



부산대학교

PUSAN NATIONAL UNIVERSITY

Asymptotic Analysis

Reasoning About Growth

Bounds, Cases, and Worked Analysis

DS3600082-140

자료구조

2026학년도 2학기

오용경

데이터사이언스전공

yongkyung.oh@pusan.ac.kr

Growth Rate

입력 크기가 커질 때 두 연산 수 함수의 차이를 비교한다

입력 크기와 연산 수의 증가율

증가율(growth rate)은 입력이 커질 때 연산 수가 늘어나는 양상을 설명한다

n	$T_A(n) = n$	$T_B(n) = n^2$
10	10	100
100	100	10,000
1,000	1,000	1,000,000

n 이 10배가 되면 T_A 는 10배, T_B 는 100배가 된다

다항식에서 가장 빠르게 커지는 항

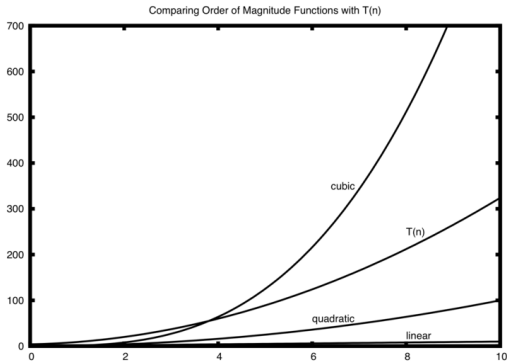
최고차항(dominant term)은 다항식에서 차수가 가장 높은 항이다

$$T(n) = 6n^3 + 20n^2 + 4n + 100$$

$$\frac{T(n)}{n^3} = 6 + \frac{20}{n} + \frac{4}{n^2} + \frac{100}{n^3} \rightarrow 6$$

n 이 커질수록 낮은 차수 항의 비중은 줄지만, 작은 입력에서는 계수와 낮은 차수 항도 영향을 준다

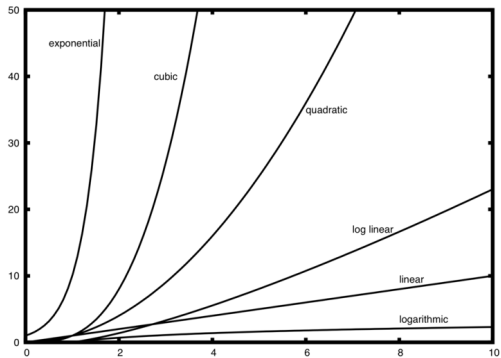
최고차항의 비중을 그래프로 확인한다



$T(n) = 3n^2 + 2n + 4$ 는 작은 n 에서 n^3 보다 크지만 n 이 커지면 n^3 에 추월되고, n^2 와 같은 차수로 커진다.

Source: Miller and Ranum (2013), Big-O Notation.

대표적인 함수의 증가를 비교한다



충분히 큰 n 에서 $1, \log n, n, n \log n, n^2, n^3, 2^n$ 순으로 빠르게 커진다.

Source: Miller and Ranum (2013), Big-O Notation.

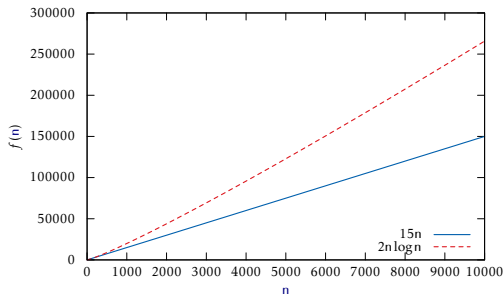
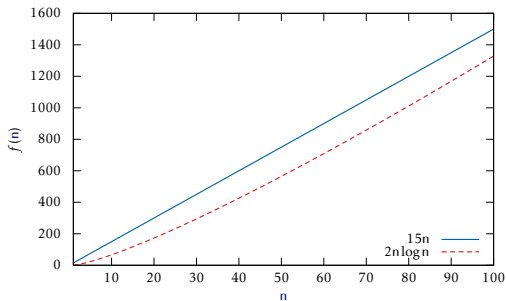
입력을 두 배로 늘리면

$T(2n)/T(n)$ 을 계산하면 입력 증가에 대한 연산 수의 변화를 읽을 수 있다

$T(n)$	$T(2n)$	비율
1	1	1
$\log_2 n$	$\log_2 n + 1$	1에 가까워짐
n	$2n$	2
$n \log_2 n$	$2n(\log_2 n + 1)$	2에 가까워짐
n^2	$4n^2$	4

작은 입력에서는 순서가 바뀔 수 있다

$15n$ 과 $2n\log_2 n$ 은 비교하는 입력 크기에 따라 대소 관계가 달라진다



$15 = 2\log_2 n$ 에서 같아지므로 교차점은 $n = 2^{7.5} \approx 181$ 이다.

Checkpoint

Growth Rate

- 입력을 두 배로 늘렸을 때 n 과 n^2 은 각각 몇 배가 되는가
- 다항식의 최고차항으로 큰 입력에서의 증가를 설명할 수 있는 이유는 무엇인가
- 입력이 작을 때와 클 때 두 함수의 대소 관계가 바뀔 수 있는가

Asymptotic Bounds

상한과 하한을 부등식으로 나타내고 성립하는 상수를 찾는다

정의

Big-O

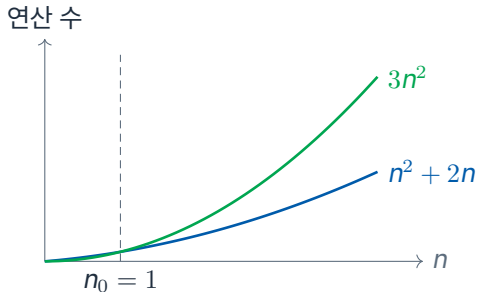
점근 표기(asymptotic notation)는 충분히 큰 입력에서 함수의 증가를 나타내는 표기다. Big-O에서 $f(n)$ 은 충분히 큰 n 에 대해 $c g(n)$ 보다 크지 않다.

$$f(n) = O(g(n)) \iff 0 \leq f(n) \leq c g(n) \quad (n \geq n_0)$$

이 부등식을 만족하는 상수 $c > 0$ 과 n_0 가 존재해야 한다. c 는 기준 함수를 늘리는 배율이고, n_0 는 부등식이 계속 성립하기 시작하는 입력 크기다.

상수 c 와 시작점 n_0 를 찾는다

$n^2 + 2n \leq 3n^2$ 는 모든 정수 $n \geq 1$ 에서 성립한다



$g(n) = n^2$ 로 두고 $c = 3$, $n_0 = 1$ 을 선택하면 Big-O의 정의를 만족한다.

가능한 상수의 쌍을 확인한다

$f(n) = 4n + 12$ 와 $g(n) = n$ 에 대해 부등식이 성립하는 상수의 쌍을 찾는다

c	n_0	확인할 부등식	결과
4	1	$4n + 12 \leq 4n$	성립하지 않음
5	12	$4n + 12 \leq 5n$	성립함
8	3	$4n + 12 \leq 8n$	성립함

정의의 요구 부등식이 성립하는 상수 한 쌍이면 충분하다. 가장 작은 c 와 n_0 를 찾을 필요는 없다

정의

Big-Omega

Ω 는 충분히 큰 입력에서 기준 함수의 상수 배로 하한을 정한다.

$$f(n) = \Omega(g(n)) \iff 0 \leq c g(n) \leq f(n) \quad (n \geq n_0)$$

이 부등식을 만족하는 상수 $c > 0$ 과 n_0 가 존재해야 한다.

$n \geq 1$ 에서 $n^2 \leq n^2 + 2n$ 이므로 $c = 1$, $n_0 = 1$ 은 $n^2 + 2n = \Omega(n^2)$ 의 근거가 된다

정의

Big-Theta

Θ 는 같은 기준 함수로 상한과 하한을 함께 나타낸다.

$$f(n) = \Theta(g(n)) \iff c_1 g(n) \leq f(n) \leq c_2 g(n) \quad (n \geq n_0)$$

적절한 양의 상수 c_1, c_2 와 시작점 n_0 가 존재해야 한다.

$n \geq 1$ 에서 $n^2 \leq n^2 + 2n \leq 3n^2$ 이므로 $n^2 + 2n = \Theta(n^2)$ 이다

Big-O의 포함 관계

$O(g(n))$ 는 부등식을 만족하는 함수들의 집합이며, $f(n) = O(g(n))$ 은 $f(n)$ 이 그 집합의 원소라는 뜻이다

함수	참인 상한	가장 구체적인 분류
$\log n$	$O(\log n), O(n), O(n^2)$	$\Theta(\log n)$
$3n + 2$	$O(n), O(n^2)$	$\Theta(n)$
$n^2 + 2n$	$O(n^2), O(n^3)$	$\Theta(n^2)$

$$O(\log n) \subset O(n) \subset O(n^2)$$

$3n + 2 = O(n^2)$ 도 참이지만, 상한과 하한을 함께 나타내는 표기는 $\Theta(n)$ 이다

Source: Morin (2013), Section 1.3.3.

부등식으로 참과 거짓을 판단한다

상한은 한쪽 부등식으로, Θ 는 상한과 하한의 두 부등식으로 판별한다

명제	참·거짓	확인할 근거
$3n + 2 = O(n)$	참	상수 c, n_0
$n = O(n^2)$	참	상한의 집합 관계
$n = \Theta(n^2)$	거짓	상한과 하한이 모두 성립하는가
$n^2 = \Omega(n)$	참	하한의 방향

n 은 $O(n^2)$ 이지만 n^2 의 상수 배를 하한으로 갖지 않으므로 $\Theta(n^2)$ 은 아니다

Checkpoint

Asymptotic Bounds

- Big-O를 보이려면 c 와 n_0 가 어떤 조건을 만족해야 하는가
- n 은 $O(n^2)$ 이지만 $\Theta(n^2)$ 은 아닌 이유는 무엇인가
- 같은 기준 함수로 상한과 하한이 모두 성립하면 어떤 기호를 쓰는가

Reading Code as a Function

코드에서 센 연산 수를 점근 표기로 나타낸다

상수 배가 달라도 점근 표기는 같다

상수 배가 달라도 같은 함수에 대한 Θ 표기가 성립한다

$$T_C(n) = n^2$$

$$T_C(n) = \Theta(n^2)$$

$$T_D(n) = 100n^2$$

$$T_D(n) = \Theta(n^2)$$

n	$T_C(n)$	$T_D(n)$	비율
10	100	10,000	100
1,000	1,000,000	100,000,000	100

Source: Miller and Ranum (2013), Big-O Notation.

로그의 밑은 상수 배만 바꾼다

1보다 큰 고정된 두 밑의 로그는 서로 상수 배 관계다

$$\log_a n = \frac{\log_b n}{\log_b a} \implies \log_a n = \Theta(\log_b n)$$

n	$\log_2 n$	$\log_{10} n$	비율
100	6.64	2	3.32
1,000,000	19.93	6	3.32

Source: UBC DSCI 221 (n.d.), The Hierarchy of Speed.

연속된 반복문의 비용

실행 블록	연산 수	표기
<pre>for x in data: total += x</pre>	an	$\Theta(n)$
<pre>for x in data: for y in data: compare(x,y)</pre>	bn^2	$\Theta(n^2)$
전체	$an + bn^2$	$\Theta(n^2)$

$n = \text{len}(\text{data})$ 이고, a 와 b 는 반복 한 번의 일정한 비용(양의 상수)이다.

두 블록의 비용을 더하면 $an + bn^2$ 이고 최고차항은 n^2 이다

반씩 줄이는 반복문

양의 정수 k 를 2로 나눈 몫으로 갱신하면 몇 번 만에 1이 되는가

```
k = n
steps = 0
while k > 1:
    k //= 2
    steps += 1
```

$n = 20$ 인 실행

- $20 \rightarrow 10 \rightarrow 5 \rightarrow 2 \rightarrow 1$
- 반복은 4회
- 일반적으로 $\lfloor \log_2 n \rfloor$ 회 ($n \geq 1$)
- 한 번의 비용이 일정하면 $\Theta(\log n)$

입력 크기가 둘일 때

두 입력 길이가 독립적이면 하나를 다른 하나로 바꾸지 않는다

반복 구조	연산 수	Θ 표기
목록 A의 원소를 하나씩 처리	m	$\Theta(m)$
목록 B의 원소를 하나씩 처리	n	$\Theta(n)$
A와 B의 모든 쌍	mn	$\Theta(mn)$
두 반복을 연속 실행	$m + n$	$\Theta(m + n)$

Caution m 과 n 의 관계가 주어지지 않으면 mn 을 n^2 으로 바꾸지 않는다

세 반복 구조의 증가 차수를 비교한다

반복 구조	연산 수	표기
원소를 한 번씩 처리	n	$\Theta(n)$
모든 순서쌍을 처리	n^2	$\Theta(n^2)$
크기를 절반씩 줄임	$\lfloor \log_2 n \rfloor$	$\Theta(\log n)$

Source: Miller and Ranum (2013), Algorithm Analysis Exercises.

Checkpoint

Reading Code as a Function

- 연산 수가 n^2 과 $n(n-1)/2$ 인 두 반복문을 모두 $\Theta(n^2)$ 으로 쓰는 이유는 무엇인가
- 크기를 절반으로 줄이는 반복의 연산 수에 로그가 나타나는 이유는 무엇인가
- 독립적인 입력 크기 m 과 n 을 하나의 변수로 바꾸면 어떤 정보가 사라지는가

Best, Worst, and Average

입력의 배치에 따라 달라지는 비용과 그 조건을 구분한다

같은 크기의 입력도 작업량이 다르다

정의

경우별 비용

목록 [7, 3, 9, 1, 8, 2, 6]에서 원소를 앞에서부터 찾으면 찾는 값의 위치에 따라 비교 수가 달라진다.
최선의 경우(best case)는 같은 크기의 입력 중 연산 수가 가장 작은 경우다.
최악의 경우(worst case)는 연산 수가 가장 큰 경우다.

찾는 값	비교 수	경우
7	1회	최선
6 또는 없는 값 5	7회	최악

Source: Morin (2013), Section 1.5.

평균을 구하려면 입력 분포가 필요하다

평균 경우(average case)는 입력별 연산 수에 그 입력의 확률을 곱해 더한 기댓값이다

가정 찾는 값은 반드시 하나 있고, n 개 위치에 있을 확률은 각각 $1/n$ 이다

$$\mathbb{E}[T(n)] = \frac{1}{n}(1 + 2 + \cdots + n) = \frac{n+1}{2} = \Theta(n)$$

같은 코드라도 입력 분포가 달라지면 평균 비교 수도 달라진다

값을 찾으면 바로 반환한다

```
def contains(data, target):  
    for value in data:  
        if value == target:  
            return True  
    return False
```

비교 수

입력 상태	비교 수
첫 원소 일치	1
가운데 원소 일치	약 $n/2$
마지막 원소 일치	n
찾는 값 없음	n

이 코드는 최선의 경우가 $\Theta(1)$ 이고 최악의 경우가 $\Theta(n)$ 이다

최악의 경우가 보장하는 것

최악의 경우의 연산 수는 같은 크기의 어떤 입력도 넘지 않는 상한이다

목록	찾는 값	비교 수	누적 최대
[5, 2, 9, 1]	5	1	1
[5, 2, 9, 1]	9	3	3
[5, 2, 9, 1]	1	4	4
[5, 2, 9, 1]	7	4	4

Big-O는 함수의 상한을 나타내는 기호이고, 최악의 경우는 비교할 입력을 고르는 기준이다

조건문의 비용은 선택된 경로로 결정된다

조건 검사 비용이 일정하고 두 경로가 모두 가능한 경우를 생각한다

```
1 if flag:
2     for i in range(n): work()
3 else:
4     for i in range(n):
5         for j in range(n): work()
```

$$T_{\text{worst}}(n) = 1 + \max\{n, n^2\} = \Theta(n^2)$$

조건	실행되는 분기	조건 검사를 포함한 비용
참	n 번 수행	$1 + n$
거짓	n^2 번 수행	$1 + n^2$

Caution 두 분기 비용을 $n + n^2$ 로 합산하지 않는다. 조건 검사 자체가 반복을 실행하면 그 비용은 포함한다

Checkpoint

Best, Worst, and Average

- 앞에서부터 값을 찾는 코드에서 비교가 1회인 입력과 n 회인 입력은 무엇인가
- 평균 비교 수를 $(n+1)/2$ 로 계산할 때 어떤 입력 분포를 가정했는가
- Big-O가 항상 최악의 경우를 뜻하는 것은 아닌 이유는 무엇인가

Section 5 • Asymptotic Analysis

Time and Space

시간과 추가 저장 공간을 함께 비교한다

시간과 추가 저장 공간을 함께 센다

정의

공간 복잡도

시간 복잡도(time complexity)는 입력 크기에 따라 연산 수가 늘어나는 정도다.

공간 복잡도(space complexity)는 입력 크기에 따라 필요한 저장 공간이 늘어나는 정도다. 여기서는 입력 저장 공간을 제외한 **추가 공간**을 센다.

코드	동작	시간	추가 공간
<code>b = a</code>	같은 목록을 가리킴	$\Theta(1)$	$\Theta(1)$
<code>b = list(a)</code>	참조 n 개를 새 목록에 담음	$\Theta(n)$	$\Theta(n)$

`list(a)`는 원소 객체 자체가 아니라 원소를 가리키는 참조를 새 목록에 복사한다.

Source: Morin (2013), Section 1.5.

문자 빈도로 두 단어를 비교한다

애너그램은 문자의 순서만 바꾸어 만들 수 있는 단어다: heart와 earth

```
def same_letters(a, b):  
    if len(a) != len(b):  
        return False  
    counts = [0] * 26  
    for ch in a:  
        counts[ord(ch) - ord('a')] += 1  
    for ch in b:  
        counts[ord(ch) - ord('a')] -= 1  
    return all(v == 0 for v in counts)
```

비교 절차

- 입력은 영어 소문자 26종
- 첫 문자열에서 더하고 둘째 문자열에서 뺀
- 모든 칸이 0이면 문자별 개수 일치
- 시간 $\Theta(n)$, 추가 공간 $\Theta(1)$

같은 문제를 푸는 네 가지 방법

같은 길이 n 의 소문자 문자열에서 작업과 추가 공간을 비교한다

방법	시간	추가 공간	조건
하나씩 찾아 지우기	$\Theta(n^2)$	$\Theta(n)$	복사한 목록, 최악의 경우
정렬해서 비교	$O(n \log n)$	구현에 따라 다름	비교 정렬의 상한
26칸에 빈도 세기	$\Theta(n)$	$\Theta(1)$	문자 종류 26개 고정
모든 순서 생성·비교	$O(n n!)$	$O(n)$	한 후보씩 생성

마지막 방법은 최대 $n!$ 개 후보를 만들고 후보당 최대 n 개 문자를 비교한 상한이다.

Python 연산의 비용은 조건과 함께 읽는다

정수 원소와 정수 키의 연산을 예로 든다

연산	list	dict
위치 또는 키로 읽기	$O(1)$	평균 $O(1)$
in으로 존재 확인	$O(n)$	평균 $O(1)$
모든 항목 처리	$O(n)$	$O(n)$
전체 복사	$O(n)$	$O(n)$
끝에서 삭제 pop()	$O(1)$	해당 없음
앞에서 삭제 pop(0)	$O(n)$	해당 없음

Caution dict의 평균 $O(1)$ 은 모든 입력에 대한 보장이 아니다. 키의 분포가 나쁘면 한 번의 조회가 $O(n)$ 까지 늘어날 수 있다

Source: Miller and Ranum (2013), Lists and Dictionaries.

리스트 이어 붙이기에는 길이에 비례한 일이 든다

```
result = []  
for value in data:  
    result = result + [value]
```

반복 시점	새 목록 길이	옮기는 참조 수
첫 번째 원소	1	1
i 번째 원소	i	i
마지막 원소	n	n

$1 + 2 + \dots + n = n(n+1)/2$ 이므로 전체 시간은 $\Theta(n^2)$ 이다

비용을 말하려면 기준과 조건이 필요하다

점근 표기에는 세는 연산, 입력 크기, 입력 조건을 함께 밝힌다

질문	목록에서 확인할 것	설명 예
입력 크기는 무엇인가	원소 수 n	길이 n 인 목록
어떤 연산을 세는가	비교, 복사, 조회	원소 비교 횟수
어떤 경우인가	최선, 평균, 최악	찾는 값이 없는 최악의 경우
추가 공간은 얼마인가	입력 외에 만든 저장소	길이 n 의 새 목록
어떤 분포의 평균인가	입력·키의 분포	해시 조회의 평균 비용

시간 복잡도 하나만 적으면 입력 조건과 공간 비용이 빠질 수 있다

Checkpoint

Time and Space

- 문자 종류를 26개로 고정했을 때 빈도표의 추가 공간이 $\Theta(1)$ 인 이유는 무엇인가
- dict의 평균 $O(1)$ 을 모든 입력에 대한 보장으로 읽으면 안 되는 이유는 무엇인가
- 원소를 하나씩 이어 붙여 길이 n 인 리스트를 만들 때 총 시간이 $\Theta(n^2)$ 이 되는 이유는 무엇인가

Why Store Items Contiguously?

Next Lecture: Arrays and Indexed Access

- 원소를 연속해서 저장하는 배열
- 인덱스로 한 번에 읽는 접근
- 가운데 삽입에서 원소를 옮기는 비용