

실전코딩

04. 파이썬 프로그래밍 리뷰 - 알고리즘과 시간복잡도 -

강원대학교 컴퓨터공학과 박치현



Outline

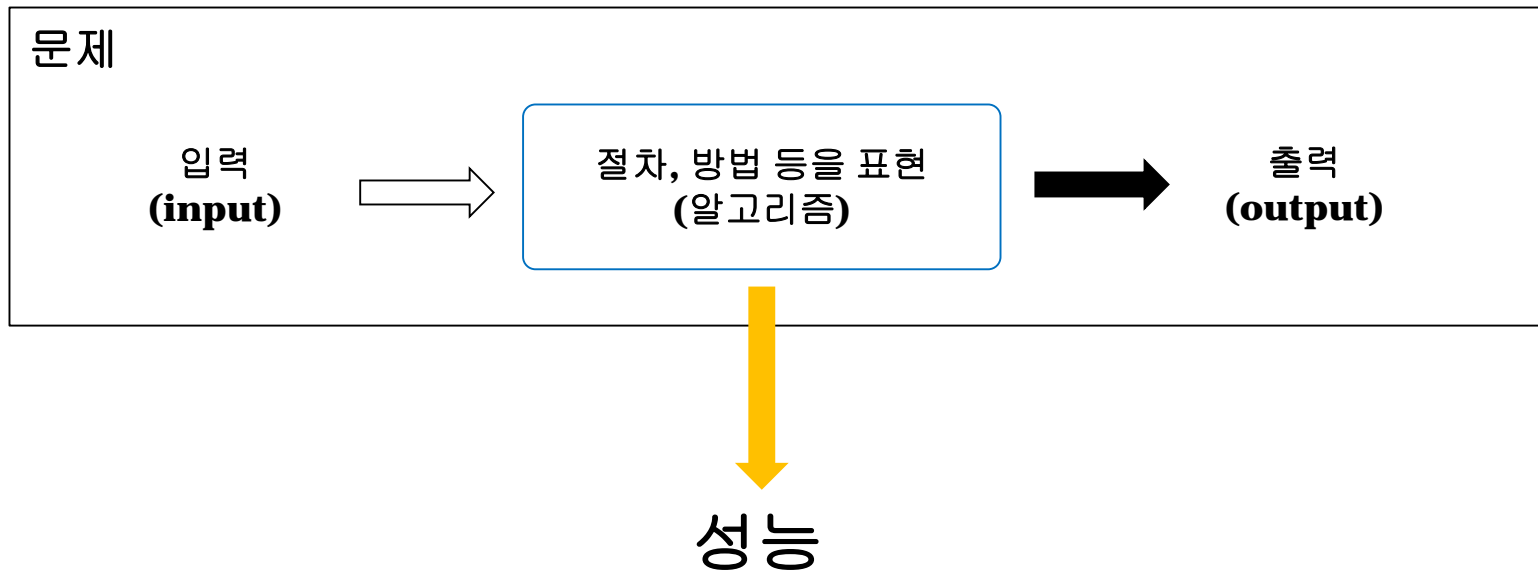
- 알고리즘과 성능
- 시간 복잡도란
- 시간 복잡도 표기법
- 알고리즘 분석



알고리즘과 성능

- 알고리즘

- 문제해결을 위해 정해진 절차, 방법 혹은 과정을 나타내는 것, 계산 실행을 위한 단계적 절차¹⁾



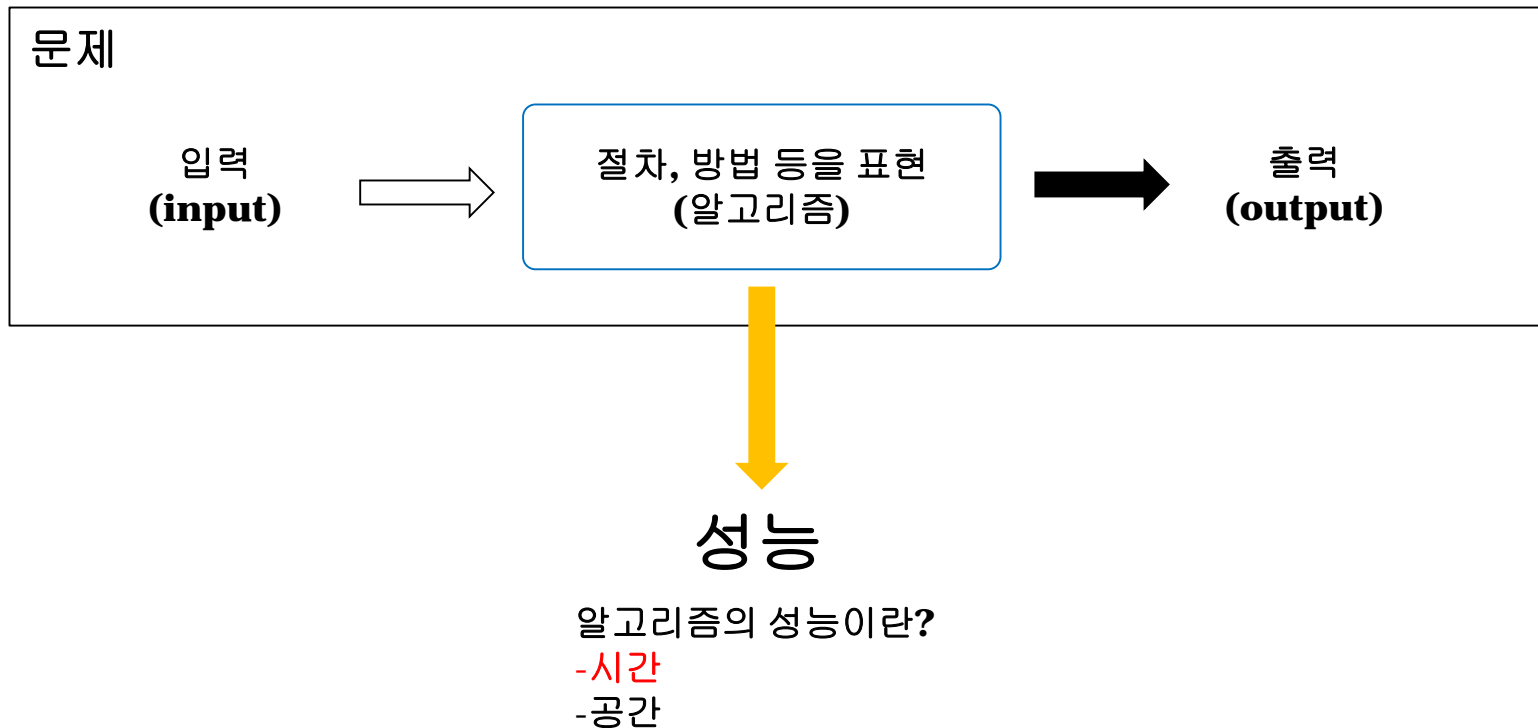
1) <https://ko.wikipedia.org/wiki/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98>



알고리즘과 성능

- 알고리즘

- 문제해결을 위해 정해진 절차, 방법 혹은 과정을 나타내는 것, 계산 실행을 위한 단계적 절차¹⁾



1) <https://ko.wikipedia.org/wiki/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98>

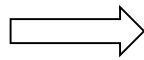


알고리즘과 성능

- 알고리즘

문제 : 입력이 n 일 때, n^2 을 출력하시오.

입력 : n
(input)





절차, 방법 등을 표현
(알고리즘)



출력 : n^2
(output)


`n = 10`


`result = n*n`


`print(result)`

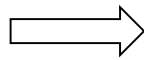


알고리즘과 성능

- 알고리즘

문제 : 입력이 n 일 때, n^2 을 출력하시오.

입력 : n
(input)



절차, 방법 등을 표현
(알고리즘)



출력 : n^2
(output)

$n = 10$

```
result = 0
for i in range(1, n+1):
    result += n
```

`print(result)`

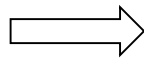


알고리즘과 성능

- 알고리즘

문제 : 입력이 n 일 때, n^2 을 출력하시오.

입력 : n
(input)



절차, 방법 등을 표현
(알고리즘)



출력 : n^2
(output)

$n = 10$

```
result = 0
for i in range(1, n+1):
    for j in range(1, n+1):
        result += 1
```

`print(result)`

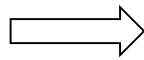


알고리즘과 성능

- 알고리즘

문제 : 입력이 n 일 때, n^2 을 출력하시오.

입력 : n
(input)



절차, 방법 등을 표현
(알고리즘)



출력 : n^2
(output)

방식 1

```
result = n*n
```

방식 2

```
result = 0
for i in range(1, n+1):
    result += n
```

방식 3

```
result = 0
for i in range(1, n+1):
    for j in range(1, n+1):
        result += 1
```

- 성능 : 실행시간?

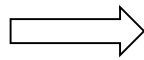


알고리즘과 성능

- 알고리즘

문제 : 입력이 n 일 때, n^2 을 출력하시오.

입력 : n
(input)



절차, 방법 등을 표현
(알고리즘)



출력 : n^2
(output)

- 성능 : 실행시간?

```
import time  
start = time.time()
```

기능을 구현 코드
(문제 해결 코드)

```
end = time.time()  
print("exe time: ", end - start)
```

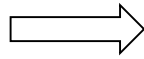


알고리즘과 성능

• 알고리즘

문제 : 입력이 n 일 때, n^2 을 출력하시오.

입력 : n
(input)



절차, 방법 등을 표현
(알고리즘)



출력 : n^2
(output)

- 성능 : 실행시간?

time 모듈의 time() 함수는
현재 unix timestamp을 소수로 리턴
(1970년 1월 1일 0시 0분 0초부터의 경과 시간)

- 정수부는 초단위
- 소수부는 마이크로(micro) 초단위

```
import time
```

```
start = time.time()
```

→ (코드 시작) 현재시간

e.g.) 10:30:00

기능을 구현 코드
(문제 해결 코드)

```
end = time.time()
```

→ (코드 종료) 현재시간

e.g.) 10:30:10

```
print("exe time: ", end - start)
```

e.g.) 10:30:10 - 10:30:00 = 10초

=> 코드 부분의 실행시간은 10초



알고리즘과 성능

- 알고리즘
 - 방식 1 성능 : 실행시간 측정

```
1 import time
2 start = time.time()
3
4 n = 10
5
6 result = n*n
7
8 print("result: ", result)
9
10 end = time.time()
11 print("exe time: ", end - start)
```

실행결과

```
result: 100
exe time: 3.910064697265625e-05
```



알고리즘과 성능

- 알고리즘

- 방식 1 성능 : 실행시간 측정

```
1  import time
2  start = time.time() → (코드 시작) 현재시간
3
4  n = 10
5
6  result = n*n
7
8  print("result: ", result)
9
10 end = time.time() → (코드 종료) 현재시간
11 print("exe time: ", end - start)
```

기능을 구현한 코드

기능코드 실행시간



알고리즘과 성능

- 알고리즘

- 방식 2 성능 : 실행시간 측정

```
1  import time
2  start = time.time()
3
4  n = 10
5
6  result = 0
7  for i in range(1, n+1):
8      result += n
9
10 print("result: ", result)
11
12 end = time.time()
13 print("exe time: ", end - start)
```

실행결과

```
result: 100
exe time: 4.220008850097656e-05
```



알고리즘과 성능

- 알고리즘

- 방식 3 성능 : 실행시간 측정

```
1 import time
2 start = time.time()
3
4 n = 10
5
6 result = 0
7 for i in range(1, n+1):
8     for j in range(1, n+1):
9         result += 1
10
11 print("result: ", result)
12
13 end = time.time()
14 print("exe time: ", end - start)
```

실행결과

```
result: 100
exe time: 5.412101745605469e-05
```



알고리즘과 성능

- 알고리즘

- 실행시간 측정

종류	실행결과
방식 1	result: 100 exe time: 3.910064697265625e-05
방식 2	result: 100 exe time: 4.220008850097656e-05
방식 3	result: 100 exe time: 5.412101745605469e-05

- 실행시간 측정방식의 문제점
 - 하드웨어 스펙, 실험 환경에 따라 다른 결과



시간 복잡도 (Time Complexity)

- 시간 복잡도
 - 코드에서 연산의 횟수를 기반으로 표현한 실행시간
 - 점근 표기법 (**Asymtotic notation**)을 이용하여 나타냄
 - **O (Big-O)**, **Ω (Big-Omega)**, **Θ (Big-Theta)** 표기법이 있음
 - 일반적으로 **O (Big-O) notation**을 가장 많이 사용

점근 표기법: 어떤 함수의 증가 양상을 다른 함수와의 비교로 표현하는 수론과 해석학의 방법
→ 원본 함수를 단순화하여 최고차항의 차수만 고려



시간 복잡도 (Time Complexity)

- 시간 복잡도

방식 1

```
result = n*n
```

방식 2

```
result = 0
for i in range(1, n+1):
    result += n
```

방식 3

```
result = 0
for i in range(1, n+1):
    for j in range(1, n+1):
        result += 1
```

-코드에서 연산의 횟수를 기반으로 표현한 실행시간

대입

	방식 1	방식 2	방식 3
Substitution	1	$n + 1$	$n * n + 1$
Addition		n	$n * n$
Multiplication	1		
Division			
Total	2	$2n + 1$	$2n^2 + 1$



시간 복잡도 (Time Complexity)

- 시간 복잡도

방식 1

result = n*n

result = n*n

n*n

	방식 1
Substitution	1
Addition	
Multiplication	1
Division	
Total	2



시간 복잡도 (Time Complexity)

- 시간 복잡도

방식 2

```
result = 0
for i in range(1, n+1):
    result += n
```

	방식 2
Substitution	$n + 1$
Addition	n
Multiplication	
Division	
Total	$2n + 1$

result = 0 - 1회, result += n - n회

result += n - n회



시간 복잡도 (Time Complexity)

- 시간 복잡도

방식 3

```
result = 0
for i in range(1, n+1):
    for j in range(1, n+1):
        result += 1
```

	방식 2
Substitution	$n * n + 1$
Addition	$n * n$
Multiplication	
Division	
Total	$2n^2 + 1$

result = 0 - 1회, result += 1 - $n * n$ 회

result += 1 - $n * n$ 회



시간 복잡도 (Time Complexity)

- 시간 복잡도

방식 3

```
result = 0
for i in range(1, n+1):
    for j in range(1, n+1):
        result += 1
```

```
for i in range(1, n+1):
```

$i \rightarrow 1$ 일 때,

```
    for j in range(1, n+1):
```

$j \rightarrow 1$ 일 때, result += 1 실행

$j \rightarrow 2$ 일 때, result += 1 실행

.....

$j \rightarrow 10$ 일 때, result += 1 실행

총 10회, 즉 n 회 실행

$i \rightarrow 2$ 일 때,

```
    for j in range(1, n+1):
```

.....

총 10회, 즉 n 회 실행



시간 복잡도 (Time Complexity)

- 시간 복잡도

방식 3

```
result = 0
for i in range(1, n+1):
    for j in range(1, n+1):
        result += 1
```

```
for i in range(1, n+1):
```

i→1 일 때,

result += 1 는 총 10회, 즉 **n**회 실행

i→2 일 때,

result += 1 는 총 10회, 즉 **n**회 실행

i→10 일 때,

result += 1 는 총 10회, 즉 **n**회 실행

n회 실행이 총 **i**의 range 만큼
총 10회, 즉 **n**회 실행
⇒ **n**회만큼 **n**번 실행, i.e.) **n*n**



시간 복잡도 (Time Complexity)

- 시간 복잡도

방식 1

```
result = n*n
```

방식 2

```
result = 0
for i in range(1, n+1):
    result += n
```

방식 3

```
result = 0
for i in range(1, n+1):
    for j in range(1, n+1):
        result += 1
```

	방식 1	방식 2	방식 3
Substitution	1	$n + 1$	$n * n + 1$
Addition		n	$n * n$
Multiplication	1		
Division			
Total	2	$2n + 1$	$2n^2 + 1$
Time Complexity	$O(2)$	$O(n)$	$O(n^2)$



시간 복잡도 표기법

- 차수가 가장 큰 항이 가장 영향을 크게 미치고 다른 항들은 상대적으로 무시될 수 있음
 - 예: $T(n) = n^2 + n + 1$
 - $n=1$ 일때 : $T(n) = 1 + 1 + 1 = 3$ (33.3%)
 - $n=10$ 일때 : $T(n) = 100 + 10 + 1 = 111$ (90%)
 - $n=100$ 일때 : $T(n) = 10000 + 100 + 1 = 10101$ (99%)
 - $n=1,000$ 일때 : $T(n) = 1000000 + 1000 + 1 = 1001001$ (99.9%)

$n=100$ 인 경우

$$T(n) = n^2 + n + 1$$

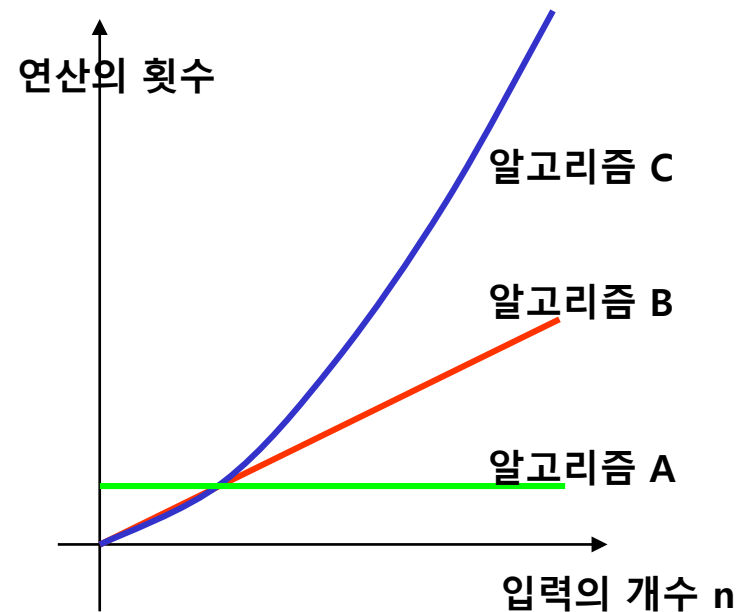
99%

1%



시간 복잡도 표기법

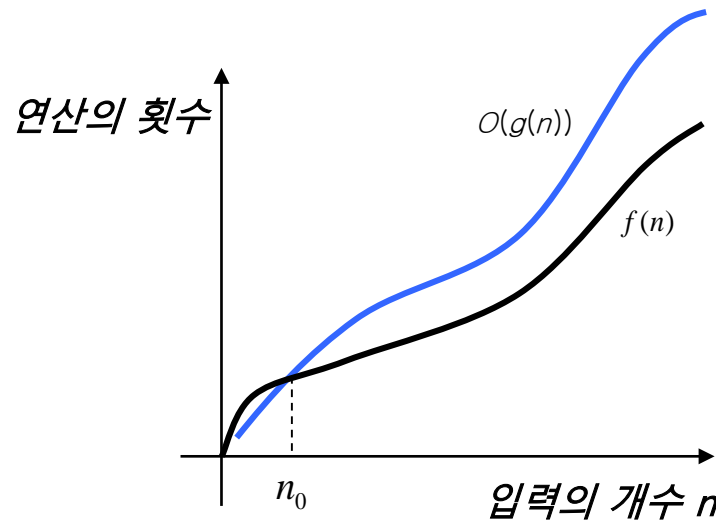
- 시간 복잡도 함수에서 중요한 것은 'n에 대한 연산이 몇 번 필요한지'가 아니라, **n이 증가함에 따라 무엇에 비례하는 수의 연산**이 필요한가이다.
- 오른쪽 그래프에서, 아래 사실이 중요
 - 알고리즘 A: n에 상관없이 동일한 수의 연산
 - 알고리즘 B: n에 비례하는 연산이 필요
 - 알고리즘 C: n^2 에 비례하는 연산이 필요
- 시간 복잡도 함수에서 불필요한 정보를 제거하여 알고리즘 분석을 쉽게 할 목적으로 시간복잡도를 표시하는 방법을 **빅오 표기법**이라 함



시간 복잡도 표기법

빅오 표기법의 수학적 정의

- 두 개의 함수 $f(n)$ 과 $g(n)$ 이 주어졌을 때,
- 모든 $n \geq n_0$ 에 대하여 $|f(n)| \leq c|g(n)|$ 을 만족하는 2개의 상수 c 와 n_0 가 존재하면 $f(n)=O(g(n))$ 이다.
 - 연산의 횟수를 대략적(점근적)으로 표기한 것
 - 빅오는 함수의 상한을 표시한다.
 - (예) $n \geq 5$ 이면 $2n+1 < 10n$ 이므로, $2n+1 = O(n) \rightarrow$ 이때 $c=10, n_0=5$



시간 복잡도 표기법

빅오 표기법의 예

- $f(n)=3n^2+100$ 이면, $O(n^2)$ 이다.
 - $n_0=100, c=5$ 일 때
 - $n \geq 100$ 에 대하여, $3*100^2+100 \leq 5*100^2$ 이 되기 때문이다.



알고리즘 분석

- 시간 복잡도

방식 1

```
result = n*n
```

방식 2

```
result = 0
for i in range(1, n+1):
    result += n
```

방식 3

```
result = 0
for i in range(1, n+1):
    for j in range(1, n+1):
        result += 1
```

	방식 1	방식 2	방식 3
Substitution	1	$n + 1$	$n * n + 1$
Addition		n	$n * n$
Multiplication	1		
Division			
Total	2	$2n + 1$	$2n^2 + 1$
Time Complexity	$O(2)$	$O(n)$	$O(n^2)$



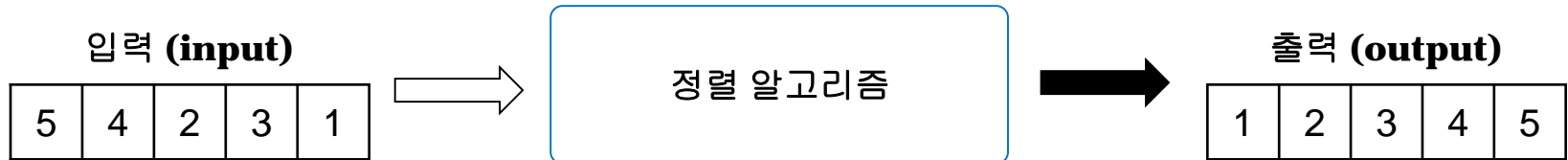
알고리즘 분석

- 선택 정렬 알고리즘 (Selection sort)

- 정렬 (sort) 문제란?

- n 개의 숫자가 입력으로 주어졌을 때 이를 기준에 맞게 정렬하여 출력하는 문제

문제 : 주어진 5개의 숫자를 오름차순(**ascending order**)으로 정렬하시오.

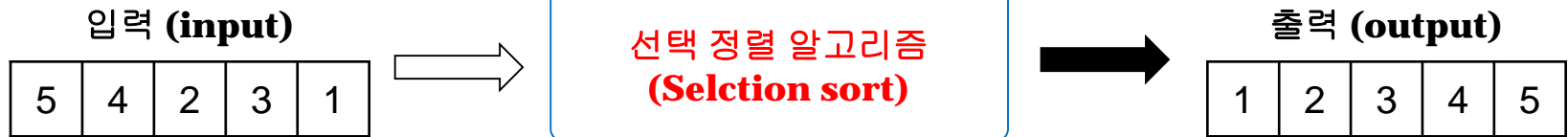


알고리즘 분석

• 선택 정렬 알고리즘 (Selection sort)

- 정렬 (**sort**) 문제란?
 - **n**개의 숫자가 입력으로 주어졌을 때 이를 기준에 맞게 정렬하여 출력하는 문제

문제 : 주어진 5개의 숫자를 오름차순(**ascending order**)으로 정렬하시오.

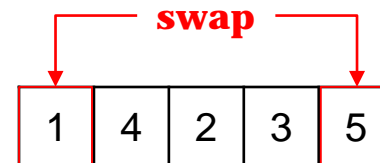
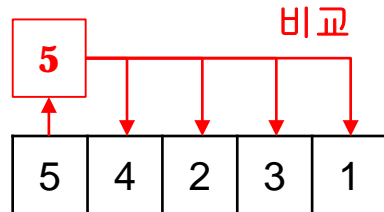


index 0의 값부터 리스트를 순회하며 가장 작은 값을 찾아 위치를 조정(**swap**)하는 방식

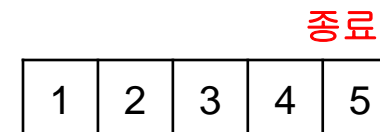
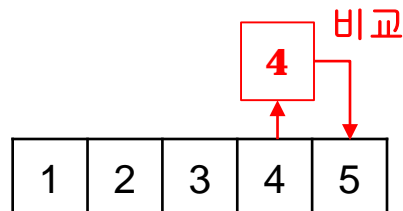
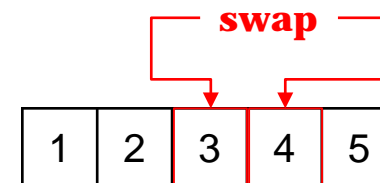
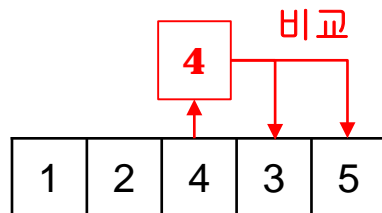
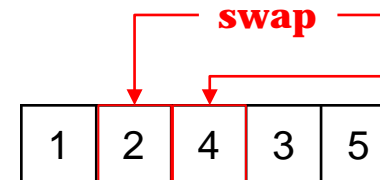
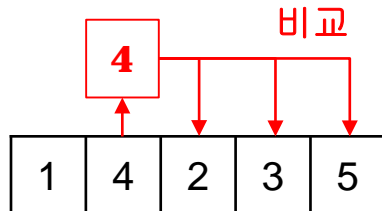


알고리즘 분석

- 선택 정렬 알고리즘 (Selection sort)



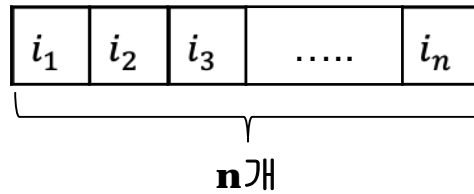
5 vs. [4,2,3,1]
min_idx = 1
min_idx = 2
min_idx = 4
4번째 idx 값이 1과 5를 swap



알고리즘 분석

- 선택 정렬 알고리즘 (Selection sort) 분석

입력 (input)



- 비교연산

$$(n-1) + (n-2) + (n-3) + \dots + 2 + 1$$

$$\Rightarrow n(n-1) / 2 = O(n^2)$$

$$\text{c.f.) } \sum_{i=1}^{n-1} n - i = n(n-1)/2$$



알고리즘 분석

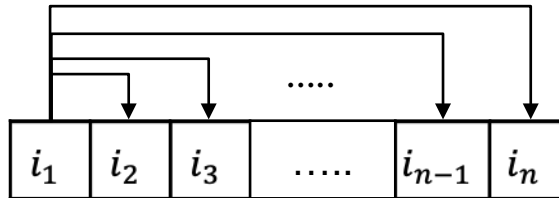
- 선택 정렬 알고리즘 (Selection sort) 분석

- 비교연산

$$(n-1) + (n-2) + (n-3) + \dots + 2 + 1$$

$$\Rightarrow n(n-1) / 2 = O(n^2)$$

i_1 은 자기자신을 제외한 나머지 **n-1**개의 값과 비교연산 진행



알고리즘 분석

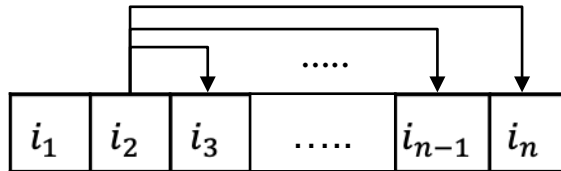
- 선택 정렬 알고리즘 (Selection sort) 분석

- 비교연산

$$(n-1) + (n-2) + (n-3) + \dots + 2 + 1$$

$$\Rightarrow n(n-1) / 2 = O(n^2)$$

i_2 는 자기자신을 제외한 나머지 **n-2**개의 값과 비교연산 진행



알고리즘 분석

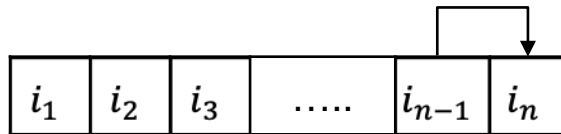
- 선택 정렬 알고리즘 (Selection sort) 분석

- 비교연산

$$(n-1) + (n-2) + (n-3) + \dots + 2 + 1$$

$$\Rightarrow n(n-1) / 2 = O(n^2)$$

i_{n-1} 은 자기자신을 제외한 나머지 1개의 값(i_n)과 비교연산 진행



퀴즈

1. 시간복잡도는 코드에서 연산의 횟수를 기반으로 표현한 실행시간이다.
2. 선택정렬 알고리즘의 시간복잡도는 $O(n^2)$ 이다.

