/* elice */

실리콘밸리 문제로 배우는 알고리즘 기초

효율적인 프로그램이란?

박지나 선생님

목차

1. 지난 수업 복습
2. 시간/공간 복잡도 (Big-O)
3. 배열
4. 해쉬
5. 배열과 해쉬의 trade-off

지난 수업 복습

지난 수업 복습

1) 자료구조란?

2) 알고리즘이란?

3) 객체란?

4) 숙제 확인

시간/공간 복잡도

효율성의 측정 방식

시간 복잡도

공간 복잡도

시간 복잡도

알고리즘에 사용되는 총 연산횟수

시간 복잡도 맛보기

단순한 연산 횟수 계산

```
sum = 0;

for i in [1, 2, 3, 4]:

    sum += i;
```

시간 복잡도 =

시간 복잡도 맛보기

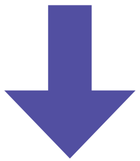
단순한 연산 횟수 계산

```
randomNumber = 0

nums = [1, 2, 3, 4]

for i in range(len(nums)):

     for j in range(len(nums)):

          randomNumber += nums[i] * nums[j]
```

시간 복잡도 =

입력 변수의 크기가 N이라면?

입력 변수의 크기 = **N**



코드의 시간 복잡도 = **f(N)**

N에 대한 복잡도

시간 복잡도 = 1

```
def doNothing(nums):

  return nums
```

N에 대한 복잡도

시간 복잡도 = N + 2

```
def doSomething(nums):
 sum = 0
 for num in nums:
     sum += num
 return sum
```

N에 대한 복잡도

$$\text{시간 복잡도} = 2 \times N^2 + 2$$

```
def doManything(nums):
  allPairs = []
  for i in range(len(nums)):
      for j in range(len(nums)):
        if nums[i] < nums[j]:
          allPairs.append((nums[i], nums[j]))
        else:
          allPairs.append((nums[i], nums[j]))
  return allPairs
```

Big-O 시간 복잡도란?

Big-O = 시간복잡도 함수의 **가장 높은** 차수

$aN + b = O(N)$

$aNlogN + b = O(NlogN)$

$aN^2 + bN + c = O(N^2)$

Big-O 시간 복잡도

시간 복잡도 = 1 / **Big-O 시간 복잡도 =**

```
def doNothing(nums):

  return nums
```

Big-O 시간 복잡도

시간 복잡도 = N + 2 / **Big-O 시간 복잡도 =**

```
def doSomething(nums):
 sum = 0
 for num in nums:
     sum += num
 return sum
```

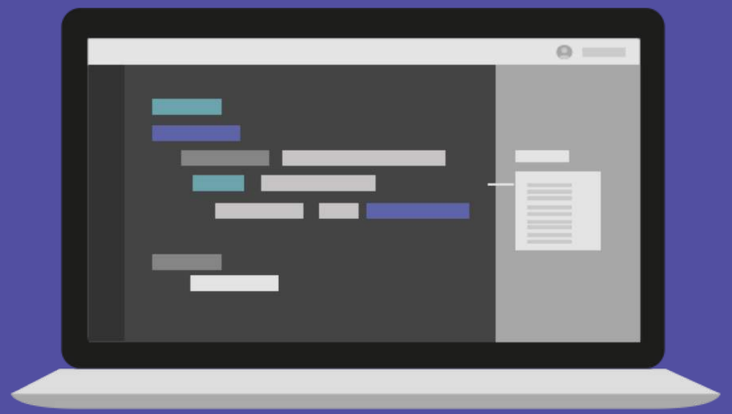
Big-O 시간 복잡도 계산 맛보기

시간 복잡도 = $2 \times N^2 + 2$ / **Big-O 시간 복잡도 =**

```
def doManything(nums):
 allPairs = []
 for i in range(len(nums)):
      for j in range(len(nums)):
       if nums[i] < nums[j]:
        allPairs.append((nums[i], nums[j]))
       else:
        allPairs.append((nums[i], nums[j]))
 return allPairs
```

[실습1]

중복된 수 제거하기

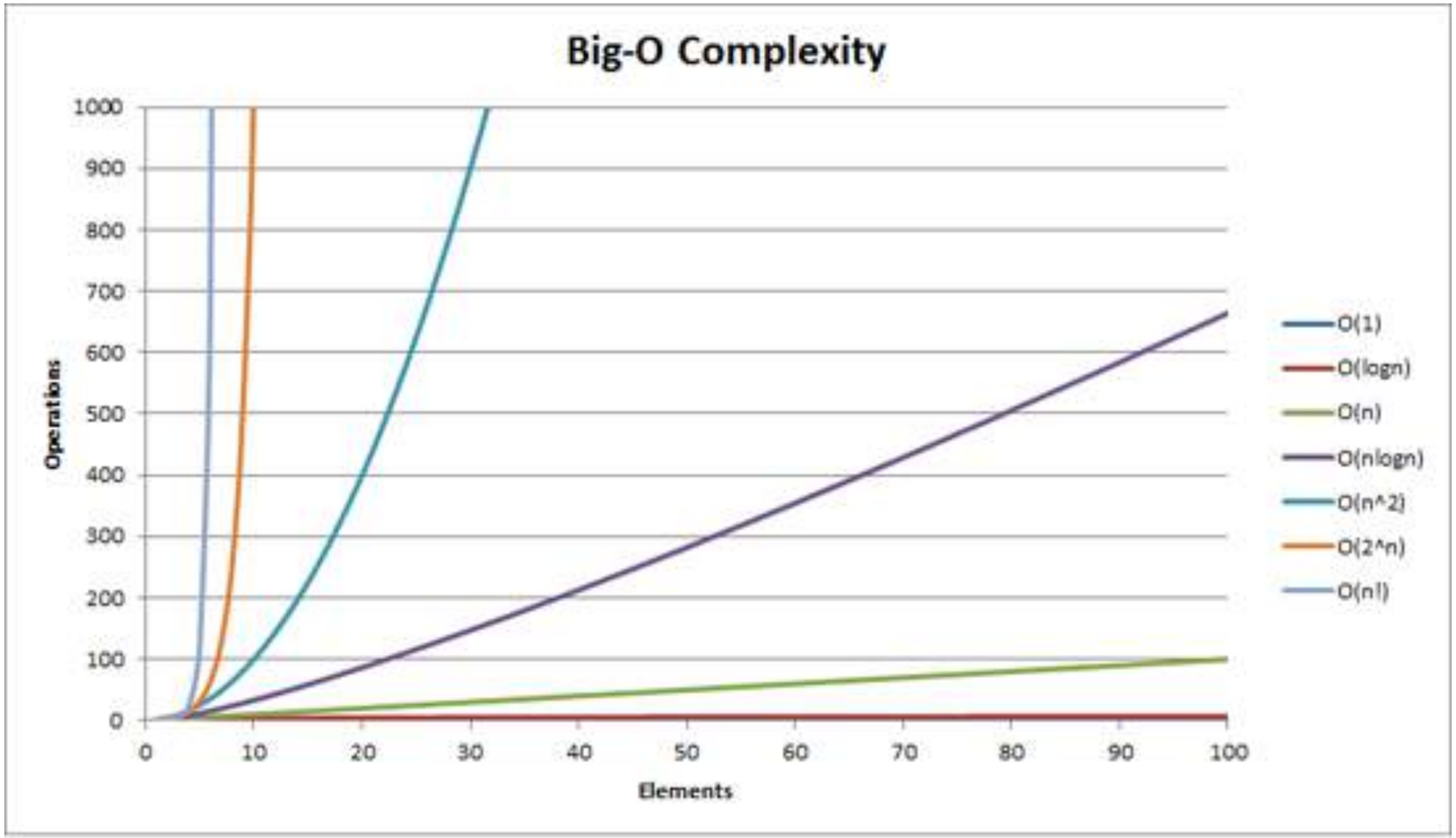


복잡도의 중요성?

	코드 1	코드 2
시간 복잡도	1,000N	$2N^2$
Big-O	O(N)	$O(N^2)$

N	1	10	100	1,000	10,000	…
코드 1	1,000	10,000	100,000	1,000,000	**10,000,000**	…
코드 2	2	200	20,000	2,000,000	**200,000,000**	…

복잡도의 중요성?


Big-O Complexity
Operations
Elements
1000
900
800
700
600
500
400
300
200
100
0
0
10
20
30
40
50
60
70
80
90
100
O(1)
O(logn)
O(n)
O(nlogn)
O(n^2)
O(2^n)
O(n!)

Big-O 시간 복잡도 계산법칙 1

For / while loop가 한 번 중첩될 때마다 O(N)

```
for num in nums:
```

```
for i in range(len(nums)):
 for j in range(len(nums)):
```

```
for i in range(len(nums)):
 for j in range(len(nums)):
  for k in range(len(nums)):
```

Big-O 시간 복잡도 계산법칙 2

자료구조 사용, 다른 함수 호출에는 각각의 O(N)을 파악

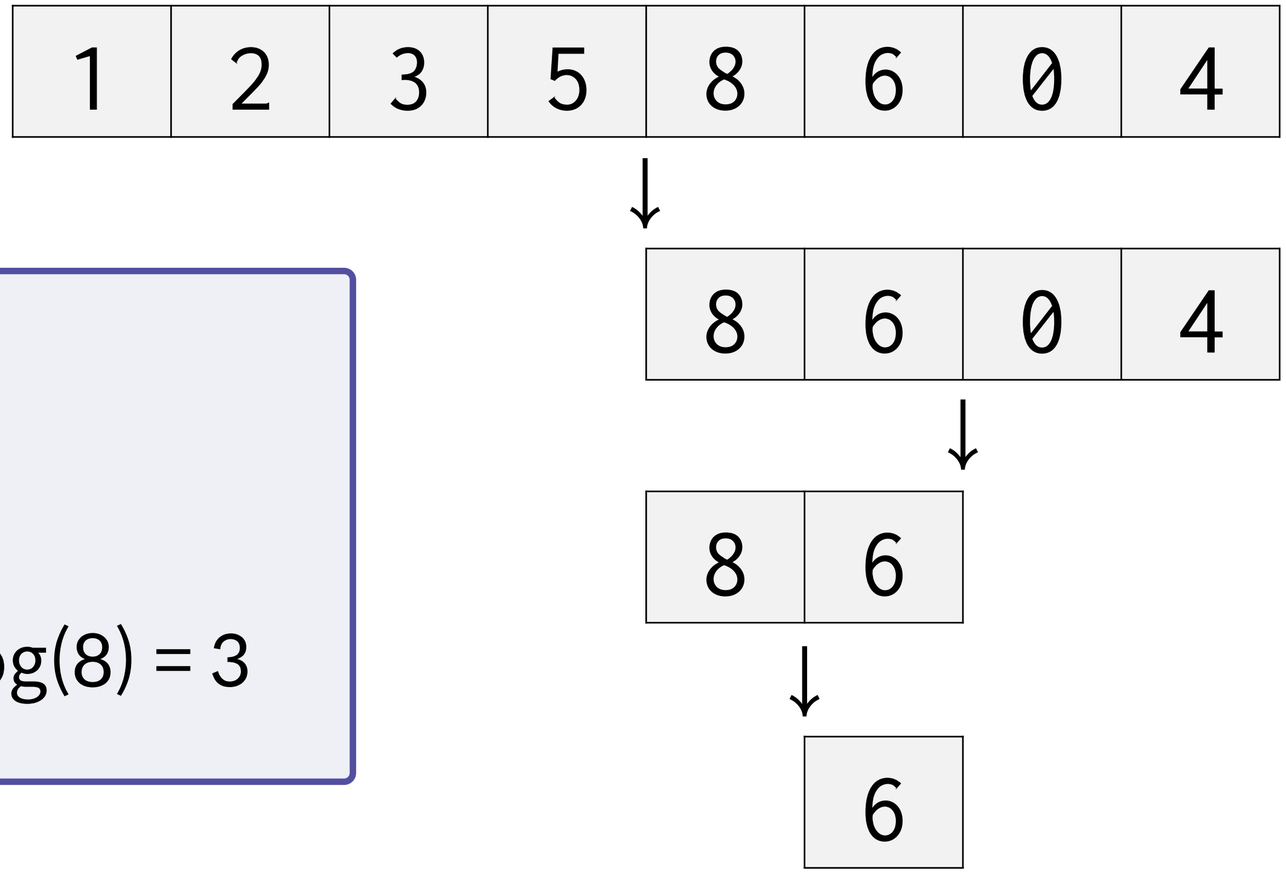
```
nums = [2, 8, 19, 37, 4, 5]
if num in nums:
```

```
nums = {2, 8, 19, 37, 4, 5}
if num in nums:
```

```
nums.sort()
```

Big-O 시간 복잡도 계산법칙 3

매번 절반씩 입력값이 줄어들면 O(logN)



이진 탐색

N = 8

실행 횟수 = log(8) = 3

공간 복잡도

알고리즘에 사용되는

메모리 공간의 총량

공간 복잡도 계산하기

Big-O 공간 복잡도 =

```
a = 1
```

공간 복잡도 계산하기

Big-O 공간 복잡도 =

```
a = [num for num in nums]
```

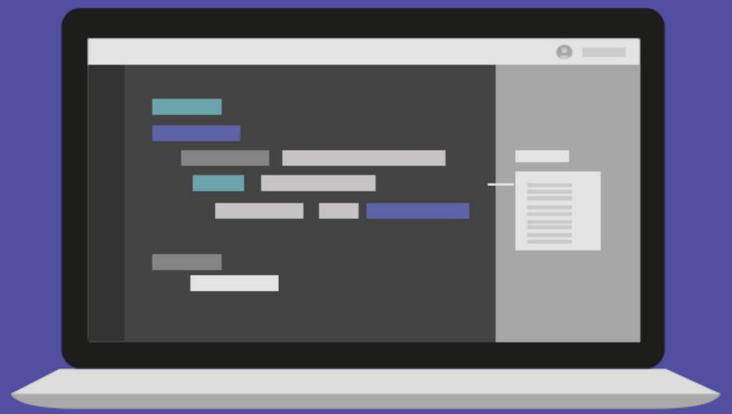
공간 복잡도 계산하기

Big-O 공간 복잡도 =

```
a = [[num for num in nums] for num in nums]
```

[실습2]

0 이동시키기



배열

배열

가장 기본적인 자료 구조

```
nums = [1, 2, 3, 4, 5, 6]
```

배열의 공간 복잡도 = O(N)

배열 : Big-O 시간 복잡도

인덱스를 알 때 : O(1)

```
nums = [1, 2, 3, 4, 5, 6]
```

```
nums[2]
```

배열 : Big-O 시간 복잡도

인덱스를 모를 때 = 하나씩 검사 : O(N)

```
nums = [1, 2, 3, 4, 5, 6]
```

```
if 5 in nums:
```

배열 : Big-O 시간 복잡도

배열 전부 순회하기 : O(N)

```
nums = [1, 2, 3, 4, 5, 6]
```

```
for num in nums:
```

배열 : Big-O 시간 복잡도

자료 끝에 추가하기 : O(1)

```
nums = [1, 2, 3, 4, 5, 6]
```

```
nums.append(7)
```

배열 : Big-O 시간 복잡도

자료 중간에 추가하기 : O(N)

```
nums = [1, 2, 3, 4, 5, 6]
```

```
nums.insert(3, 9)
```

배열 인덱싱

```
nums[2]
```

```
nums[2:5]
```

```
nums[len(nums)-1]
```

```
nums[-1]
```

문자열

배열의 한 종류, 문자들의 배열

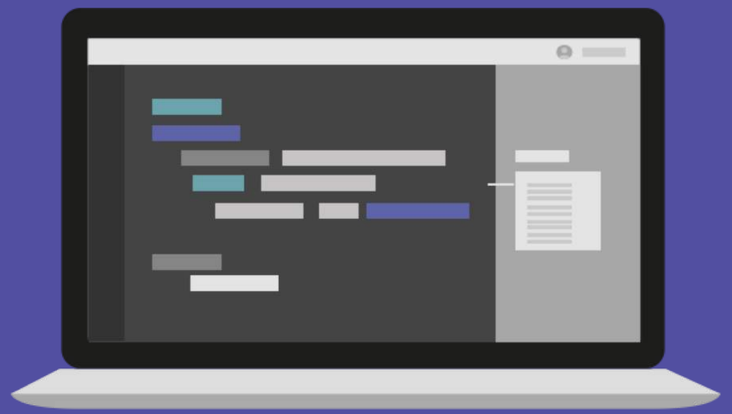
```
tempString = “abcdef”
for ch in tempString:
```

2차원 배열

```
nums = [[1, 2, 3, 4, 5]\
        [6, 7, 8, 9, 10]\
        [11, 12, 13, 14, 15]\
        [16, 17, 18, 19, 20]]
```

[실습3]

배열의 회전



해쉬

해쉬

Dictionary.Key + Value (in Python)

Key는 중복될 수 없음

공간 복잡도는 대략 O(N)

```
studentIds = {
“박지나”:123,
“송호준”:145,
“이주원”:563 }
```

“박지나”	123
“송호준”	145
“이주원”	563
…	…

해쉬 : Big-O 시간 복잡도

Key를 이용해서 Value 가져오기 : 대략 O(1)

```
print(studentIds[“박지나”])
```

해쉬 : Big-O 시간 복잡도

Key가 존재하는지 확인하기 : 대략 O(1)

```
if(“박지나” in studentIds):
if(“손지윤” in studnetIds):
```

해쉬 : Big-O 시간 복잡도

Key, Value 추가하기 : 대략 O(1)

```
studentIds[“손지윤”] = 938
```

해쉬 : Big-O 시간 복잡도

해당 Key의 Value 변경하기 : 대략 O(1)

```
studentIds[“박지나”] = 555
```

해쉬 : 공간 복잡도

해쉬의 공간 복잡도 = O(N)

해쉬는 데이터가 입력되지 않은

여유 공간이 많아야 성능 유지

Set

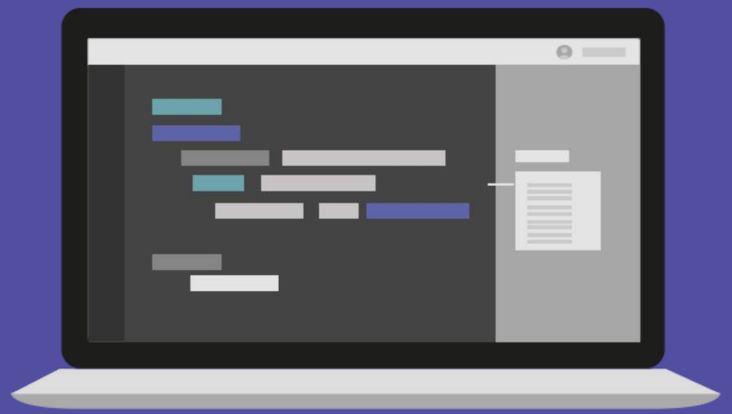
Value 없이 Key만 있는 Dictionary

```
studentNames = {“박지나”, “송호준”, “이주원”, “손지윤”}
```

“박지나”
“송호준”
“이주원”
“손지윤”

[실습4]

아나그램 탐지



배열과 해쉬의 trade-off

배열 VS 해쉬

해쉬

식별자가 있는 데이터, 시간 복잡도 ↓ / 공간 복잡도 ↑

배열

식별자가 없는 데이터, 시간 복잡도 ↑ / 공간 복잡도 ↓

오늘 배운 내용

1) 시간/공간 복잡도란?

2) 효율적인 프로그램이란?

3) 배열과 해쉬의 특징

4) 배열과 해쉬의 trade-off

/* elice */

문의 및 연락처

academy.elice.io

contact@elice.io

facebook.com/elice.io

medium.com/elice