



CSE2010 자료구조론

Week 1: Algorithm

ICT융합학부 한진영

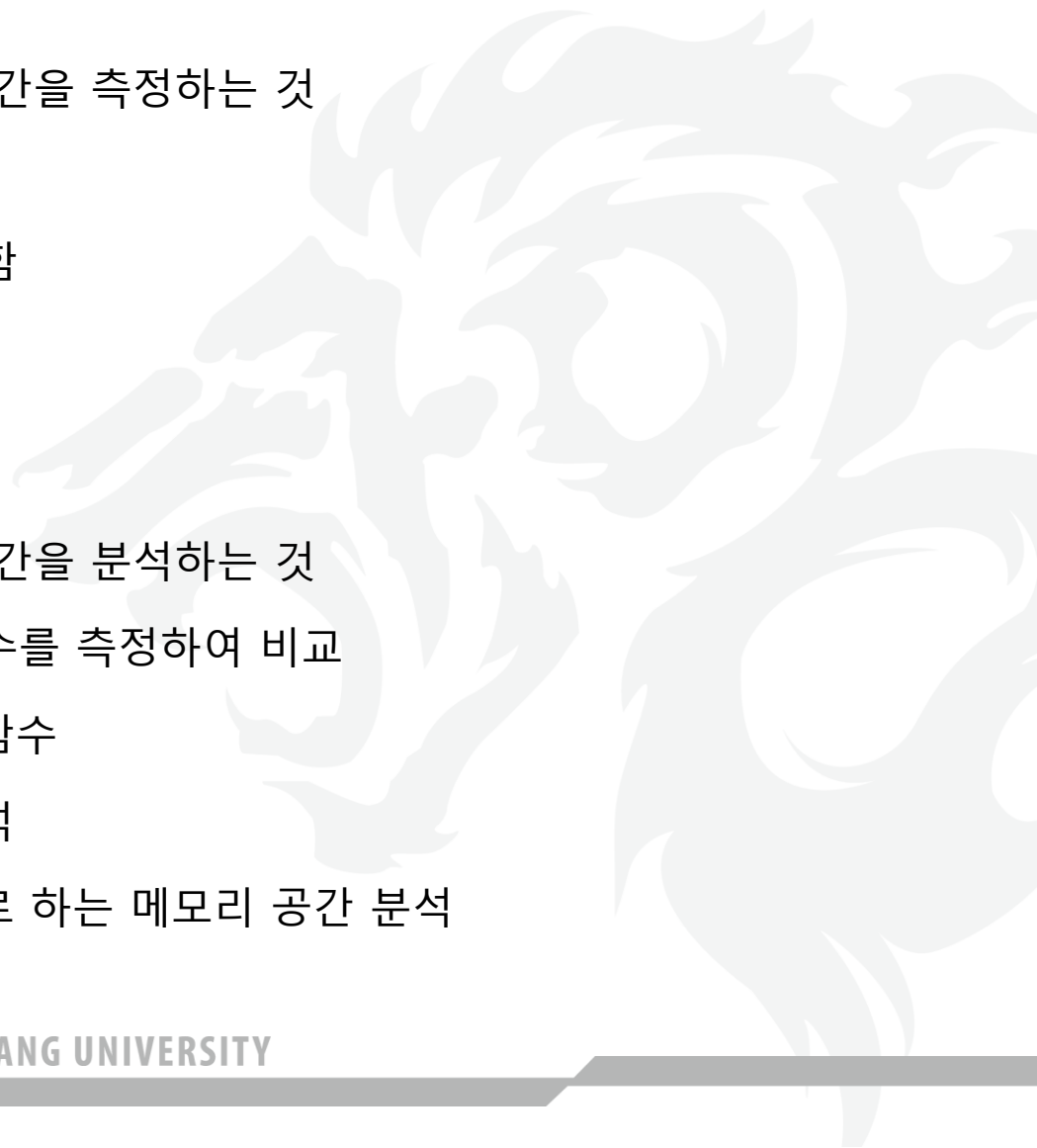
알고리즘 성능 분석 기법

■ 수행 시간 측정

- 두개의 알고리즘의 실제 수행 시간을 측정하는 것
- 실제로 구현하는 것이 필요
- 동일한 하드웨어를 사용하여야 함

■ 알고리즘의 복잡도 분석

- 직접 구현하지 않고서도 수행 시간을 분석하는 것
- 알고리즘이 수행하는 연산의 횟수를 측정하여 비교
- 일반적으로 연산의 횟수는 n 의 함수
- 시간 복잡도 분석: 수행 시간 분석
- 공간 복잡도 분석: 수행 시 필요로 하는 메모리 공간 분석



수행시간측정

- clock() 함수 사용
 - clock_t clock(void);
 - clock 함수는 호출되었을 때의 시스템 시각을 CLOCKS_PER_SEC 단위로 반환
- 수행 시간을 측정하는 전형적인 프로그램

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void main( void )
{
    clock_t start, finish;
    double duration;
    start = clock();
    // 수행시간을 측정하고 하는 코드....
    // ....
    finish = clock();
    duration = (double)(finish - start) / CLOCKS_PER_SEC;
    printf("%f 초입니다.\n", duration);
}
```

알고리즘 효율성과 복잡도 분석

- 어떤 문제를 해결하는데 사용될 수 있는 알고리즘은?
 - 매우 다양함
- 다양한 알고리즘 중에서 우리는 어떤 것을 골라야 할까?
 - 효율적인 알고리즘
 - 그래서 알고리즘 분석이 필요! 즉, 알고리즘의 효율성을 분석해야 함!
- 알고리즘의 효율성 분석 방법은?
 - 알고리즘의 복잡도 분석(Complexity Analysis)
 - 직접 구현하지 않고 모든 입력을 고려하여 대략적으로 알고리즘 효율성을 비교하는 방법
 - 실행 하드웨어나 소프트웨어 환경과 관계없이 알고리즘의 효율성을 평가할 수 있음

시간 복잡도 분석(1)

■ 시간 복잡도 계산 방법

- 알고리즘을 이루고 있는 연산들이 몇 번이나 수행되는지를 숫자로 표
 - 알고리즘의 절대적인 실행 시간을 분석하는 것이 아님!
- 산술 연산, 대입 연산, 비교 연산, 이동 연산 등 기본적인 연산 고려
- 연산의 수행횟수는 고정된 숫자가 아니라 입력의 개수 n 에 대한 함수(시간 복잡도 함수)

■ 시간 복잡도 함수(time complexity function) $T(n)$

- n : 문제의 크기(입력자료의 개수)
- 알고리즘을 수행하는데 필요한 시간을 추정한 것

시간 복잡도 분석(2)

- 최악의 경우의 시간 복잡도 (worst-case time complexity)
 - 크기가 n 인 문제에 대한 알고리즘을 분석할 때, 그 알고리즘에 의해 수행되는 기본 연산의 회수를 최대로 하는 입력에 대한 시간복잡도 함수
 - D_n : 크기가 n 인 문제에 대한 모든 입력 집합
 - $I : D_n$ 의 원소
 - $t(I)$: 입력 I 에 대하여 수행하는 기본 연산 수
 - $T_W(n)$: 최악의 경우의 시간 복잡도, $T_W(n) = \max\{t(I) | I \in D_n\}$
- 평균적인 경우의 시간 복잡도(expected-case time complexity)
 - 알고리즘이 평균적으로 얼마 만큼의 기본 연산을 수행하는 가를 분석
 - $P(I)$: 입력 I 가 일어날 확률
 - $T_E(n)$: 평균적 경우의 시간 복잡도, $T_E(n) = \sum_{I \in D_n} p(I)t(I)$

복잡도 분석의 예

- n 을 n 번 더하는 문제: 각 알고리즘이 수행하는 연산의 개수를 세어 봄. 단, for 루프 제어 연산은 고려하지 않음.

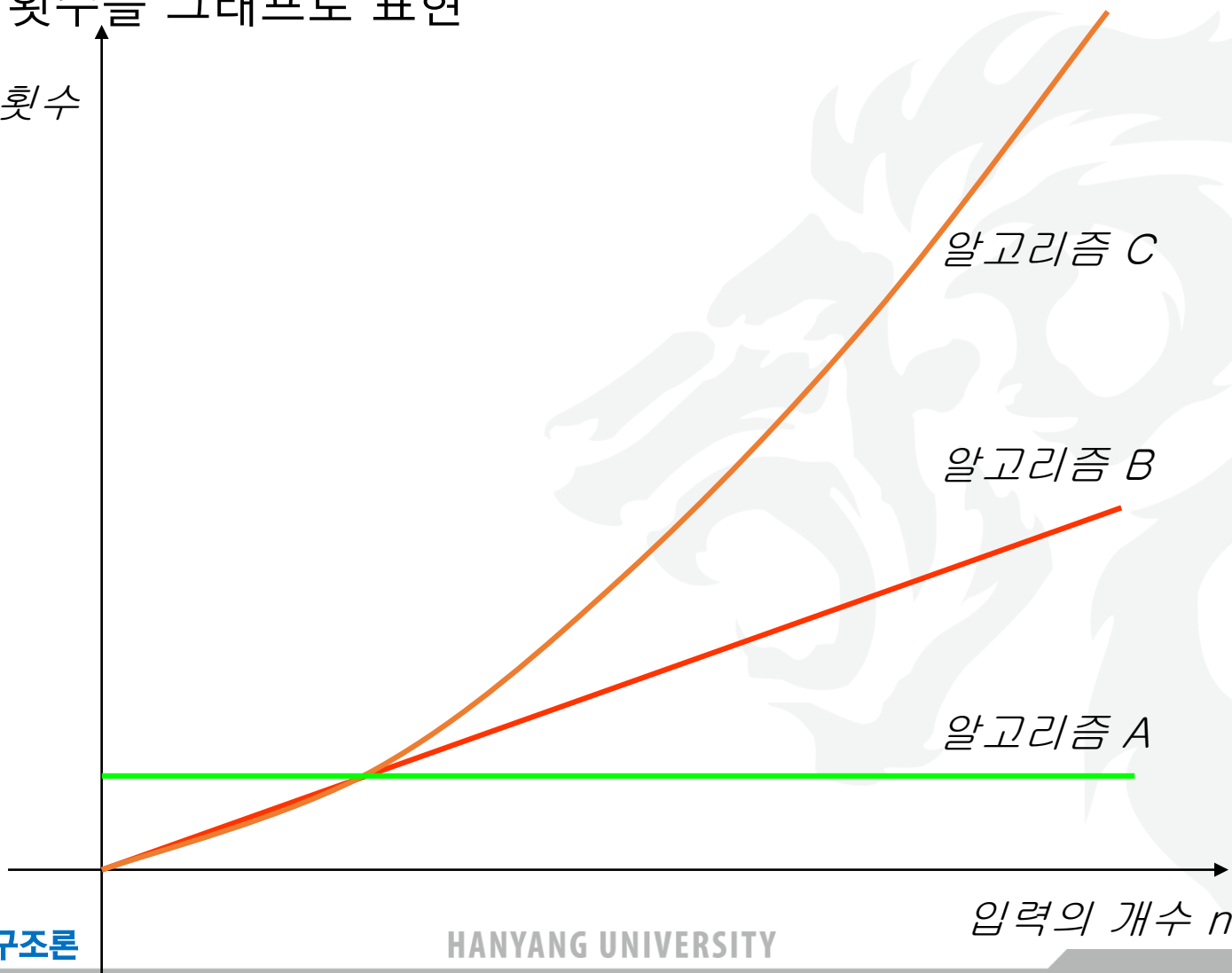
알고리즘 A	알고리즘 B	알고리즘 C
$\text{sum} \leftarrow n * n;$	$\text{sum} \leftarrow 0;$ $\text{for } i \leftarrow 1 \text{ to } n \text{ do}$ $\quad \text{sum} \leftarrow \text{sum} + n;$	$\text{sum} \leftarrow 0;$ $\text{for } i \leftarrow 1 \text{ to } n \text{ do}$ $\quad \text{for } j \leftarrow 1 \text{ to } n \text{ do}$ $\quad \quad \text{sum} \leftarrow \text{sum} + 1;$

	알고리즘 A	알고리즘 B	알고리즘 C
대입연산	1	$n + 1$	$n * n + 1$
덧셈연산		n	$n * n$
곱셈연산	1		
나눗셈연산			
전체연산수	2	$2n + 1$	$2n^2 + 1$

복잡도 분석의 예

- 연산 횟수를 그래프로 표현

연산의 횟수



알고리즘 C

알고리즘 B

알고리즘 A

입력의 개수 n

시간 복잡도 함수 계산 예

- 코드를 분석해보면 수행되는 연산들의 횟수를 입력 크기의 함수로 만들 수 있음

ArrayMax(A,n)

```
tmp ← A[0];  
for i ← 1 to n-1 do  
    if tmp < A[i] then  
        tmp ← A[i];  
return tmp;
```

1번의 대입 연산
루프 제어 연산은 제외
n-1번의 비교 연산
n-1번의 대입 연산(최대)
1번의 반환 연산

총 연산수 = $2n$ (최대)

빅오 표기법(1)

- 자료의 개수가 많은 경우에는 차수가 가장 큰 항이 가장 영향을 크게 미치고 다른 항들은 상대적으로 무시될 수 있음
 - (예) $n=1,000$ 일 때, $T(n)$ 의 값은 1,001,001이고 이중에서 첫 번째 항인 n^2 의 값이 전체의 약 99%인 1,000,000이고 두 번째 항의 값이 1000으로 전체의 약 1%를 차지함
 - 따라서 보통 시간 복잡도 함수에서 가장 영향을 크게 미치는 항만을 고려하면 충분함

$n=1000$ 인 경우

$$T(n) = n^2 + n + 1$$

입력의 개수 n

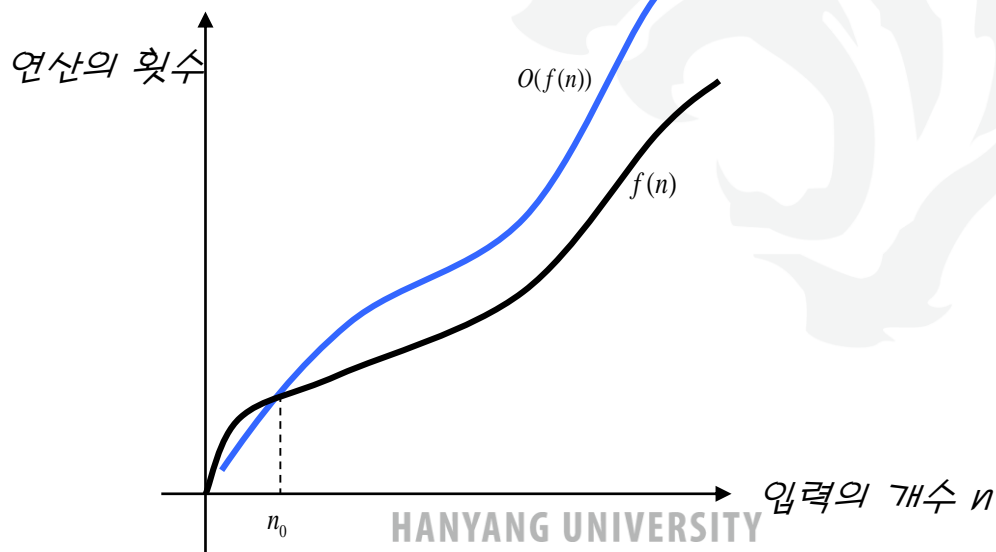
99%

1%

빅오 표기법(2)

■ 빅오(big-oh) 표기법: 연산의 횟수를 대략적(점근적)으로 표기한 것


- 두개의 함수 $f(n)$ 과 $g(n)$ 이 주어졌을 때,
모든 $n \geq n_0$ 에 대하여 $|f(n)| \leq c|g(n)|$ 을 만족하는 두개의 상수 c 와 n_0 가 존재하면 $f(n) = O(g(n))$
- "빅오"는 **함수의 상한**을 표시
 - (예) $n \geq 5$ 이면 $2n+1 < 10n$ 이므로 $2n+1 = O(n)$ ← Big-oh of n

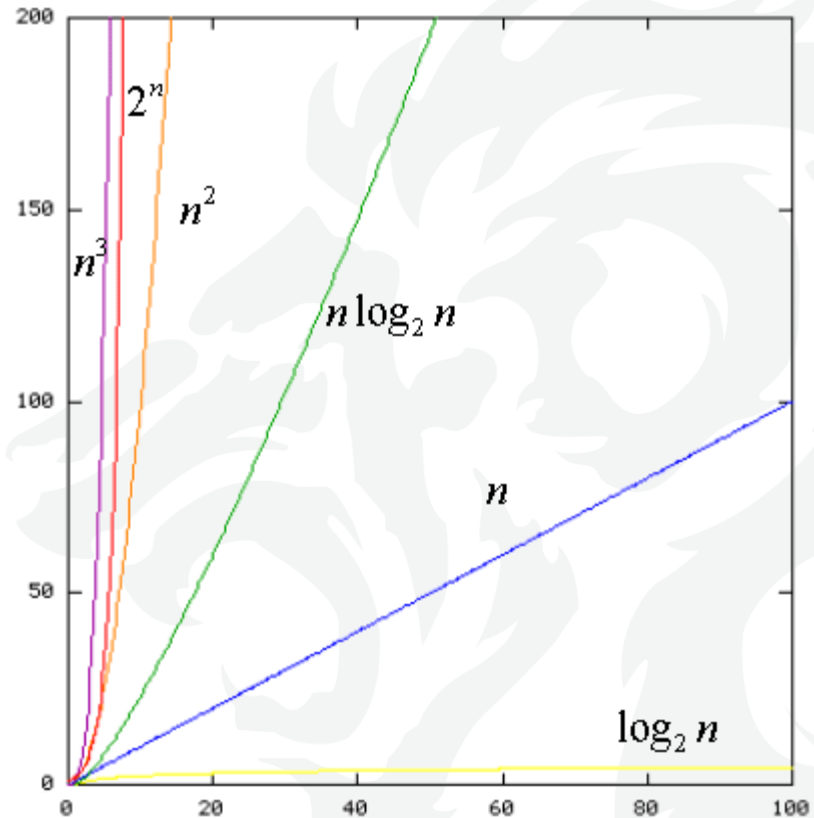


빅오 표기법 예제

- $f(n) = 5$ 이면 이다. 왜냐하면 $n_0 = 1$, $c = 10$ 일 때, $n \geq 1$ 에 대하여 $5 \leq 10 \cdot 1$ 이 되기 때문이다.
- $f(n) = 2n + 1$ 이면 이다. 왜냐하면 $n_0 = 2$, $c = 3$ 일 때, $n \geq 2$ 에 대하여 $2n + 1 \leq 3n$ 이 되기 때문이다.
- $f(n) = 3n^2 + 100$ 이면 이다. 왜냐하면 $n_0 = 100$, $c = 5$ 일 때, $n \geq 100$ 에 대하여 $3n^2 + 100 \leq 5n^2$ 이 되기 때문이다.
- $f(n) = 5 \cdot 2^n + 10n^2 + 100$ 이면 이다. 왜냐하면 $n_0 = 1000$, $c = 10$ 일 때, $n \geq 1000$ 에 대하여 $5 \cdot 2^n + 10n^2 + 100 \leq 10 \cdot 2^n$ 이 되기 때문이다.

빅오 표기법 종류

- 
- $O(1)$: 상수형
 - $O(\log n)$: 로그형
 - $O(n)$: 선형
 - $O(n \log n)$: 로그선형
 - $O(n^2)$: 2차형
 - $O(n^3)$: 3차형
 - $O(n^k)$: k차형
 - $O(2^n)$: 지수형
 - $O(n!)$: 팩토리얼형



함수 수행속도 비교(1)

시간복잡도	n					
	1	2	4	8	16	32
1	1	1	1	1	1	1
$\log n$	0	1	2	3	4	5
n	1	2	4	8	16	32
$n \log n$	0	2	8	24	64	160
n^2	1	4	16	64	256	1024
n^3	1	8	64	512	4096	32768
2^n	2	4	16	256	65536	4294967296
$n!$	1	2	24	40326	20922789888000	26313×10^{33}

함수 수행속도 비교(2)

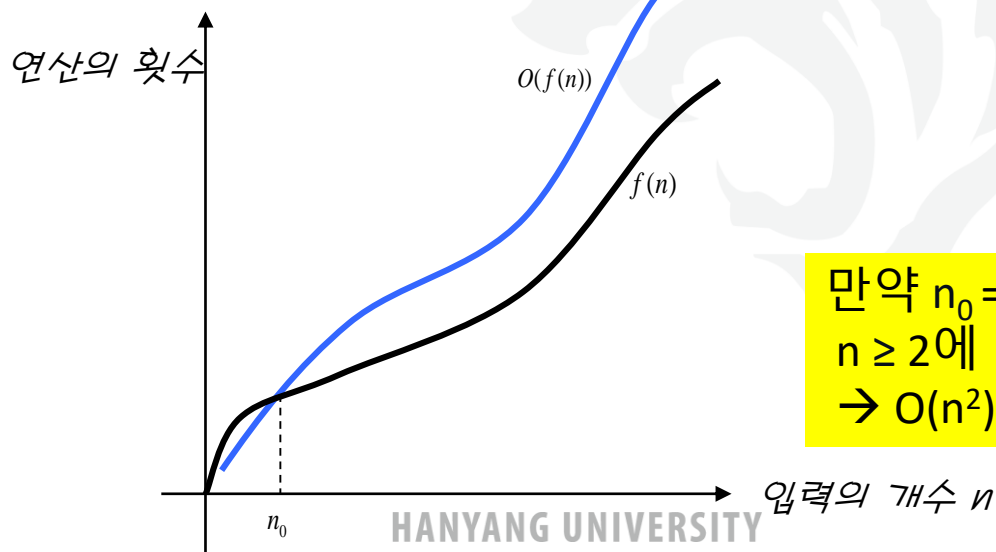
(단위 연산속도 : 10^{-6} 초)

입력 크기 n	알고리즘 복잡도	A	B	C	D	E
		$100n$	$10n\log_2 n$	$5n^2$	n^3	2^n
10		10^{-3} 초	1.5×10^{-3} 초	5×10^{-4} 초	10^{-3} 초	10^{-3} 초
100		10^{-2} 초	0.03 초	5×10^{-2} 초	1 초	4×10^{14} 세기
1,000		10^{-1} 초	0.45 초	5 초	1.6 분	***
10,000		1 초	6.1 초	8.3 분	11.57 일	***
100,000		10 초	1.5 분	13.8 시간	31.7 년	***

빅오 표기법 – 함수의 상한 표시

■ 빅오(big-oh) 표기법: 연산의 횟수를 대략적(점근적)으로 표기한 것

- 두개의 함수 $f(n)$ 과 $g(n)$ 이 주어졌을 때,
모든 $n \geq n_0$ 에 대하여 $|f(n)| \leq c|g(n)|$ 을 만족하는 두개의 상수 c 와 n_0 가 존재하면 $f(n) = O(g(n))$
- “빅오”는 **함수의 상한**을 표시
 - (예) $n \geq 5$ 이면 $2n+1 < 10n$ 이므로 $2n+1 = O(n)$ ← Big-oh of n



만약 $n_0=2$, $c=2$ 로 잡으면,
 $n \geq 2$ 에 대해서 $2n+1 \leq 2n^2$
 $\rightarrow O(n^2)$

빅오메가 표기법

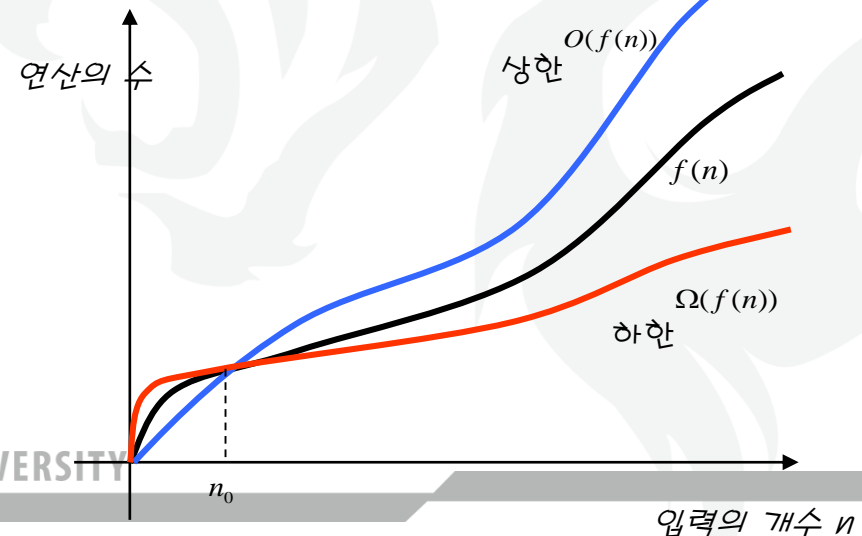
■ 빅오메가(big omega) 표기법

- 두개의 함수 $f(n)$ 과 $g(n)$ 이 주어졌을 때,
모든 $n \geq n_0$ 에 대하여 $|f(n)| \geq c|g(n)|$ 을 만족하는 두개의 상수 c 와 n_0 가 존재하면 $f(n) = \Omega(g(n))$
- “빅오메가”는 **함수의 하한**을 표시
 - (예) $n \geq 1$ 이면 $2n+1 \geq n$ 이므로 $2n+1 = \Omega(n)$

빅세타 표기법

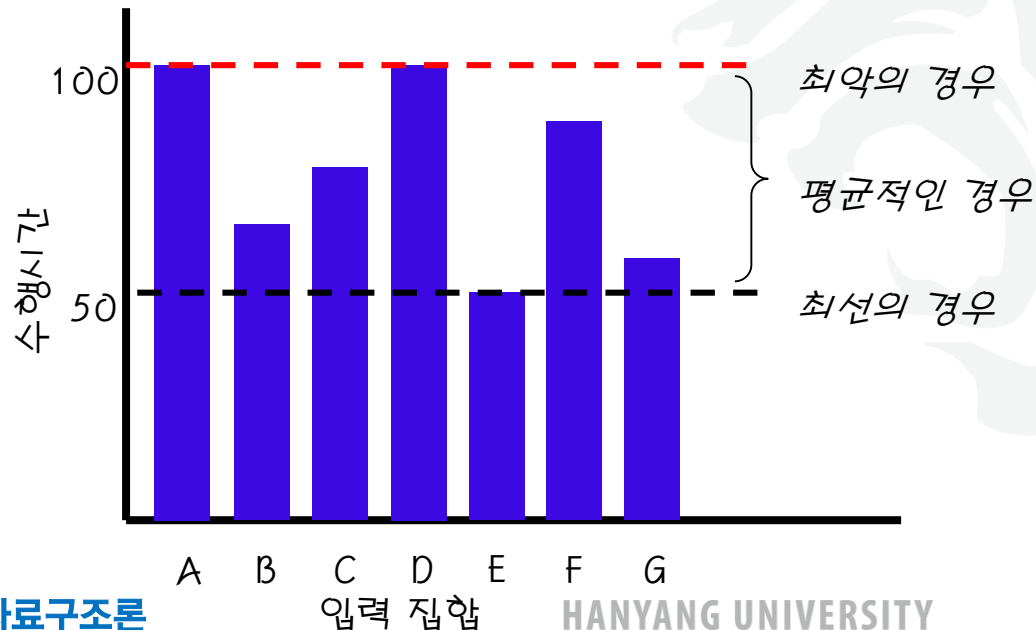
■ 빅세타(big theta) 표기법

- 두개의 함수 $f(n)$ 과 $g(n)$ 이 주어졌을 때,
모든 $n \geq n_0$ 에 대하여 $c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|$ 을 만족하는 세개의 상수 c_1, c_2 와 n_0 가 존재하면 $f(n) = \theta(g(n))$
- “빅세타”는 **함수의 하한인 동시에 상한**을 표시
- $f(n) = O(g(n))$ 이면서 $f(n) = \Omega(g(n))$ 이면, $f(n) = \theta(n)$
- (예) $n \geq 1$ 이면 $n \leq 2n+1 \leq 3n$ 이므로 $2n+1 = \theta(n)$



최선, 평균, 최악의 경우

- 알고리즘의 수행시간은 입력 자료 집합에 따라 다를 수 있음
 - (예) 정렬 알고리즘
- 최선의 경우(best case):** 수행 시간이 가장 빠른 경우
- 평균의 경우(average case):** 수행시간이 평균적인 경우
- 최악의 경우(worst case):** 수행 시간이 가장 늦은 경우



순차탐색 알고리즘 예

5	9	10	17	21	29	33	37	38	43
---	---	----	----	----	----	----	----	----	----

■ 최선의 경우

- 찾고자 하는 숫자가 맨앞에 있는 경우
 $\therefore O(1)$

■ 최악의 경우

- 찾고자 하는 숫자가 맨뒤에 있는 경우
 $\therefore O(n)$

■ 평균적인 경우

- 각 요소들이 균일하게 탐색된다고 가정
하면 모든 숫자들이 탐색되었을 경우의
비교 연산 수행횟수를 더한 후, 전체 숫
자 개수로 나누면 됨
 $(1+2+\dots+n)/n=(n+1)/2$
 $\therefore O(n)$

인덱스 0에서 값 5 발견
숫자 비교 횟수 = 1

5	9	10	17	21	29	33	37	38	43
0	1	2	3	4	5	6	7	8	9

인덱스 9에서 값 43 발견
숫자 비교 횟수 = 10

5	9	10	17	21	29	33	37	38	43
0	1	2	3	4	5	6	7	8	9

인덱스 5에서 값 26 발견
숫자 비교 횟수 = 6

2	7	16	19	20	26	35	42	46	50
0	1	2	3	4	5	6	7	8	9

Week 1: Algorithm

