Best... Order notation이 필요 없다.

- 종료 조건이 모호할 때: n^2 번까지는 opr( )이 수행하지 않는 것 같은데...

b) 다음 차선책으로 $\Theta(f(n))$을 찾아본다.

c) 다음 차선책으로 $o(f(n))$을 찾아본다.

d) 다음 차선책으로 $O(f(n))$을 찾아본다.

낮을수록 좋다. $O(n^3)$, $O(n^2 \log n)$, $O(n^2)$, $O(n^n)$?? 무의미

d) $\Omega(f(n))$ 분석에서는 무의미. 단 lower bound를 구할 때 유용

예) 모든 sorting algorithm의 복잡도는 $\Omega(n \log n)$.

이는 $\Theta(n)$ sorting algorithm을 개발하려는 노력을 하지 말라는 의미.

Why asymptotic order ? 복잡도는 대부분 recurrence relation으로 제시된다.

$T(n) = 1 + T((n-1)/2)$

$T(n) \le W(n)$, $W(n) = W(n/2) + 1$

6) 2020년 봄학기에 추가함.

2.18 점근적 복잡도 실전 문제에서의 활용

양팔 저울(balance scale)을 이용해서 n개의 동전 중에서 가짜(가벼운) 동전 찾기: 동전에 n개 있으며 이 중에서 약간 가벼운 동전 1개가 있다. 양팔 저울 몇 번 만에 이 가짜 동전을 찾아낼 수 있는지 최적 알고리즘을 제시하고 그 복잡도 $f(n)$을 분석하시오.

a) 기본 동작 : 양팔 저울 양쪽에 동전을 올려놓고 비교하는 작업
b) $n=3$, 9, 27
c) $n=28, 29, 30$
d) $n=3^k$, $n=3^{k+1}$
e) $n=3^k+1$, 3^k+2, 3^k+3^{k-1}

Q) 이 알고리즘의 $\Omega(f(n))$, $O(f(n))$를 계산하시오.

2.19 Decision Tree를 이용한 lower bound 계산

a) 100개의 동전에게 약간 가벼운 동전이 1개일 때

b) 100개의 동전에게 가짜 동전이 1개일 때, 무거운지 가벼운지를 모름

c) 100개의 동전에게 약간 가벼운 동전이 2개일 때

d) 100개의 동전에게 가짜 동전이 2개일 때, 무거운지 가벼운지를 모름

Lecture 3 : 시간 복잡도, 공간 복잡도 관련

이 장에서는 구체적인 문제를 사용하여 해당 문제를 해결하는 몇 가지 알고리즘의 시간복잡도, 공간복잡도를 직접 구해보는 연습을 해본다. 이런 류의 문제는 입사 시험 면접에서 흔히 나타난다. 그러한 면접 문제의 대표적인 유형은 어떤 배열(정렬이 된 예도 있고 아닌 경우도 있음)에서 특정 조건을 만족하는 원소 1개, 혹은 원소의 쌍을 구하는 형식이다. 이런 문제의 가장 직관적인 답의 형식을 2개 혹은 복수의 for loop을 이용해서 해결하는 것인데 이런 답은 거의 오답일 가능성이 크다. 만일 여러분이 면접에서 복수 개의 loop을 사용하는 것이 답으로 떠오른다면 다른 더 나은 답이 존재할 것이라고 예상해야 하고 그 naive 한 답을 말해서는 곤란하다.

예를 들어 가장 대표적인 문제는 주어진 unsorted array에서 최댓값과 최솟값을 찾는 문제이다. 이 문제에서 기본동작은 두 원소를 비교하는 동작이다. 즉 `if A[i]>A[j]` 이라고 보면 된다. 가장 단순한 답을 한 번의 loop에서 최댓값(max)을 구하고 또 다른 loop에서 최솟값(min)을 구하는 것이다. 즉 원소의 갯수가 N개 일 때[7] N-1번의 비교를 2번하는 것이다. 그런데 조금만 생각해보면 비교 횟수를 더 줄일 수 있다. finding max 문제를 토너먼트, 즉 2개씩 짝을 지어 더 큰 수가 다른 round로 올라가는 방식으로 푼다고 생각해보자.

편의상 원소의 갯수가 N=2k일 때 첫 round에서 k개의 원소가 승자가 된다.[8] 이렇게 round를 진행하면 N-1번에 최댓값을 찾을 수 있다. 문제는 min 값을 구하는 문제는 max를 구할 때의 과정에서 나온 정보를 이용할 수 있다는 것이다. 그것은 바로, 첫 round의 시합에서 한번이라도 이긴 원소는 결코 최솟값이 될 수 없다는 것이다. 따라서 최솟값은 첫 라운드에서 진 모든 원소 즉 N/2개만 모아서 tourment식으로 진행, 더 작은 값이 이기는 방식으로 진행하면 구할 수 있으므로 전체 비교횟수는 $N-1+(N/2-1)$이 된다.

3.1 알고리즘의 시간복잡도(time complexity)

- 주어진 알고리즘에서 우리가 정의한 기본동작(basic operation)이
- 데이터의 개수가 n개 일 때
- 최악의 경우 몇 번 수행된 후에 알고리즘이 종료되는가를
  즉 종료되기까지 수행해야하는 기본동작[9]의 횟수, 최악의 수행횟수를
- n의 함수로 표현한 것.

3.2 인터넷 인기 검색어 프로그램의 복잡도 (Space Complexity)

- 어떤 on-line site에서 질의어(number)가 real-time으로 들어온다.
  시간은 T_o 부터 시작한다. T_i 마다 하나씩의 원소가 입력된다.

---

7) 이 강의에서 앞으로 N이나 n의 입력의 크기를 의미한다.
8) 편의상 모든 원소는 서로 다른 양의 정수라고 생각해보자.
9) 기본동작은 하나 이상 정의될 수 있다.

문제 a) T_k 일 때까지 입력된 단어의 개수, $f(k)$

문제 b) T_k 일 때까지 입력된 단어의 평균

문제 c) T_k 일 때까지 입력된 단어 중에서 가장 큰 수, 작은 수

문제 d) T_k 일 때까지 입력된 단어 중에서 가장 많이 나타난 수

(Finding the most frequent item)

3.3 어떤 정렬된 배열에서 주어진 K에 대하여 $D[i]+D[j] \equiv K$ 를 만족하는 (i,j)가 존재하는지 판단하는 알고리즘을 제시하시오.

1	2	3	4	5	6	7	8	9	10	11	12	13	14
34	*45*	*61*	*65*	*80*	*90*	*102*	*114*	*137*	*180*	*211*	*234*	*267*	*370*

3.4 집합 S의 원소 중 3개 $x,y,z \in S$의 관계가 $x+y=z$인 3개의 수 Triple(x,y,z)를 찾아내기 : 어떤 수의 집합 S= { s_i } 가 있다. 예를 들어 S= {34, 34, 56, 123, 4, 234, 76, 456, 897, 100, 12, 4, 565, 234, 9, 23, 700} 이 있다고 하자. 이 중에서 어떤 두 개의 합이 다른 하나가 되는 Triple(s_i,s_j,s_k)이 존재하는지를 알아내는 알고리즘을 생각해보자.

a) 일단 $O(n^3)$ '짐승'같은 방법의 (brute-force) 알고리즘을 먼저 생각해보자.

b) "미치광이"기법의 randomized[10] algorithm도 한번 시도해보자.

c) 좀 더 빠르고 멋있는 알고리즘은 없을까 생각해보자.

3.5 재미있는 구글 (Google) 면접 문제

a) 어떤 단단하고 무거운 항아리가 있다. 이 항아리는 높이 L미터에서 떨어뜨리면 깨어지지 않고 L+1 미터에서 떨어뜨리면 깨어진다고 하자. 이 경우 이 항아리의 안전높이는 L미터라고 말한다. (원래 문제에서는 계란이 소재로 사용되었다.)

b) 이 항아리는 무척 무거워서 특수 화물용 엘리베이터를 이용해서 측정용 건물(높이가 100m)에 끌어 올려야 한다. 엘리베이터로 올릴 수 있는 기본 단위는 m이다. 만일 항아리가 한 개 뿐일 때 이 항아리의 안전높이를 구하기 위해서는 엘리베이터를 몇 번이나 이용해야 하는지 그 최악의 경우, 최선, 평균의 경우를 계산해보자.

c) 만일 동일한 항아리가 2개가 있다면 이 작업의 횟수를 좀 줄일 수 있다. 즉 먼저 한 항아리를 25m에서 떨어뜨리면 작업횟수가 줄어든다. (단 엘리베이

10) 이것은 제일 마지막 강의에서 설명할 것이다. 쉽게 설명하자면 이런 것이다. 만일 어떤 알고리즘은 우리가 질문한 답에 대하여 답을 한 것이 정답일 가능성이 $1/2+\epsilon$ 이라고 하자. 이 경우 이 알고리즘의 100번 수행했을 때 100번 모두 틀린 가능성은 $(1/2)^{100}$보다 작다. 따라서 우리는 충분히 답을 믿을 수 있다.

터에는 한 번에 하나의 항아리만 위로 올려보낼 수 있다.)

http://www.programmerinterview.com/index.php/puzzles/2-eggs-100-floors-puzzle/

http://classic-puzzles.blogspot.kr/2006/12/google-interview-puzzle-2-egg-problem.html

3.6 Pancake Sorting (+2015)

여러 장의 팬케익이 프라이팬에 쌓여있는 상황을 생각해보자. 여러분은 이 쌓여진 팬케익의 중간에 뒤집개를 넣어서 전체를 뒤집을 수 있다. 이 방식으로 팬케익 전체를 크기순으로 sorting한다고 생각해보자. 아래 배열에서 앞 쪽이 위, 뒤가 아래쪽이다. 즉 5 1 2 4 3 은 3번을 뒤집어서 sorting을 할 수 있다.

[1 2 3 4 5] → [4 3 2 1 5] → [3 4 2 1 5] → [5 1 2 4 3]

질문 1) 임의의 팬케익이 놓인 상황에서 이 방법으로 sorting할 수 있는가 ?

k를 그 개수라고 할 때, k=1 가능(trivial), k=2 가능 (2가지 경우만 고려하면 됨.) $k=n$ 일 때 가능하다고 가정하고, $k=n+1$일 때 가능함을 보이면 됨.

Hint) 가장 큰 원소를 바닥에 놓을 수 있으면 증명이 됨. 2번의 작업으로 가능함. 만일 그 원소가 바닥에 있으면 OK. 만일 아니라고 한다면 그 원소 위의 모든 원소를 뒤집어 제일 위에 있도록 함. 다시 전체 $n+1$개를 모두 뒤집으면 그것이 제일 아래로 감. 그 다음 그 위 n개의 다시 pancake sorting을 함.

질문 2) 만일 그렇다면 이 방법을 사용해서 가장 적은 횟수로 sorting하는 알고리즘을 제시하시오.

3.7 고장이 난 컴퓨터 찾기 문제[11]

화성에 건설된 우주기지는 인공지능 시스템이 관리하고 있다. 그런데 지구와 달리 화성에는 강력한 우주방사선이 존재하여 컴퓨터가 오작동을 일으킨다. 그런데 화성에는 아직 사람이 살지 못하는 상황이고 지구와 화성과의 통신 지연 시간이[12] 길어 지구의 관제센터에서 조정할 수 없고 그들 화성의 컴퓨터끼리 고장을 진단해야 한다. 일단 10대의 독립된 컴퓨터가 존재한다고 하자. 각 컴퓨터는 다른 컴퓨터에서 간단한 계산 문제를 보내어 이 문제에 답을 하게 하는 방식으로 진단을 한다. 그런데 정상적인 컴퓨터는 정상과 비정상을 구분할 수 있는데 진단을 하는 컴퓨터 자체가 고장일 경우에는 그 결과를 믿을 수 없다. 즉 고장 컴퓨터(faulty computer)는 정상을 비정상으로 진단할 수도 있고, 비정상으로 오히려 정상으로 진단하기도 한다.

---

11) 2022년에 추가됨. 이 문제를 원류는 fault-tolerant computing이다.

12) 지구에서 화성까지 가장 가까울 때는 5,460만 km 가장 멀 때는 4억 100만 km이다. 가장 가까울 때 전파 도달 시간은 약 3분이고 가장 멀 때는 약 22분이다.

예를 들어 2대의 컴퓨터 A, B가 있을 경우 A가 B를 정상, B가 A를 정상으로 판단했을 때 둘 모두 정상일 수도 있지만 둘 모두 고장일 수도 있다.[13] 이론에 따르면 과반 이하가 고장난 경우에는, 이것이 항상 보장이 된다면 정확하게 고장난 컴퓨터를 찾아낼 수 있다. 이 과정에서 컴퓨터가 다른 컴퓨터를 진단하는 작업의 횟수를 최소화해야 하는 것이다. 자 여기에서 문제 나갑니다.

Q1) 10대 $\{C_i\}$ $(i=1,10)$ 중에서 1대가 고장이 난 경우 그것을 최소의 진단(minimal diagnosis)으로 찾아내는 알고리즘을 제시하시오.

Q2) 만일 10개 중에서 최대 4대가 고장 날 수 있음을 알 때 어떤 방법으로 진단을 해야 할지 가장 좋을 것으로 생각되는 알고리즘을 제시하시오.

13) 이 문제는 어떤 집단에서 미치광이(미친갱이)를 찾는 문제와 동일하다. 두 사람 모두 미친 경우 누가 미쳤는지를 알아낼 수 없다. 3명 중 1명만 미친 경우에는 그 한 명을 찾아낼 수 있다.