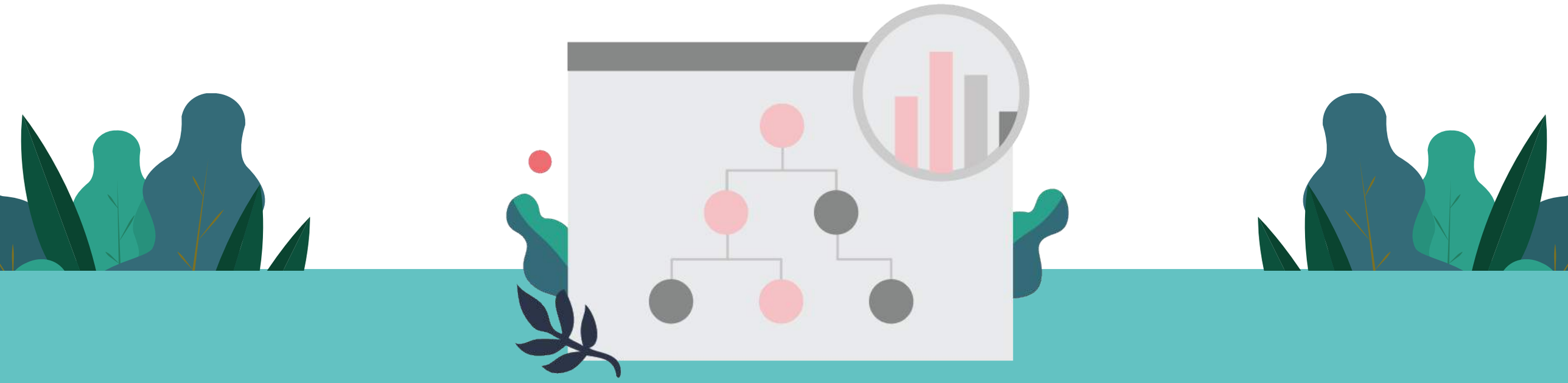
/* 데이터 구조 및 알고리즘 */

신현규 강사, 화/목 20:00

시간복잡도 (2)



/* elice */

주차별 커리큘럼

1주차	과정 소개, 배열, 연결리스트, 클래스
2주차	스택, 큐, 해싱
3주차	**시간복잡도**
4주차	트리, 트리순회, 재귀호출
5주차	힙
6주차	그래프 소개, DFS
7주차	그래프 심화, BFS
8주차	강의 요약, 알고리즘 과정 소개

컴퓨터를 이용한 문제 해결 과정

1. 문제를 정확히 이해한다
2. 문제를 해결하는 알고리즘을 개발한다
3. 알고리즘이 **문제를 해결한다는 것을 증명한다**
4. 알고리즘이 **제한시간 내에 동작한다는 것을 보인다**
5. 알고리즘을 코드로 작성한다
6. 제출 후 만점을 받고 매우 기뻐한다

시간복잡도 (Time Complexity)

알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

시간복잡도 (Time Complexity)

알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

```
sum = 0

for i in range(n) :
   sum = sum + i
```

시간복잡도 (Time Complexity)

알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

```
sum = 0

for i in range(n) :
   sum = sum + i
```

n 개

시간복잡도 (Time Complexity)

알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

```
sum = 0

for i in range(n) :
   sum = sum + i
```

O(n)

시간복잡도 (Time Complexity)

알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

```
sum = 0

for i in range(n) :
   for j in range(n) :
      sum = sum + i + j
```

시간복잡도 (Time Complexity)

알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

```
sum = 0

for i in range(n) :
   for j in range(n) :
      sum = sum + i + j
```

O(n²)

시간복잡도 (Time Complexity)

알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

```
sum = 0

for i in range(n) :
   for j in range(i) :
      sum = sum + i + j
```

시간복잡도 (Time Complexity)

알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

```
sum = 0

for i in range(n) :
   for j in range(i) :
      sum = sum + i + j
```

O(1/2 * n²)

시간복잡도 (Time Complexity)

알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

```
sum = 0

for i in range(n) :
   for j in range(i) :
      sum = sum + i + j
```

$O(n^2)$

시간복잡도 (Time Complexity)

알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

```
def findNumber(myList, target) :

   for v in myList :
      if v == target :
         return True

   return False
```

시간복잡도 (Time Complexity)

알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

```
def findNumber(myList, target) :

  for v in myList :
    if v == target :
      return True

  return False
```

O(n)

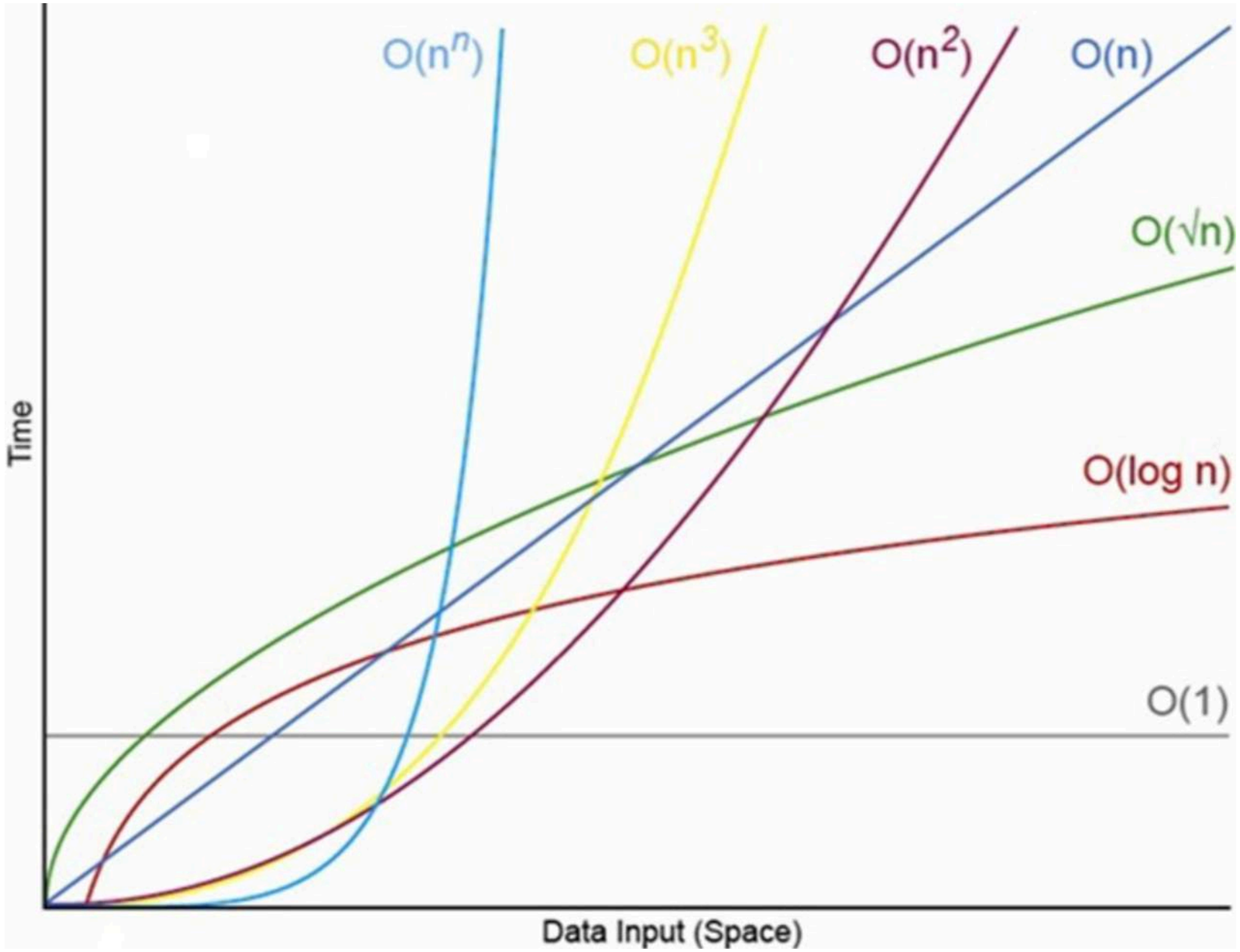
시간복잡도 (Time Complexity)

알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

Big-O 표기 : 최악의 경우에 수행하는 명령 수

시간복잡도 (Time Complexity)



https://apelbaum.wordpress.com/2011/05/05/big-o/

시간복잡도와 실제 수행 시간

많은 명령을 수행한다 = 오래 걸린다

시간복잡도와 실제 수행 시간

많은 명령을 수행한다 = 오래 걸린다

몇 개의 명령을 수행해야 1초가 걸리는가 ?

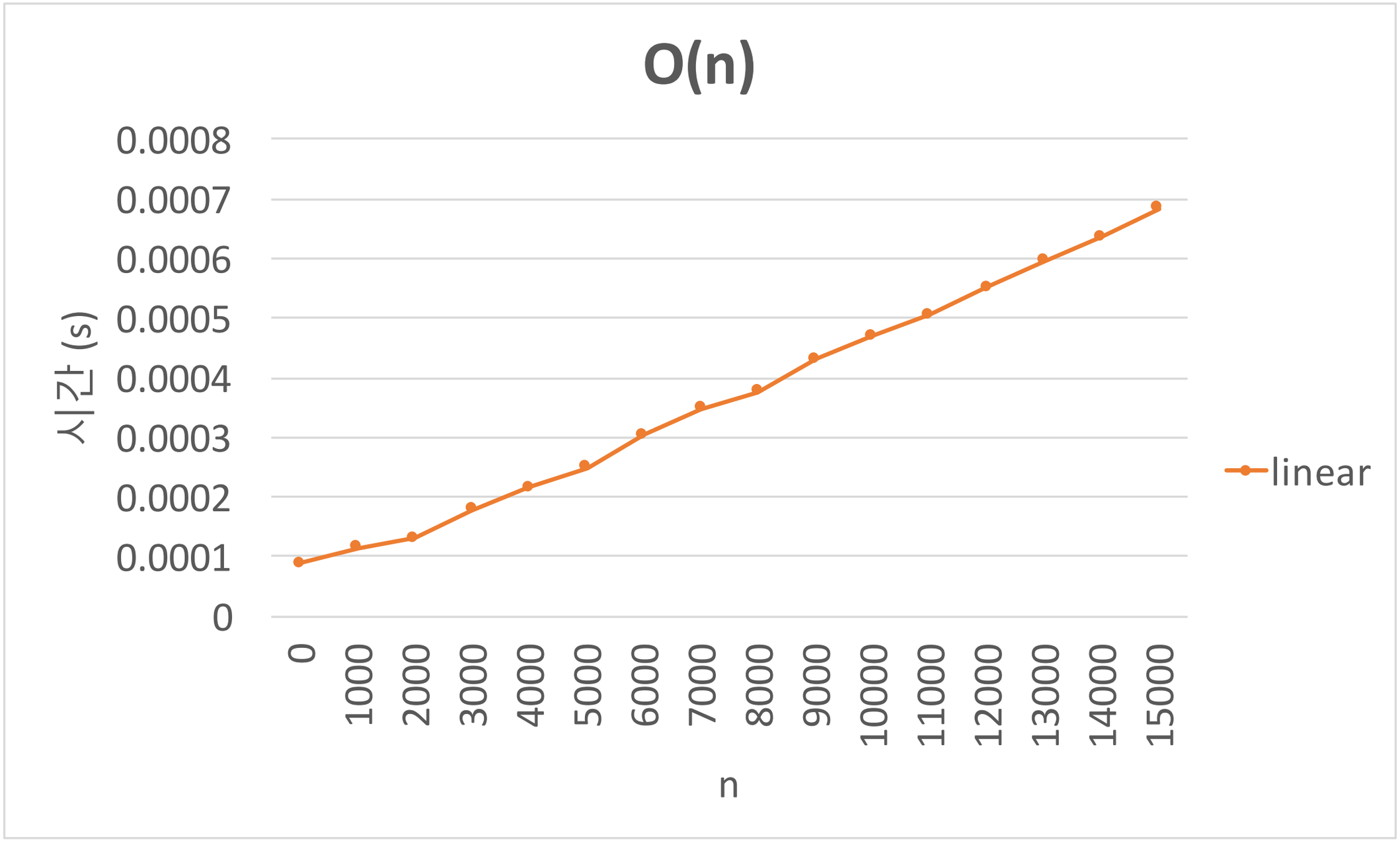
실험

아래 코드에 대하여 Elice에서 수행 시간을 측정

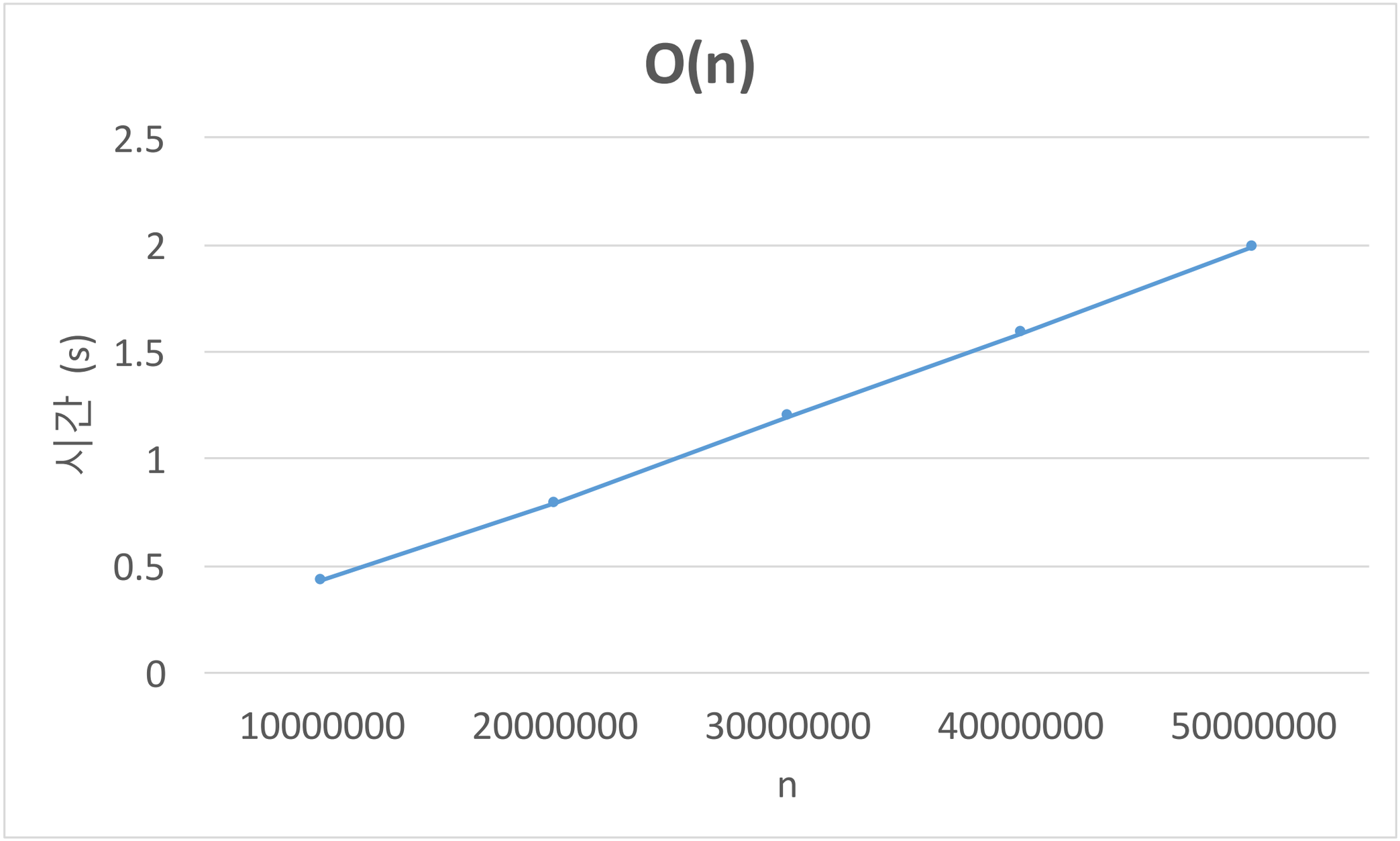
```python
for i in range(n) :
    sum = sum + 1
```

```python
for i in range(n) :
    for j in range(n) :
        sum = sum + 1
```

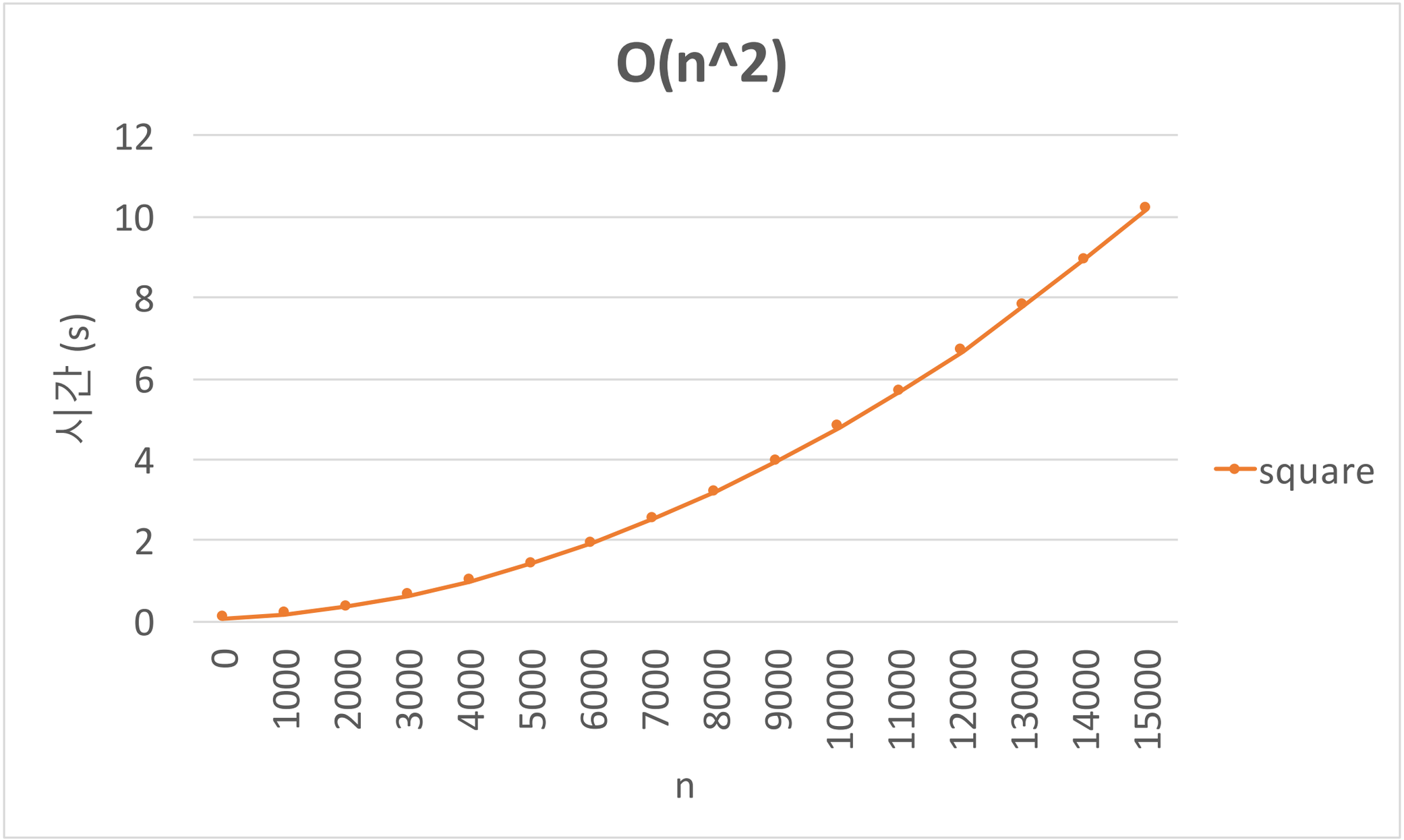
결과 : O(n)


O(n)
0.0008
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0
시간 (s)
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
12000
13000
14000
15000
n
linear

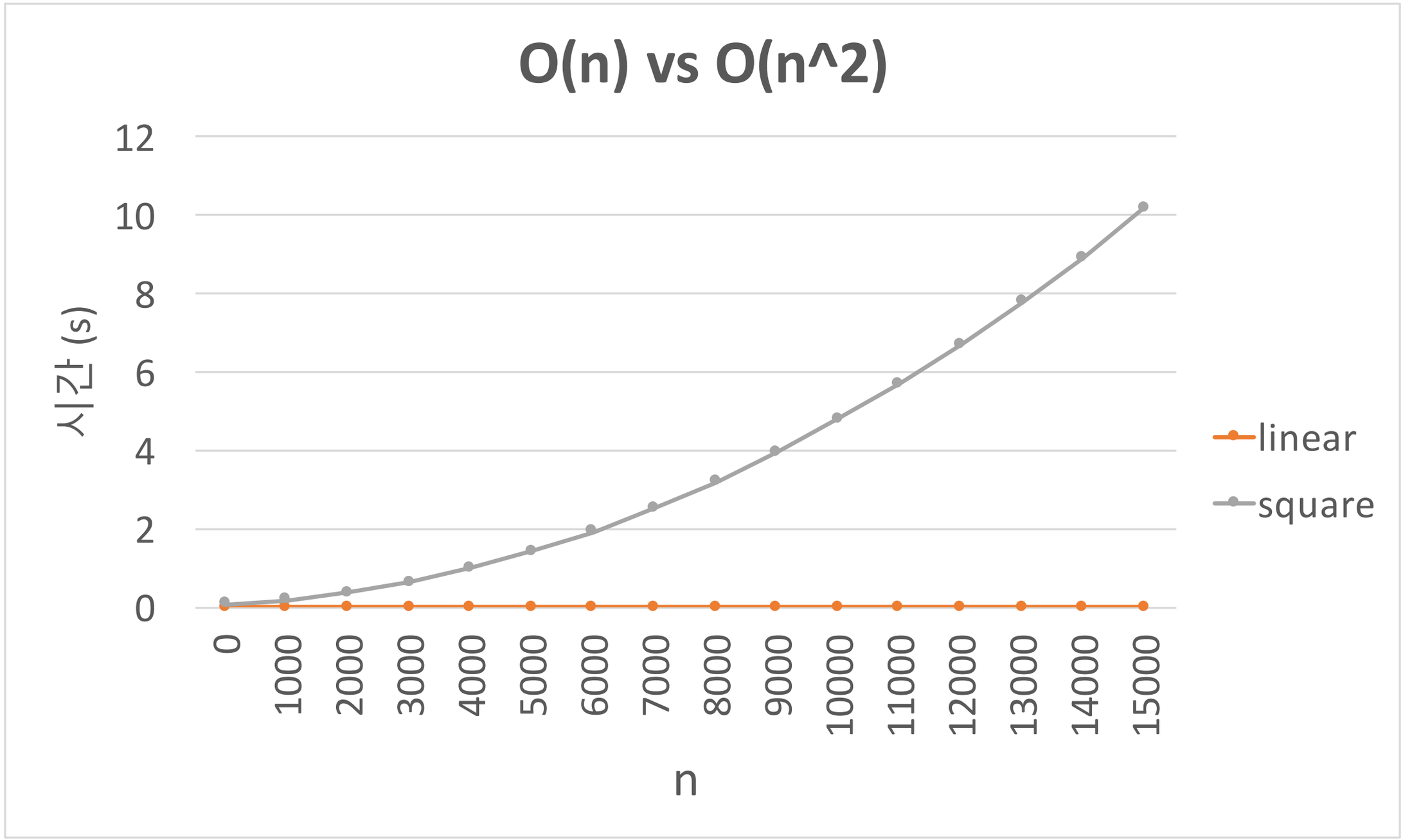
결과 : O(n)


O(n)
2.5
2
1.5
1
0.5
0
시간 (s)
10000000
20000000
30000000
40000000
50000000
n

결과 : $O(n^2)$


O(n^2)
12
10
8
6
4
2
0
시간 (s)
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
12000
13000
14000
15000
n
square

결과 : O(n) vs O(n^2)


O(n) vs O(n^2)
시간 (s)
12
10
8
6
4
2
0
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
12000
13000
14000
15000
n
linear
square

결론

대략 2500만개의 명령을

수행하면 1초가 걸린다

내 알고리즘이 **최악의 경우**에 2500만개를

수행하는지 고민해보자

[맛보기 문제 3] 소수 판정

숫자가 주어질 때, 소수인지 판정하라

(단, $1 \leq n \leq 1{,}000{,}000{,}000$ // 제한시간 1초)

입력의 예	출력의 예
7	True
4	False

[맛보기 문제 3] 소수 판정

알고리즘 개발

2 ~ n-1의 모든 수로 나누어본다.

[맛보기 문제 3] 소수 판정

알고리즘 개발

2 ~ n-1의 모든 수로 나누어본다.

풀이 증명

가능한 모든 수로 나누어보기 때문에 옳은 풀이이다

[맛보기 문제 3] 소수 판정

알고리즘 개발

2 ~ n-1의 모든 수로 나누어본다.

풀이 증명

가능한 모든 수로 나누어보기 때문에 옳은 풀이이다

시간복잡도

O(n). 2500만보다 더 많은 명령을 수행하므로 **1초에 안됨**

[맛보기 문제 3] 소수 판정

더 나은 알고리즘 개발

$2 \leq i \leq \sqrt{n}$ 의 모든 수로 나누어본다.

[맛보기 문제 3] 소수 판정

풀이 증명

증명해야 하는 명제 : **우리 알고리즘이 소수 판정을 옳게 한다**

[맛보기 문제 3] 소수 판정

풀이 증명

증명해야 하는 명제 : **우리 알고리즘이 소수 판정을 옳게 한다**

Case 1. n이 소수인 경우

n이 소수인 경우, 2 ~ √n 의 모든 숫자로도 나누어 떨어지지 않는다.

따라서 우리 알고리즘은 True를 반환한다.

[맛보기 문제 3] 소수 판정

풀이 증명

증명해야 하는 명제 : **우리 알고리즘이 소수 판정을 옳게 한다**

Case 2. n이 소수가 아닌 경우

[맛보기 문제 3] 소수 판정

풀이 증명

증명해야 하는 명제 : **우리 알고리즘이 소수 판정을 옳게 한다**

Case 2. n이 소수가 아닌 경우

n은 소수가 아니므로 약수가 존재한다.

이 약수들 중에서 $\sqrt{n}$보다 작거나 같은 약수가 **반드시 존재한다.**

[맛보기 문제 3] 소수 판정

풀이 증명

증명해야 하는 명제 : **우리 알고리즘이 소수 판정을 옳게 한다**

Case 2. n이 소수가 아닌 경우

관찰 1. a가 n의 약수면, (n / a)는 자연수이다.

관찰 2. 만약 $a \geq \sqrt{n}$ 이면, $(n / a) \leq \sqrt{n}$ 이다.

[맛보기 문제 3] 소수 판정

풀이 증명

증명해야 하는 명제 : **우리 알고리즘이 소수 판정을 옳게 한다**

Case 2. n이 소수가 아닌 경우

만약 모든 약수가 √n보다 크다고 가정하자. 이 약수를 a라고 하자.

[맛보기 문제 3] 소수 판정

풀이 증명

증명해야 하는 명제 : **우리 알고리즘이 소수 판정을 옳게 한다**

Case 2. n이 소수가 아닌 경우

만약 모든 약수가 $\sqrt{n}$보다 크다고 가정하자. 이 약수를 a라고 하자.

그러면 (관찰 2)에 의하여 $(n / a) \leq \sqrt{n}$ 이고, 이 또한 n의 약수이다.

[맛보기 문제 3] 소수 판정

풀이 증명

증명해야 하는 명제 : **우리 알고리즘이 소수 판정을 옳게 한다**

Case 2. n이 소수가 아닌 경우

만약 모든 약수가 $\sqrt{n}$보다 크다고 가정하자. 이 약수를 a라고 하자.

그러면 (관찰 2)에 의하여 $(n / a) \leq \sqrt{n}$ 이고, 이 또한 n의 약수이다.

따라서 $\sqrt{n}$보다 작거나 같은 약수가 적어도 하나 존재한다.

[맛보기 문제 3] 소수 판정

풀이 증명

증명해야 하는 명제 : **우리 알고리즘이 소수 판정을 옳게 한다**

Case 2. n이 소수가 아닌 경우

우리 알고리즘은 $2 \leq i \leq \sqrt{n}$ 의 수가 n으로
나누어 떨어지는지 테스트한다.

해당 범위에는 적어도 하나의 약수가 반드시 존재하므로,
False가 반환된다.

[맛보기 문제 3] 소수 판정

시간복잡도

O(√n). n ≤ 1,000,000,000 이므로, √n ≤ 31,622

[맛보기 문제 3] 소수 판정

시간복잡도

O(√n). n ≤ 1,000,000,000 이므로, √n ≤ 31,622

2500만번보다 적은 횟수의 명령이므로

1초 내에 결과가 나온다

[문제 4] 소수의 개수 구하기

범위가 주어질 때, 소수가 몇개인지 출력하라

(단, $1 \leq a, b \leq 100,000$ // 제한시간 1초)

입력의 예

```
1  7
```

```
4  17
```

출력의 예

```
4
```

```
5
```

2. 알고리즘 개발

모든 범위 내의 숫자들에 대하여 소수인지 판별

숫자 i가 소수인지 판별하기 위해서는 O(√i)가 걸림

2. 알고리즘 개발

모든 범위 내의 숫자들에 대하여 소수인지 판별

숫자 i가 소수인지 판별하기 위해서는 O(√i)가 걸림

따라서 [a, b] 내의 모든 숫자들을 판단하기 위해서는

$$O(\sqrt{a}) + O(\sqrt{(a+1)}) + O(\sqrt{(a+2)}) + \ldots + O(\sqrt{(b-1)}) + O(\sqrt{b})$$

3. 풀이 증명

범위 내 모든 숫자가 소수인지

테스트 해보므로 **옳은 풀이**임

4. 시간복잡도

[a, b] 내의 모든 숫자에 대해서 소수 판정을 함

4. 시간복잡도

[a, b] 내의 모든 숫자에 대해서 소수 판정을 함 **O(n√n)**

4. 시간복잡도

[a, b] 내의 모든 숫자에 대해서 소수 판정을 함 **O(n√n)**

최악의 경우는 [1, 100000]이므로,

$100000 * \sqrt{100000} = 31,622,776$

4. 시간복잡도

[a, b] 내의 모든 숫자에 대해서 소수 판정을 함 **O(n√n)**

최악의 경우는 [1, 100000]이므로,

100000 * √100000 = 31,622,776

2500만보다 크기 때문에 **1초 안에 안나온다**

2. 알고리즘 개발

에라토스테네스의 체를 사용

		2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39

2. 알고리즘 개발

에라토스테네스의 체를 사용

2	3	4	5	6	7	8	9		
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39

2. 알고리즘 개발

에라토스테네스의 체를 사용

	2	3	4	5	6	7	8	9	
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39

2. 알고리즘 개발

에라토스테네스의 체를 사용

	2	**3**	4	5	6	7	8	9	
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39

2. 알고리즘 개발

에라토스테네스의 체를 사용

2	**3**	4	5	6	7	8	9		
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39

2. 알고리즘 개발

에라토스테네스의 체를 사용

	2	**3**	4	**5**	6	7	8	9	
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39

2. 알고리즘 개발

에라토스테네스의 체를 사용

2	**3**	4	**5**	6	7	8	9		
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39

2. 알고리즘 개발

에라토스테네스의 체를 사용

	2	**3**	4	**5**	6	**7**	8	9	
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39

3. 문제 해결 증명

에라토스테네스의 체를 이용하면

소수를 구할 수 있음

4. 시간복잡도

숫자 하나를 잡고, 그 배수를 없애는데

얼마나 걸리는가 ?

		2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39

4. 시간복잡도

숫자 하나를 잡고, 그 배수를 없애는데

얼마나 걸리는가 ?

		2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39

4. 시간복잡도

숫자 하나를 잡고, 그 배수를 없애는데

얼마나 걸리는가 ?

		2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39

4. 시간복잡도

숫자 하나를 잡고, 그 배수를 없애는데

얼마나 걸리는가 ?

배수의 개수만큼의 연산이 필요함

		2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39

4. 시간복잡도

숫자 하나를 잡고, 그 배수를 없애는데

얼마나 걸리는가 ?

배수의 개수만큼의 연산이 필요함

숫자 x에 대하여 O(b / x) 번이 필요하다

		2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39

4. 시간복잡도

[1, n] 의 모든 숫자에 대해서 배수를 지운다면

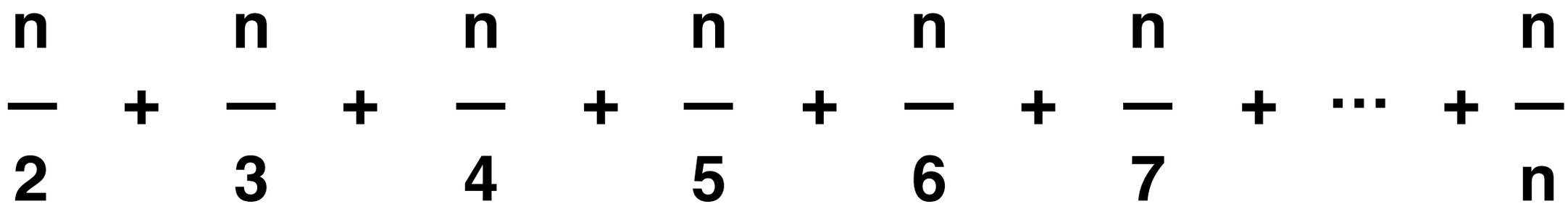
얼마나 걸리는가

		2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39

4. 시간복잡도

[2, n] 의 모든 숫자에 대해서 배수를 지운다면

얼마나 걸리는가



4. 시간복잡도

[2, n] 의 모든 숫자에 대해서 배수를 지운다면

얼마나 걸리는가

$$\underset{2}{\overset{n}{-}} + \underset{3}{\overset{n}{-}} + \underset{4}{\overset{n}{-}} + \underset{5}{\overset{n}{-}} + \underset{6}{\overset{n}{-}} + \underset{7}{\overset{n}{-}} + \cdots + \underset{n}{\overset{n}{-}} = n * \left(\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \right)$$

4. 시간복잡도

[2, n] 의 모든 숫자에 대해서 배수를 지운다면

얼마나 걸리는가

$$\underbrace{n}_{2} + \underbrace{n}_{3} + \underbrace{n}_{4} + \underbrace{n}_{5} + \underbrace{n}_{6} + \underbrace{n}_{7} + \cdots + \underbrace{n}_{n}$$

$$= n * \left(\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \right)$$

$$\leq n \log n$$

이건 이해 안하셔도 됩니다

$$\int_{1}^{n+1} \frac{1}{x} \mathrm{d}x = \ln(n+1)$$

4. 시간복잡도

[2, n] 의 모든 숫자에 대해서 배수를 지운다면

얼마나 걸리는가

O(n log n) = 100,000 * log 100,000 = 1,660,964

4. 시간복잡도

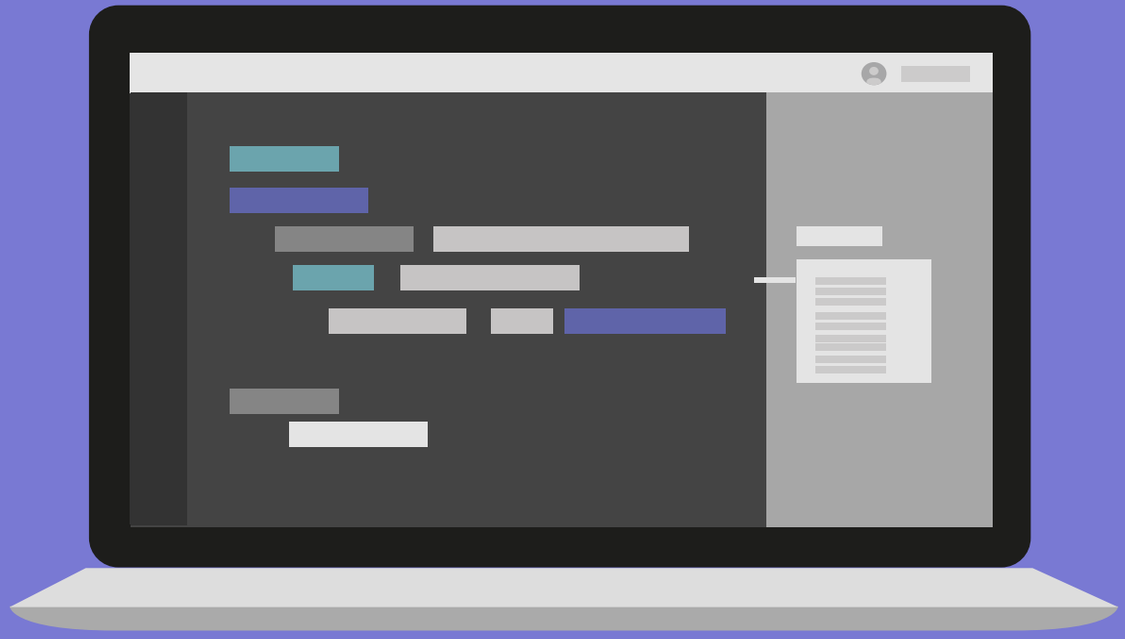
[2, n] 의 모든 숫자에 대해서 배수를 지운다면

얼마나 걸리는가

O(n log n) = 100,000 * log 100,000 = 1,660,964

2500만보다 적으므로 **1초 안에 나온다 !**

[문제 4] 소수의 개수 구하기



[문제 3] 이진 탐색

n개의 **정렬된** 숫자 중에서,

특정 숫자가 포함되어 있는지 판단

입력의 예

```
6
1 3 4 5 9 10
4
```

출력의 예

```
True
```

[문제 3] 이진 탐색

1	3	4	5	9	10	12

4를 찾자!

[문제 3] 이진 탐색

단순한 풀이 : 하나하나 비교하며 찾아본다

1	3	4	5	9	10	12

4를 찾자!

[문제 3] 이진 탐색

단순한 풀이 : 하나하나 비교하며 찾아본다

O(n)

1	3	4	5	9	10	12

4를 찾자!

[문제 3] 이진 탐색

정렬이 되어 있다는 사실을 이용한다

1	3	4	5	9	10	12

4를 찾자!

[문제 3] 이진 탐색

정렬이 되어 있다는 사실을 이용한다

4를 찾자!

[문제 3] 이진 탐색

정렬이 되어 있다는 사실을 이용한다

4를 찾자!

[문제 3] 이진 탐색

정렬이 되어 있다는 사실을 이용한다

4를 찾자!

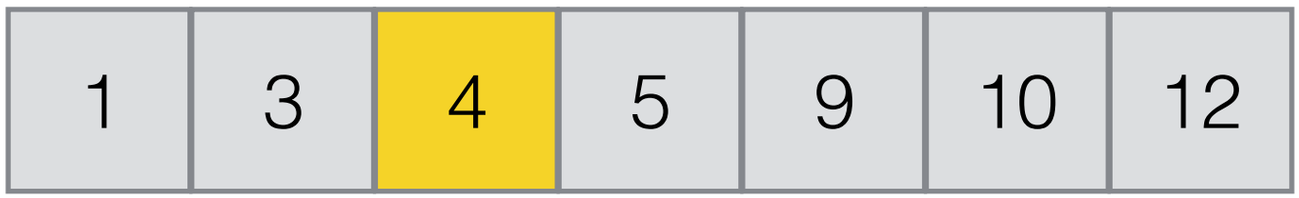
[문제 3] 이진 탐색

정렬이 되어 있다는 사실을 이용한다

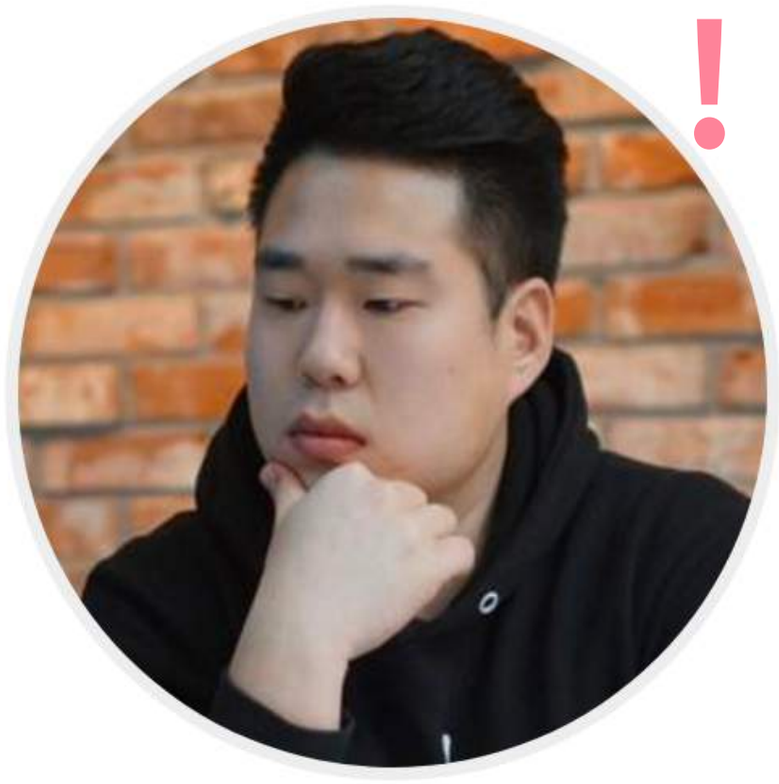
4를 찾자!

[문제 3] 이진 탐색

정렬이 되어 있다는 사실을 이용한다



4를 찾자!



아직 키보드에 손 올리시면 안됩니다

[문제 3] 이진 탐색

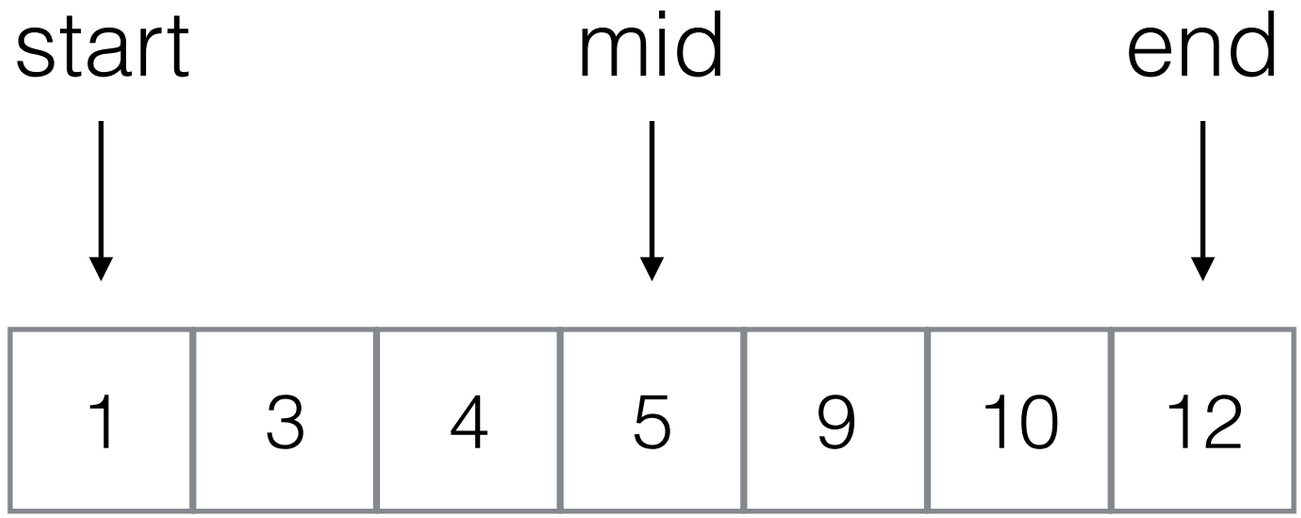
1	3	4	5	9	10	12

4를 찾자!

[문제 3] 이진 탐색

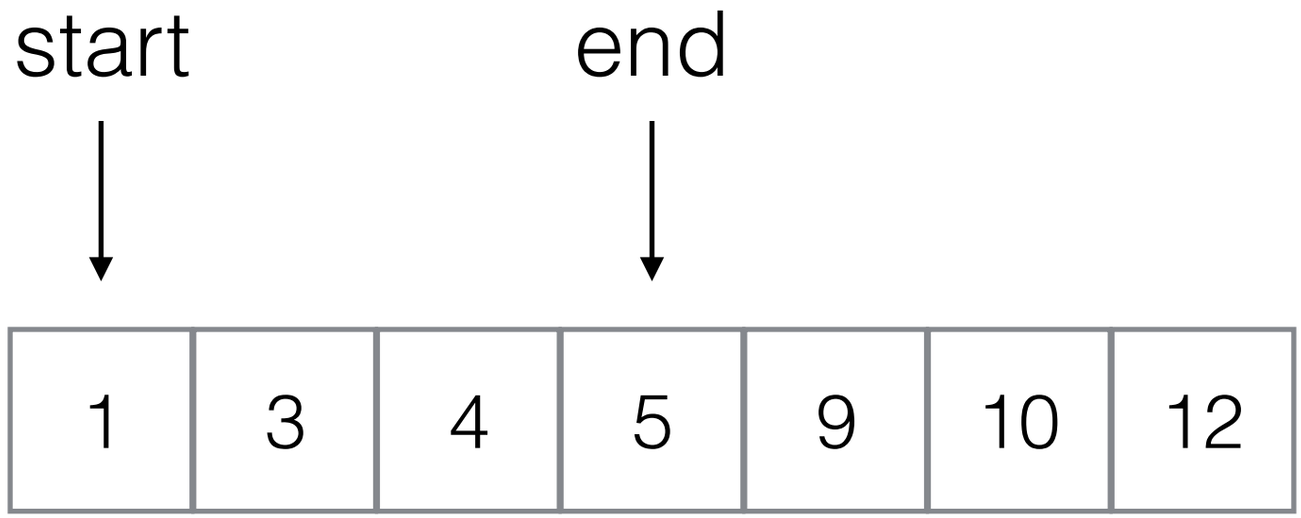
4를 찾자!

[문제 3] 이진 탐색



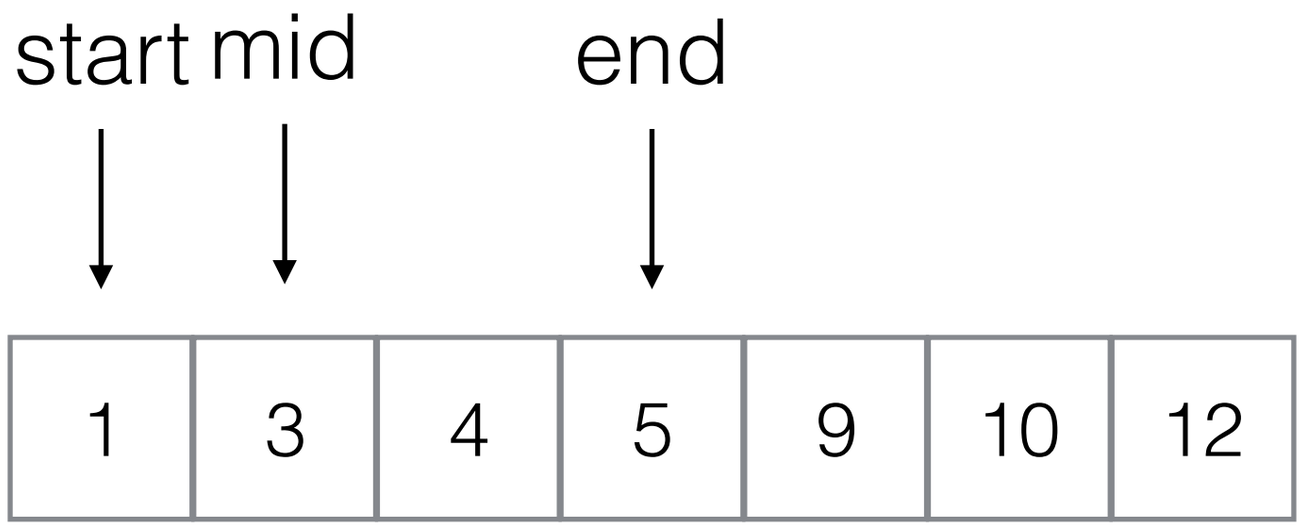
4를 찾자!

[문제 3] 이진 탐색



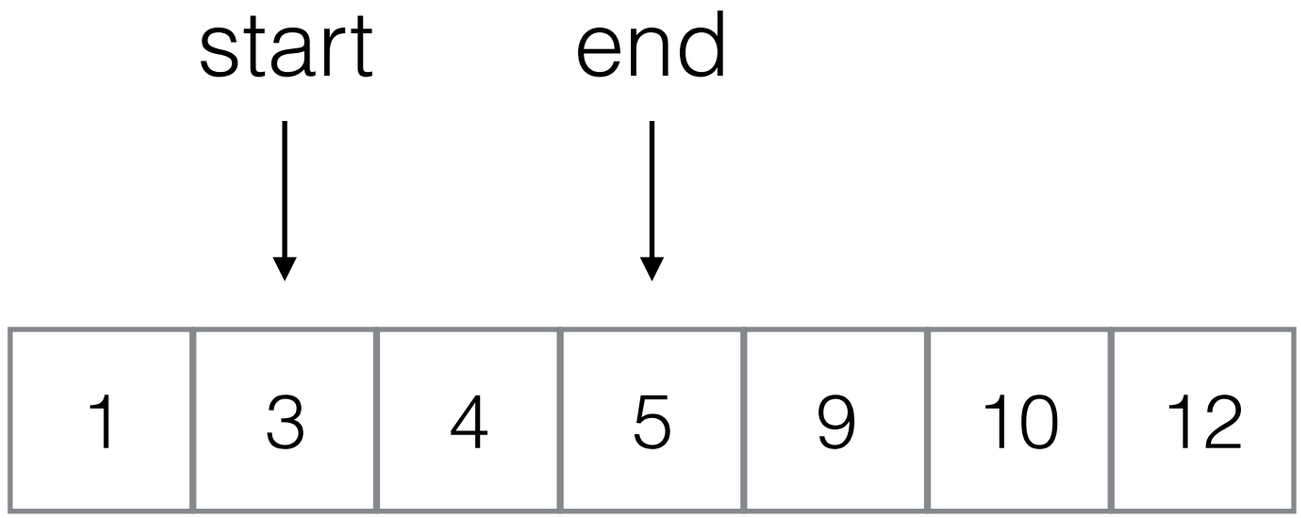
4를 찾자!

[문제 3] 이진 탐색



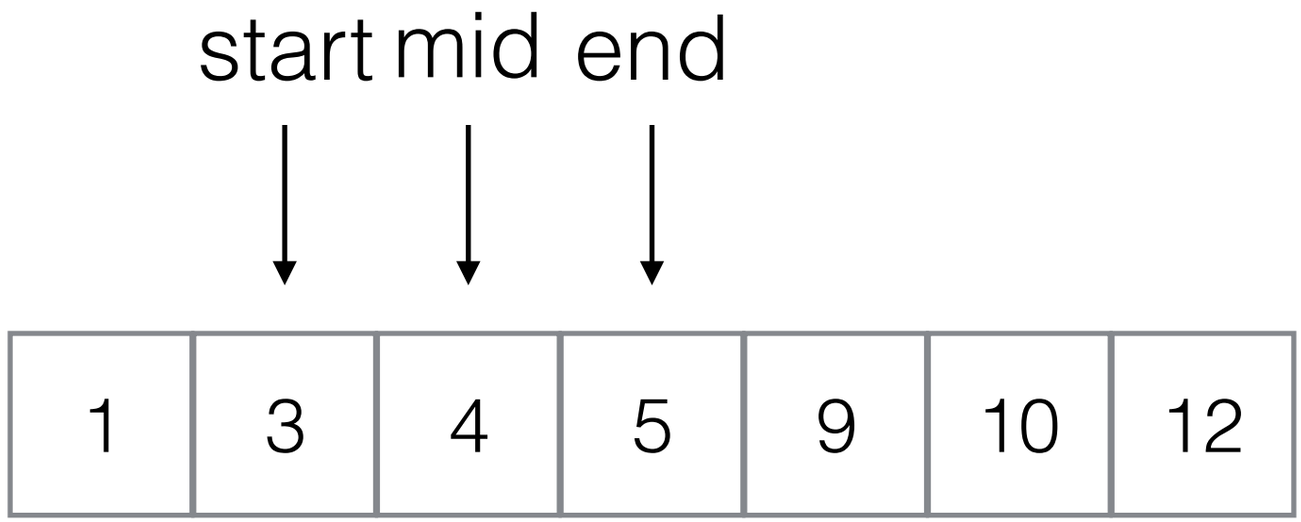
4를 찾자!

[문제 3] 이진 탐색



4를 찾자!

[문제 3] 이진 탐색

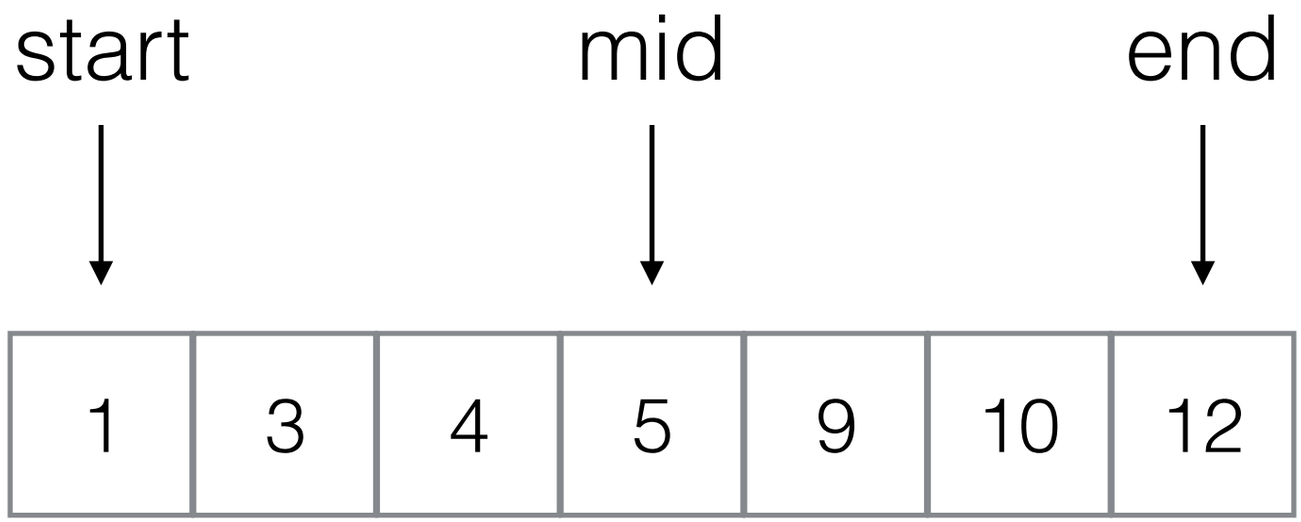


4를 찾자!

[문제 3] 이진 탐색

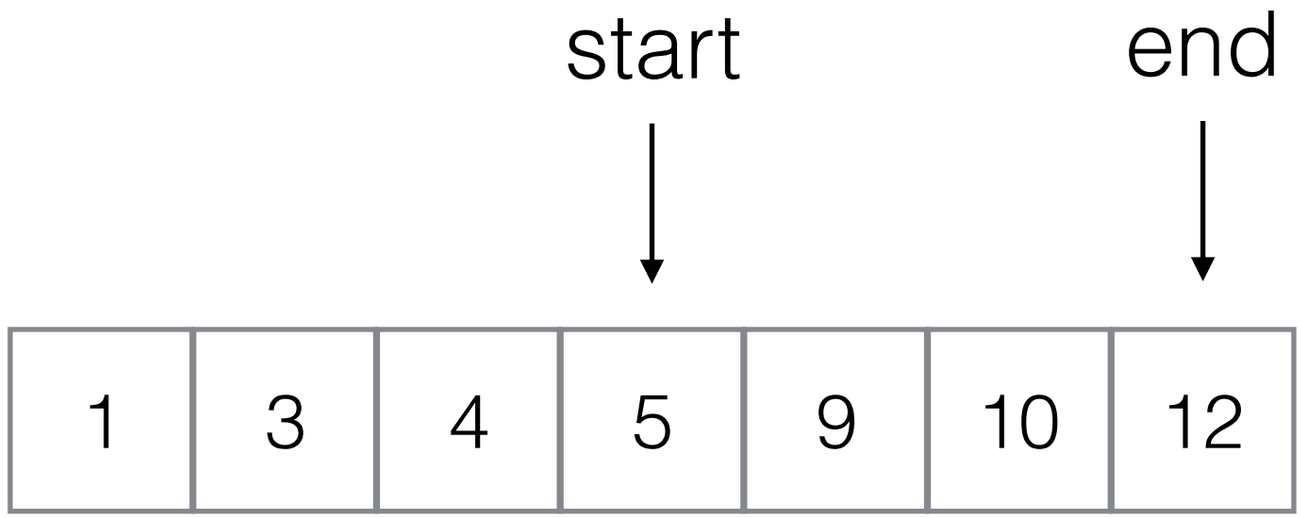
12를 찾자!

[문제 3] 이진 탐색



12를 찾자!

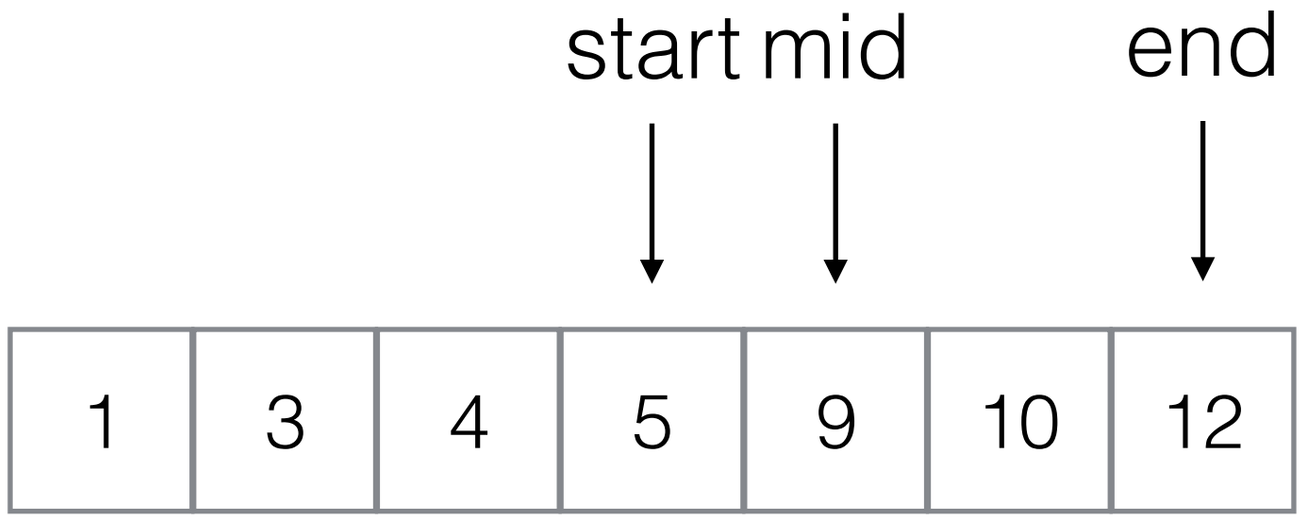
[문제 3] 이진 탐색



12를 찾자!

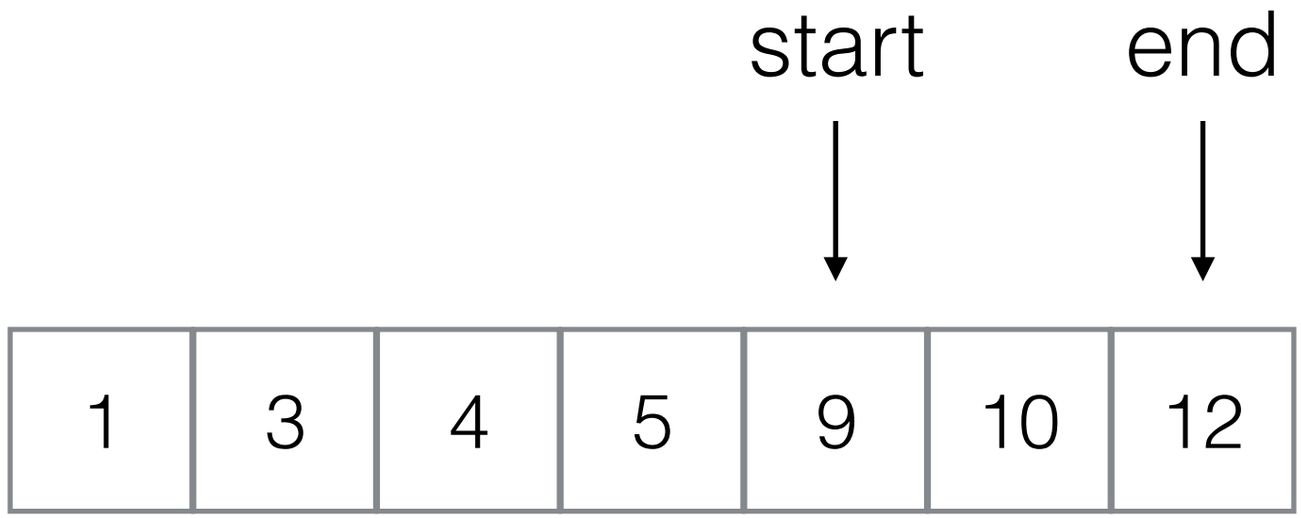
```
/* elice */                                                    90
```

[문제 3] 이진 탐색



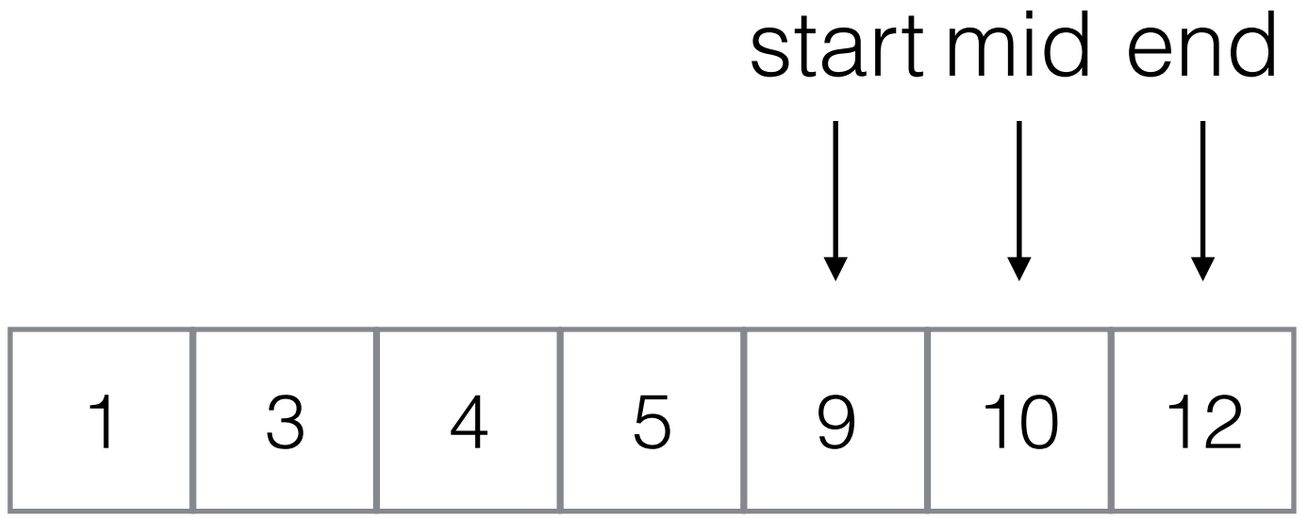
12를 찾자!

[문제 3] 이진 탐색



12를 찾자!

[문제 3] 이진 탐색

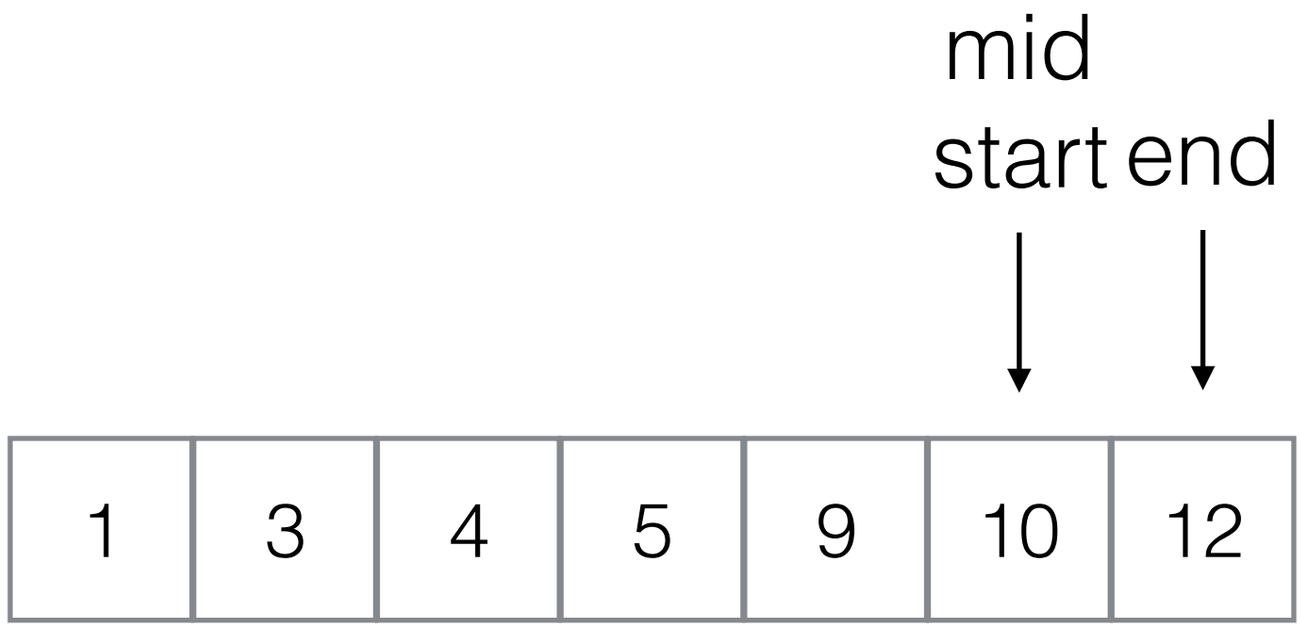


12를 찾자!

[문제 3] 이진 탐색

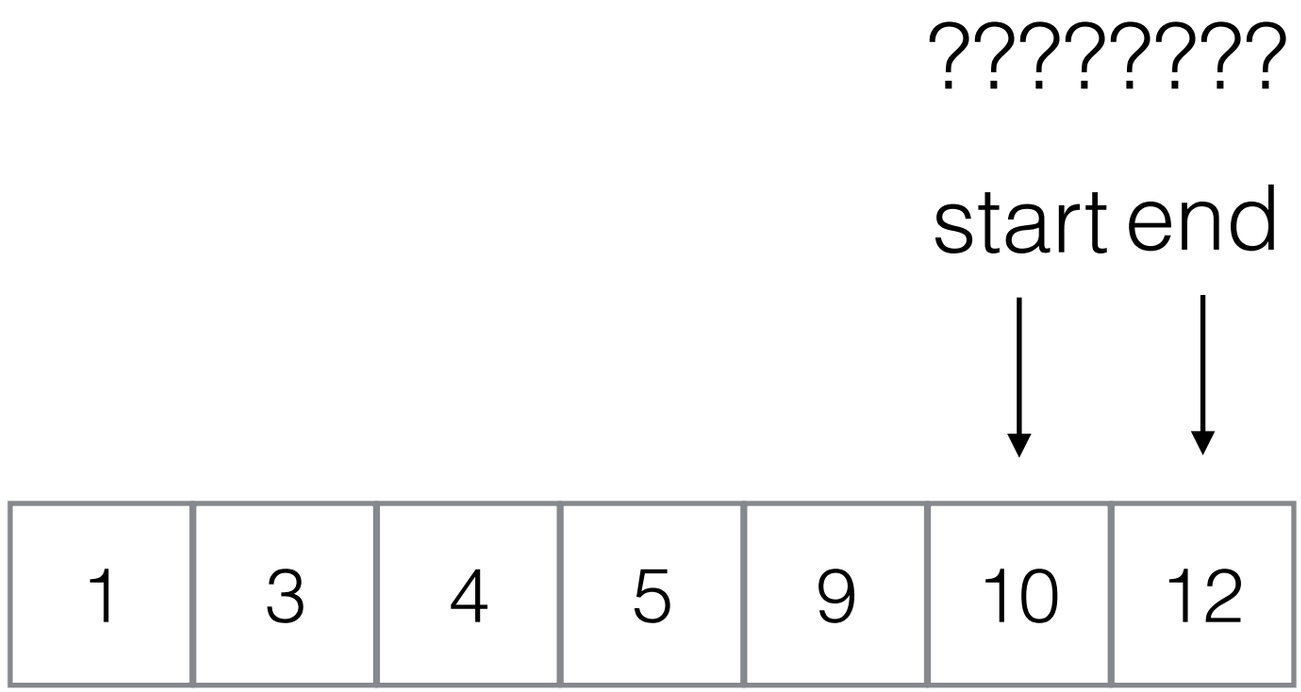
12를 찾자!

[문제 3] 이진 탐색



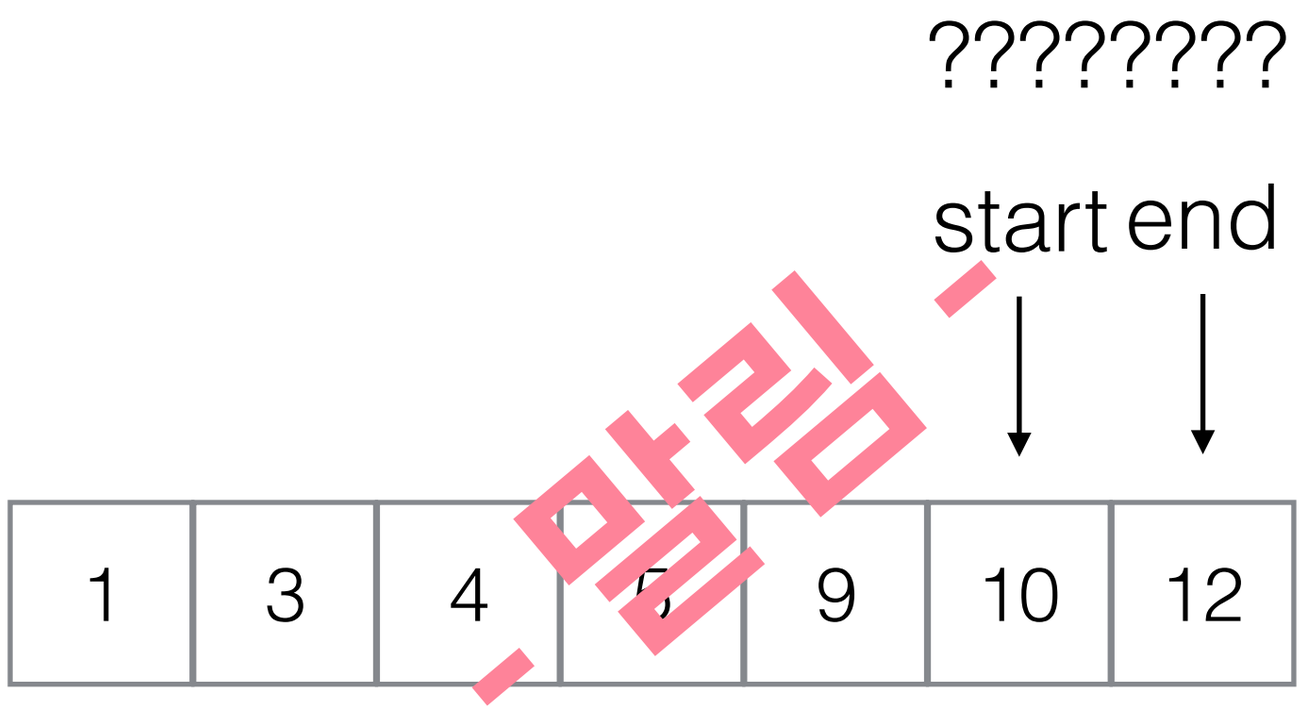
12를 찾자!

[문제 3] 이진 탐색



12를 찾자!

[문제 3] 이진 탐색



12를 찾자!

[문제 3] 이진 탐색

모든 단계에 **“왜”**를 생각하자

4를 찾자!

[문제 3] 이진 탐색

모든 단계에 **“왜”**를 생각하자

4를 찾자!

start : 항상 찾는 수보다 작은 숫자만 가리킨다

end : 항상 찾는 수보다 큰 숫자만 가리킨다

[문제 3] 이진 탐색

모든 단계에 “왜”를 생각하자

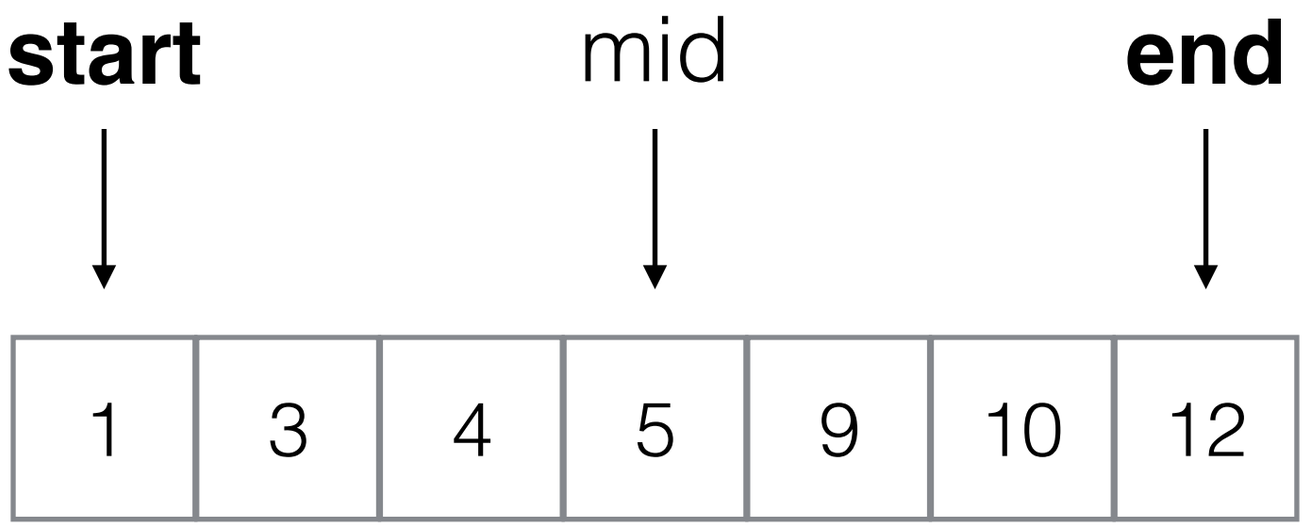
4를 찾자!

start : 항상 찾는 수보다 작은 숫자만 가리킨다

end : 항상 찾는 수보다 큰 숫자만 가리킨다

[문제 3] 이진 탐색

모든 단계에 **"왜"**를 생각하자



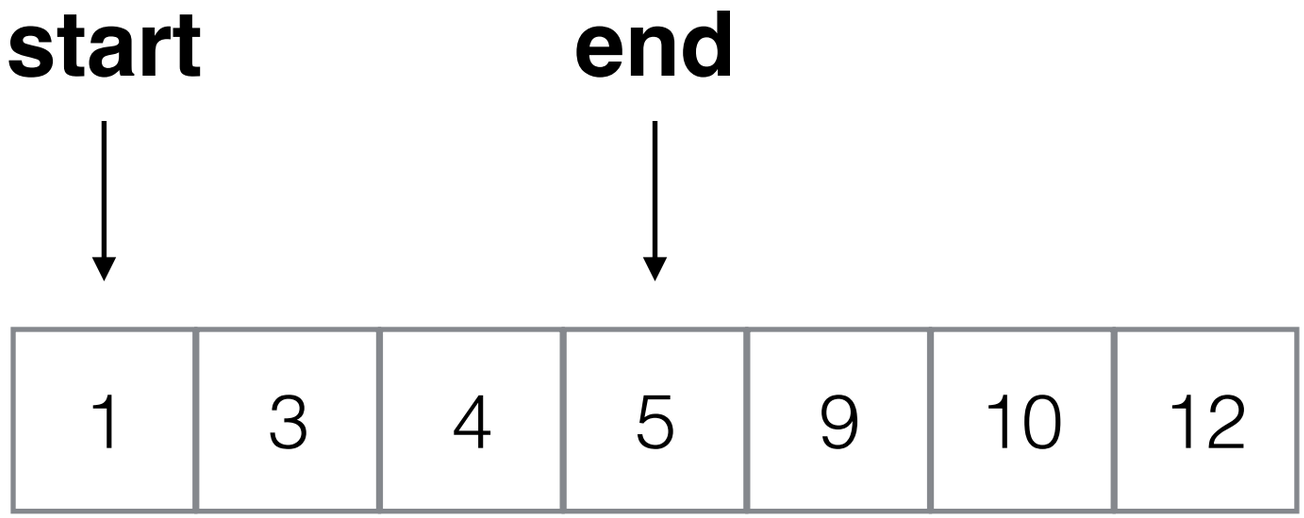
4를 찾자!

start : 항상 찾는 수보다 작은 숫자만 가리킨다

end : 항상 찾는 수보다 큰 숫자만 가리킨다

[문제 3] 이진 탐색

모든 단계에 “왜”를 생각하자



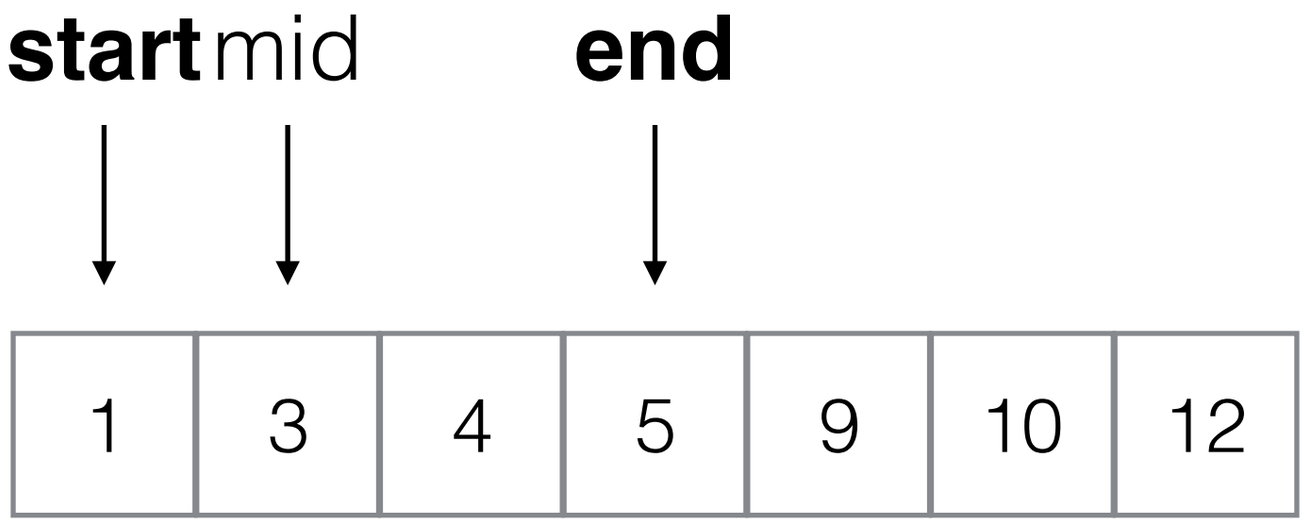
4를 찾자!

start : 항상 찾는 수보다 작은 숫자만 가리킨다

end : 항상 찾는 수보다 큰 숫자만 가리킨다

[문제 3] 이진 탐색

모든 단계에 “왜”를 생각하자



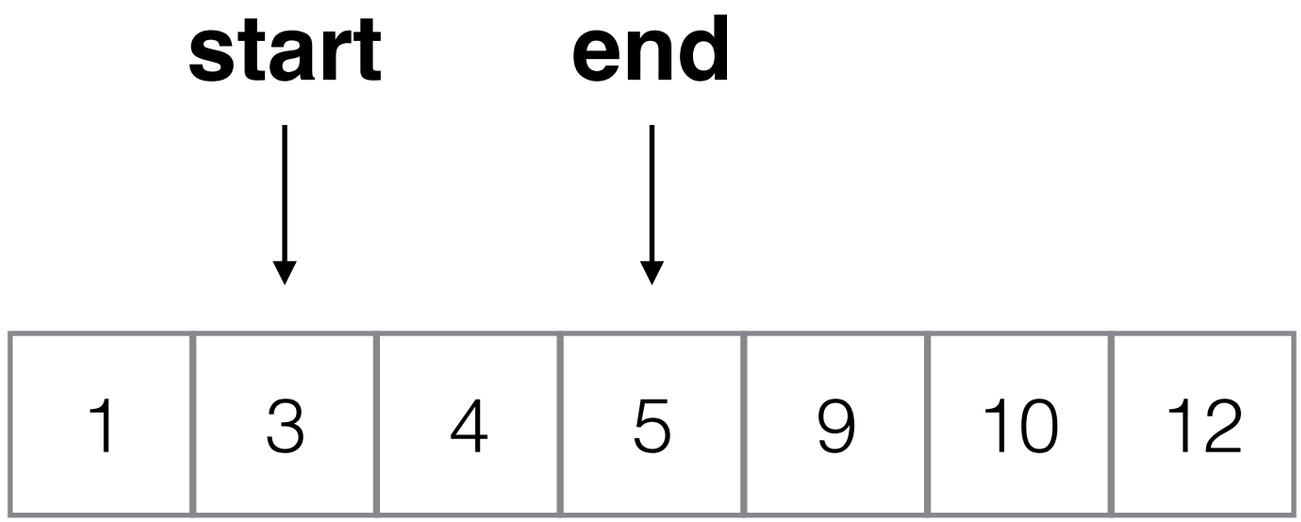
4를 찾자!

start : 항상 찾는 수보다 작은 숫자만 가리킨다

end : 항상 찾는 수보다 큰 숫자만 가리킨다

[문제 3] 이진 탐색

모든 단계에 **“왜”**를 생각하자



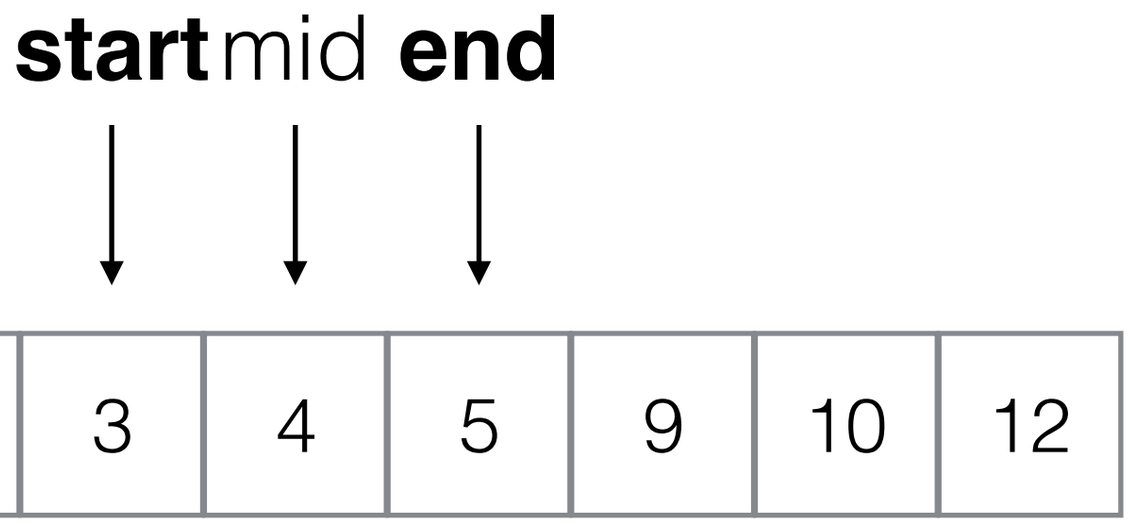
4를 찾자!

start : 항상 찾는 수보다 작은 숫자만 가리킨다

end : 항상 찾는 수보다 큰 숫자만 가리킨다

[문제 3] 이진 탐색

모든 단계에 **"왜"**를 생각하자



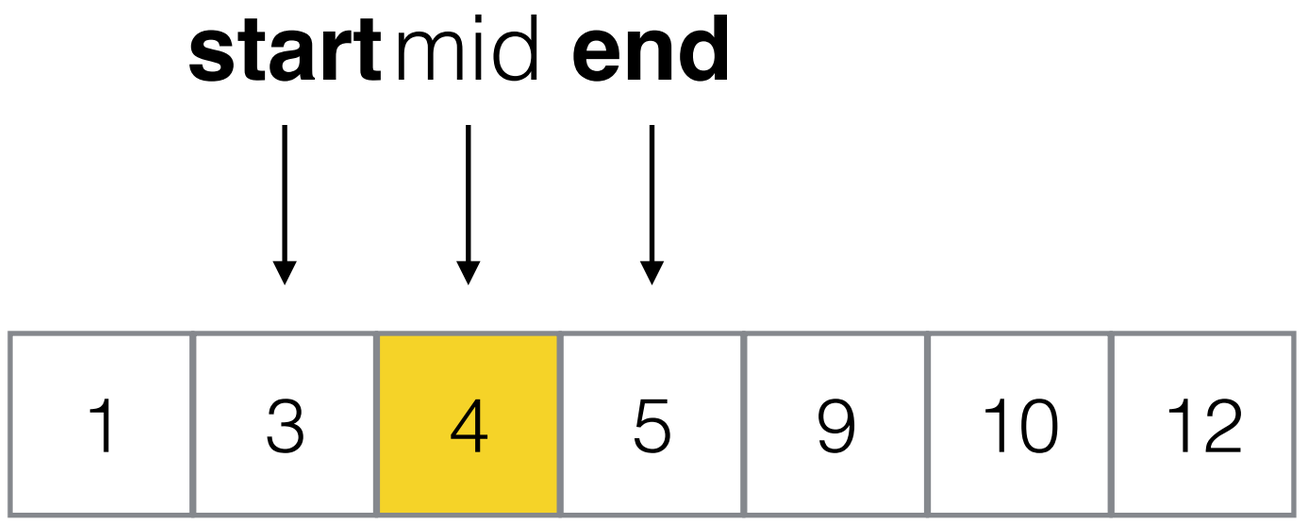
4를 찾자!

start : 항상 찾는 수보다 작은 숫자만 가리킨다

end : 항상 찾는 수보다 큰 숫자만 가리킨다

[문제 3] 이진 탐색

모든 단계에 **"왜"**를 생각하자



4를 찾자!

start : 항상 찾는 수보다 작은 숫자만 가리킨다

end : 항상 찾는 수보다 큰 숫자만 가리킨다

[문제 3] 이진 탐색

모든 단계에 **“왜”**를 생각하자

12를 찾자!

start : 항상 찾는 수보다 작은 숫자만 가리킨다

end : 항상 찾는 수보다 큰 숫자만 가리킨다

[문제 3] 이진 탐색

모든 단계에 **“왜”**를 생각하자

12를 찾자!

start : 항상 찾는 수보다 작은 숫자만 가리킨다

end : 항상 찾는 수보다 큰 숫자만 가리킨다

[문제 3] 이진 탐색

모든 단계에 **“왜”**를 생각하자

start

end

1	3	4	5	9	10	12

11을 찾자!

start : 항상 찾는 수보다 작은 숫자만 가리킨다

end : 항상 찾는 수보다 큰 숫자만 가리킨다

[문제 3] 이진 탐색

모든 단계에 **“왜”**를 생각하자

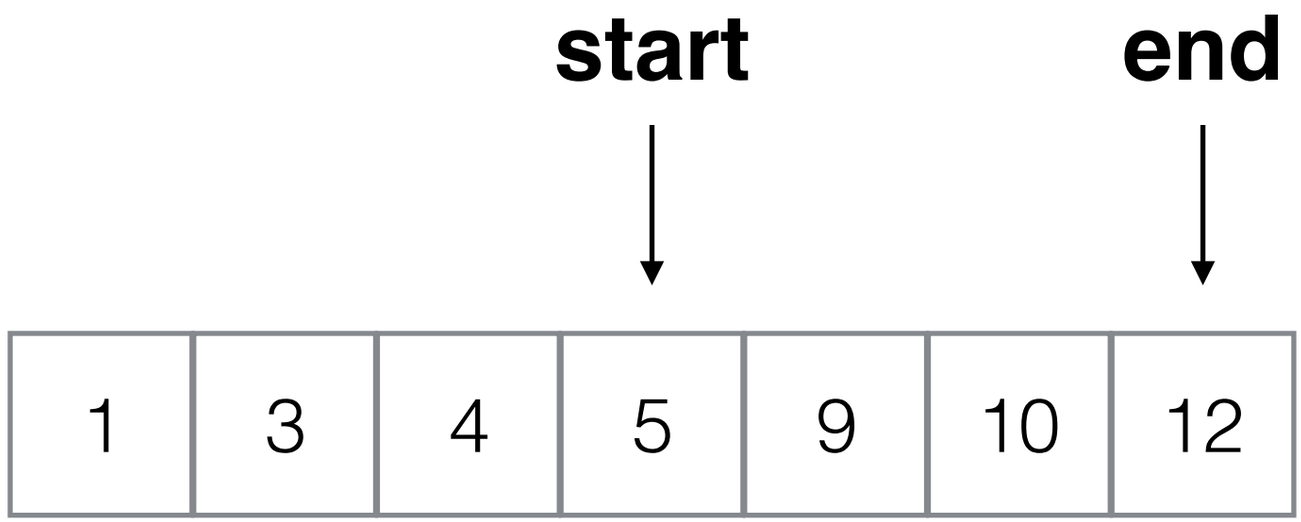
11을 찾자!

start : 항상 찾는 수보다 작은 숫자만 가리킨다

end : 항상 찾는 수보다 큰 숫자만 가리킨다

[문제 3] 이진 탐색

모든 단계에 **“왜”**를 생각하자



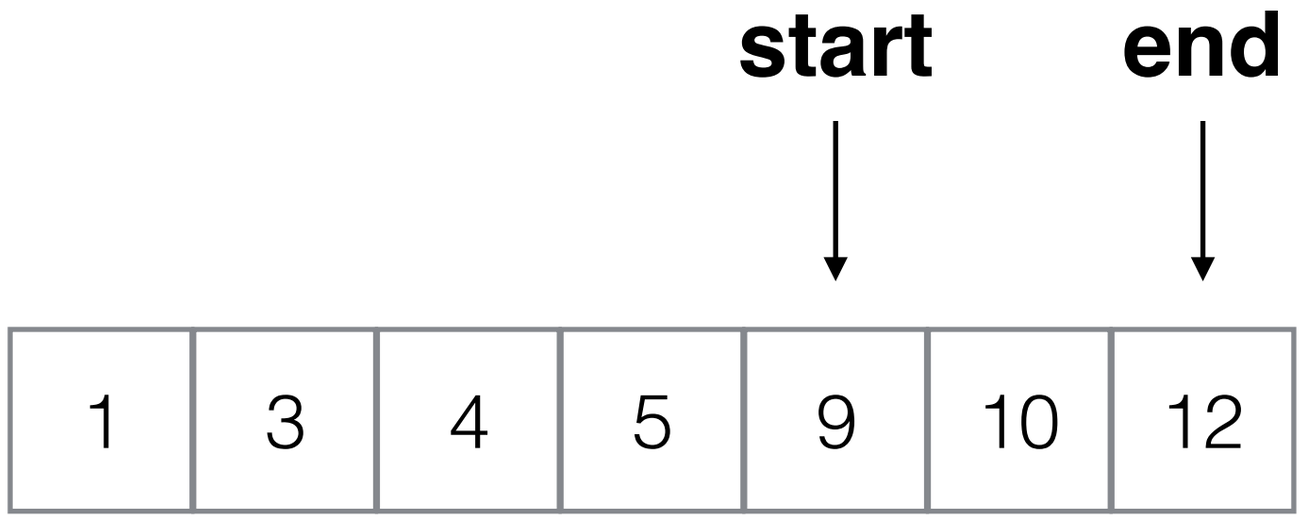
11을 찾자!

start : 항상 찾는 수보다 작은 숫자만 가리킨다

end : 항상 찾는 수보다 큰 숫자만 가리킨다

[문제 3] 이진 탐색

모든 단계에 “왜”를 생각하자



11을 찾자!

start : 항상 찾는 수보다 작은 숫자만 가리킨다

end : 항상 찾는 수보다 큰 숫자만 가리킨다

[문제 3] 이진 탐색

모든 단계에 “왜”를 생각하자

11을 찾자!

start : 항상 찾는 수보다 작은 숫자만 가리킨다

end : 항상 찾는 수보다 큰 숫자만 가리킨다

[문제 3] 이진 탐색

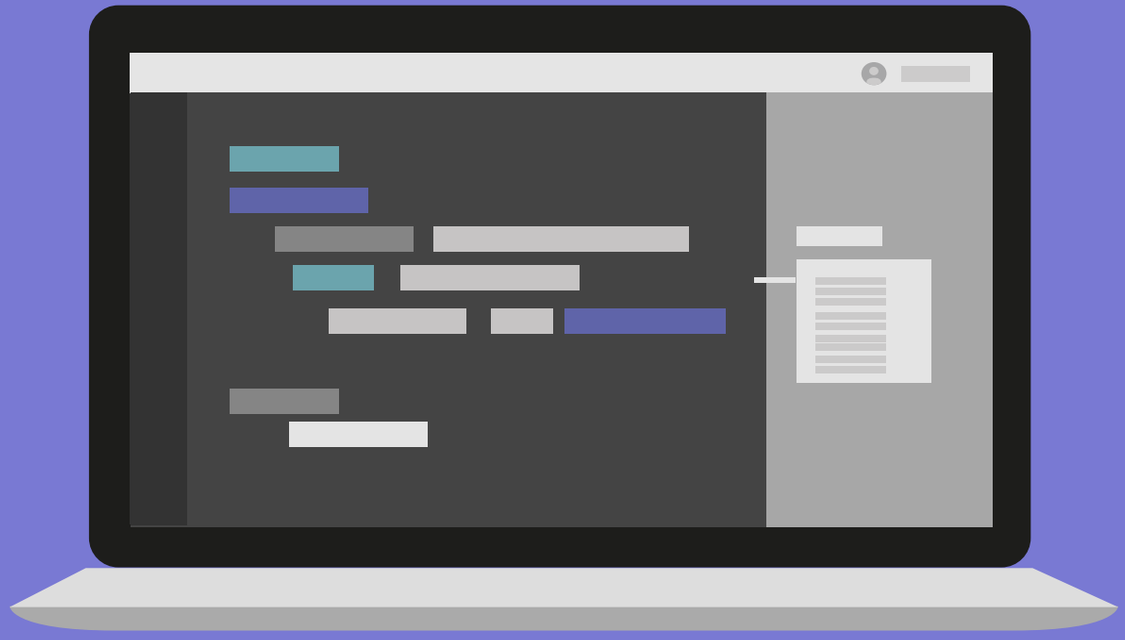
모든 단계에 “왜”를 생각하자

11을 찾자! **없음**

start : 항상 찾는 수보다 작은 숫자만 가리킨다

end : 항상 찾는 수보다 큰 숫자만 가리킨다

[문제 3] 이진 탐색

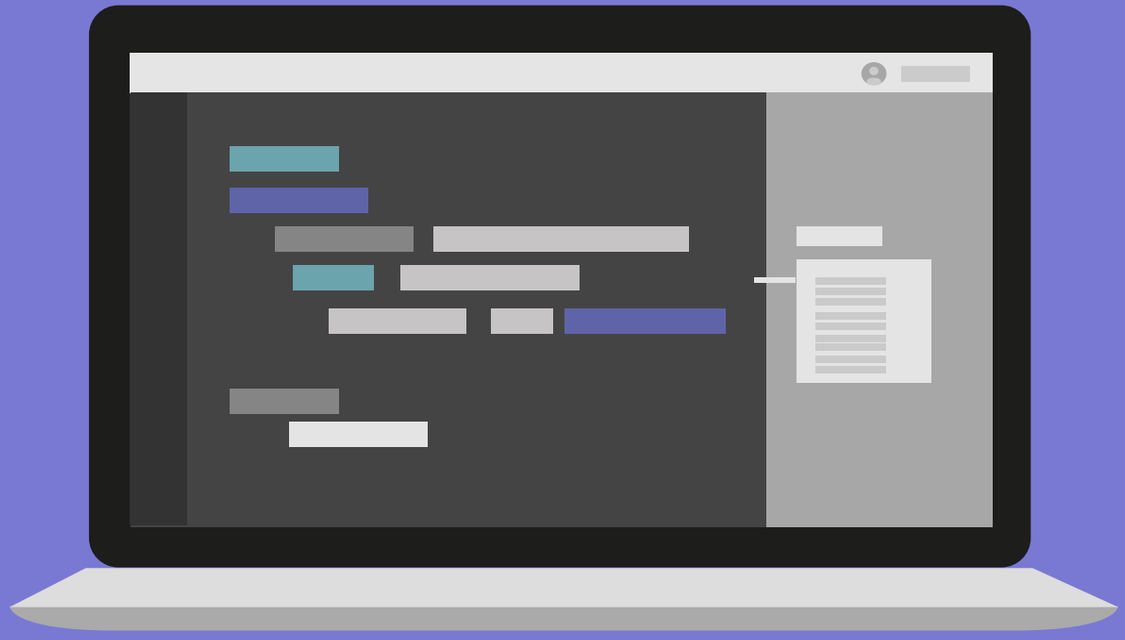


요약 : 문제 풀이 과정

1. 문제를 정확히 이해한다
2. 문제를 해결하는 알고리즘을 개발한다
3. 알고리즘이 **문제를 해결한다는 것을 증명한다**
4. 알고리즘이 **제한시간 내에 동작한다는 것을 보인다**
5. 알고리즘을 코드로 작성한다
6. 제출 후 만점을 받고 매우 기뻐한다

퀴즈와 설문

퀴즈 URL은 강의 목록을 참고해주세요



감사합니다!

신현규

E-mail : hyungyu.sh@kaist.ac.kr

Kakao : yougatup

/* elice */

문의 및 연락처

academy.elice.io

contact@elice.io

facebook.com/elice.io

blog.naver.com/elicer