기본 알고리즘

2022. Fall

국민대학교 소프트웨어학부

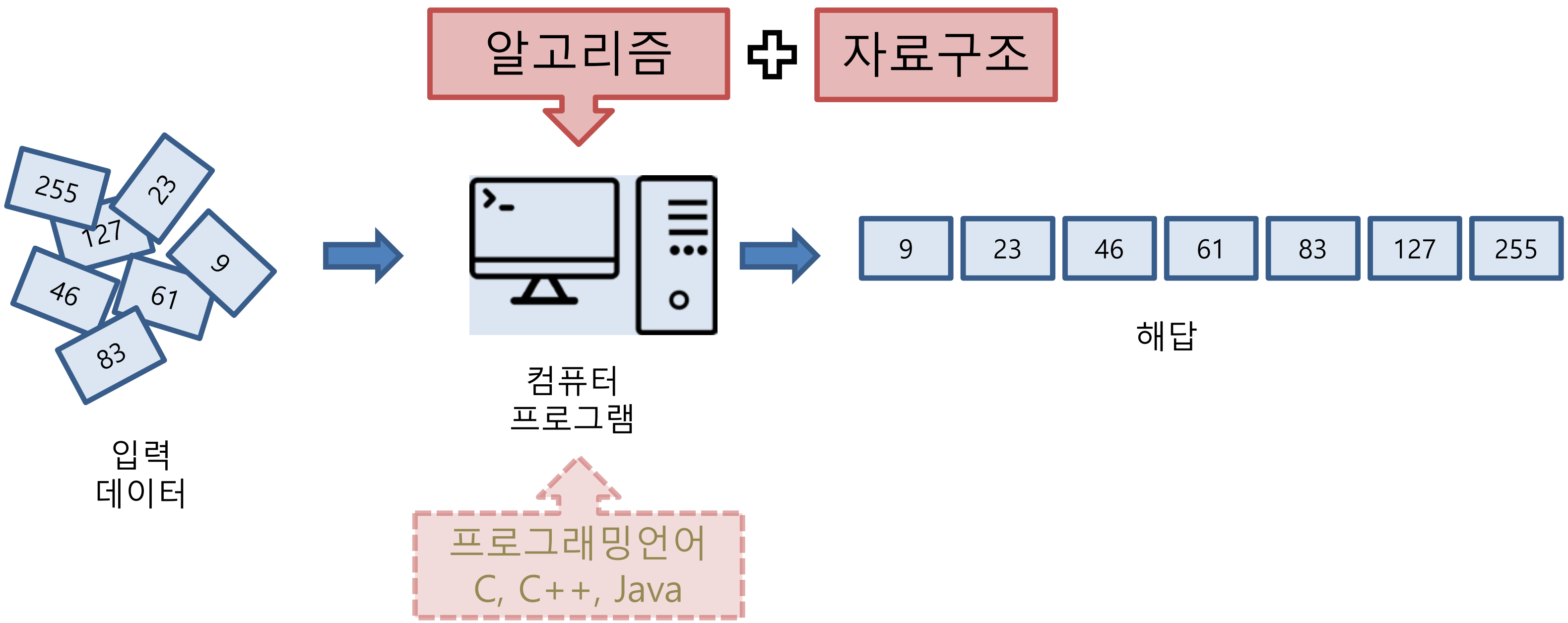
알고리즘?

컴퓨터로 푸는 퍼즐 (혹은 수수께끼)

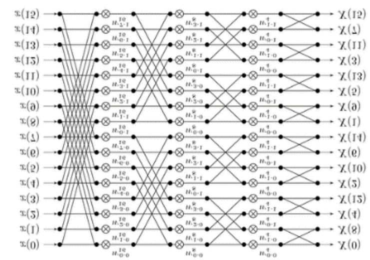
체계적인 문제해결방법이 필요함

꾀(trick)가 필요함

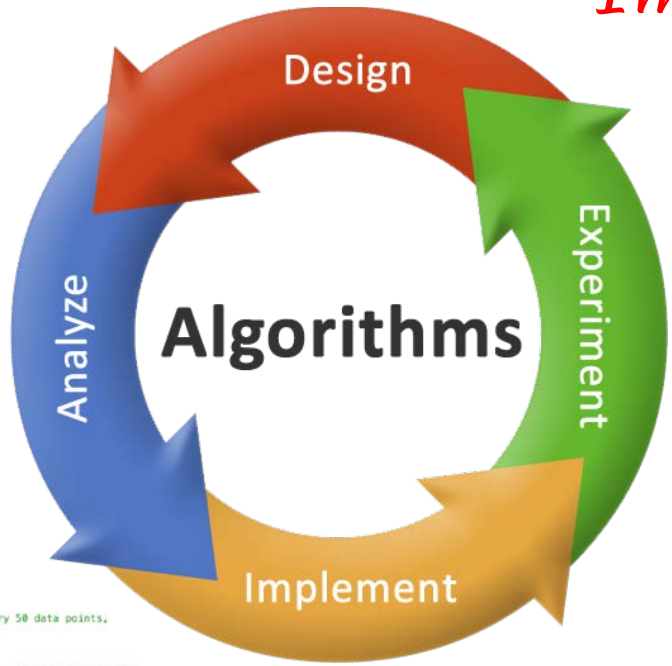
해답을 알기 전에는 매우 어렵지만,
알고 난 후에는 매우 쉬움.

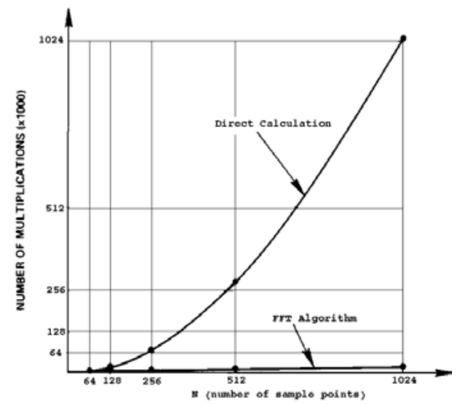


Design and Analysis of Algorithms



Improvement





$$
\begin{aligned}
T(n) &= 2T(n/2)+n = 2\{2T(n/4)+n/2\}+n \\
&= 4T(n/4)+2n = 4\{2T(n/8)+n/4\}+2n \\
&= 8T(n/8)+3n \\
&\cdots \\
&= nT(1)+(\log n)n \\
&= n\log n \\
&\in O(n\log n)
\end{aligned}
$$

```
% calculate FFT for length 256, shift the buffer every 50 data points,
% under sampling rate 250 Hz
function [FFTVals] = EEG_FFT(eeg, L, Interval)
    nchn = min(size(eeg));
    max_index = floor((length(eeg) - L) / Interval);    % index starts from
    FFTVals = cell(nchn, 1);

    for n = 1:nchn
        FFTVals{n} = zeros(max_index + 1,L/2 + 1);
    end

    % calculate the fft values
    for index = 0:max_index
        for n = 1:nchn
            temp_dat = eeg(n,index*Interval + (1:L));   % data in buffer
            % hamming
            temp_han = temp_dat'.*hamming(L);
            temp_fft = fft(temp_han);
            temp_mag = abs(temp_fft/L);
            temp_mag = temp_mag(1:L/2+1);
            temp_mag(2:end-1) = 2*temp_mag(2:end-1);
            FFTVals{n}(index+1,:) = temp_mag;
        end
    end
end
```

알고리즘 개발 (설계)

- 소프트웨어 개발
  - 소프트웨어 목적에 맞는 알고리즘을 개발 (설계)
  - 알고리즘을 프로그램으로 구현
- 알고리즘 개발 과정이 소프트웨어 개발 과정에서 가장 핵심이면서 가장 어려운 과정
- 알고리즘 개발 과정에서는 문제해결 능력을 요구함

알고리즘 개발 중요성의 예

Search Engine Algorithm

문제 :

주어진 "단어"를 포함하고 있는 (웹-)문서를 모두 검색한 후, 이 문서들을 어떤 순서로 나열할 것인가?

Before 1997

WebCrawler, Lycos, Excite, Infoseek, Ask Jeeves, Altavista, Yahoo(Inktomi), …

찾고자하는 단어가 문서에서 나타나는 빈도수나 그 문서와의 상관관계도에 따라서 페이지를 우선적으로 나열하는 알고리즘 채택

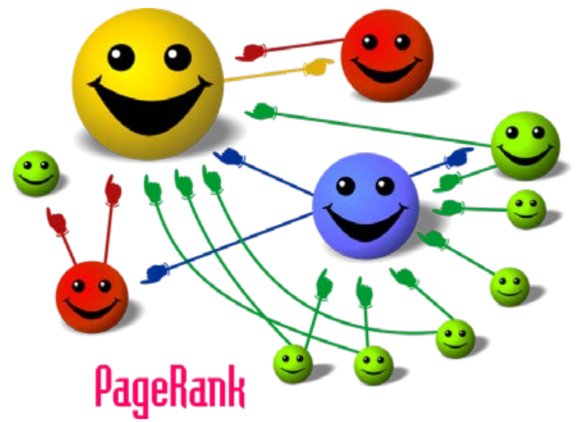
1997

Google(Larry Page, Sergey Brin),

Baidu(Li, China, 1996)

PageRank Algorithm

다른 웹-페이지로부터 웹-링크(참조)가 많은 웹-페이지가 높은 우선순위를 가지며, 이 우선순위가 높은 페이지를 우선적으로 나열한다.



알고리즘 개발 중요성의 예

News Feed Algorithm

문제 :

Facebook 등과 같은 SNS에서 어떤 사람에서 전달되는 수많은 News(지인들 소식 등과 같은 정보)를 어떤 순서대로 보여줄 것인가?

중요성:

- Facebook needs an algorithm, because otherwise you would miss content that is important to you.
- The algorithm tries to figure out which stories you are most likely to like, comment on, and share.

✓ Facebook 의 성공은 News Feed Algorithm 으로부터 시작됨..

✓ 매년 새로운 알고리즘으로 Update 하고 있으며,

✓ 최근 추세는 내가 읽어본 news의 특징을 Machine Learning Algorithm을 적용하여 추출한 후, 이를 기반으로 새로운 news중에서 유사한 특징을 가진 news를 보여주는 형태로 진화하고 있음. (Youtube 등 자동추천)

알고리즘 설계 및 분석

- 어떤 문제 *P* 가 주어졌을 때,
  - 컴퓨터로 문제 *P* 를 해결할 수 있는가?

Computability

- 해결할 수 있다면,
  문제 *P* 를 해결하는 알고리즘 *A* 에 대하여
  - 알고리즘 *A* 가 정확한가?
  - 알고리즘 *A* 는 얼마나 좋은 알고리즘인가?
  - 알고리즘 *A* 보다 더 좋은 알고리즘은?

Verification

Efficiency

- 알고리즘이 좋다는 것은 어떻게 평가하는가?
  - 이 알고리즘이 수행되는 시간은?
  - 이 알고리즘이 사용하는 메모리의 양은?

Time Complexity

Space Complexity

Complexity (복잡도)

- 문제 *P* 를 해결하는 알고리즘 *A* 가 주어졌을 때,

- 어떻게 더 좋은 알고리즘을 개발할 수 있을까?
  - 더 빠른 알고리즘은?
  - 메모리를 덜 사용하는 알고리즘은?

Complexity of Algorithm

- 더 좋은 알고리즘이 있을까?
  - 이 알고리즘이 가장 최선의 알고리즘인가?
    (더 빠른 알고리즘은 없는가?)

Complexity of Problem

알고리즘 설계

- 알고리즘 설계
  - 설계에 사용할 도구 : Data Structure
  - 설계 기법

도구 (Data Structures)	설계기법 (Techniques)
Arrays Stacks, Queues Linked lists Sets, Dictionaries Hash Tables Trees, Binary Search Trees Graphs	Brute Force Recursion (재귀, 되부름) Divide & Conquer (분할정복기법) Dynamic Programming (동적계획법) Greedy Approach (욕심장이기법) Backtracking (되추적기법) Branch and Bound (분기한정기법)

컴퓨터 프로그램 = 자료구조 + 알고리즘

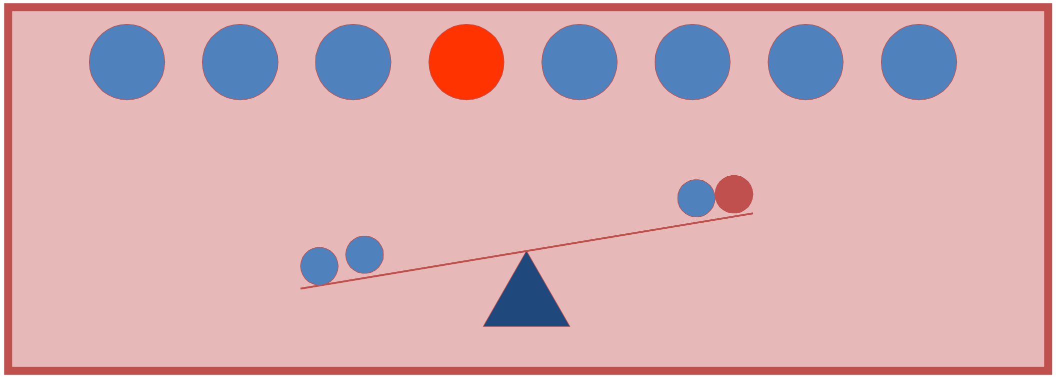
by Niklaus Wirth

Searching Problem

- Searching
  - 전화번호부, 사전 찾기; Dictionary Searching
    - 전화번호부에서 "이 순신" 이름을 찾고자 할 때, **어떤 방법**으로 찾는가?
    - 전화번호부에서 이름을 쉽게 찾을 수 있도록, **어떤 방법**로 이름을 나열하고 있는가?

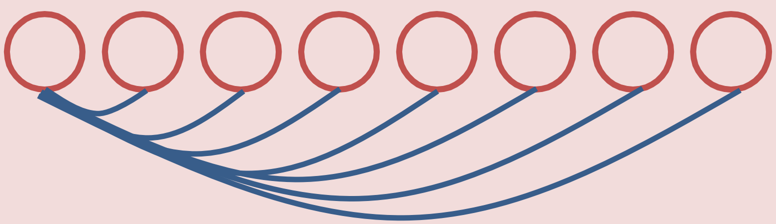
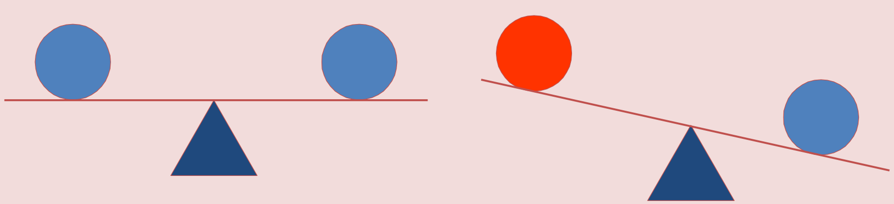
무게가 가벼운 구슬 찾기 (1)

- Searching
  - 무게가 가벼운 구슬 찾기; Searching a pebble
    - 같은 모양의 구슬이 8 개와 구슬의 무게를 잴 수 있는 천칭이 주어져 있다. 이 구슬 중에서 7개의 무게는 같으며, 한 개의 무게는 다른 구슬보다 *가볍다*. 천칭을 이용하여 이들 구슬 중에서 무게가 가벼운 구슬을 찾으려고 한다. 최소 횟수로 천칭을 이용하여 가벼운 구슬을 찾는 방법을 제시하시오.



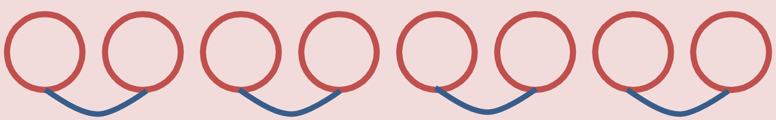
무게가 가벼운 구슬 찾기 (2)

- 문제: 천칭을 이용한 *무게가 가벼운* 구슬찾기

알고리즘 설계 Design of Algorithm	알고리즘 분석 Analysis of Algorihm
알고리즘 1: 한 개의 구슬을 고정하고, 이 구슬과 다른 모든 구슬의 무게를 각각 비교한다.  	천칭을 몇 번 사용하여 가벼운 구슬을 찾았나? 7 번

무게가 가벼운 구슬 찾기 (3)

- 문제: 천칭을 이용한 *무게가 가벼운* 구슬찾기

알고리즘 설계 Design of Algorithm	알고리즘 분석 Analysis of Algorihm
알고리즘 2: 두 개의 구슬마다 어느 구슬이 가벼운지 를 비교한다. 	천칭을 몇 번 사용하여 가벼운 구슬을 찾았나? 4 번

무게가 가벼운 구슬 찾기 (4)

- 문제: 천칭을 이용한 *무게가 가벼운* 구슬찾기

알고리즘 설계 Design of Algorithm	알고리즘 분석 Analysis of Algorihm
알고리즘 3:	천칭을 몇 번 사용하여 가벼운 구슬을 찾았나? 3 번

무게가 가벼운 구슬 찾기 (5)

- 문제: 천칭을 이용한 *무게가 가벼운* 구슬찾기

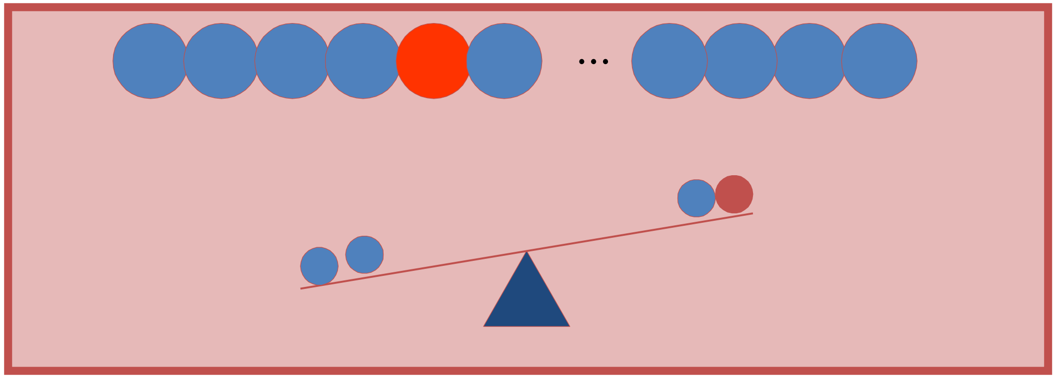
알고리즘 설계 Design of Algorithm	알고리즘 분석 Analysis of Algorihm
알고리즘 4:	천칭을 몇 번 사용하여 가벼운 구슬을 찾았나? 2 번

무게가 가벼운 구슬 찾기 (6)

- 문제: 천칭을 이용한 *무게가 가벼운* 구슬찾기
  - 앞에서 제시한 알고리즘 1, 2, 3, 4 중에서 어느 알고리즘이 *효율적인* 알고리즘인가?
  - 알고리즘 4에서 제시한 횟수보다 더 적은 횟수로 무게가 가벼운 구슬을 찾을 수 있을까?

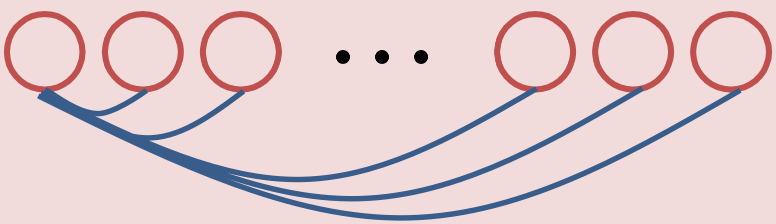
무게가 가벼운 구슬 찾기 (7)

- Searching
  - 무게가 가벼운 구슬 찾기; Searching a pebble
    - 같은 모양의 구슬이 n 개와 구슬의 무게를 잴 수 있는 천칭이 주어져 있다. 이 구슬 중에서 (n-1) 개의 무게는 같으며, 한 개의 무게는 다른 구슬보다 *가볍다*. 천칭을 이용하여 이들 구슬 중에서 무게가 가벼운 구슬을 찾으려고 한다. 최소 횟수로 천칭을 이용하여 가벼운 구슬을 찾는 방법을 제시하시오.



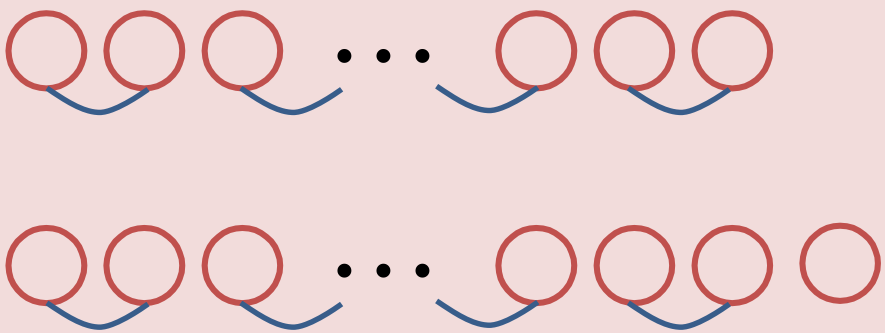
무게가 가벼운 구슬 찾기 (8)

- 문제: 천칭을 이용한 *무게가 가벼운* 구슬찾기

알고리즘 설계 Design of Algorithm	알고리즘 분석 Analysis of Algorihm
알고리즘 1: 	천칭을 몇 번 사용하여 가벼운 구슬을 찾았나? (n-1) 번

무게가 가벼운 구슬 찾기 (9)

- 문제: 천칭을 이용한 *무게가 가벼운* 구슬찾기

알고리즘 설계 Design of Algorithm	알고리즘 분석 Analysis of Algorihm
알고리즘 2: 	천칭을 몇 번 사용하여 가벼운 구슬을 찾았나? $\left\lfloor \frac{n}{2} \right\rfloor$ 번

무게가 가벼운 구슬 찾기 (10)

- 문제: 천칭을 이용한 *무게가 가벼운* 구슬찾기

알고리즘 설계 Design of Algorithm	알고리즘 분석 Analysis of Algorihm
알고리즘 3: Recursive Algorithm - Base case - Recursive step	천칭을 몇 번 사용하여 가벼운 구슬을 찾았나? ? 번

국민대학교 KOOKMIN UNIVERSITY

Algorithmic Engineering Laboratory

무게가 가벼운 구슬 찾기 (11)

- 문제: 천칭을 이용한 *무게가 가벼운* 구슬찾기

알고리즘 설계 Design of Algorithm	알고리즘 분석 Analysis of Algorihm
알고리즘 4: Recursive Algorithm - Base case - Recursive step	천칭을 몇 번 사용하여 가벼운 구슬을 찾았나? ? 번

무게가 가벼운 구슬 찾기 (12)

- 문제: 천칭을 이용한 *무게가 가벼운* 구슬찾기
  - 위 알고리즘들은 임의의 개수의 구슬이 주어지더라도 반드시 무게가 가벼운 구슬을 찾을 수 있는가? (verification)
  - 앞에서 제시한 알고리즘 1, 2, 3, 4 중에서 어느 알고리즘이 *효율적인* 알고리즘인가?
  - 알고리즘 4에서 제시한 횟수보다 더 적은 횟수로 무게가 가벼운 구슬을 찾을 수 있을까?
    - Recursive하게 계속 2개의 group 으로 묶어서 무게를 다는 것 보다 3개의 group 으로 묶어서 무게를 다는 것이 횟수를 줄일 수 있음. 그러면, 4개의 group, 5개의 group, ... 으로 묶어서 무게를 다는 것이 더 횟수를 줄일 수 있지 않을까?

무게가 가벼운 구슬 찾기 (13)

- 알고리즘 1,2,3,4 비교
  - 천칭을 사용하는 횟수

알고리즘	8개 구슬	n개 구슬
1	7	$n-1$
2	4	$\lfloor n/2 \rfloor$
3	3	$\lfloor \log_2 n \rfloor$
4	2	$\lceil \log_3 n \rceil$

- 알고리즘 2는 1보다 얼마나 적은 횟수를 사용하는가 ?
- 알고리즘 3은 2보다 얼마나 적은 횟수를 사용하는가 ?
- 알고리즘 4는 3보다 얼마나 적은 횟수를 사용하는가 ?

무게가 가벼운 구슬 찾기 (14)

- 알고리즘 1,2,3,4 비교
  - 알고리즘 2는 1보다 얼마나 적은 횟수를 사용하는가 ?

알고리즘	n개 구슬
1	$n-1$
2	$\lfloor n/2 \rfloor$

$$\frac{\left\lfloor \frac{n}{2} \right\rfloor}{n-1} \geq \frac{\frac{n-1}{2}}{n-1} \geq \frac{1}{2}$$

Note that $\frac{n-1}{2} \leq \left\lfloor \frac{n}{2} \right\rfloor \leq \frac{n+1}{2}$

  - 즉
    - 알고리즘 2는 1보다 반(1/2) 이상의 횟수를 사용한다.
      (1/2 배 보다 더 적은 횟수를 사용하지는 않는다)
      (최대 2배 빠르다)

무게가 가벼운 구슬 찾기 (15)

- 알고리즘 1,2,3,4 비교
  - 알고리즘 4는 3보다 얼마나 적은 횟수를 사용하는가 ?

알고리즘	n개 구슬
3	$\lfloor \log_2 n \rfloor$
4	$\lceil \log_3 n \rceil$

$$\frac{\lceil \log_3 n \rceil}{\lfloor \log_2 n \rfloor} \geq \frac{\log_3 n}{\log_2 n} \geq \log_3 2$$

Note that
$x - 1 < \lfloor x \rfloor \leq x$
$x \leq \lceil x \rceil < x + 1$

  - 즉
    - 알고리즘 4는 3보다 $\log_3 2 \sim 0.631$ 배 이상의 횟수를 사용한다.
      ($\log_3 2\ \sim$ 0.631배 보다 더 적은 횟수를 사용하지는 않는다)
      (최대 $\log_2 3 \sim$ 1.585 배 빠르다)

무게가 가벼운 구슬 찾기 (16)

- 알고리즘 1,2,3,4 비교
  - 알고리즘 3는 2보다 얼마나 적은 횟수를 사용하는가 ?

알고리즘	n개 구슬
2	$\lfloor n/2 \rfloor$
3	$\lfloor \log_2 n \rfloor$

$$\frac{\lfloor \log_2 n \rfloor}{\lfloor n/2 \rfloor} \geq C \quad \text{for all } n > k \text{ for some } k$$

임의의 상수 C

  - 즉, 아래와 같이 표현할 수 있는가? (임의의 상수 C에 대하여)
    - 알고리즘 3은 2보다 C배 이상의 횟수를 사용한다.
      (C 배 보다 더 적은 횟수를 사용하지는 않는다)
      (최대 1/C 배 빠르다)

  이 경우에는 불가능함 (나중에!)

무게가 가벼운 구슬 찾기 (17)

- 이론이 아닌 실제 실행시 알고리즘 1,2,3,4 비교
  - 이론적 천칭을 사용하는 횟수

알고리즘	n개 구슬
1	$n-1$
2	$\lfloor n/2 \rfloor$
3	$\lfloor \log_2 n \rfloor$
4	$\lceil \log_3 n \rceil$

최대 2 배 빠름

최대 $\log_2 3$ 배 빠름

- 알고리즘 1(3)를 실행(실제로 천칭으로 구슬의 무게를 다는 일) 하는 사람이 알고리즘 2(4)을 실행하는 사람보다 천칭을 한 번 다는 속도가 2($\log_2 3$)배 이상 빠르다면
  - 알고리즘 1(3)을 실행하는 속도가 빠르게 됨.
  - *즉, 실제 실행 속도 비교는 이론적인 속도 비교의 반대가 되게 됨.*

무게가 가벼운 구슬 찾기 (18)

- 이론이 아닌 실제 실행시 알고리즘 1,2,3,4 비교
  - 이론적 천칭을 사용하는 횟수

알고리즘	n개 구슬
1	$n-1$
2	$\lfloor n/2 \rfloor$
3	$\lceil \log_2 n \rceil$
4	$\lceil \log_3 n \rceil$

최대 상수배 빠른 것이 불가능함

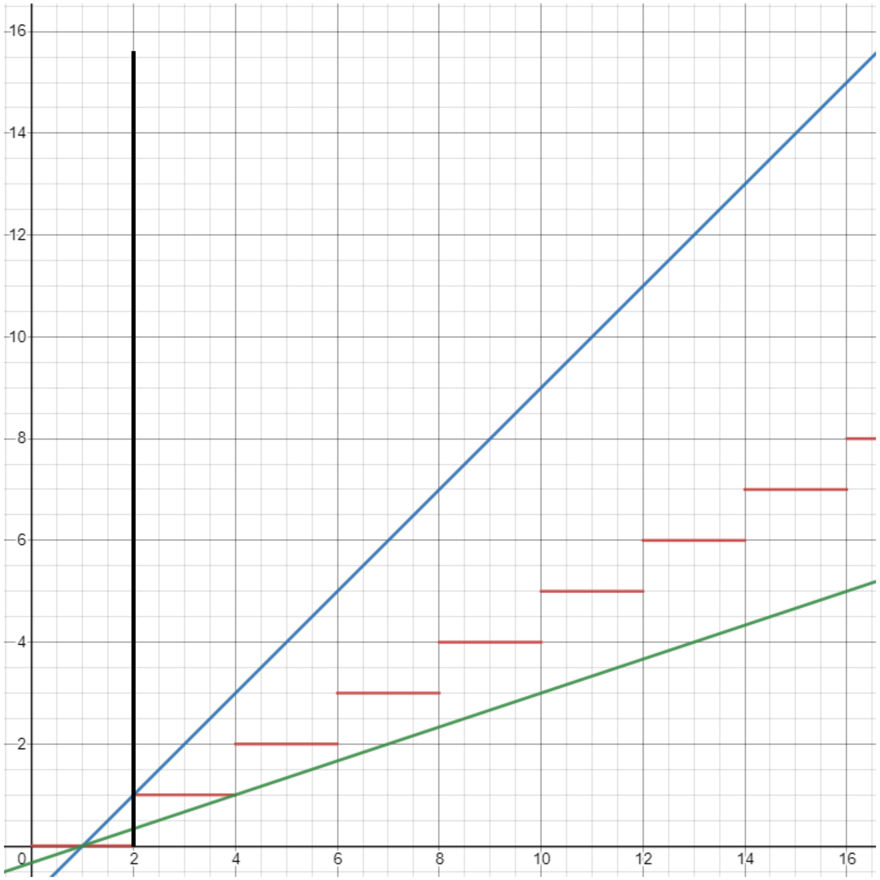
- 그러나 알고리즘 2와 3의 경우에는 알고리즘 2를 실행하는 사람의 속도가 아무리 상수배 만큼 빨라도 위와 같이 *빠르기가 역전될 수 없음*.
- 이러한 현상의 수학적으로 어떻게 표현할 것인가?
  - Big-O notation (Asymptotic Analysis 가 필요함)

KMU 국민대학교 KOOKMIN UNIVERSITY

Algorithmic Engineering Laboratory

함수 크기 비교

알고리즘	n개 구슬
1	$n-1$
2	$\lfloor n/2 \rfloor$



$f(n) = n - 1$

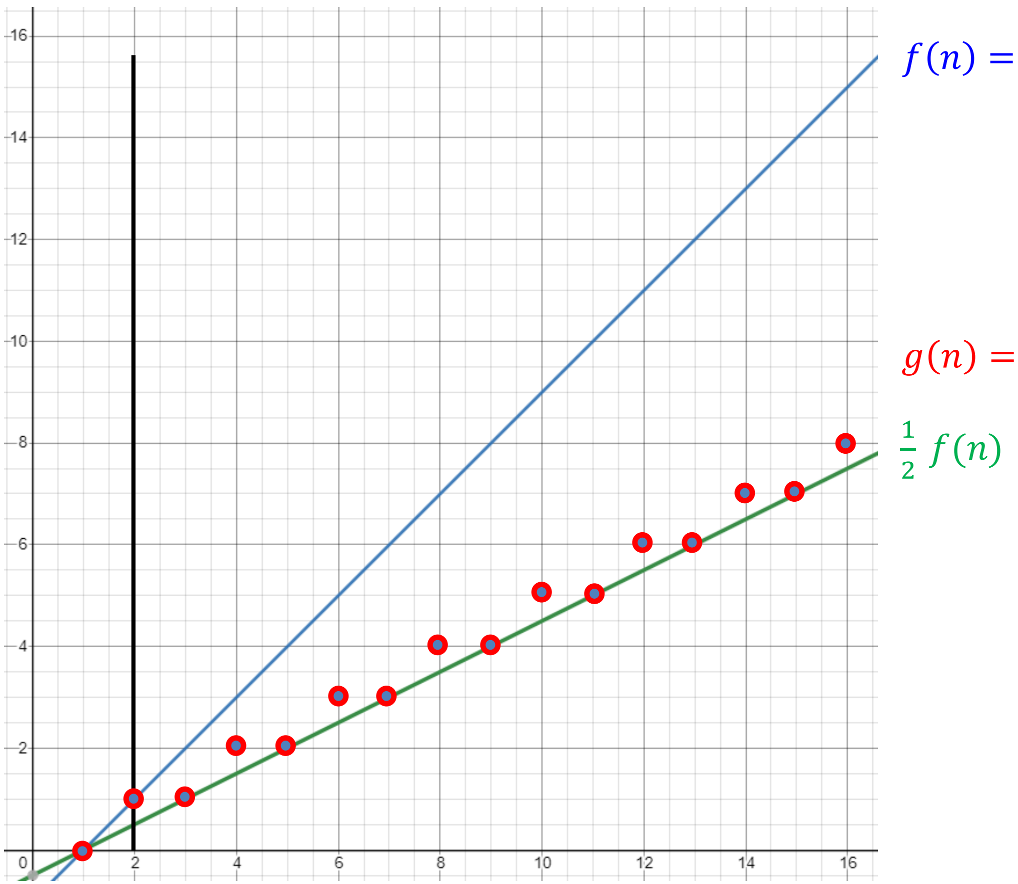
$g(n) = \lfloor n/2 \rfloor$

$\frac{1}{3} f(n)$

$$\frac{1}{3} f(n) \le g(n) \le f(n), \qquad n \ge 2$$

함수 크기 비교

알고리즘	n개 구슬
1	$n-1$
2	$\lfloor n/2 \rfloor$

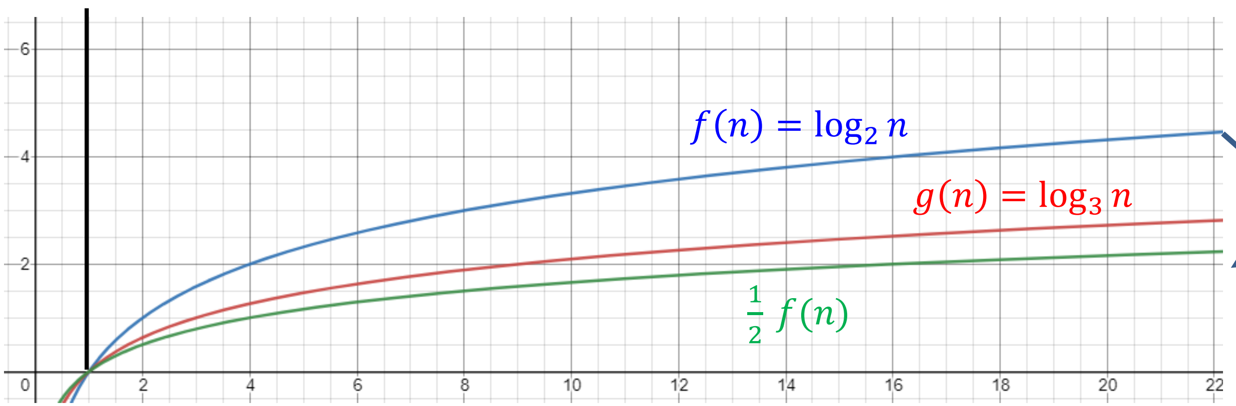


g(n) ≤ *f(n)* 이지만,
*f(n)*을 2배 빠르게 실행하게 되면
0.5f(n) ≤ *g(n)* 이 된다.

$$\frac{1}{2}f(n) \le g(n) \le f(n), \qquad n \ge 2, \text{integer}$$

함수 크기 비교

알고리즘	n개 구슬
3	$\lfloor \log_2 n \rfloor$
4	$\lceil \log_3 n \rceil$



g(n) ≤ *f(n)* 이지만,
*f(n)*을 2배 빠르게 실행하게 되면
0.5f(n) ≤ *g(n)* 이 된다.

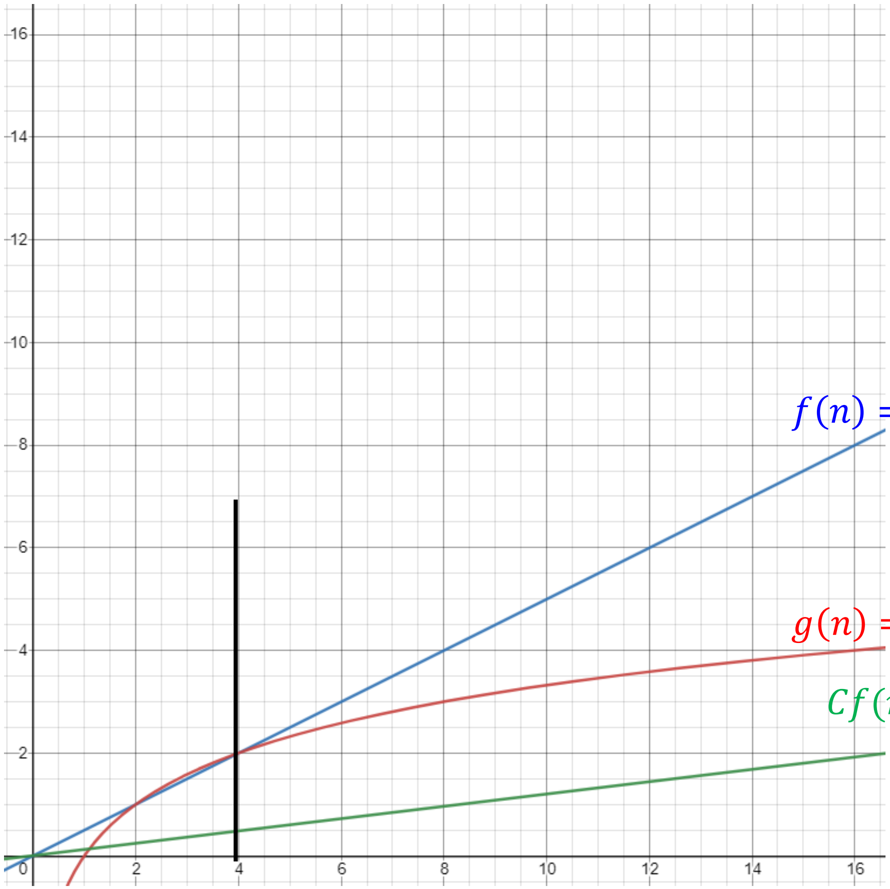
$$\frac{1}{2} f(n) \le g(n) \le f(n), \qquad n \ge 0$$

국민대학교 KOOKMIN UNIVERSITY

Algorithmic Engineering Laboratory

함수 크기 비교

알고리즘	n개 구슬
2	$\lfloor n/2 \rfloor$
3	$\lfloor \log_2 n \rfloor$



- ✓ *g(n)* ≤ *f(n)* 이지만, *C·f(n)* ≤ *g(n)* 이 되는 어떠한 상수 *C* 도 존재하지 않는다
- ✓ 즉, 아무리 상수 1/*C* 배 빨리 *f(n)* 을 실행하더라도 *g(n)*보다 빠를 수는 없다.
- ✓ Why? Note that

$$\lim_{n\to\infty} \frac{n/2}{\log_2 n} = \infty$$

$Cf(n) \le g(n) \le f(n), \quad n \ge 4$
인 상수 C는 존재하지 않음

알고리즘 분석

- Space Complexity (공간복잡도)
  - 알고리즘을 수행하기 위해 필요한 메모리의 양
- Time Complexity (시간복잡도)
  - 알고리즘이 수행되는데 걸리는 시간
- 알고리즘 분석에서는 time complexity 를 space complexity 보다 더 중요하게 고려함

알고리즘 분석

- Time complexity analysis
  - 프로그램이 수행되는 하드웨어적인 요소와 프로그램이 구현되는 소프트웨어적인 요소와 무관한 이론적인 분석이 필요함
  - Method 1:
    - 알고리즘을 구현하는 모든 primitive operation (assignment, array indexing, 덧셈, 곱셈, 함수호출 등) 의 수행 회수를 계산
  - Method 2:
    - 알고리즘의 수행시간이 어떤 연산의 수행 횟수에 비례하는 가장 핵심적인 연산을 찾아서, 그 연산이 수행되는 회수를 계산
    - 핵심연산 (Basic Operation)

알고리즘 분석

- Time complexity analysis
  - Primitive operation
  - Basic operation
- 예 : 삽입정렬

Primitive operation

Basic operation

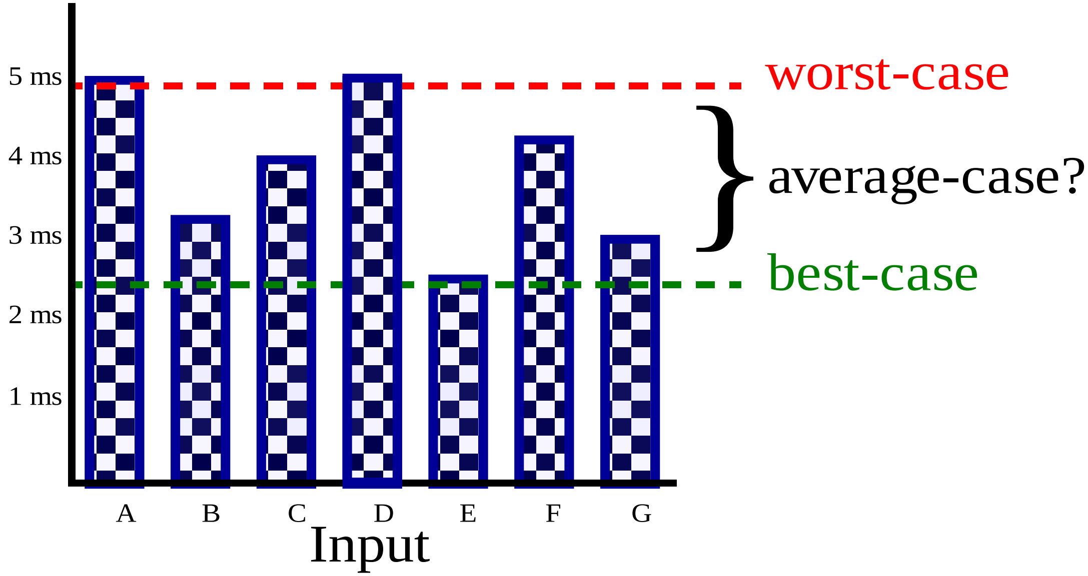
```
void insertionSort(int a[], int n)
{
    int i, j, value;

    for(i=1; i<n; i++)
    {
        value = a[i];

        for(j=i-1; j>=0; j--)
            if (a[j] > value)
                a[j+1] = a[j];
            else
                break;

        a[j+1] = value;
    }
}
```

알고리즘 분석

- Time complexity analysis
  - 이론적인 분석
  - 이론적인 수행시간을 입력의 크기(개수)를 변수로 하는 함수로 표현
  - T(n)
    - n : 입력데이터의 크기 (개수)
    - T(n) : 입력데이터의 크기가 n 일 때, primitive operation 혹은 basic operation 의 수행 회수

Time Complexity

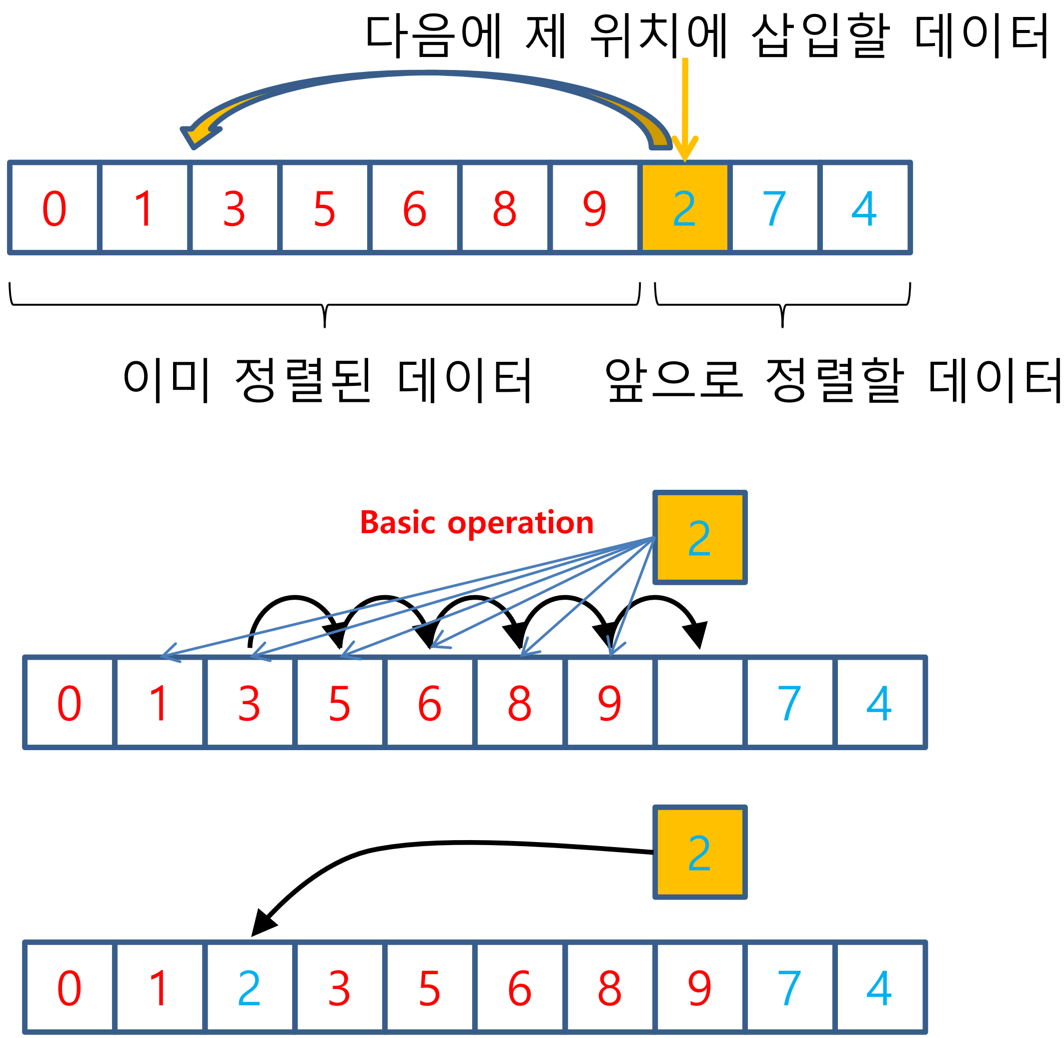
- Worst-case time complexity analysis
  - 알고리즘 수행시간은 입력되는 데이터의 종류에 따라 다름
  - 알고리즘 수행시간을 가장 길게 요하는 데이터가 입력되는 것을 가정하고 분석
    - 예 : 무게가 가벼운 구슬 찾기



Time Complexity (2)

- Example : 삽입정렬

```
void insertionSort(int a[], int n)
{
    int i, j, value;

    for(i=1; i<n; i++)
    {
        value = a[i];

        for(j=i-1; j>=0; j--)
            if (a[j] > value)
                a[j+1] = a[j];
            else
                break;

        a[j+1] = value;
    }
}
```



Time Complexity (3)

- Example : 삽입정렬
  - T(n) : basic operation 수행 횟수

```
void insertionSort(int a[], int n)
{
    int i, j, value;

    for(i=1; i<n; i++)
    {
        value = a[i];

        for(j=i-1; j>=0; j--)
            if (a[j] > value)
                a[j+1] = a[j];
            else
                break;

        a[j+1] = value;
    }
}
```

Best-case input data

0	1	2	3	4	5	6	7	8	9

$T(n) = 1+1+\dots + 1 = n-1$

Worst-case input data

9	8	7	6	5	4	3	2	1	0

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

Time Complexity (4)

- Example : 삽입정렬
  - T(n) : primitive operation 수행 횟수 (worst-case)

void insertionSort(int a[], int n) { int i, j, value;	Primitive operation 수	Primitive operation 총 수행 횟수
`    for(i=1;`	1	1
`            i<n; i++)`	2	2(n-1)
`    {`		
`        value = a[i];`	2	2(n-1)
`        for(j=i-1;`	2	2(n-1)
`                j>=0; j--)`	2	n(n-1)
`            if (a[j] > value)`	2	n(n-1)
`                a[j+1] = a[j];`	4	2n(n-1)
`            else`		
`                break;`		
`        a[j+1] = value;`	3	3(n-1)
`    }`		
`}`		

Time Complexity (5)

- Example : 삽입정렬
  - T(n) : worst-case

    - Basic operation 수행 횟수
      - $T(n) = n(n-1)/2$
        $= 0.5n^2 - 0.5n$

    - Primitive operation 수행 횟수
      - $T(n) = 4n(n-1) + 9(n-1) + 1$
        $= 4n^2 + 5n - 8$

KMU 국민대학교 KOOKMIN UNIVERSITY

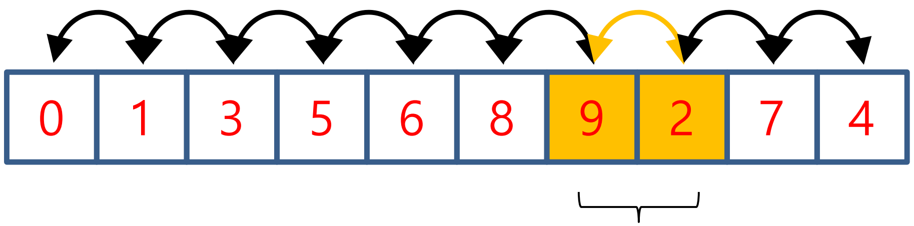
Algorithmic Engineering Laboratory

Time Complexity (6)

- Example : 버블정렬

```
void bubbleSort(int a[], int n)
{
    int i, j, tmp;

    for(i=0; i<n; i++)
        for(j=0; j<n-1; j++)
            if (a[j] > a[j+1])
            {
                tmp = a[j];
                a[j] = a[j+1];
                a[j+1] = a[j];
            }
}
```

Basic operation



인접한 두 정수의 위치를 계속 바꾸어줌

$T(n) = n(n-1)$

Time Complexity (7)

- Example : 선택정렬

```
void selectionSort(int a[], int n)
{
    int i, j, min, tmp;

    for(i=0; i<n-1; i++)
    {
        min = i;
        for(j=i+1; j<n; j++)
            if (a[j] < a[min])
               min = j;
        if (i != min)
        {
            tmp = a[i];
            a[i] = a[min];
            a[min] = tmp;
        }
    }
}
```

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

Big O Notation

- 수행시간의 증가율 (Growth rate of running time)
  - 알고리즘을 구현하는 소프트웨어와 하드웨어의 변화는
    - 시간복잡도 T(n) 의 상수배 정도 영향을 미치고
    - T(n) 의 증가율에는 영향을 미치지 않음
  - 예
    - 버블정렬의 시간복잡도 T(n)=n(n-1) 와 선택정렬의 시간복잡도 T(n)=n(n-1)/2 은 약 2배의 차이가 있다. 이는 두 알고리즘이 구현되는 소프트웨어 환경이나 하드웨어 영향에 따라 어느 알고리즘의 수행시간이 더 빠른지에 영향을 미친다.
    - 그러나, 병합정렬의 시간복잡도는 위 두 알고리즘의 시간복잡도보다 더 느리게 증가하는 함수로서, 구현되는 소프트웨어나 하드웨어에 영향을 받지 않고, n 이 매우 클 경우에는 항상 병합정렬이 위 두 알고리즘보다 훨씬 빠르다.

Big O Notation (2)

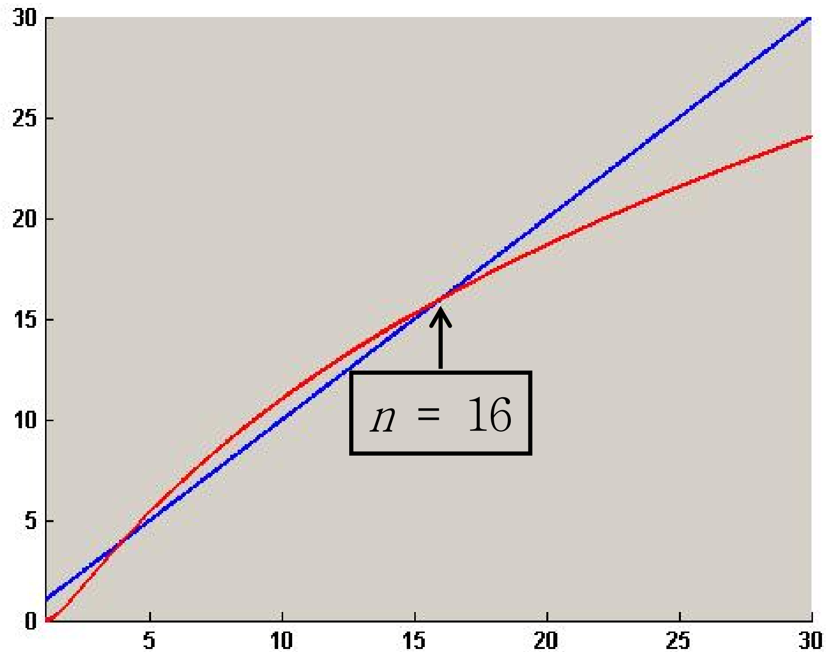
- Big O Notation (빅-오 표기법)
  - Asymptotic analysis (점근적 분석)
    - 시간복잡도 T(n) 에서 n 이 매우 큰 경우 ($n \rightarrow \infty$) 에만 고려함.
  - $T(n) = O(f(n))$ ($T(n) \in O(f(n))$)
    - T(n) 이 입력데이터의 크기 n 이 매우 큰 경우에는 함수 f(n)의 상수배를 초과하지 않음을 나타냄.
    - O(f(n)) 을 order 라 부름

Definition: $T(n) \in O(f(n))$ if there exist constants $c > 0$ and $n_0 > 0$ such that

$$T(n) \leq cf(n) \quad \text{for all } n \geq n_0.$$

Big O Notation (3)

- Example:
  - $(\log n)^2 = O(n)$



$T(n) = (\log n)^2$

$f(n) = n$

$(\log n)^2 \le n$ for all $n \ge 16$, so $(\log n)^2 = O(n)$

Big O Notation (4)

- Big O Notation (빅-오 표기법)
  - T(n)=n(n-1)
    - $T(n) = O(n^2)$
    - $T(n) = O(n^2-n)$
    - $T(n) = O(n^3)$
    - $T(n) = O(5n^4+4n^3+n)$
  - T(n)=O(f(n))

$$\lim_{n\to\infty}\frac{f(n)}{T(n)} = C \quad \text{or} \quad \lim_{n\to\infty}\frac{f(n)}{T(n)} = \infty$$

KMU 국민대학교 KOOKMIN UNIVERSITY

Algorithmic Engineering Laboratory

Big O Notation (4)

- Time complexity (order) 의 비교
  - Order 가 각각 O(f(n)), O(g(n)) 인 알고리즘 F, G 의 비교

$$\lim_{n\to\infty}\frac{g(n)}{f(n)}=\begin{cases}0 & (\text{case } 1)\\ c & (\text{case } 2)\\ \infty & (\text{case } 3)\end{cases}$$

- case 1: 알고리즘 G가 알고리즘 F 보다 우수
- case 2: 알고리즘 G와 알고리즘 F는 우열을 가릴 수 없음(비슷함)
- case 3: 알고리즘 F가 알고리즘 G 보다 우수

Big O Notation (4)

- Time complexity (order) 의 비교 예-1
  – 공정한 떡 나누기 알고리즘

알고리즘	위치표시의 수
알고리즘-1	$T_1(n) = n(n-1)/2-1$
알고리즘-3	$T_3(n) = n\log_2 n$

$$\lim_{n\to\infty}\frac{T_2(n)}{T_1(n)} = \lim_{n\to\infty}\frac{n\log_2 n}{n(n-1)/2-1} = 0$$

Big O Notation (4)

- Time complexity (order) 의 비교 예-2
  - 무게가 가벼운 구슬 찾기 알고리즘

알고리즘	위치표시의 수
알고리즘-1	$T_1(n) = n/2$
알고리즘-2	$T_2(n) = \log_2 n$
알고리즘-3	$T_3(n) = \log_3 n$

$$\lim_{n\to\infty} \frac{T_2(n)}{T_1(n)} = \lim_{n\to\infty} \frac{\log_2 n}{n/2} = 0$$

$$\lim_{n\to\infty} \frac{T_3(n)}{T_2(n)} = \lim_{n\to\infty} \frac{\log_3 n}{\log_2 n} = \frac{\log n / \log 3}{\log n / \log 2} = \frac{\log 2}{\log 3}$$

Big O Notation (5)

- Big O Notation (빅-오 표기법)
  - T(n)을 어떻게 표현하는 것이 가장 좋은가?
    - T(n) = O(f(n)) 에서 f(n) 은 T(n)에서 가장 고차단항이면서 계수가 1인 식으로 표현.
    - 즉, 다음을 만족하는 f(n) 중에서 단항이면서 계수가 1인 식

$$\lim_{n\to\infty}\frac{f(n)}{T(n)}=C$$

  - 예
    - $T(n) = 5n^4+4n^3+n$
      - $T(n) = O(n^4)$

Big O Notation (6)

- Big O Notation (빅-오 표기법)
  - Order 의 순서의 예

$$O(1) \quad O(\log n) \quad O(\sqrt{n}) \quad O(n) \quad O(n \log n) \quad O(n^2)$$

$$O(n^3) \quad O(n^{1000}) \quad O(2^n) \quad O(3^n) \quad O(1000^n) \quad O(n^n)$$

Analysis of Algorithms : Example

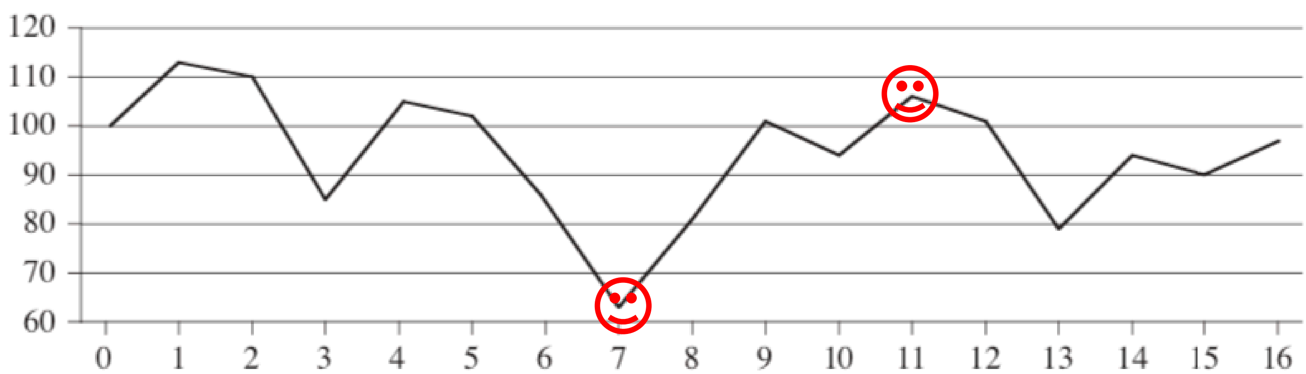
- Maximum Contiguous Subsequence Sum
  - n 개의 정수 $a_1, a_2, \ldots, a_n$ 이 주어졌을 때, 연속적인 부분 수열의 합 $\sum_{k=i}^{j} a_k$ 이 최대가 되는 구간 (i, j) 와 그 구간의 합을 계산하시오.

5	-7	2	3	-4	5	2	-7	8	-7

$$\sum_{k=3}^{9} a_k = 9$$

Analysis of Algorithms : Example

- 예
  - 과거의 주식가격이 주어졌을 때, 과거 어느 시점에서 그 주식을 구매하고, 어느 시점에서 매각하는 것이 가장 최대의 이익을 얻을 수 있는가? 또한 그 때의 최대이익은 얼마인가?



Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

최소시점에서 구매하고, 최대시점에 매각한다. (Not always)

Max. Conti. Subseq. Sum (MCSS)

- 알고리즘 1 (Brute-force, 주먹구구, 모든 경우 시도)
  - 모든 구간 (i, j) $(1 \le i \le j \le n)$ 에 대하여 그 구간의 합 $\sum_{k=i}^{j} a_k$ 을 계산하고, 이 합들 중에서 가장 큰 합을 계산한다.

```
int maxSubsequenceSum (int a[], int n,
                    int *start, int *end)
{
    int i, j, k;
    int maxSum = 0;

    *start = *end = -1;

    for(i=0; i<n; i++)
        for(j=i; j<n; j++)
        {
            int thisSum = 0;

            for(k=i; k<=j; k++)
                thisSum += a[k];

            if(thisSum > maxSum)
            {
                maxSum = thisSum;
                *start = i;
                *end = j;
            }

        }
    return maxSum;
}
```

$$T(n) = \sum_{i=1}^{n} \sum_{j=i}^{n} \sum_{k=i}^{j} 1$$

$$= \frac{n(n+1)(n+2)}{6}$$

$$= O(n^3)$$

Max. Conti. Subseq. Sum (MCSS) (2)

- 알고리즘 2
  - 알고리즘 1에서 $\sum_{k=i}^{j} a_k = \sum_{k=i}^{j-1} a_k + a_j$ 을 이용하면 좀 더 효율적인 알고리즘을 만들 수 있다.

```
int maxSubsequenceSum (int a[], int n,
                       int *start, int *end)
{
    int i, j;
    int maxSum = 0;

    *start = *end = -1;

    for(i=0; i<n; i++)
    {
        int thisSum = 0;

        for(j=i; j<n; j++)
        {
            thisSum += a[j];

            if(thisSum > maxSum)
            {
                maxSum = thisSum;
                *start = i;
                *end = j;
            }
        }
    }
    return maxSum;
}
```

$$
\begin{aligned}
T(n) &= \sum_{i=1}^{n} \sum_{j=i}^{n} 1 \\
&= n + (n-1) + \cdots + 1 \\
&= \frac{n(n+1)}{2} \\
&= O(n^2)
\end{aligned}
$$

Max. Conti. Subseq. Sum (MCSS) (3)

- 알고리즘 3

```
int maxSubsequenceSum (int a[], int n,
                       int *start, int *end)
{
    int i, j;
    int maxSum = 0, thisSum = 0;

    *start = *end = -1;

    for(i=0, j=0; j<n; j++)
    {
        thisSum += a[j];

        if(thisSum > maxSum)
        {
            maxSum = thisSum;
            *start = i;
            *end = j;
        }
        else if(thisSum < 0)
        {
            i = j+1;
            thisSum = 0;
        }
    }
    return maxSum;
}
```

5	-7	2	3	-4	5	2	-7	8	-7

$$T(n) = n$$
$$= O(n)$$