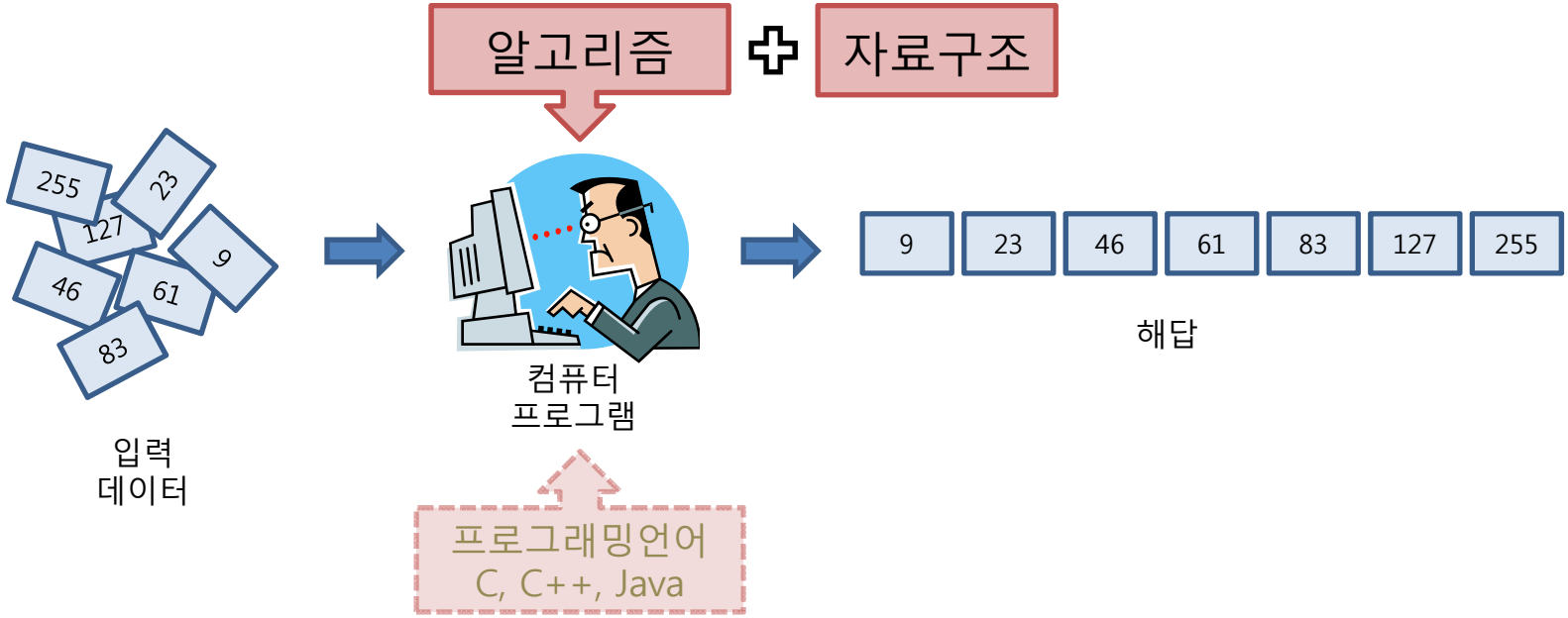
기본 알고리즘
제2장

2016. Fall

국민대학교 컴퓨터공학부 최준수

알고리즘?

컴퓨터로 푸는 퍼즐 (혹은 수수께끼)
- 체계적인 문제해결방법이 필요함
- 꾀(trick)가 필요함
- 해답을 알기 전에는 매우 어렵지만,
  알고 난 후에는 매우 쉬움.



알고리즘 관련 문제의 예

문제 (난이도: 어려움)

양팔저울과 $2k$ 개의 무게추가 있다. 무게추의 무게는 모두 다르다. 무게추를 양팔저울의 왼쪽이나 오른쪽에 임의의 순서로 계속 양팔저울의 왼쪽이나 오른쪽으로 올릴 때, 무게추를 한 개씩 올릴 때마다 무거운 쪽을 나타내는 문자를 순서대로 기록한다. 문자는 'L' 이나 'R' 을 사용하며, 'L' 은 양팔저울의 왼쪽이 현재 무거운 쪽임을 나타내고, 'R' 은 그 반대를 나타낸다.
'L' 이나 'R' 로 구성된 길이가 $2k$ 인 문자열이 주어졌을 때, 양팔저울의 무거운 쪽이 이 문자순서가 되도록 양팔저울에 올리는 무게추의 순서를 계산하는 프로그램을 작성하시오.

평가방법

제한시간은 1초이고, 부분점수는 없다.

입력의 예

```
6
LRLLRR
10 1 7 4 2 9
```

출력의 예

```
1
4 L
7 R
9 L
1 L
10 R
2 R
```

입력의 예

```
100
RLLRLRRLLLLRRRRRLRLLRLRRRLLRRRLLLRLLRRRRLLLLLRRRRLRLLRRRLLLL
LLRLLLRLRLLLRRLRRLRLLLLRRRLLRLLRRRRRRLLR
226 245 170 172 9 228 231 86 122 110 248 236 251 276 60 61 136 42
105 76 184 104 202 274 39 24 72 88 149 289 116 140 153 30 235 295
173 112 13 298 82 238 77 71 232 243 .....
```

알고리즘 관련 문제의 예 (2)

```
#include <stdio.h>

#define filename1 "balance.in"
#define filename2 "balance.out"
#define MAX 5000

int weight[MAX],solution[MAX][2], n;
char sequence[MAX];

void input_data()
{
    int i ;
    FILE *inFile;

    inFile = fopen(filename1,"r");
    fscanf(infile,"%d ",&n);
    fscanf(infile,"%s", sequence);
    for(i=0 ; i<n ; i++)
        fscanf(infile,"%d",&weight[i]);
    fclose(infile);
}

void output_solution()
{
    int i;
    FILE *outFile;

    outFile=fopen(filename2,"w");
    fprintf(outfile,"1\n");
    for(i=0; i<n; i++)
        fprintf(outfile,"%d %c\n",
            weight[solution[i][0]],(solution[i][1])?('L'):('R'));
    fclose(outfile);
}

int intcomp(void *a,void *b)
{
    return ( *(int *)a - *(int *)b);
}
```

```
void process()
{
    int i, low=0, high=n-1, reverse;

    qsort(weight, n, sizeof(int), intcomp);
    if (sequence[n-1]=='R') reverse=1;
    for(i=n-2 ; i>=0 ; i--) {
        if (sequence[i]==sequence[i+1])
            solution[i+1][0] = low++ ;
        else solution[i+1][0] = high-- ;
        solution[i+1][1] = (solution[i+1][0]+reverse)%2;
    }
    solution[0][0] = low ;
    solution[0][1] = (solution[0][0]+reverse)%2;
}
```

```
int main()
{
    input_data();
    process();
    output_solution();
    return 0 ;
}
```

알고리즘 관련 문제의 예 (3)

- 해결 알고리즘
  - 문제해결기법 :
    - Working backward(역방향 추론), backward induction (후진 귀납법)
    - 무게추를 천칭에 올려 놓은 후, 무게추를 하나씩 내리면서 문제를 해결함
  - 해법
    - n 개의 무게추: $w_1 > w_2 > \ldots > w_n$
    - W=RLLRR· · ·RRL, W^T=LRR · · ·RRLLR (항상 W^T 는 L 로 시작한다고 가정하여 설명)
    - 천칭의 왼쪽에 무게추 $w_1 > w_3 > w_5 > \ldots$ 를 올려 놓고,
    - 천칭의 오른쪽에 무게추 $w_2 > w_4 > w_6 > \ldots$ 를 올려 놓는다.
    - 초기에는 천칭의 왼쪽이 무겁다 (W^T 는 L 로 시작한다는 가정과 일치)

알고리즘 관련 문제의 예 (4)

- 해결 알고리즘
  - 해법
    - 무게추를 하나씩 내리는 과정에서
      - 천칭의 무거운 쪽이 바뀌면, 천칭에서 가장 무거운 무게추를 내린다.
      - 천칭의 무거운 쪽이 바뀌지 않으면, 천칭에서 가장 가벼운 무게추를 내린다.
  - 예
    - W=RLLRRRLRRL, W^T=LRRLRRRLLR
    - 천칭의 왼쪽 무게추 : $w_1 > w_3 > w_5 > w_7 > w_9$
    - 천칭의 오른쪽 무게추 : $w_2 > w_4 > w_6 > w_8 > w_{10}$
    - 천칭에서 무게추를 내리는 순서 (천칭을 올리는 역순)
      - w_1 w_{10} w_2 w_3 w_9 w_8 w_4 w_7 w_5 w_6

알고리즘 관련 문제의 예 (4)

- 알고리즘 증명
  - Observation
    - 위 알고리즘에 따라 무게추를 내리는 과정에서
      - 천칭의 양쪽에 올려진 무게추의 개수는 아래 경우만 발생한다
        - » 무개추의 개수가 같다.
        - » 무개추의 개수가 1개의 차이만 발생한다.
          - 가장 무거운 추와 가장 가벼운 추가 같은 쪽에 올려져 있다.
      - 무게추의 개수가 2개 이상인 경우가 발생하지 않는 이유는?

(case 1) 무게추의 개수가 같은 경우

- 가장 무거운 추를 내리면
- 가장 가벼운 추를 내리면

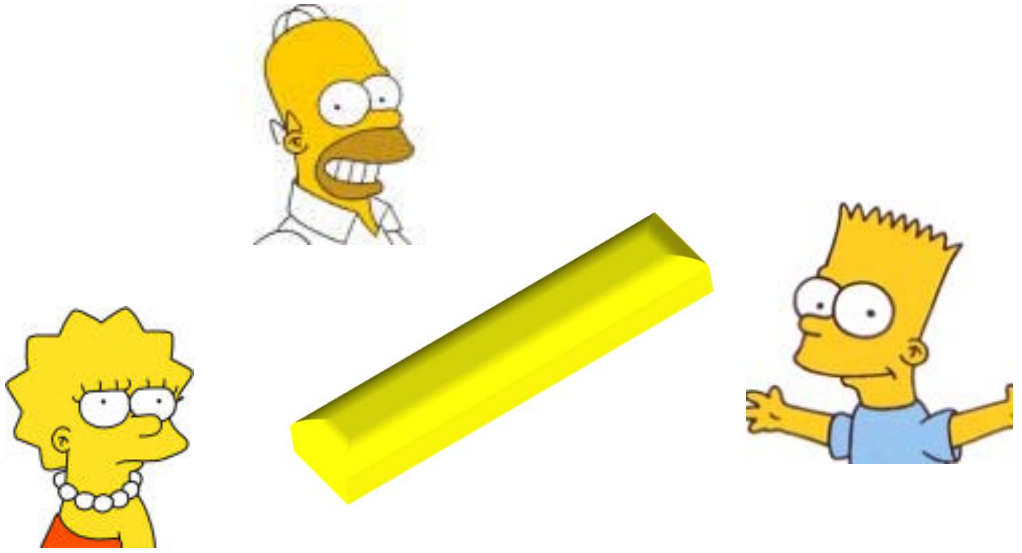
(case 2) 무게추의 개수가 1개가 다른 경우

- 가장 무거운 추를 내리면
- 가장 가벼운 추를 내리면

예제

- 공정한 떡 나누기 (Fair Cake Cutting)
  - 컴퓨터공학부 학생들이 교외 학부 오리엔테이션에 참석하게 되었다.
  - 오리엔테이션 첫날 직사각형의 긴 떡 한 덩어리를 저녁으로 제공하고, 떡 한 덩어리를 여러 학생들이 나누어서 저녁거리로 해야 한다는 것을 알게 되었다.

주어진 문제

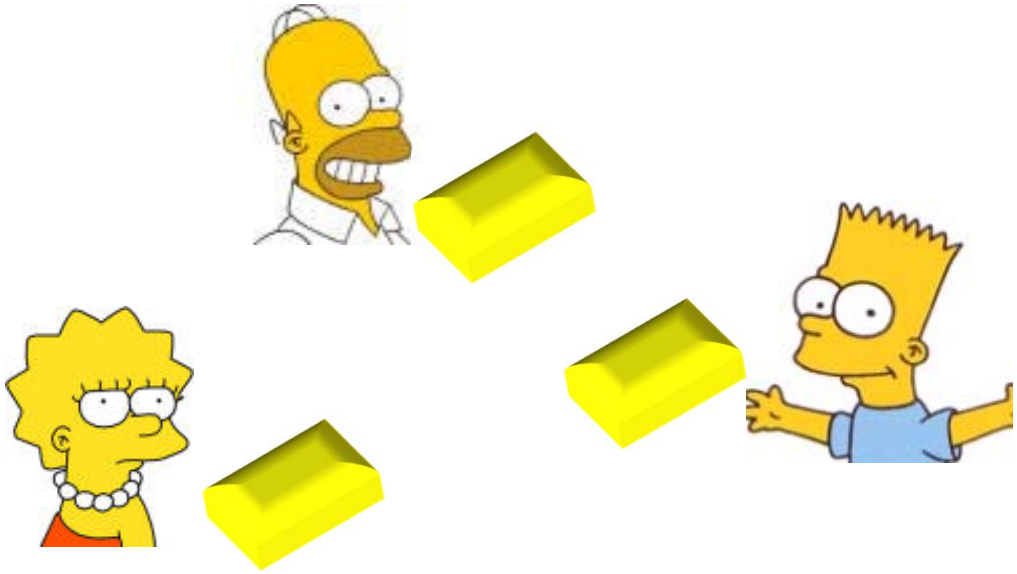


예제

- 공정한 떡 나누기 (Fair Cake Cutting)
  - 저녁식사를 빨리하기 위하여, 주어진 떡을 여러 학생들에게 공정하게 나누는 방법을 서로 고안하여, 그 중에서 제일 빠르게 나누는 방법을 채택하여 한 달 동안 사용하기로 하였다.

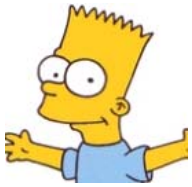
알고리즘 설계

알고리즘 분석



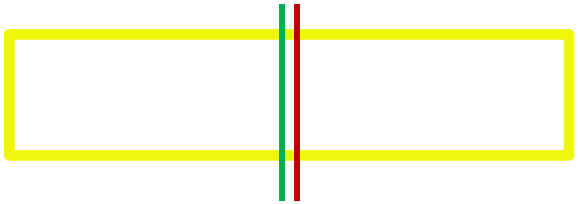
공정한 떡 나누기

- 방법-1



  - n 명이 각각 떡의 왼쪽으로부터, 가장 공정하게 판단하여, 떡의 1/n 이라고 생각하는 부분에 표시를 한다 (서로 다른 사람이 표시한 부분을 모른다고 가정).
    - 각자는 자기가 표시한 부분에 잘라서 왼쪽의 1/n 부분을 가져도 불만이 없음.

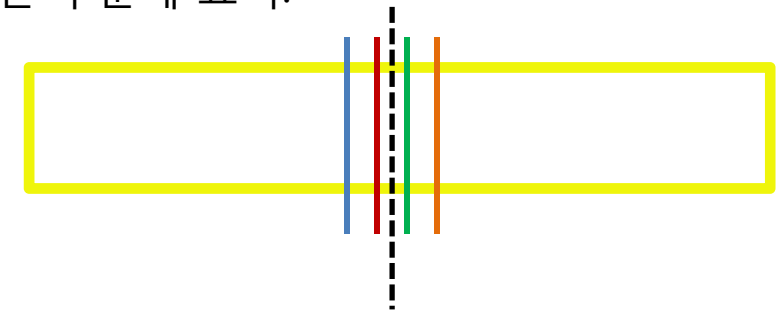
  - 왼쪽 1/n을 가장 왼쪽에 표시한 사람에게 잘라서 준다.
  - 오른쪽 (n-1)/n 부분을 가장 왼쪽에 표시한 사람을 제외한 나머지 (n-1) 명이 똑 같은 방법으로 나눈다.



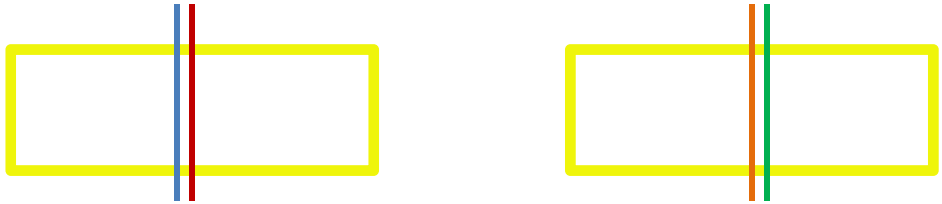
공정한 떡 나누기

- 방법-2

  - n 이 짝수 (n=2k)
    - n 명이 각각 떡의 왼쪽으로부터, 가장 공정하게 판단하여, 떡의 1/2 이라고 생각하는 부분에 표시.



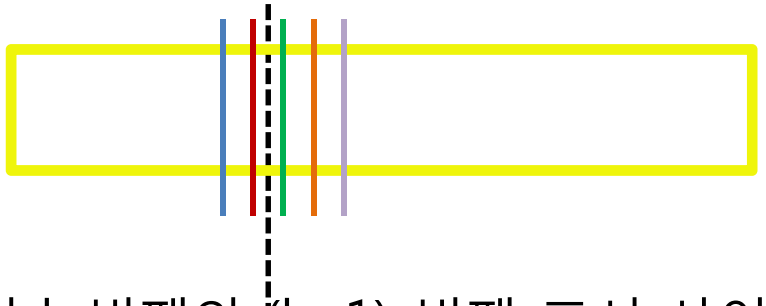
  - 왼쪽에서 k-번째와 (k+1)-번째 표시 사이에 임의로 자른다.
  - 나누어진 떡의 왼쪽 조각에 대해서, 왼쪽 떡은 왼쪽 떡 부분에 표시를 하는 사람끼리, 오른 쪽 떡은 오른 쪽에 표시한 사람들끼리 같은 방법으로 계속한다.



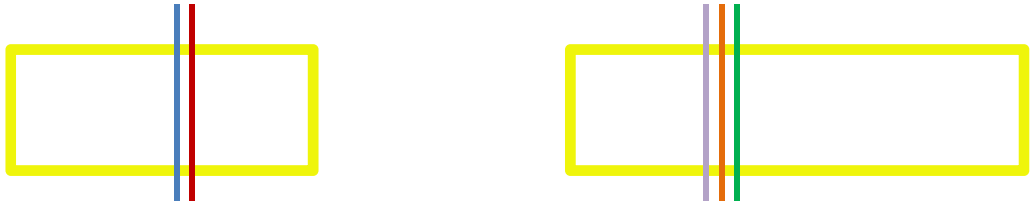
공정한 떡 나누기

- 방법-2

  - n 이 홀수 (n=2k+1)
    - n 명이 각각 떡의 왼쪽으로부터, 가장 공정하게 판단하여, 떡의 k/(2k+1) 이라고 생각하는 부분에 표시.



  - 왼쪽에서 k-번째와 (k+1)-번째 표시 사이에 임의로 자른다.
  - 나누어진 떡의 왼쪽 조각에 대해서, 왼쪽 떡은 왼쪽 떡 부분에 표시를 하는 사람끼리, 오른 쪽 떡은 오른 쪽에 표시한 사람들끼리 같은 방법으로 계속한다.



공정한 떡 나누기

- 방법-3

– 2명

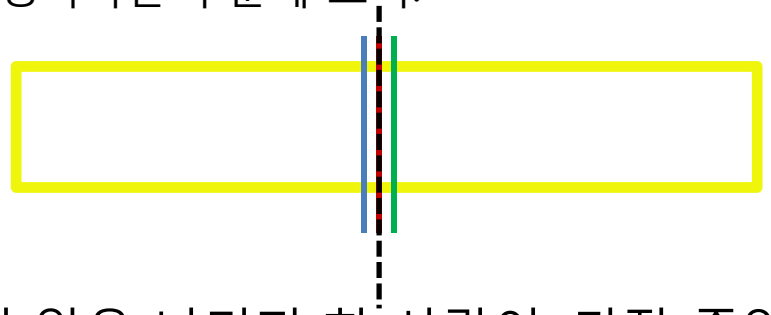
- 먼저, 한 명이 떡의 ½ 이라고 생각하는 부분에서 떡을 나눈다.

- 나머지 한 명이 떡 두 조각 중에서 한 조각을 선택을 하고, 나머지 한 조각은 떡을 나눈 사람에게 준다.

공정한 떡 나누기

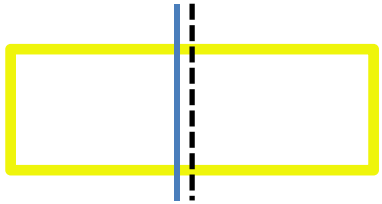
- 방법-3

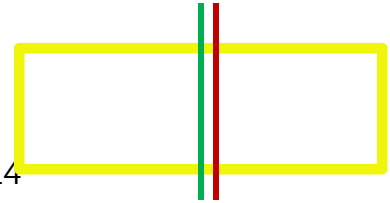
  - n 이 짝수 (n=2k)
    - 먼저, 2k-1 명이 각각 떡의 왼쪽으로부터, 가장 공정하게 판단하여, 떡의 1/2 이라고 생각하는 부분에 표시.



  - 표시하지 않은 나머지 한 사람이, 가장 중앙에 위치한 표시를 기준으로 떡을 자른 후, 왼쪽 조각이나 오른쪽 조각을 선택한다.

  - 떡을 자른 사람이 왼쪽 조각을 선택하면 (오른쪽을 선택하면?)
    - 왼쪽 조각에 표시한 사람과 자른 사람이 왼쪽 조각에서 계속하고
    - 가장 중앙에 표시한 사람과 오른쪽 조각에 표시한 사람이 오른쪽 조각에서 같은 방법으로 계속한다.

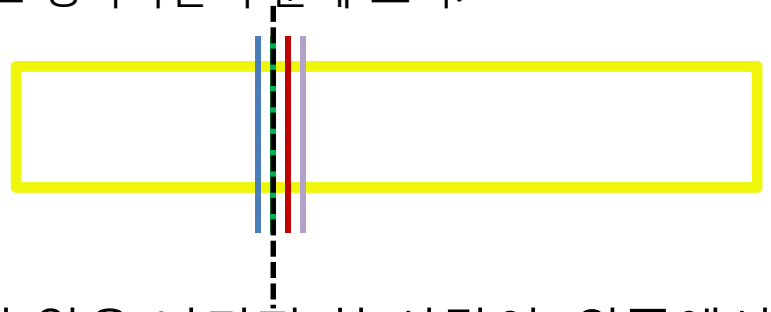




공정한 떡 나누기

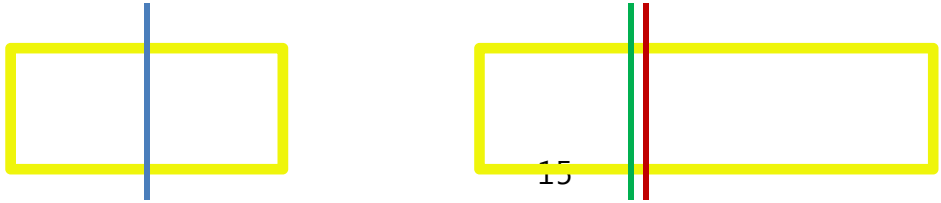
- 방법-3

  - n 이 홀수 (n=2k+1)
    - 먼저, 2k 명이 각각 떡의 왼쪽으로부터, 가장 공정하게 판단하여, 떡의 k/(2k+1) 이라고 생각하는 부분에 표시.



  - 표시하지 않은 나머지 한 사람이, 왼쪽에서 k-번째 위치한 표시를 기준으로 떡을 자른 후, 왼쪽 조각이나 오른쪽 조각을 선택한다.

  - 떡을 자른 사람이 왼쪽 조각을 선택하면 (오른쪽을 선택하면?)
    - 왼쪽 조각에 표시한 사람과 자른 사람이 왼쪽 조각에서 계속하고
    - 가장 중앙에 표시한 사람과 오른쪽 조각에 표시한 사람이 오른쪽 조각에서 같은 방법으로 계속한다.

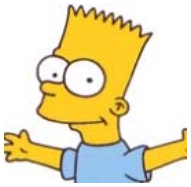


공정한 떡 나누기

- 각 방법들이 얼마나 빠르게 나눌 수 있는가?
  - 빠르게?
    - 빠르기를 측정하는 기준은 무엇인가?
      - 실제로 실행해 본다
      - 개략적으로 빠르기는, 떡에 표시하거나 자르는 횟수에 비례할 것이므로, 이 횟수를 기준으로 빠르기를 나타낸다.

공정한 떡 나누기

- 빠르기 분석 – (방법-1)



– n = 8

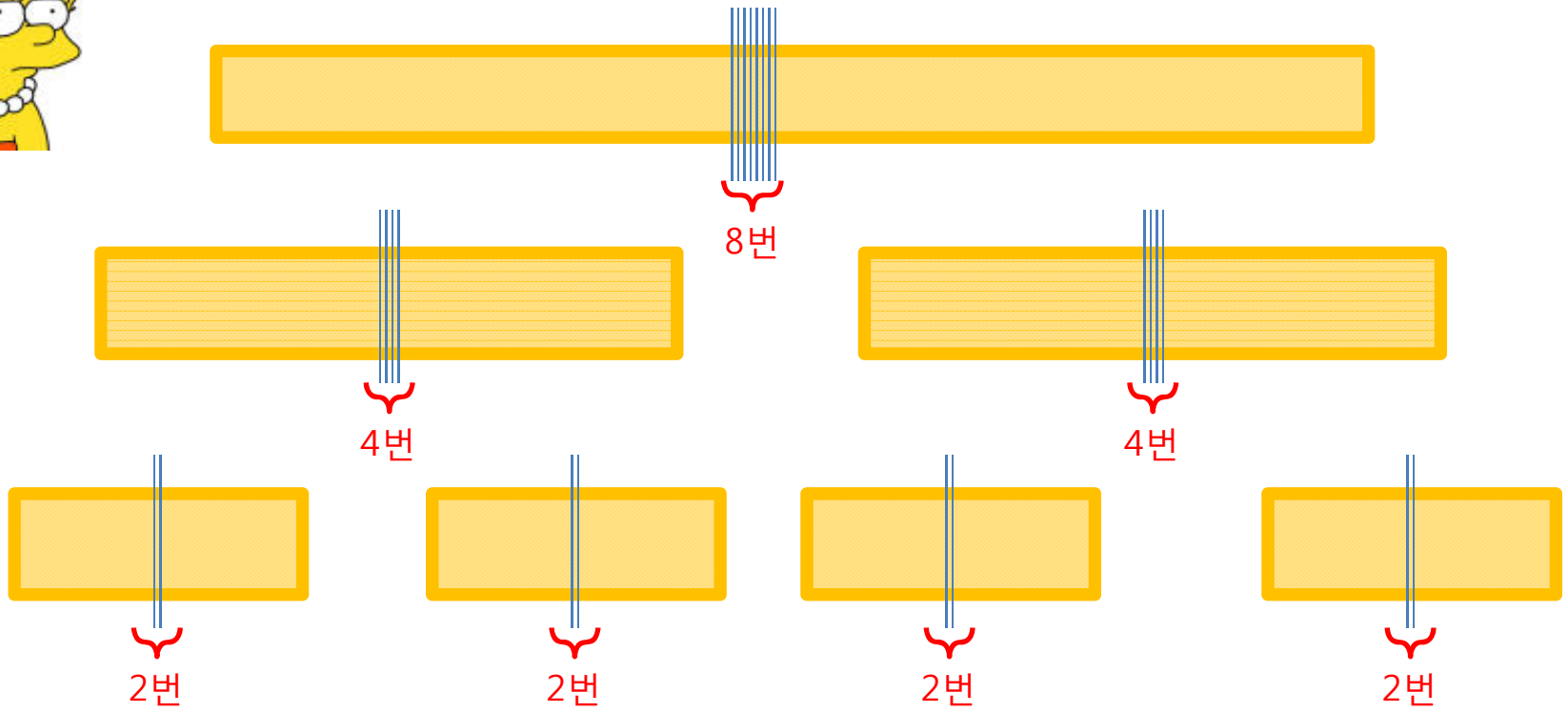
횟수 = 8+7+…+2 = 35

(일반식)
횟수 = n+(n-1)+…+2 = n(n+1)/2-1

공정한 떡 나누기

- 빠르기 분석 – (방법-3)

– n = 8



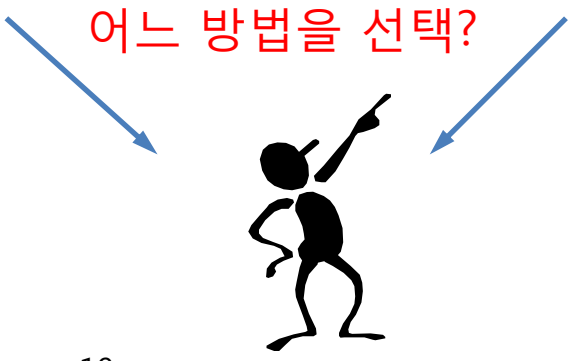
횟수 = 8+8+8 = 24

(일반식)
횟수 = $n+n+\cdots+n = n\log_2 n$

공정한 떡 나누기

- 방법들의 비교

	방법-1	방법-3
방법(알고리즘 설계)	앞 설명 참조	앞 설명 참조
빠르기(알고리즘 분석)		
n=8	35	24
n (일반식)	n(n+1)/2-1	$n\log_2 n$
n=1,024	524,799	10,240
n=65,536	2,147,516,416	1,048,576



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), …

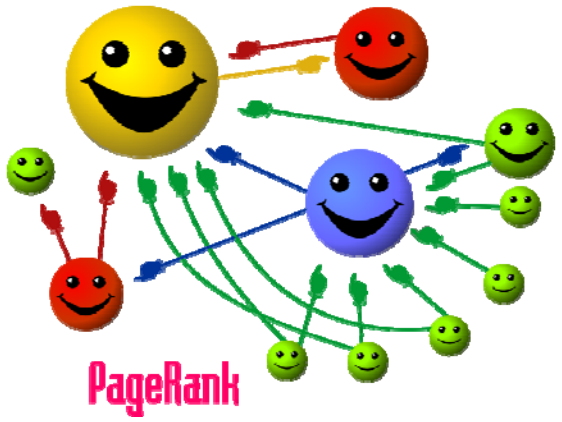
찾고자하는 단어가 문서에서 나타나는 빈도수나 그 문서와의 상관관계도에 따라서 페이지를 우선적으로 나열하는 알고리즘 채택

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 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

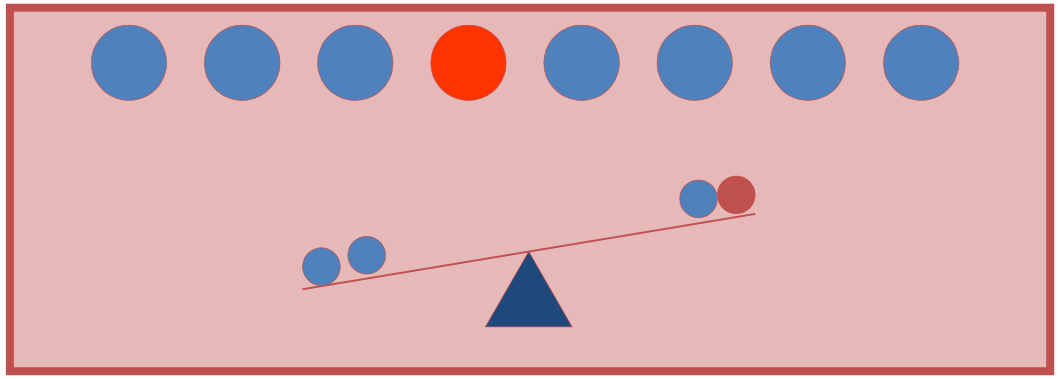
by Niklaus Wirth

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?

Searching Problem (2)

- 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.



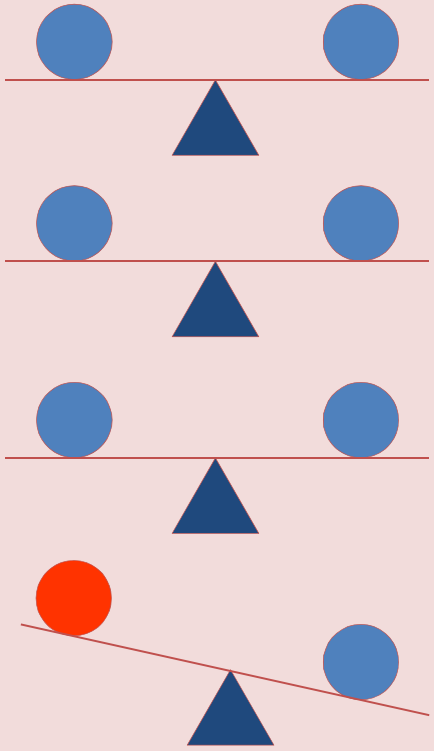
Searching Problem (3)

- 문제: 천칭을 이용한 *무게가 가벼운* 구슬찾기

알고리즘 설계 Design of Algorithm	알고리즘 분석 Analysis of Algorihm
알고리즘 0: 한 개의 구슬을 고정하고, 이 구슬과 다른 모든 구슬의 무게를 각각 비교한다.	천칭을 몇 번 사용하여 가벼운 구슬을 찾았나? ? 번

Searching Problem (3)

- 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? 4 번

Searching Problem (3)

- 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? ? 번

Searching Problem (4)

- 문제: 천칭을 이용한 *무게가 가벼운* 구슬찾기

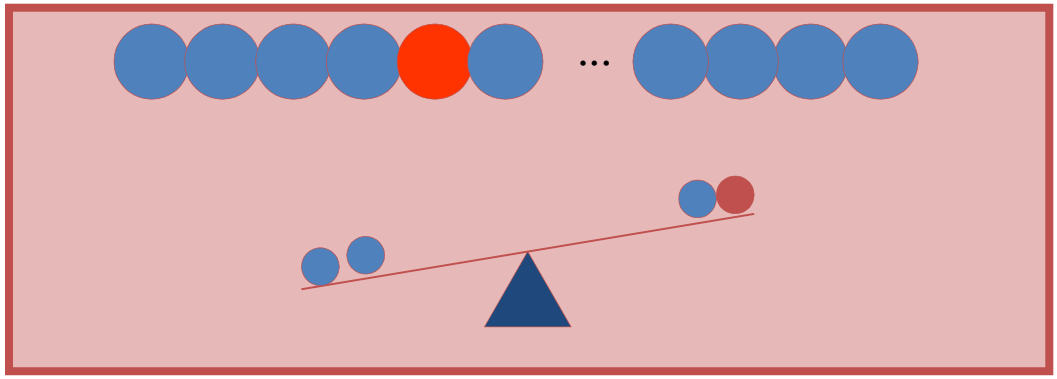
알고리즘 설계 Design of Algorithm	알고리즘 분석 Analysis of Algorihm
알고리즘 3:	천칭을 몇 번 사용하여 가벼운 구슬을 찾았나? ? 번

Searching Problem (5)

- 문제: 천칭을 이용한 *무게가 가벼운* 구슬찾기
  - 앞에서 제시한 알고리즘 0, 1, 2, 3 중에서 어느 알고리즘이 *효율적인* 알고리즘인가?
  - 알고리즘 3에서 제시한 횟수보다 더 적은 횟수로 무게가 가벼운 구슬을 찾을 수 있을까?

Searching Problem (6)

- 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.



Searching Problem (7)

- 문제: 천칭을 이용한 *무게가 가벼운* 구슬찾기

알고리즘 설계 Design of Algorithm	알고리즘 분석 Analysis of Algorihm
알고리즘 0:	천칭을 몇 번 사용하여 가벼운 구슬을 찾았나? ? 번

Searching Problem (7)

- 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? ? 번

Searching Problem (7)

- 문제: 천칭을 이용한 *무게가 가벼운* 구슬찾기

알고리즘 설계 Design of Algorithm	알고리즘 분석 Analysis of Algorihm
알고리즘 2: Recursive Algorithm - Base case - Recursive step	천칭을 몇 번 사용하여 가벼운 구슬을 찾았나? ? 번

Searching Problem (8)

- 문제: 천칭을 이용한 *무게가 가벼운* 구슬찾기

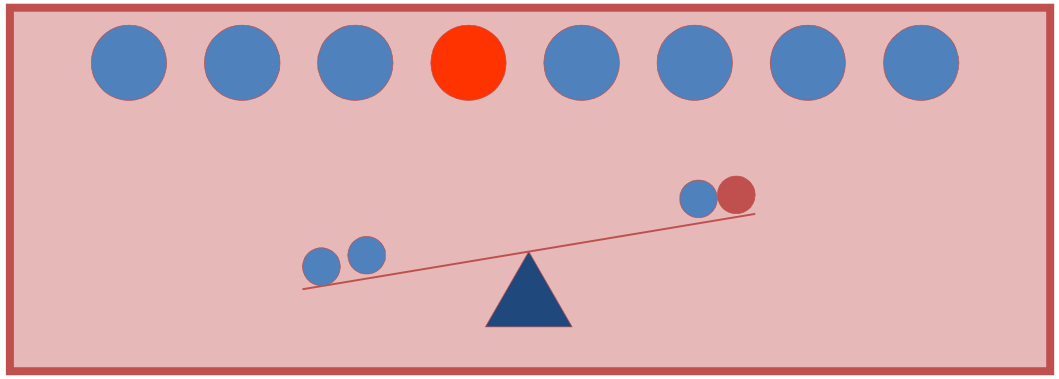
알고리즘 설계 Design of Algorithm	알고리즘 분석 Analysis of Algorihm
알고리즘 3: Recursive Algorithm - Base case - Recursive step	천칭을 몇 번 사용하여 가벼운 구슬을 찾았나? ? 번

Searching Problem (5)

- 문제: 천칭을 이용한 *무게가 가벼운* 구슬찾기
  - 위 알고리즘들은 임의의 개수의 구슬이 주어지더라도 반드시 무게가 가벼운 구슬을 찾을 수 있는가? (verification)
  - 앞에서 제시한 알고리즘 0, 1, 2, 3 중에서 어느 알고리즘이 *효율적인* 알고리즘인가?
  - 알고리즘 3에서 제시한 횟수보다 더 적은 횟수로 무게가 가벼운 구슬을 찾을 수 있을까?
    - Recursive하게 계속 2개의 group 으로 묶어서 무게를 다는 것보다 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?

Searching Problem (9)

- Searching worst case 3번만에 가능 2개2개// 1개1개 (시험출제예상)
  - 무게가 다른 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 구슬과 *다르다* (즉, 무거울 수도 있으며, 가벼울 수도 있다). 천칭을 이용하여 이들 구슬 중에서 무게가 다른 구슬을 찾으려고 한다. 최소 횟수로 천칭을 이용하여 무게가 다른 구슬을 찾는 방법을 제시하시오.



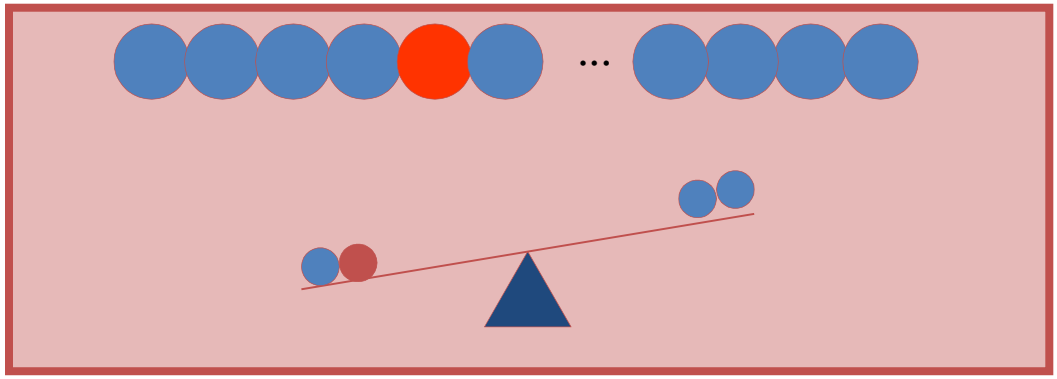
Searching Problem (10)

- 문제: 천칭을 이용한 *무게가 다른* 구슬찾기

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

Searching Problem (10)

- Searching
  - 무게가 다른 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 구슬과 *다르다* (즉, 무거울 수도 있으며, 가벼울 수도 있다). 천칭을 이용하여 이들 구슬 중에서 무게가 다른 구슬을 찾으려고 한다. 최소 횟수로 천칭을 이용하여 무게가 다른 구슬을 찾는 방법을 제시하시오.



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

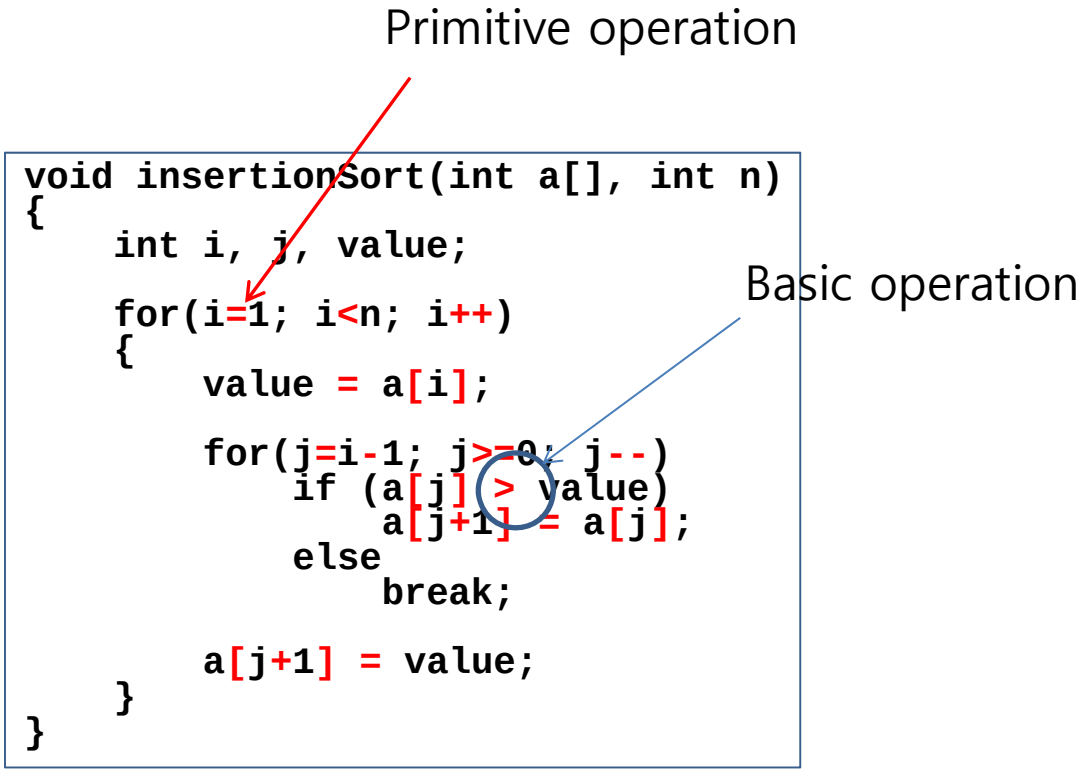
알고리즘 분석

- 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

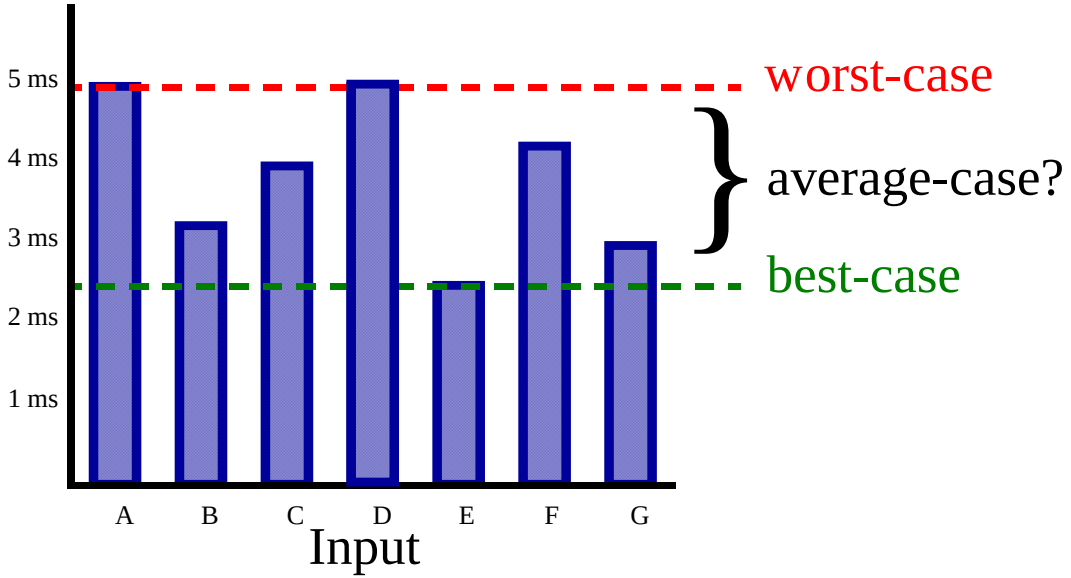


알고리즘 분석

- 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)

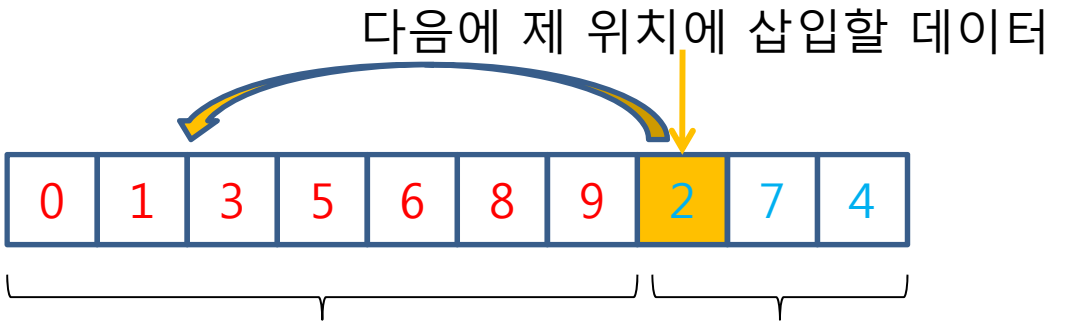
- Example : 삽입정렬

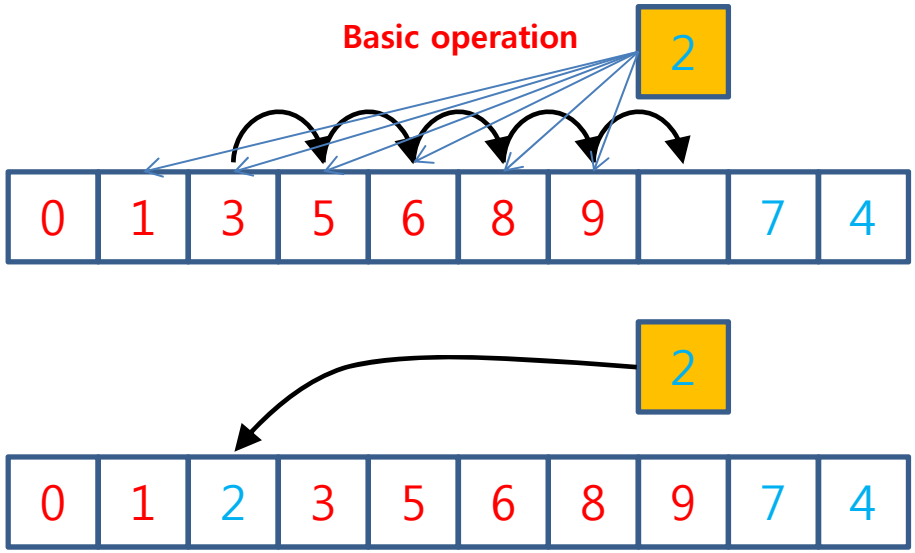
```
void insertionSort(int a[], int n)
{
    int i, j, value;

    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+… + 1 = n-1

Worst-case input data

9	8	7	6	5	4	3	2	1	0

T(n) = 1+2+… + (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)

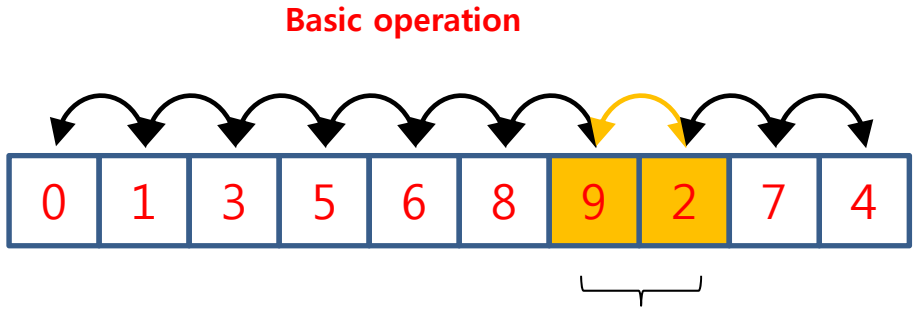
- Example : 삽입정렬
  - T(n) : worst-case
    - Basic operation 수행 횟수
      - $T(n) = n(n-1)/2$
        $= 0.5n^2 - 0.5n$
    - Primitive operation 수행 횟수
      - $T(n) = 4n(n-1) + 9(n-1) + 1$
        $= 4n^2 + 5n - 8$

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];
            }
}
```



인접한 두 정수의 위치를 계속 바꾸어줌

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

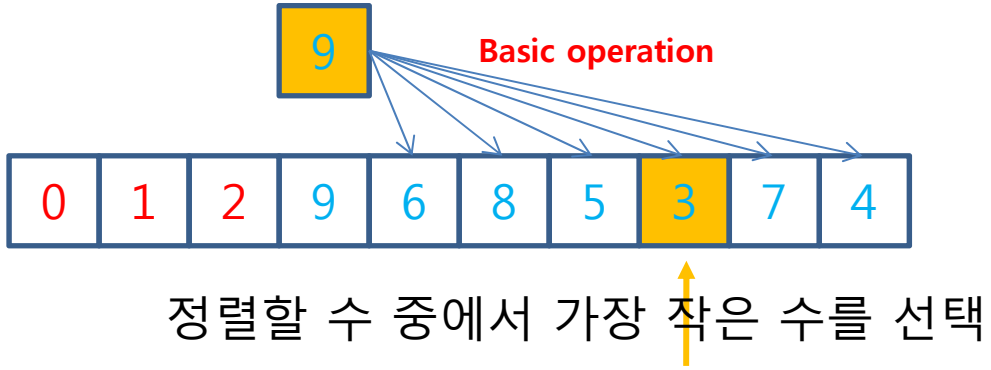
뒤에 정렬된거는 포함안하면 $n(n-1)/2$ 까지 가능

Time Complexity (7)

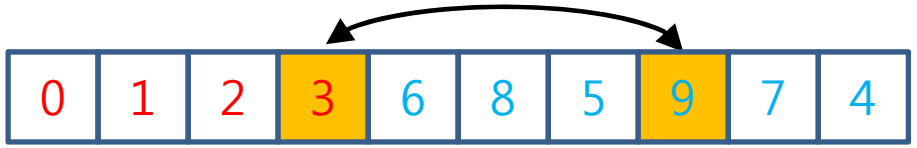
- Example : 선택정렬

```
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 수식으로는 2배 선택정렬이 빠른거 같지만 상수배정도는 무시할 만 한다
    - 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 ($n \to \infty$) 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

수식 order

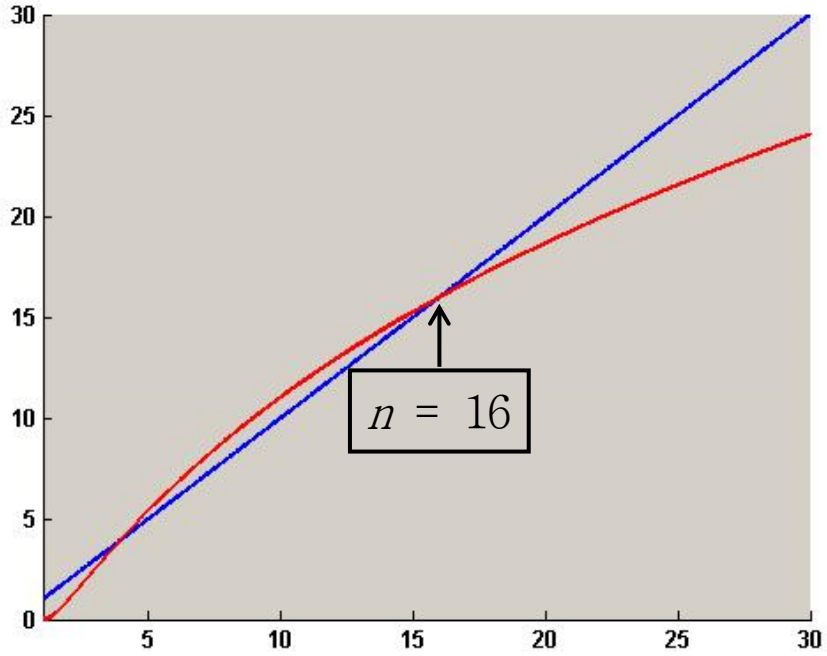
이거하는이유 알고리즘 비교하기 위해

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

$T(n) \leq cf(n)$ for all $n \geq n_0$.

Big O Notation (3)

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



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

$f(n) = n$

$(\log\ n)^2 \leq n$ for all $n \geq 16$, 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))

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

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 F 는 우열을 가릴 수 없음
- 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	$T_1(n) = n(n-1)/2-1$	O(n^2)
알고리즘-3	$T_3(n) = n\log_2 n$	O(nlog밑2n)

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

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

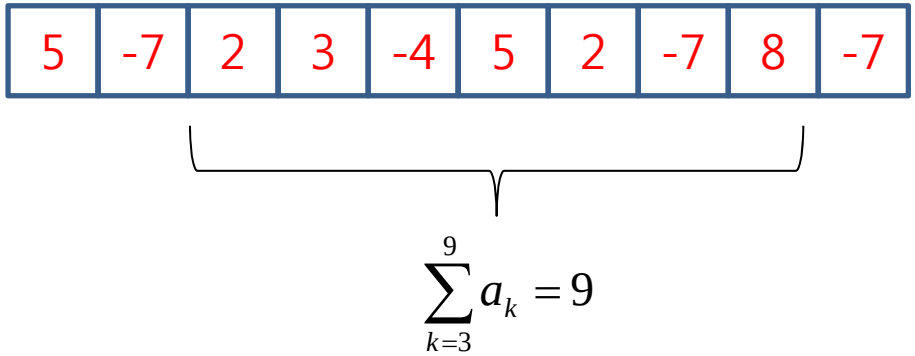
Big O Notation (6)

- 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.



Max. Conti. Subseq. Sum (MCSS)

- 알고리즘 1
  - 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, k;
    int maxSum = 0;

    *start = *end = -1;

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

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

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

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

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

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