제1장

알고리즘의 분석: 시간복잡도

알고리즘의 분석

- 알고리즘의 **자원(resource)** 사용량을 분석
- 자원이란 실행 시간, 메모리, 저장장치, 통신 등
- 여기서는 **실행시간의 분석**에 대해서 다룸

시간복잡도(time complexity)

- 실행시간은 실행환경에 따라 달라짐
  - 하드웨어, 운영체제, 언어, 컴파일러 등
- 실행 시간을 측정하는 대신 연산의 실행 횟수를 카운트
- 연산의 실행 횟수는 입력 데이터의 크기에 관한 함수로 표현
- 데이터의 크기가 같더라도 실제 데이터에 따라서 달라짐
  - 최악의 경우 시간복잡도 (worst-case analysis)
  - 평균 시간복잡도 (average-case analysis)

점근적(Asymptotic) 분석

- **점근적 표기법을 사용**
  - 데이터의 개수 n→∞일때 수행시간이 증가하는 growth rate로 시간복잡도를 표현하는 기법
  - Θ-표기, O-표기 등을 사용
- **유일한 분석법도 아니고 가장 좋은 분석법도 아님**
  - 다만 (상대적으로) 가장 간단하며
  - 알고리즘의 실행환경에 비의존적임
  - 그래서 가장 광범위하게 사용됨

점근적 분석의 예: 상수 시간복잡도

입력으로 n개의 데이터가 저장된 배열 data가 주어지고,
그 중 n/2번째 데이터를 반환한다.

```
int sample( int data[], int n )
{
   int k = n/2 ;
   return data[k] ;
}
```

n에 관계없이 상수 시간이 소요된다.
이 경우 알고리즘의 시간복잡도는 O(1)이다.

점근적 분석의 예: 선형 시간복잡도

입력으로 n개의 데이터가 저장된 배열 data가 주어지고,
그 합을 구하여 반환한다.

```
int sum(int data[], int n)
{
	int sum = 0 ;
	for (int i = 0; i < n; i++)
		sum = sum + data[i];
	return sum;
}
```

이 알고리즘에서 가장 자주 실행되는 문장이며,
실행 횟수는 항상 n번이다.
가장 자주 실행되는 문장의 실행횟수가 n번이라면
모든 문장의 실행 횟수의 합은 n에 선형적으로 비례하며,
모든 연산들의 실행횟수의 합도 역시 n에 선형적으로 비례한다.

선형 시간복잡도를 가진다고 말하고
O(n)이라고 표기한다.

선형 시간복잡도: 순차탐색

배열 data에 정수 target이 있는지 검색한다.

```
int search(int n, int data[], int target)
{
     for (int i=0; i<n; i++) {
       if (data[i] == target)
         return i;
     }
     return -1;
}
```

이 알고리즘에서 가장 자주 실행되는 문장이며,
실행 횟수는 최악의 경우 n번이다.

최악의 경우 시간복잡도는 O(n)이다.

Quadratic

배열 x에 중복된 원소가 있는지 검사하는 함수이다.

```
bool is_distinct( int n, int x[] )
{
  for (int i=0; i<n-1; i++)
    for (int j=i+1; j<n; j++)
      if (x[i]==x[j])
        return false;
  return true;
}
```

이 알고리즘에서 가장 자주 실행되는 문장이며,
최악의 경우 실행 횟수는 n(n-1)/2번이다.

최악의 경우 배열에 저장된 모든 원소 쌍을 비교 하므로
비교 연산의 횟수는 n(n-1)/2이다.
최악의 경우 시간복잡도는 $O(n^2)$으로 나타낸다.

?

```
for (i=1; i<n; i*=2)
{
      // Do something
}
```

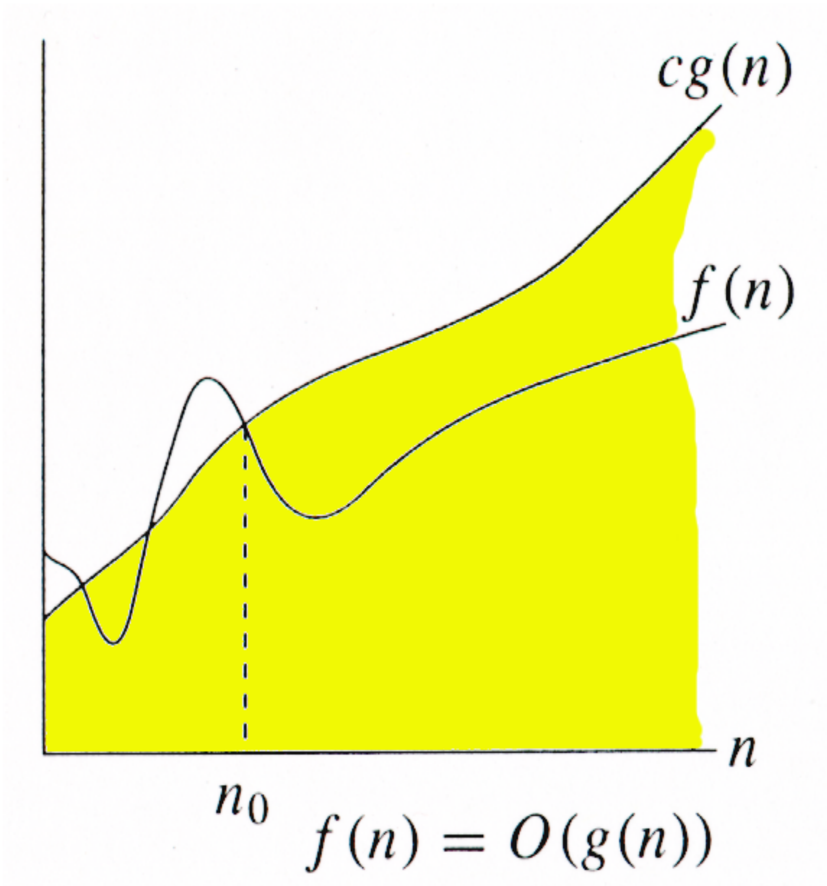
이 문장의 실행 횟수는?

점근적 표기법

- 알고리즘에 포함된 연산들의 실행 횟수를 표기하는 하나의 기법
- 최고차항의 차수만으로 표시
- 따라서 가장 자주 실행되는 연산 혹은 문장의 실행횟수를 고려하는 것으로 충분

점근표기법: O-표기

$$f(n) \in O(g(n)) \text{ if there exist constant } c > 0 \text{ and } n_0 \geq 0$$
$$\text{such that for all } n \geq n_0 \text{ we have } f(n) \leq cg(n)$$

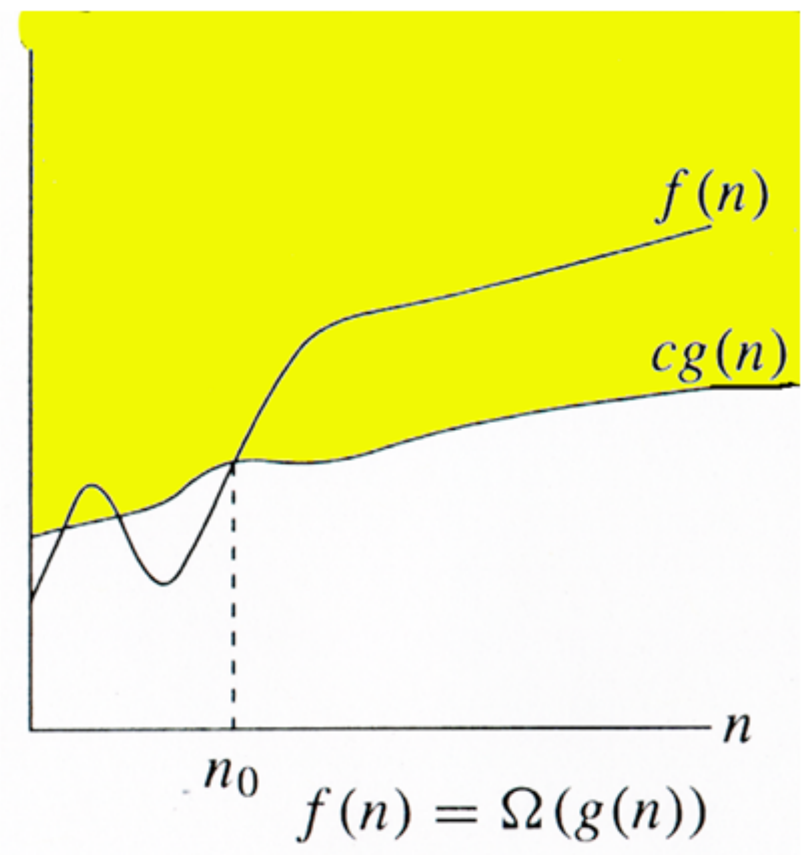


$$f(n) = 32n^2 + 17n \; - 32$$
$$f(n) \in O(n^2)$$
$$\text{but also } f(n) \in O(n^3) \text{ and } O(n^2 \log n)$$

upper bound를 표현

점근표기법: Ω-표기

$$f(n) \in \Omega(g(n)) \text{ if there exist constant } c > 0 \text{ and } n_0 \geq 0$$
$$\text{such that for all } n \geq n_0 \text{ we have } f(n) \geq cg(n).$$



$$f(n) = 32n^2 + 17n \;\; - 32$$
$$f(n) \in \Omega(n^2)$$
$$\text{but also } f(n) \in \Omega(n) \text{ and } \Omega(n \log n)$$

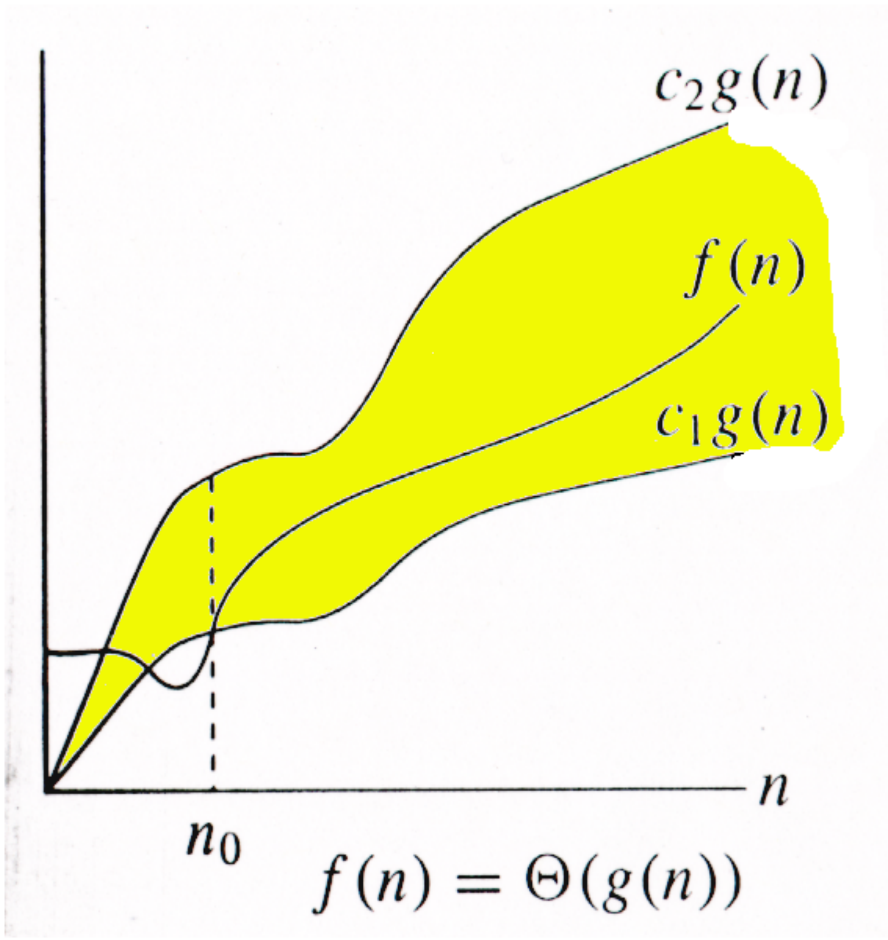
lower bound를 표현

점근표기법:Θ-표기

$$f(n) \in \Theta(g(n)) \text{ if there exist constants } c_1 > 0, c_2 > 0, \text{ and } n_0 \geq 0$$
$$\text{such that for all } n \geq n_0 \text{ we have } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n).$$

or

$$f(n) \in \Theta(g(n)) \text{ if } f(n) \in O(g(n)) \text{ and } f(n) \in \Omega(g(n))$$



$$f(n) = 32n^2 + 17n \ - 32$$
$$f(n) \in \Theta(n^2)$$
$$\text{but } f(n) \notin \Theta(n^3), f(n) \notin \Theta(n)$$

upper bound와
lower bound를 동시에 표현

- $f(n) \in O(g(n))$을 $f(n) = O(g(n))$으로 쓰는 경우가 많음
- 차수가 $k \geq 0$인 모든 다항식은 $O(n^k)$이다.

$$
\begin{aligned}
f(n) &= c_k n^k + c_{k-1} n^{k-1} + \cdots + c_1 n + c_0 \\
&= O(n^k)
\end{aligned}
$$

- 차수가 p인 다항식과 q인 다항식의 합은 $O(n^{\max\{p,q\}})$ 이다.

$$
\text{If } g(n) = O(n^p) \text{ and } h(n) = O(n^q),
$$
$$
\text{then } f(n) = g(n) + h(n) = O(n^{\max(p,q)})
$$

Θ-표기에 대해서도 성립함

Exercise

- 다음 테이블을 YES 혹은 NO로 채워라.

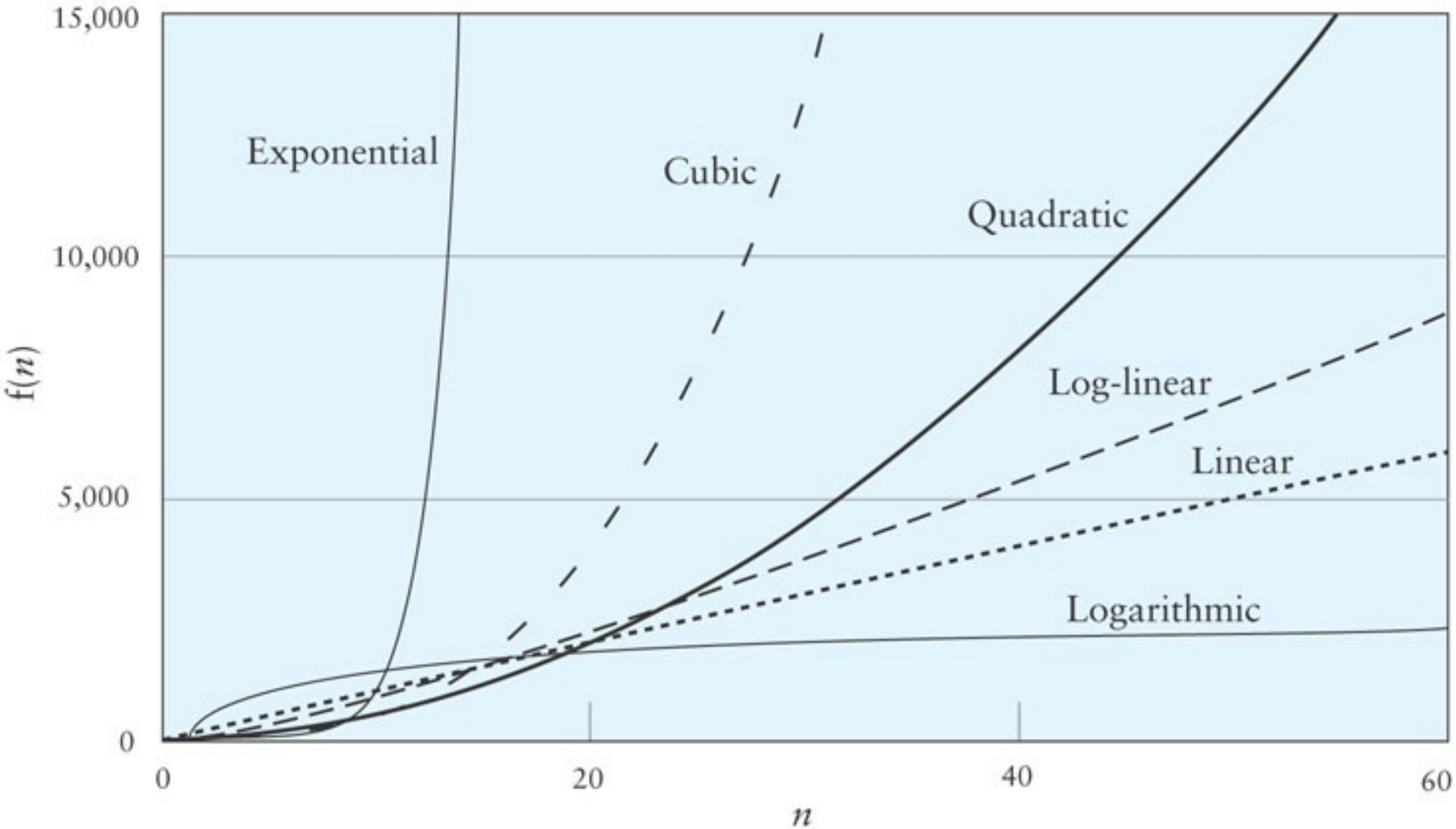
A	B	$A = O(B)?$	$A = \Theta(B)?$
$\log^k n$	n^ϵ		
n^k	c^n		
$\sqrt{n}$	$n^{\sin n}$		
2^n	$2^{n/2}$		
$n^{\log c}$	$c^{\log n}$		
$\log(n!)$	$\log(n^n)$		

A와 B는 n에 관한 함수. 나머지는 상수들

Common Growth Rate

Big-O	Name
$O(1)$	Constant
$O(\log n)$	Logarithmic
$O(n)$	Linear
$O(n \log n)$	Log-linear
$O(n^2)$	Quadratic
$O(n^3)$	Cubic
$O(2^n)$	Exponential
$O(n!)$	Factorial

Common Growth Rate



Common Growth Rate

O(f(n))	f(50)	f(100)	f(100)/f(50)
O(1)	1	1	1
O(log n)	5.64	6.64	1.18
O(n)	50	100	2
O(n log n)	282	664	2.35
O(n^2)	2500	10,000	4
O(n^3)	12,500	100,000	8
O(2^n)	1.126×10^{15}	1.27×10^{30}	1.126×10^{15}
O(n!)	3.0×10^{64}	9.3×10^{157}	3.1×10^{93}

다항시간 (polynomial-time) 알고리즘

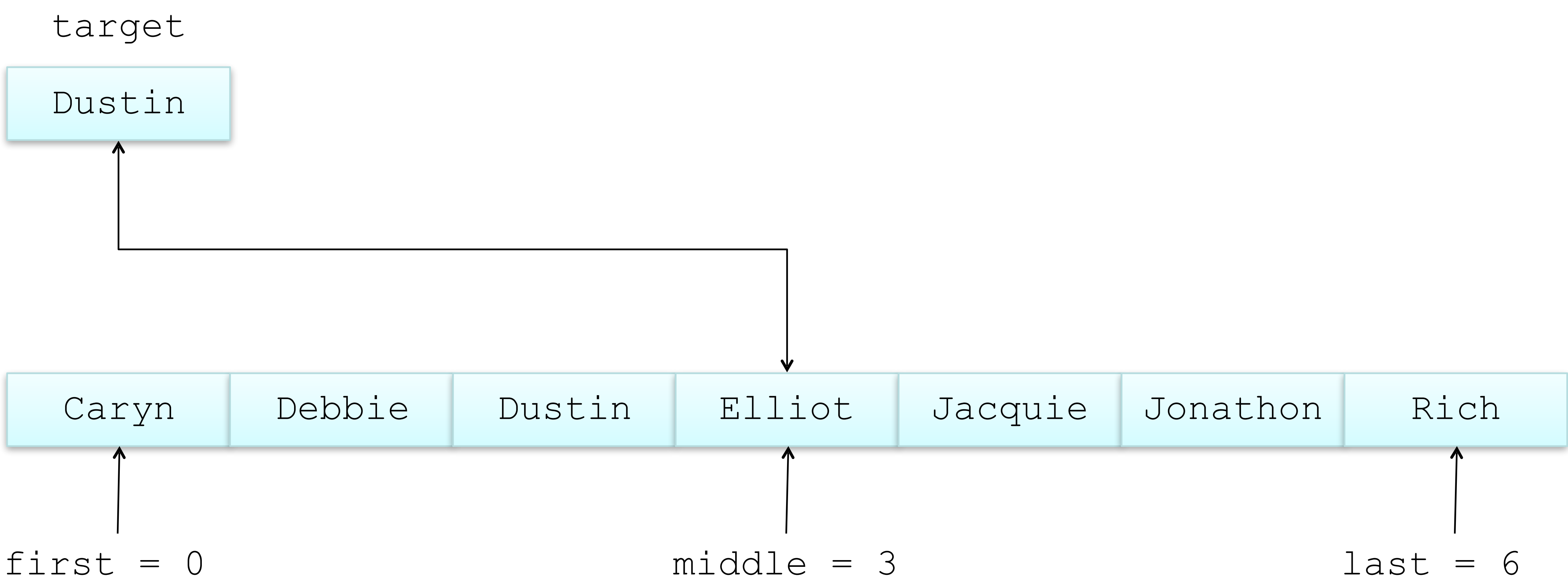
- An algorithm is efficient if its running time is polynomial.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

이진검색과 정렬

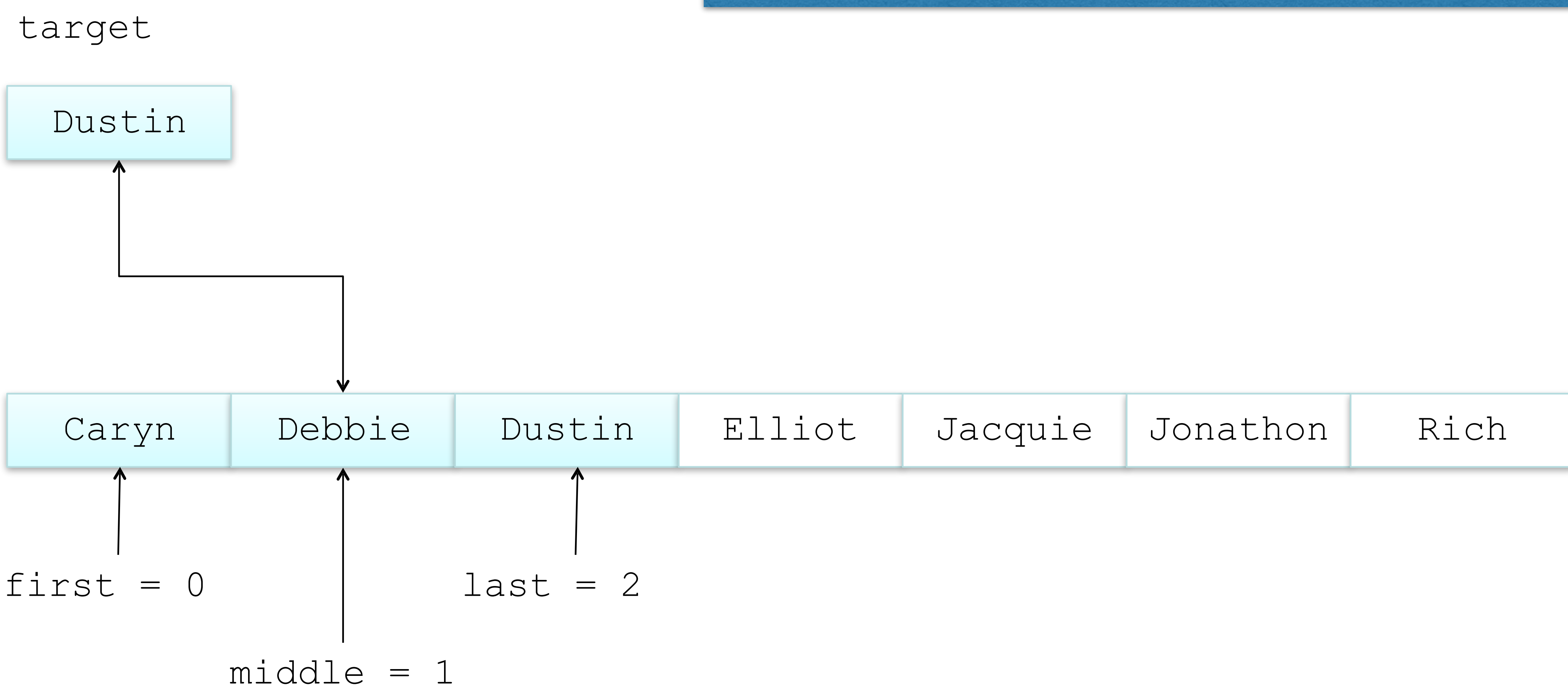
이진검색 (Binary Search)

배열에 데이터들이 오름차순으로 정렬되어 저장되어 있다.



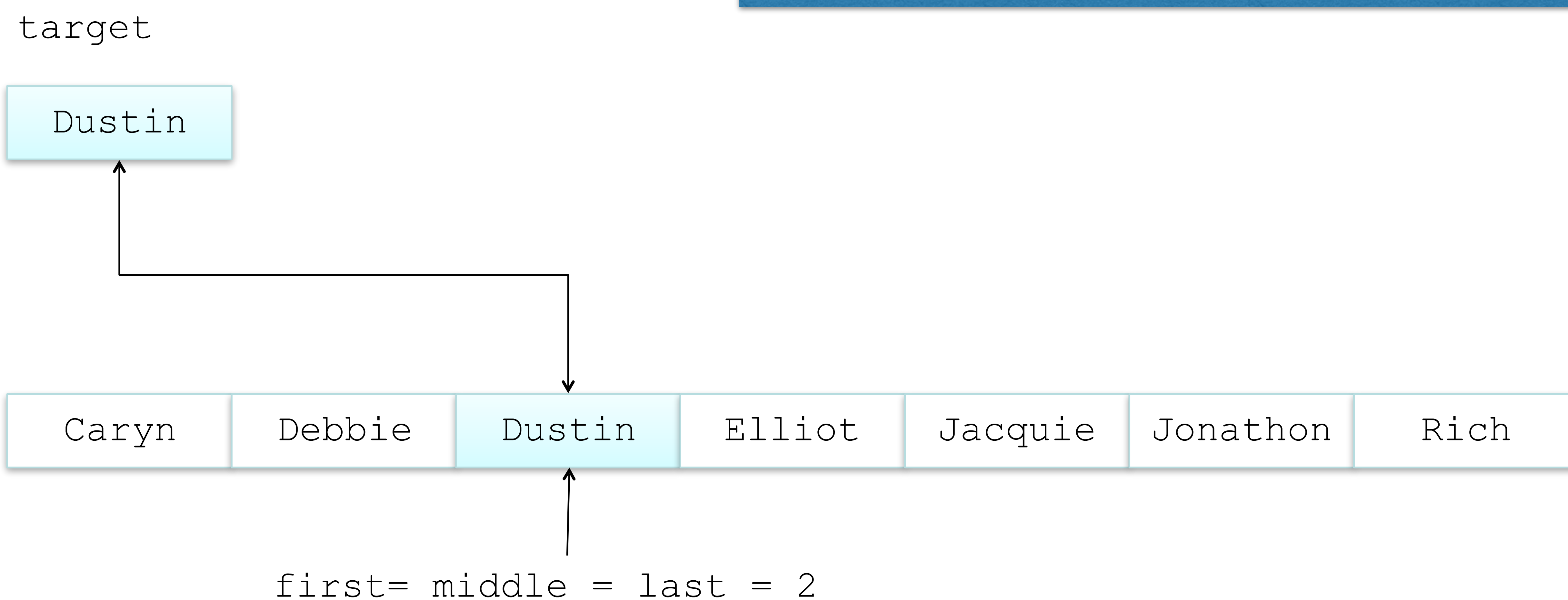
이진검색 (Binary Search)

배열에 데이터들이 오름차순으로 정렬되어 저장되어 있다.



이진검색 (Binary Search)

배열에 데이터들이 오름차순으로 정렬되어 저장되어 있다.



이진검색

배열 data에 n개의 문자열이 오름차순으로 정렬되어 있다.

```
int binarySearch(int n, char *data[], char *target) {
    int begin = 0, end = n-1;
    while(begin <= end)   {
        int middle = (begin + end)/2;
        int result = strcmp(data[middle], target);
        if (result == 0)
            return middle;
        else if (result < 0)
            begin = middle+1;
        else
            end = middle-1;
    }
    return -1;
}
```

한 번 비교할 때마다 남아있는 데이터가 절반으로 줄어든다.
따라서 시간복잡도는 $O(\log_2 n)$이다.

이진검색

- **데이터가 연결리스트에 오름차순으로 정렬되어 있다면?**
  - 연결리스트에서는 가운데(middle) 데이터를 O(1)시간에 읽을 수 없음
  - 따라서 이진검색을 할 수 없다.
- **순차검색의 시간복잡도는 O(n)이고 이진검색은 $O(\log_2 n)$이다. 둘의 차이는 매우 크다.**

버블 정렬 (bubble sort)

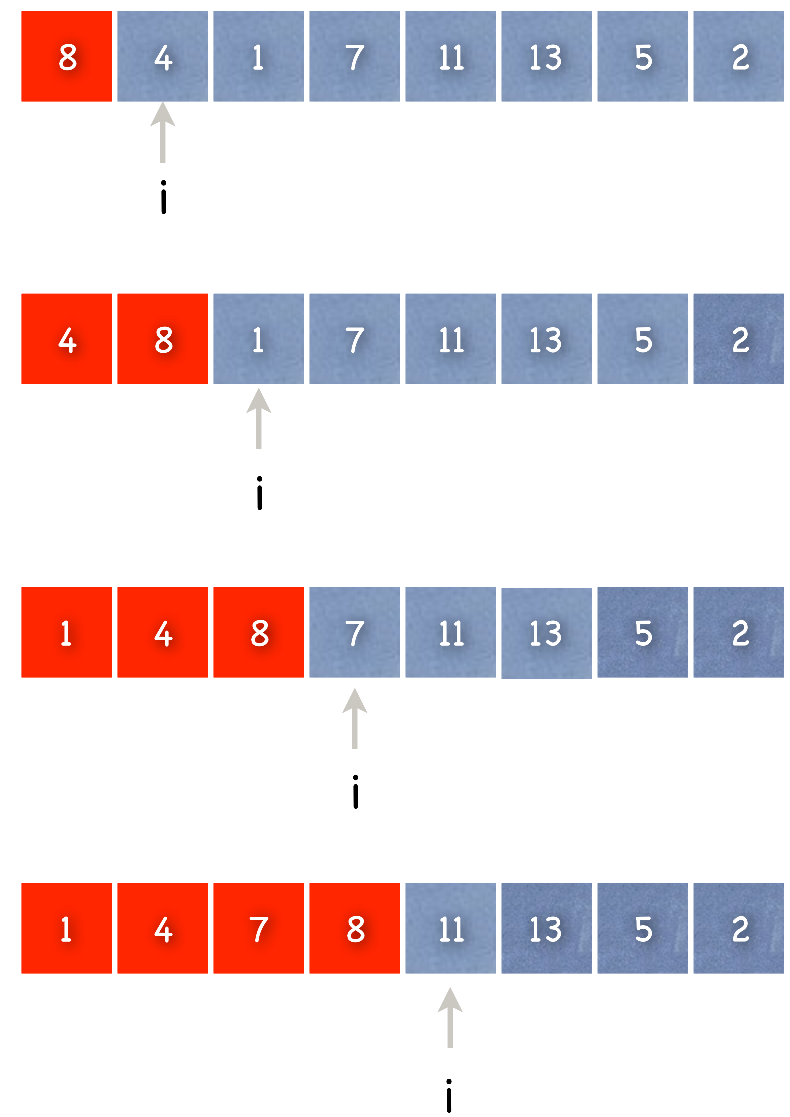
```
void bubbleSort(int data[], int n) {
    for ( int i=n-1; i>0; i--) {
        for ( int j=0; j<i; j++ ) {
            if (data[j] > data[j+1]) {
                int tmp = data[j];
                data[j] = data[j+1];
                data[j+1] = tmp;
            }
        }
    }
}
```

시간복잡도는 ?

삽입정렬 (Insertion sort)

```
void insertion_sort(int n, int data[])  {
    for ( int i=1; i<n; i++) {
        int tmp = data[i];
        int j = i-1;
        while (j>=0 && data[j]>data[i]) {
            data[j+1] = data[j];
            j–;
        }
        data[j+1] = tmp;
    }
}
```

data[i]를 data[0] ~ data[i-1] 중에 제자리를 찾아 삽입하는 일



시간복잡도는 ?

다른 정렬 알고리즘

- **퀵소트(quicksort) 알고리즘**
  - 최악의 경우 $O(n^2)$, 하지만 평균 시간복잡도는 $O(n\log_2 n)$
- **최악의 경우 $O(n\log_2 n)$의 시간복잡도를 가지는 정렬 알고리즘**
  - 합병정렬(merge sort)
  - 힙 정렬(heap sort) 등
- **데이터가 배열이 아닌 연결리스트에 저장되어 있다면?**