알고리즘의 성능

1. 수행 시간이란?
2. 알고리즘의 복잡도

1. 알고리즘의 수행 시간이란?

알고리즘의 수행 시간

입력의 크기 n에 대해 시간이 얼마나 걸리는지로 표현한다

입력의 크기는 대부분 자명함

- 정렬: 정렬할 원소의 수
- 색인: 색인에 포함된 원소의 수
- …

알고리즘 수행 시간의 예

```
sample1(A[], n):
    k ← n/2
    return A[k]
```

상수 시간

```
sample2(A[], n):
    sum ← 0
    for i ← 0 to n-1
        sum ← sum + A[i]
    return sum
```

n에 비례하는 시간

```
sample3(A[], n):
    sum ← 0
    for i ← 0 to n-1
        for j ← 0 to n-1
            sum ← sum + A[i] * A[j]
    return sum
```

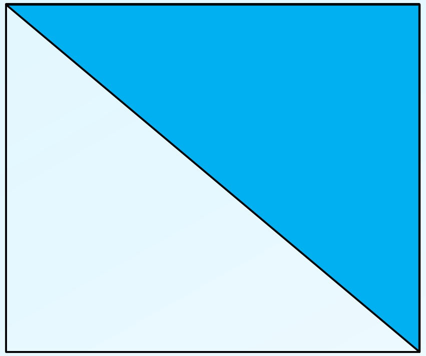
n^2에 비례하는 시간

```
sample4(A[], n):
    sum ← 0
    for i ← 0 to n-1
        for j ← 0 to n-1
            k ← A[0...n-1]에서 임의로 n/2개를 뽑은 것들 중 최댓값
            sum ← sum + k
    return sum
```

n^3에 비례하는 시간

```
sample5(A[], n):
    sum ← 0
    for i ← 0 to n-1
        for j ← i to n-1
            sum ← sum + A[i] * A[j]
    return sum
```

n^2에 비례하는 시간



```
factorial(n):
    if (n = 1) return 1
    return n* factorial(n-1)
```

n에 비례하는 시간

2. 알고리즘의 복잡도

알고리즘 점근적 복잡도 표현

점근적 복잡도Asymptotic Complexity

입력의 크기가 충분히 클 때의 복잡도

직관적 맛보기

입력의 크기: n

알고리즘이 기껏해야 n^2에 비례하는 시간이 든다 ⟶ $O(n^2)$

알고리즘이 적어도 n^2에 비례하는 시간이 든다 ⟶ $\Omega(n^2)$

알고리즘이 항상 n^2에 비례하는 시간이 든다 ⟶ $\Theta(n^2)$

점근적 복잡도

O-표기

$O(n^2)$: 최고차항의 차수가 n^2을 넘지 않는 모든 함수의 집합

Ω-표기

$\Omega(n^2)$: 최고차항의 차수가 n^2보다 작지 않은 모든 함수의 집합

Θ-표기

$\Theta(n^2)$: 최고차항의 차수가 n^2인 모든 함수의 집합

O-표기

$O(f(\mathrm{n}))$: 빅 오 Big Oh

- 최고차항의 차수가 $f(n)$보다 작거나 같은 모든 함수
- 점근적 상한
- 예, $O(n)$, $O(n \log n)$, $O(n^2)$, $O(2^n)$, …
- $O(n^2) = \{n^2, 5n^2 + 7\mathrm{n} + 12, 100n^2 - 2\mathrm{n} + 9, n \log n + 5n, 3n + 9, 150, \ldots\}$
  - 최고차항의 차수가 2차 이하인 모든 함수
- $5n^2 + 7\mathrm{n} + 12 \in O(n^2)$ 으로 표기해야 하나 편의상 $5n^2 + 7\mathrm{n} + 12 = O(n^2)$ 으로 표기함

Ω-표기

$\Omega(f(\mathrm{n}))$: 빅 오메가Big Omega

- 최고차항의 차수가 $f(n)$보다 크거나 같은 모든 함수
- 점근적 하한
- 예, $\Omega(n)$, $\Omega(n \log n)$, $\Omega(n^2)$, $\Omega(2^n)$, …
- $\Omega(n^2) = \{n^2, 5n^2 + 7\mathrm{n} + 12, 100n^2 - 2\mathrm{n} + 9, n^3 + 5n, 7n^5 + n^3 + 9, 2^n, n!, \ldots\}$

  - 최고차항의 차수가 2차 이상인 모든 함수
- $5n^3 + 12 \in \Omega(n^2)$ 으로 표기해야 하나 편의상 $5n^3 + 12 = \Omega(n^2)$으로 표기함

Θ-표기

$\Theta(f(\mathrm{n}))$: 빅 세타 Big Theta

- 최고차항의 차수가 $f(n)$과 동일한 모든 함수
- 예, $\Theta(n)$, $\Theta(n \log n)$, $\Theta(n^2)$, $\Theta(2^n)$, …
- $\Theta(n^2) = \{n^2, 5n^2 + 7\mathrm{n} + 12, 100n^2 - 2\mathrm{n} + 9, \ldots\}$
- 최고차항의 차수가 2차인 모든 함수
- $5n^2 + 12 \in \Theta(n^2)$ 으로 표기해야 하나 편의상 $5n^2 + 12 = \Theta(n^2)$ 으로 표기함

표기법 사용 예

```
sample4(A[], n):
    sum ← 0
    for i ← 0 to n-1
        for j ← 0 to n-1
            k ← A[0...n-1]에서 임의로 n/2개를 뽑은 것들 중 최댓값
            sum ← sum + k
    return sum
```

알고리즘 sample4()의 수행 시간은 $\Theta(n^3)$이다

수학적 정의

$$O(g(n)) = \{\, f(n) \mid \exists c > 0,\ n_0 > 1 \text{ s.t. } \forall n \geq n_0,\ f(n) \leq cg(n) \,\}$$

조금 풀어서 표현하면

$O(g(n))$ = { $f(n)$| 충분히 큰 모든 n에 대해 $f(n) \leq cg(n)$인 양의 상수 c가 존재한다}

✓ 점근적 복잡도를 수학적으로 증명할 때는(알고리즘 과목) 이런 방식의 정의가 필요하지만 자료구조 과목 수준에서는 앞의 직관적 정의로 충분하다

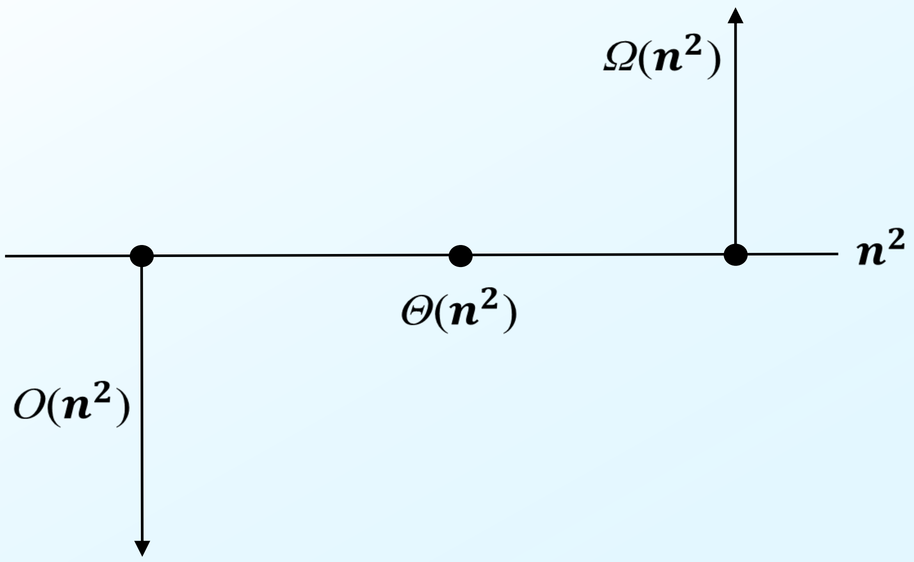
점근적 표기에서 =는 ∈의 대용

어떤 알고리즘의 수행 시간 T(n) = O(nlogn)이라 하면
T(n) ∈ O(nlogn)이란 뜻이다.

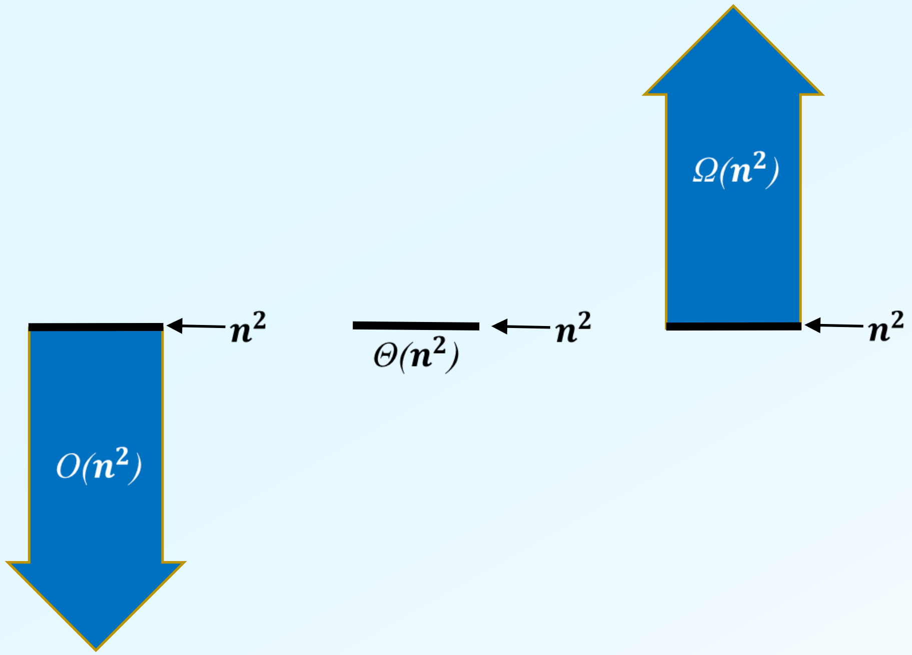
즉, T(n)은 O(nlogn)에 속하는 함수란 뜻이다.

그러므로, O(nlogn) = T(n)과 같은 표현은 적절하지 않다.

참고: 시각적 의미



$\Omega(\boldsymbol{n}^2)$
$\boldsymbol{n}^2$
$\Theta(\boldsymbol{n}^2)$
$O(\boldsymbol{n}^2)$



$\Omega(\boldsymbol{n}^2)$
$\boldsymbol{n}^2$
$\boldsymbol{n}^2$
$\Theta(\boldsymbol{n}^2)$
$\boldsymbol{n}^2$
$O(\boldsymbol{n}^2)$

O, Ω, Θ 표기법 사용해 보기

```
alg1(A[], n):
    k ← n/2
    return A[k]
```

n에 관계없이 상수 시간이 소요된다
Ω(1)이기도 하고 O(1)이기도 하다
Θ(1): 가장 정확한 표현

```
alg2(A[], n):
    sum ← 0
    for i ← 0 to n-1
        sum ← sum + A[i]
    return sum
```

항상 n에 비례하는 시간이 소요된다
Ω(n)이기도 하고 O(n)이기도 하다
Θ(n): 가장 정확한 표현

```
alg3(A[], n) :
    sum ← 0
    if (n이 홀수)
        for i ← 0 to n-1
            sum ← sum + A[i]
    return sum
```

최악의 경우 n에 비례하는 시간이 소요된다
최선의 경우 상수 시간이 소요된다
Ω(1)이고 O(n)이다
최선의 경우 Θ(1), 최악의 경우 Θ(n)이라고 말한다

```
alg4(A[], n):
    sum ← 0
    for i ← 0 to n-2
        for j ← i+1 to n-1
            sum ← sum + A[i] * A[j]
    return sum
```

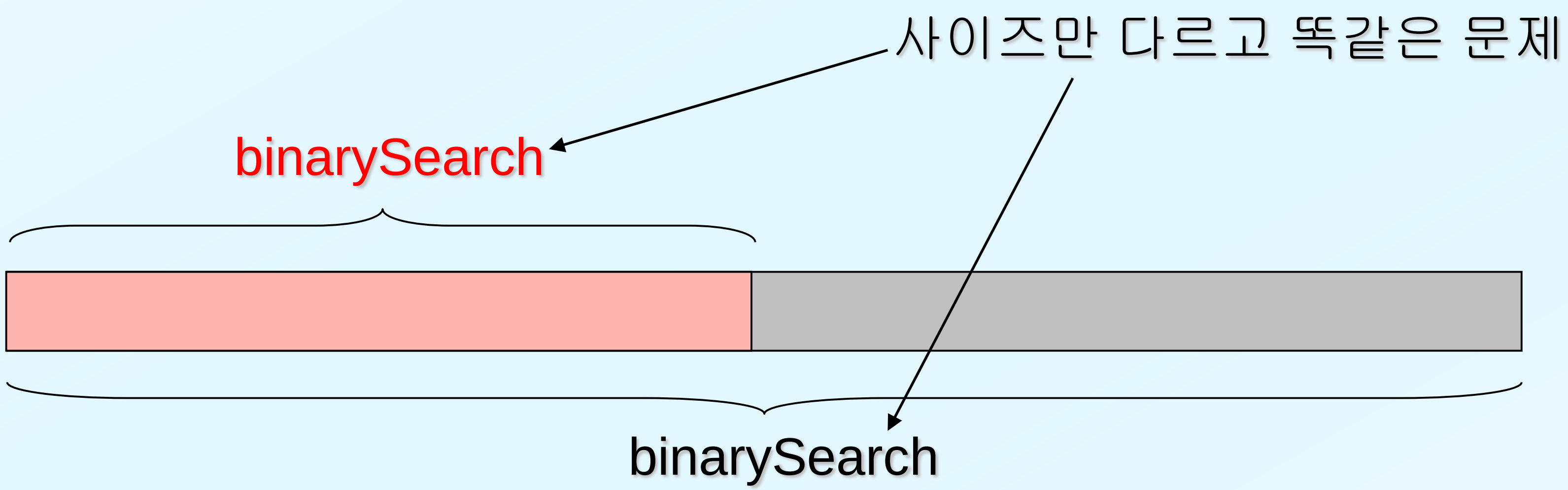
항상 n^2에 비례하는 시간이 소요된다
$\Omega(n^2)$이기도 하고 $O(n^2)$이기도 하다
$\Theta(n^2)$: 가장 정확한 표현

재귀 알고리즘의 복잡도 분석 예

```
binarySearch(A[], x, low, high) :
        // A: array,  x: search key,  low, high: array bounds
        if (low > high) return "Not found"
        mid ← (low + high)/2
        if (A[mid] < x) return binarySearch(A, x, mid+1, high)
        else if (A[mid] > x) return binarySearch(A, x, low, mid-1)
        else return mid
```

배열의 중앙에 있는 원소와 비교하고 나면
자신과 똑같지만 크기가 반이 되는 문제를
만난다. 즉, $T(n) = c + T(n/2)$
이런 식으로 크기를 반씩 줄여나가면
$\sim\log_2 n$번만에 크기 1인 문제를 만나게 된다.
문제의 크기를 반으로 줄이는데 필요한 작업은
상수 시간(c)이므로 최대 $\log_2 n$에 비례하는 시간에 끝난다.

수행 시간:
- 최악의 경우 $\Theta(\log n)$,
- 최선의 경우 $\Theta(1)$,
- 앞에 수식어 없이 그냥 말하면 $O(\log n)$



$T(n) = c + T(n/2)$

c가 거슬리면..

$T(n) = \Theta(1) + T(n/2)$