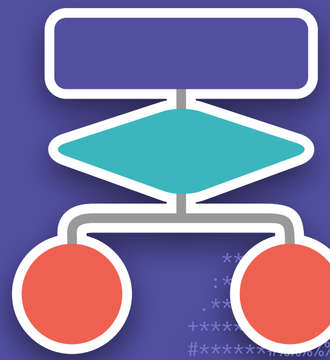
/* elice */

알고리즘의 정석

문제 해결 절차, 완전 탐색, 시간복잡도



엘리스 선생님

수강 목표

알고리즘 문제 **해결 절차**를 배워봅니다

시간 복잡도를 통해 효율적인 알고리즘에 대해 알아봅니다

완전 탐색의 개념에 대해 배워봅니다

목차

문제 해결의 절차

문제 해결의 절차

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

[실습] k번째 숫자 찾기

n개의 숫자 중

“지금까지 입력된 숫자들 중에서 k번째 숫자”는 ?

(단, $1 \leq k \leq n \leq 100$)

입력의 예

```
10 3
1 9 8 5 2 3 5 6 2 10
```

출력의 예

```
-1 -1 9 8 5 3 3 3 2 2
```

문제 해결의 절차

2. 문제를 해결하는 알고리즘을 개발한다

문제 해결의 절차

3. 알고리즘이 **문제를 해결한다는 것을 증명한다**

문제 해결의 절차

4. 알고리즘이 **제한시간내에 동작한다는 것을 보인다**

시간복잡도

시간복잡도

알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

```
sum = 0

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

시간복잡도

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

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

알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

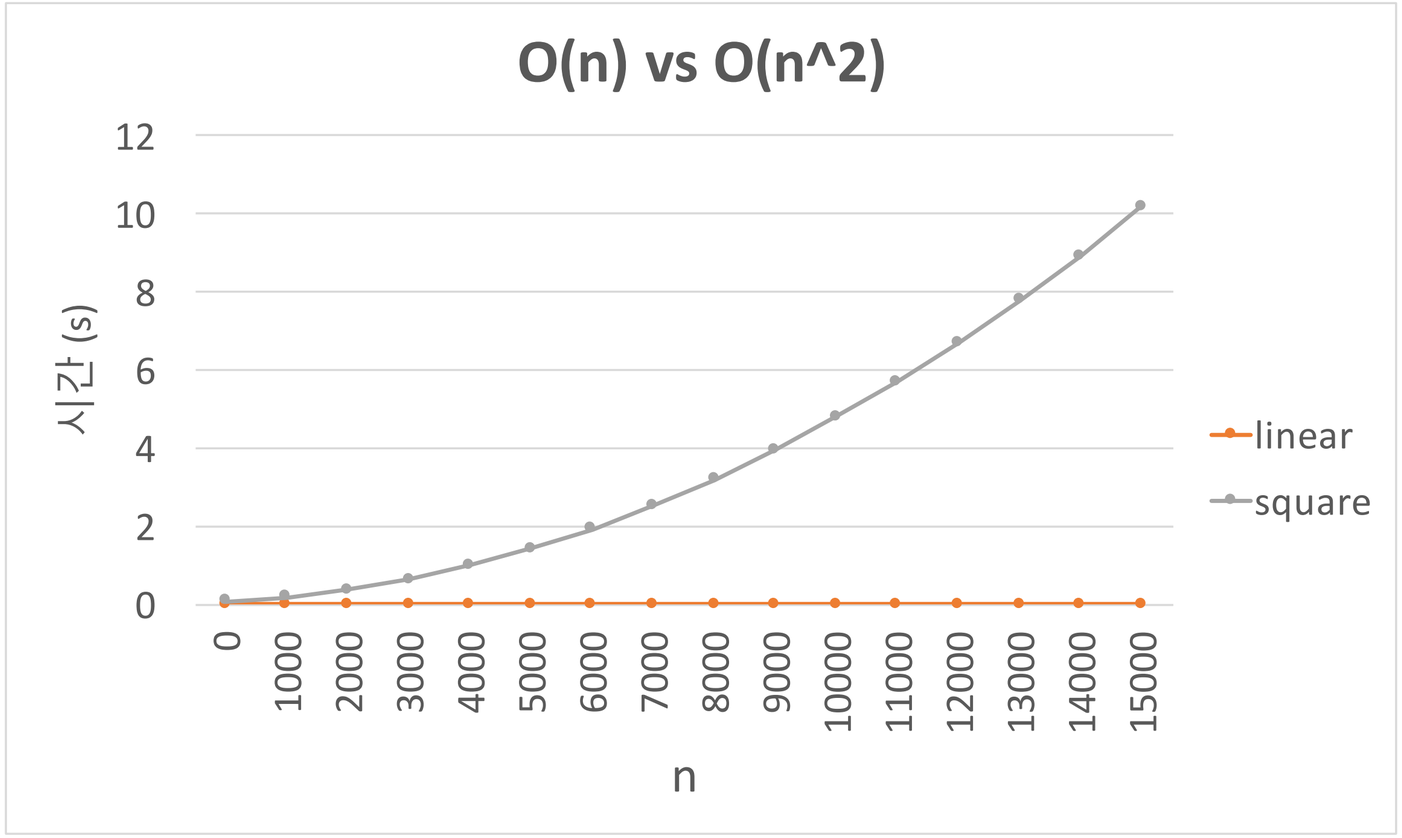
```
def findNumber(myList, target) :
    for v in myList :
        if v == target :
            return True
    return False
```

시간복잡도

Big-O 표기 :

최악의 경우에 수행하는 명령 수

시간복잡도


O(n) vs 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
linear
square

[실습] 올바른 괄호인지 판단하기

짝이 올바른 괄호라면 YES, 아니면 NO

입력의 예	출력의 예
(())()	YES
(())())()	NO

문제 해결의 절차

2. 문제를 해결하는 알고리즘을 개발한다

문제 해결의 절차

3. 알고리즘이 **문제를 해결한다는 것을 증명한다**

문제 해결의 절차

4. 알고리즘이 **제한시간내에 동작한다는 것을 보인다**

완전 탐색

완전 탐색 (Brute-Force)

가능한 모든 경우를 시도해 보는 것

가능한 모든 경우가 무엇인가?

완전 탐색 (Brute-Froce)

가능한 모든 경우를 **전부** 고려해도 괜찮을 경우에는
단순히 모든 경우를 고려함으로써 문제를 해결한다

[실습1] 연속 부분 최대합

연속된 부분을 선택하였을 때, 그 **최대 합**을 출력

단, $1 \leq n \leq 100$

입력의 예

```
1 2 -4 5 3 -2 9 10
```

출력의 예

```
25
```

완전 탐색의 중요성

문제가 주어지면

무.조.건

완전 탐색법으로 먼저 시도해야 한다.

상식적인 문제 해결의 흐름

[실습1] 연속 부분 최대합

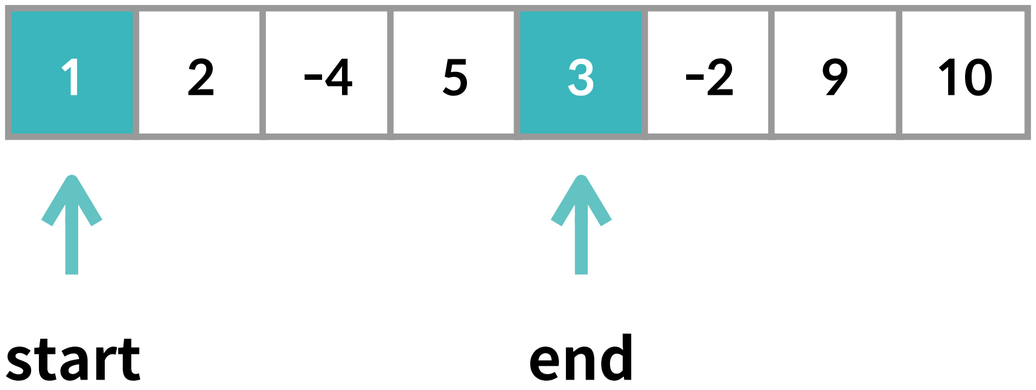
연속된 부분을 선택하였을 때, 그 **최대 합**을 출력

1	2	-4	5	3	-2	9	10

완전 탐색

[실습1] 연속 부분 최대합

연속된 부분을 선택하였을 때, 그 **최대 합**을 출력



[실습2] 멱집합 구하기

n개의 원소를 가지는 집합의 **멱집합** 구하기

단, $1 \leq n \leq 10$

입력의 예

```
3
```

출력의 예

```
1
1 2
1 2 3
1 3
2
2 3
3
```

[실습2] 멱집합 구하기

전체 경우 : 1을 선택하는 경우와 그렇지 않은 경우

`getPowerSet(n, k)` : k를 첫 원소로 갖는 집합을 반환

`getPowerSet(3, 1)`

`getPowerSet(3, 2)`

[실습2] 멱집합 구하기

전체 경우 : 1을 선택하는 경우와 그렇지 않은 경우

`getPowerSet(n, k)` : k를 첫 원소로 갖는 집합을 반환

`getPowerSet(3, 1)` + `getPowerSet(3, 2)` + `getPowerSet(3, 3)`

[실습3] 균형 맞추기

n개의 숫자를 두개의 그룹으로 나누어,

그 **합을 가장 가깝게** 하라

단, $1 \leq n \leq 10$

입력의 예

```
1 -3 4 5 -2
```

출력의 예

```
1
```

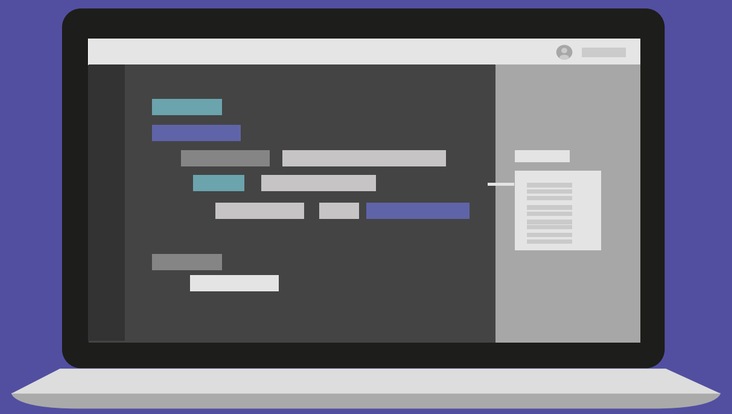
[실습3] 균형 맞추기

n개의 숫자를 두개의 그룹으로 나누어,

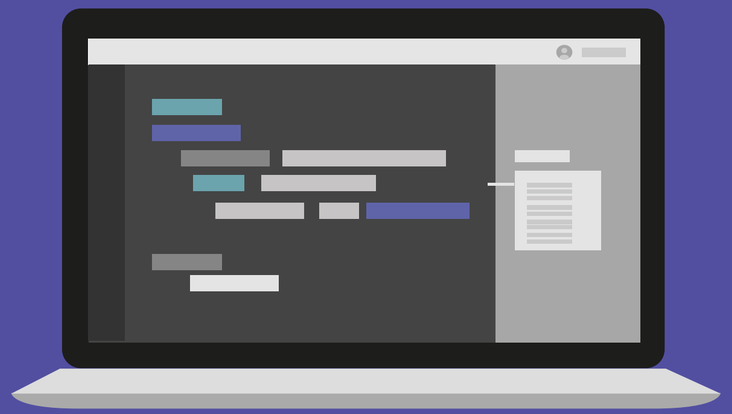
그 **합을 가장 가깝게** 하라

단, $1 \leq n \leq 10$

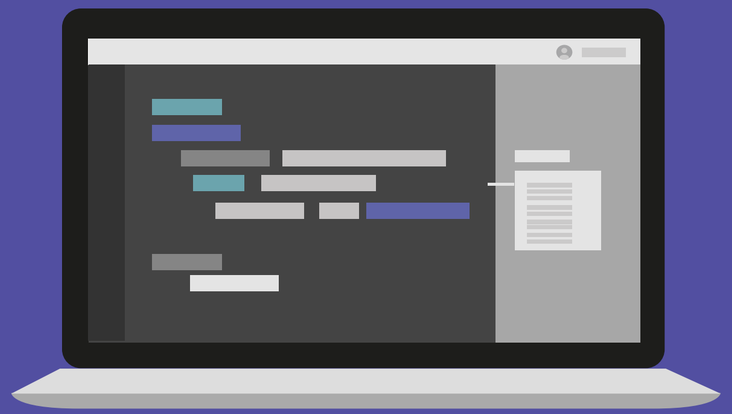
[실습 1] 연속 부분 최대합



[실습 2] 멱집합 구하기



[실습 3] 균형 맞추기



Complexity

Complexity Theory

각 문제마다 풀이의 **시간복잡도**가 다르다

내 풀이가 **얼마나 좋은 풀이**인가?

Complexity Theory

문제 자체에도 복잡도가 존재한다

P class

NP-Complete class

알고리즘 과정에서 다루는 문제들

(거의 대부분) **P 문제**들만을 다룬다

알고리즘에서는 **고려해야 하는 경우**를
줄이는 방법을 배운다

하지만 대표적인 **NP-Complete 문제**는
알면 좋다

요약

요약

문제 **해결 절차**를 잘 따라야 한다

모든 경우를 고려해도 괜찮은 경우에는
모든 경우를 고려하여 해결한다

/* elice */

문의 및 연락처

academy.elice.io

contact@elice.io

facebook.com/elice.io

medium.com/elice