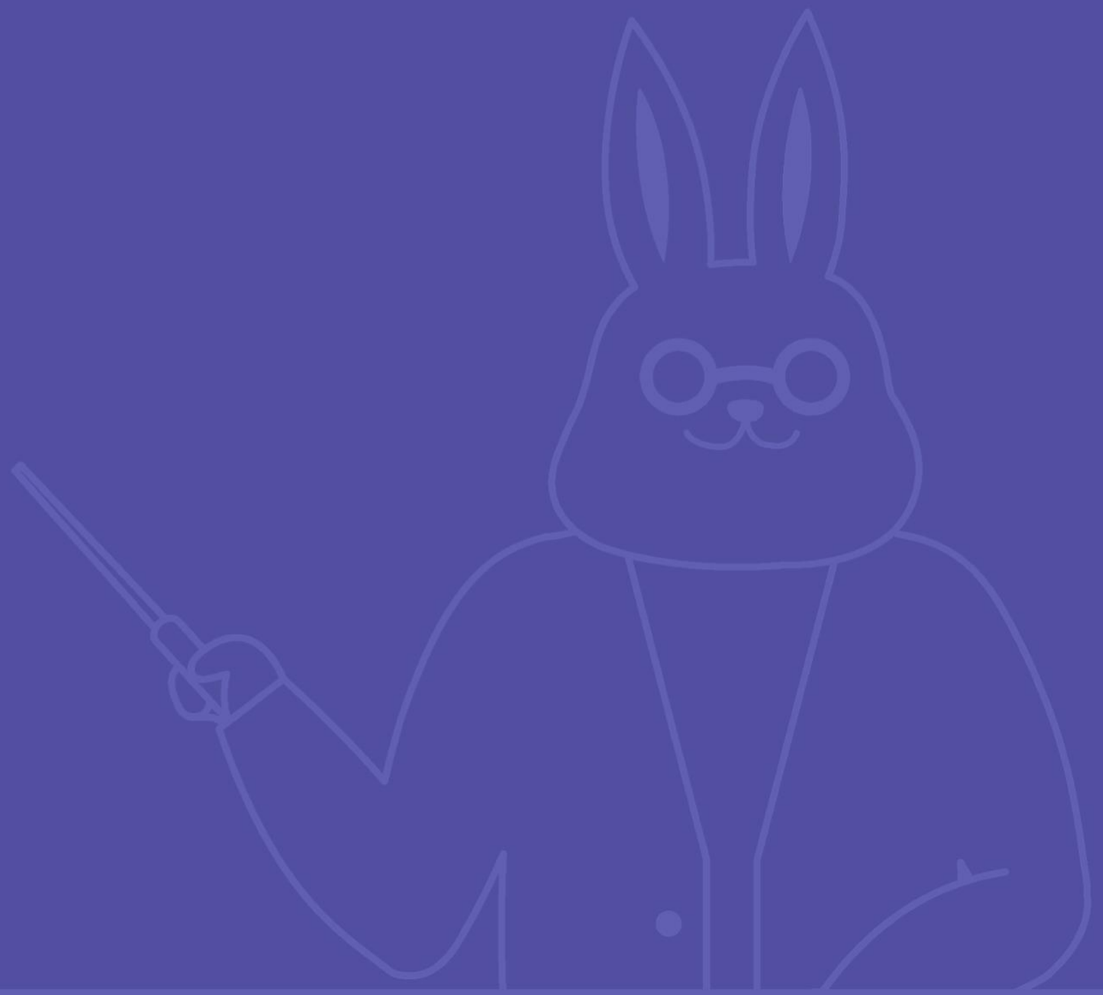
알고리즘

01 6/30 알고리즘 이론 강의

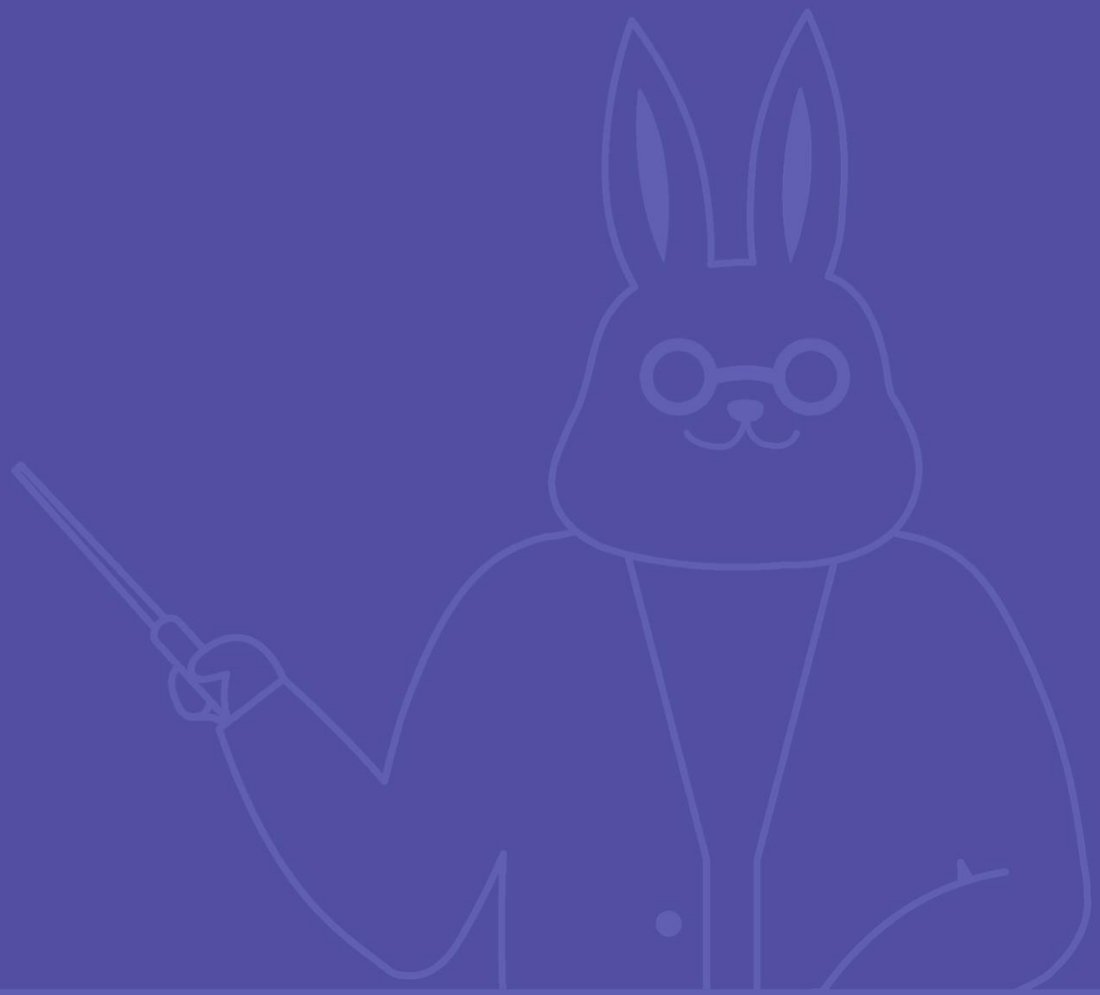


목차

알고리즘도 **컴퓨터 공학부에서도 1학기 전공 필**수 과목입니다.

이번 시간에 모두 이해한다는 것은 거의 불가능하니
개인 시간을 내서라도 꼭 공부해야 하는 과목입니다!

알고리즘이란?



어떤 문제를 해결하기 위한 일련의 방법

어떤 문제를 해결하기 위한 일련의 방법

이라고 설명하기엔 세상에 너무 문제가 많다...

임의로 뿌려져 있는 정보들을 오름차순으로 정렬하기

임의로 뿌려져 있는 정보들을 오름차순으로 정렬하기

지금 위치에서 집까지 가장 빠른 경로를 찾기

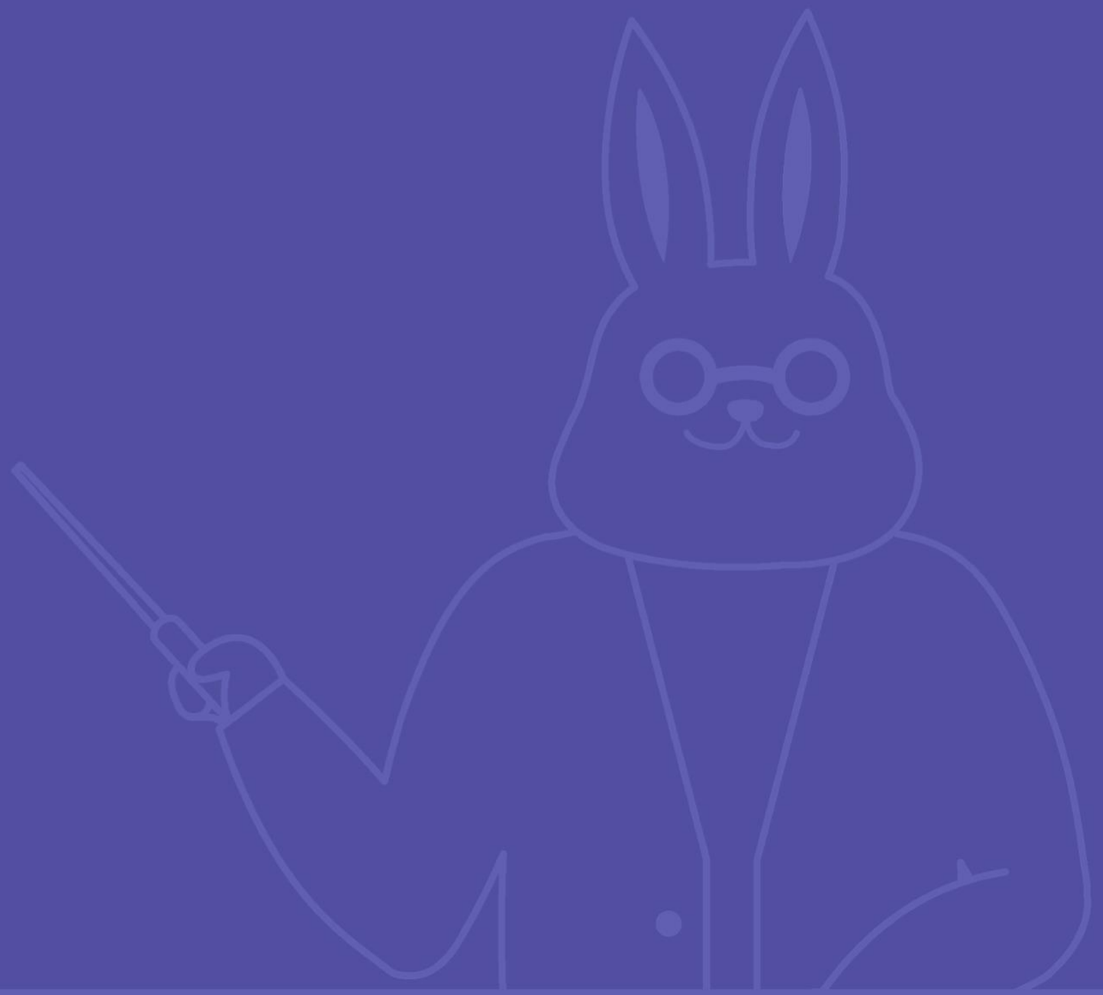
임의로 뿌려져 있는 정보들을 오름차순으로 정렬하기

지금 위치에서 집까지 가장 빠른 경로를 찾기

현재 바둑판에서 가장 승률이 좋은 경우의 수를 찾기

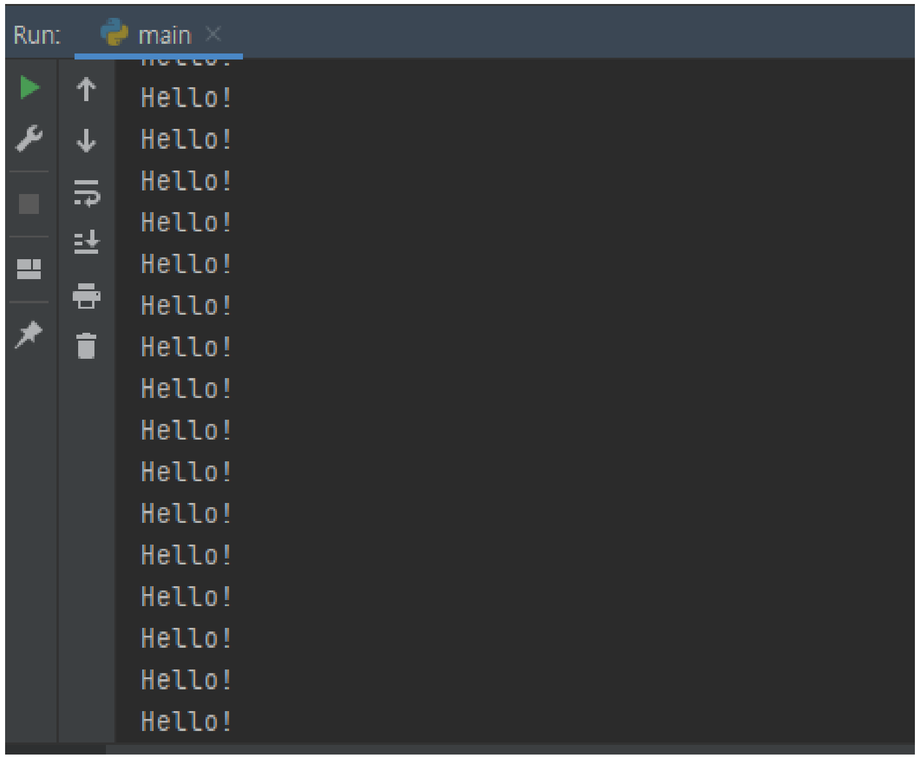
02

재귀함수



자신을 **재참조하는 함**수

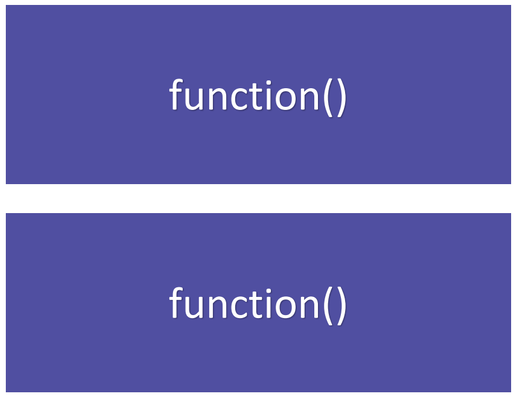
```
1
2    def function():
3        print('Hello!')
4        function()
5
6    function()
7
```


Run:
main
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!

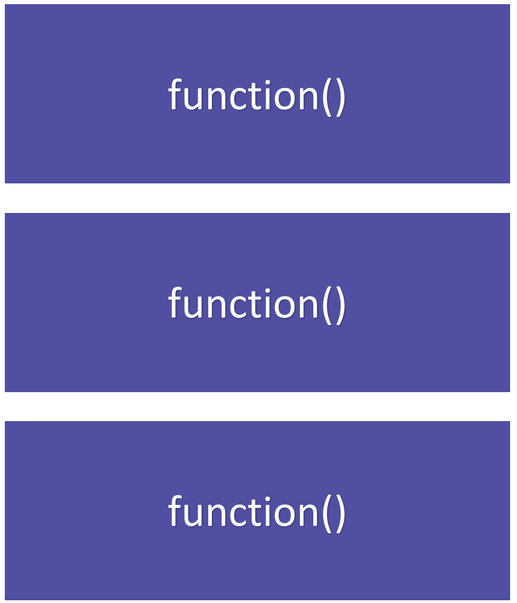
```
1
2   def function():
3       print('Hello!')
4       function()
5
6   function()
7
```

function()

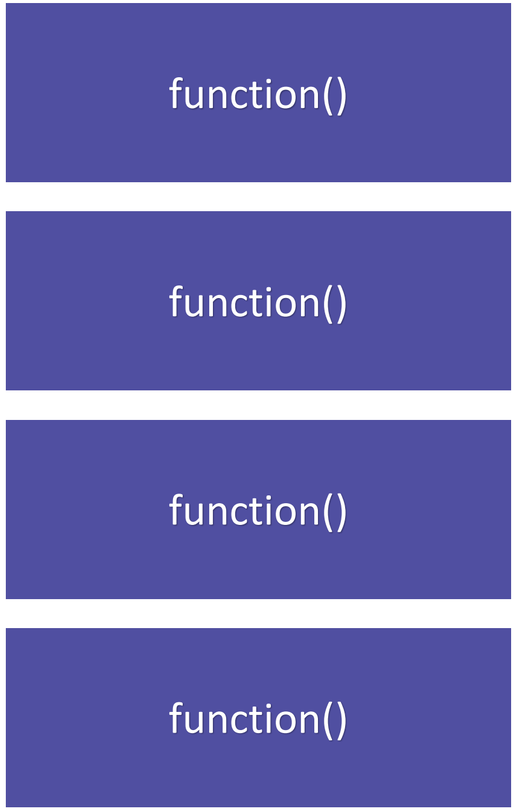
```
1
2   def function():
3       print('Hello!')
4       function()
5
6   function()
7
```



```
1
2   def function():
3       print('Hello!')
4       function()
5
6   function()
7
```



```
1
2   def function():
3       print('Hello!')
4       function()
5
6   function()
7
```



```
어느 한 컴퓨터공학과 학생이 유명한 교수님을 찾아가 물었다.
"재귀함수가 뭔가요?"
"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을 통달한 선인이 있었어.
마을 사람들은 모두 그 선인에게 수많은 질문을 했고, 모두 지혜롭게 대답해 주었지.
그의 답은 대부분 옳았다고 하네. 그런데 어느 날, 그 선인에게 한 선비가 찾아와서 물었어."
____"재귀함수가 뭔가요?"
____"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을 통달한 선인이 있었어.
____마을 사람들은 모두 그 선인에게 수많은 질문을 했고, 모두 지혜롭게 대답해 주었지.
____그의 답은 대부분 옳았다고 하네. 그런데 어느 날, 그 선인에게 한 선비가 찾아와서 물었어."
________"재귀함수가 뭔가요?"
________"재귀함수는 자기 자신을 호출하는 함수라네"
________라고 답변하였지.
____라고 답변하였지.
라고 답변하였지.
```

출처 : 백준 17478번, 재귀함수가 뭔가요?

```
def tab(index):
    ret = ''
    for i in range (0, index):
        ret += '  '
    return ret

def function(index):
    if index == 10:
        return
    print(tab(index), '"재귀함수가 뭔가요?"')
    print(tab(index), '"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을 통달한 선인이 있었어. 마을 사람들은 모
    function(index + 1)
    print(tab(index), '라고 답변하셨지.')

print(' 어느 한 컴퓨터공학과 학생이 유명한 교수님을 찾아가 물었다.')
function(0)
```

```python
def pibonacci(n):
    if n == 1:
        return 1
    return n + pibonacci(n-1)


print(pibonacci(1))
print(pibonacci(2))
print(pibonacci(3))
print(pibonacci(4))
print(pibonacci(5))
print(pibonacci(6))
```

재귀함수로 구현한 피보나치 수열

pibonacci(5)

```python
def pibonacci(n):
    if n == 1:
        return 1
    return n + pibonacci(n-1)


print(pibonacci(1))
print(pibonacci(2))
print(pibonacci(3))
print(pibonacci(4))
print(pibonacci(5))
print(pibonacci(6))
```

```python
def pibonacci(n):
    if n == 1:
        return 1
    return n + pibonacci(n-1)


print(pibonacci(1))
print(pibonacci(2))
print(pibonacci(3))
print(pibonacci(4))
print(pibonacci(5))
print(pibonacci(6))
```

pibonacci(5)

- 5 + pibonacci(4)

```python
def pibonacci(n):
    if n == 1:
        return 1
    return n + pibonacci(n-1)


print(pibonacci(1))
print(pibonacci(2))
print(pibonacci(3))
print(pibonacci(4))
print(pibonacci(5))
print(pibonacci(6))
```

pibonacci(5)

- 5 + pibonacci(4)

- 5 + 4 + pibonacci(3)

```python
def pibonacci(n):
    if n == 1:
        return 1
    return n + pibonacci(n-1)


print(pibonacci(1))
print(pibonacci(2))
print(pibonacci(3))
print(pibonacci(4))
print(pibonacci(5))
print(pibonacci(6))
```

pibonacci(5)

- 5 + pibonacci(4)

- 5 + 4 + pibonacci(3)

- 5+ 4 + 3 + pibonacci(2)

```python
def pibonacci(n):
    if n == 1:
        return 1
    return n + pibonacci(n-1)


print(pibonacci(1))
print(pibonacci(2))
print(pibonacci(3))
print(pibonacci(4))
print(pibonacci(5))
print(pibonacci(6))
```

pibonacci(5)

- 5 + pibonacci(4)

- 5 + 4 + pibonacci(3)

- 5+ 4 + 3 + pibonacci(2)

- 5 + 4 + 3 + 2 + pibonacci(1)

```python
def pibonacci(n):
    if n == 1:
        return 1
    return n + pibonacci(n-1)


print(pibonacci(1))
print(pibonacci(2))
print(pibonacci(3))
print(pibonacci(4))
print(pibonacci(5))
print(pibonacci(6))
```

pibonacci(5)

- 5 + pibonacci(4)

- 5 + 4 + pibonacci(3)

- 5+ 4 + 3 + pibonacci(2)

- 5 + 4 + 3 + 2 + pibonacci(1)

- 5 + 4 + 3 + 2 + 1

```python
def factorial(n):
    if n == 1:
        return 1
    return n * factorial(n-1)


print(factorial(1))
print(factorial(2))
print(factorial(3))
print(factorial(4))
print(factorial(5))
print(factorial(6))
```

재귀함수로 구현한 팩토리얼 함수

재귀함수의 동작 분석

함수가 다시 나를 부를때, 미처 실행되지 않은 다음 코드들은

언젠가 실행될 예정!

재귀함수의 동작 분석

함수가 다시 나를 부를때, 미처 실행되지 않은 다음 코드들은

언젠가 실행될 예정!

```
어느 한 컴퓨터공학과 학생이 유명한 교수님을 찾아가 물었다.
"재귀함수가 뭔가요?"
"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을 통달한 선인이 있었어.
마을 사람들은 모두 그 선인에게 수많은 질문을 했고, 모두 지혜롭게 대답해 주었지.
그의 답은 대부분 옳았다고 하네. 그런데 어느 날, 그 선인에게 한 선비가 찾아와서 물었어."
____"재귀함수가 뭔가요?"
____"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을 통달한 선인이 있었어.
____마을 사람들은 모두 그 선인에게 수많은 질문을 했고, 모두 지혜롭게 대답해 주었지.
____그의 답은 대부분 옳았다고 하네. 그런데 어느 날, 그 선인에게 한 선비가 찾아와서 물었어."
_________"재귀함수가 뭔가요?"
_________"재귀함수는 자기 자신을 호출하는 함수라네"
_________라고 답변하였지.
____라고 답변하였지.
라고 답변하였지.
```

재귀함수의 동작 분석

```
def function(index):
    if index == 10:
        return
    print(tab(index), '"재귀함수가 뭔가요?"')
    print(tab(index), '"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을
    function(index + 1)
    print(tab(index), '라고 답변하셨지.')
```

재귀함수의 동작 분석

실행

```
def function(index):
    if index == 10:
        return
    print(tab(index), '"재귀함수가 뭔가요?"')
    print(tab(index), '"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을
    function(index + 1)
    print(tab(index), '라고 답변하셨지.')
```

재귀함수가 뭔가요?
잘 들어보게...

재귀함수의 동작 분석

```
def function(index):
    if index == 10:
        return
    print(tab(index), '"재귀함수가 뭔가요?"')
    print(tab(index), '"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을
    function(index + 1)
    print(tab(index), '라고 답변하셨지.')
```

재귀함수가 뭔가요?
잘 들어보게...

재귀함수의 동작 분석

```
def function(index):
    if index == 10:
        return
    print(tab(index), '"재귀함수가 뭔가요?"')
    print(tab(index), '"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을
    function(index + 1)
    print(tab(index), '라고 답변하셨지.')
```

재귀함수가 뭔가요?
잘 들어보게...

```
def function(index):
    if index == 10:
        return
    print(tab(index), '"재귀함수가 뭔가요?"')
    print(tab(index), '"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을
    function(index + 1)
    print(tab(index), '라고 답변하셨지.')
```

재귀함수의 동작 분석

```
def function(index):
    if index == 10:
        return
    print(tab(index), '"재귀함수가 뭔가요?"')
    print(tab(index), '"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을
    function(index + 1)
    print(tab(index), '라고 답변하셨지.')
```

재귀함수가 뭔가요?
잘 들어보게...

    재귀함수가 뭔가요?
    잘 들어보게...

실행

```
def function(index):
    if index == 10:
        return
    print(tab(index), '"재귀함수가 뭔가요?"')
    print(tab(index), '"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을
    function(index + 1)
    print(tab(index), '라고 답변하셨지.')
```

재귀함수의 동작 분석

```
def function(index):
    if index == 10:
        return
    print(tab(index), '"재귀함수가 뭔가요?"')
    print(tab(index), '"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을
    function(index + 1)
    print(tab(index), '라고 답변하셨지.')
```

재귀함수가 뭔가요?
잘 들어보게...

재귀함수가 뭔가요?
잘 들어보게...

재귀함수가 뭔가요?
잘 들어보게...

```
def function(index):
    if index == 10:
        return
    print(tab(index), '"재귀함수가 뭔가요?"')
    print(tab(index), '"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을
    function(index + 1)
    print(tab(index), '라고 답변하셨지.')
```

실행

```
def function(index):
    if index == 10:
        return
    print(tab(index), '"재귀함수가 뭔가요?"')
    print(tab(index), '"잘 들어보게. 옛날옛날 한 산 꼭대기
    function(index + 1)
    print(tab(index), '라고 답변하셨지.')
```

재귀함수의 동작 분석

```
def function(index):
    if index == 10:
        return
    print(tab(index), '"재귀함수가 뭔가요?"')
    print(tab(index), '"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을
    function(index + 1)
    print(tab(index), '라고 답변하셨지.')
```

재귀함수가 뭔가요?
잘 들어보게...

재귀함수가 뭔가요?
잘 들어보게...

재귀함수가 뭔가요?
잘 들어보게...

이 녀석들은...?

```
def function(index):
    if index == 10:
        return
    print(tab(index), '"재귀함수가 뭔가요?"')
    print(tab(index), '"잘 들어보게. 옛날옛날 한 산 꼭대기
    function(index + 1)
    print(tab(index), '라고 답변하셨지.')
```

재귀함수의 동작 분석

```
def function(index):
    if index == 10:
        return
    print(tab(index), '"재귀함수가 뭔가요?"')
    print(tab(index), '"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을
    function(index + 1)
    print(tab(index), '라고 답변하셨지.')
```

재귀함수가 뭔가요?
잘 들어보게...

재귀함수가 뭔가요?
잘 들어보게...

재귀함수가 뭔가요?
잘 들어보게...

```
def function(index):
    if index == 10:
        return
    print(tab(index), '"재귀함수가 뭔가요?"')
    print(tab(index), '"def function(index):
    function(index + 1)
    print(tab(index), '
        if index == 10:
            return
        print(tab(index), '"재귀함수가 뭔가요?"')
        print(tab(index), '"잘 들어보게. 옛날옛날 한 산 꼭대기
        function(index + 1)
        print(tab(index), '라고 답변하셨지.')
```

재귀함수의 동작 분석

```
def function(index):
    if index == 10:
        return
    print(tab(index), '"재귀함수가 뭔가요?"')
    print(tab(index), '"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을
    function(index + 1)
    print(tab(index), '라고 답변하셨지.')
```

재귀함수가 뭔가요?
잘 들어보게...

재귀함수가 뭔가요?
잘 들어보게...

재귀함수가 뭔가요?
잘 들어보게...

실행 ↓

```
def function(index):
    if index == 10:
        return
    print(tab(index), '"재귀함수가 뭔가요?"')
    print(tab(index), '"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을
    function(index + 1)
    print(tab(index), '라고 답변하셨지.')
```

재귀함수의 동작 분석

```
def function(index):
    if index == 10:
        return
    print(tab(index), '"재귀함수가 뭔가요?"')
    print(tab(index), '"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을
    function(index + 1)
    print(tab(index), '라고 답변하셨지.')
```

재귀함수가 뭔가요?
잘 들어보게...

재귀함수가 뭔가요?
잘 들어보게...

재귀함수가 뭔가요?
잘 들어보게...
라고 답변하셨지.

실행

```
def function(index):
    if index == 10:
        return
    print(tab(index), '"재귀함수가 뭔가요?"')
    print(tab(index), '"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을
    function(index + 1)
    print(tab(index), '라고 답변하셨지.')
```

재귀함수의 동작 분석

```
def function(index):
    if index == 10:
        return
    print(tab(index), '"재귀함수가 뭔가요?"')
    print(tab(index), '"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을
    function(index + 1)
    print(tab(index), '라고 답변하셨지.')
```

실행

재귀함수가 뭔가요?
잘 들어보게...

재귀함수가 뭔가요?
잘 들어보게...

재귀함수가 뭔가요?
잘 들어보게...
라고 답변하셨지.

라고 답변하셨지.

유클리드 호제법

두 양의 정수 a, b(a>b)에 대하여, a = bq + r (0 <= r < b)라 하면,

a, b의 최대공약수는 b, r의 최대공약수와 같다.

gcd(a, b) = gcd(b, r)

r = 0이면, a, b의 최대 공약수는 b이다.

12345와 1234의 최대공약수?

12345와 1234의 최대공약수?

12345 = 1234 * 10 + 5

12345와 1234의 최대공약수?

12345 = 1234 * 10 + 5

1234 = 5 * 246 + 4

12345와 1234의 최대공약수?

12345 = 1234 * 10 + 5

1234 = 5 * 246 + 4

5 = 4 * 1 + 1

12345와 1234의 최대공약수?

12345 = 1234 * 10 + 5

1234 = 5 * 246 + 4

5 = 4 * 1 + 1

4 = 1 * 4

-> gcd(12345, 1234) = 1

35와 14의 최대공약수?

35와 14의 최대공약수?

35 = 14 * 2 + 7

35와 14의 최대공약수?

35 = 14 * 2 + 7

14 = 7 * 2

-> gcd(35, 14) = 7

유클리드 호제법

```
1  def gcd(a, b):
2      if b == 0:
3          return a
4      else:
5          return gcd(b, a % b)
6
7  print(gcd(4, 10))
8  print(gcd(3, 10))
```

35와 14의 최대공약수?

```python
def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)

print(gcd(4, 10))
print(gcd(3, 10))
```

35와 14의 최대공약수?

gcd(35, 14)

```python
def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)

print(gcd(4, 10))
print(gcd(3, 10))
```

35와 14의 최대공약수?

gcd(35, 14)
- gcd(14, 7)

```python
def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)

print(gcd(4, 10))
print(gcd(3, 10))
```

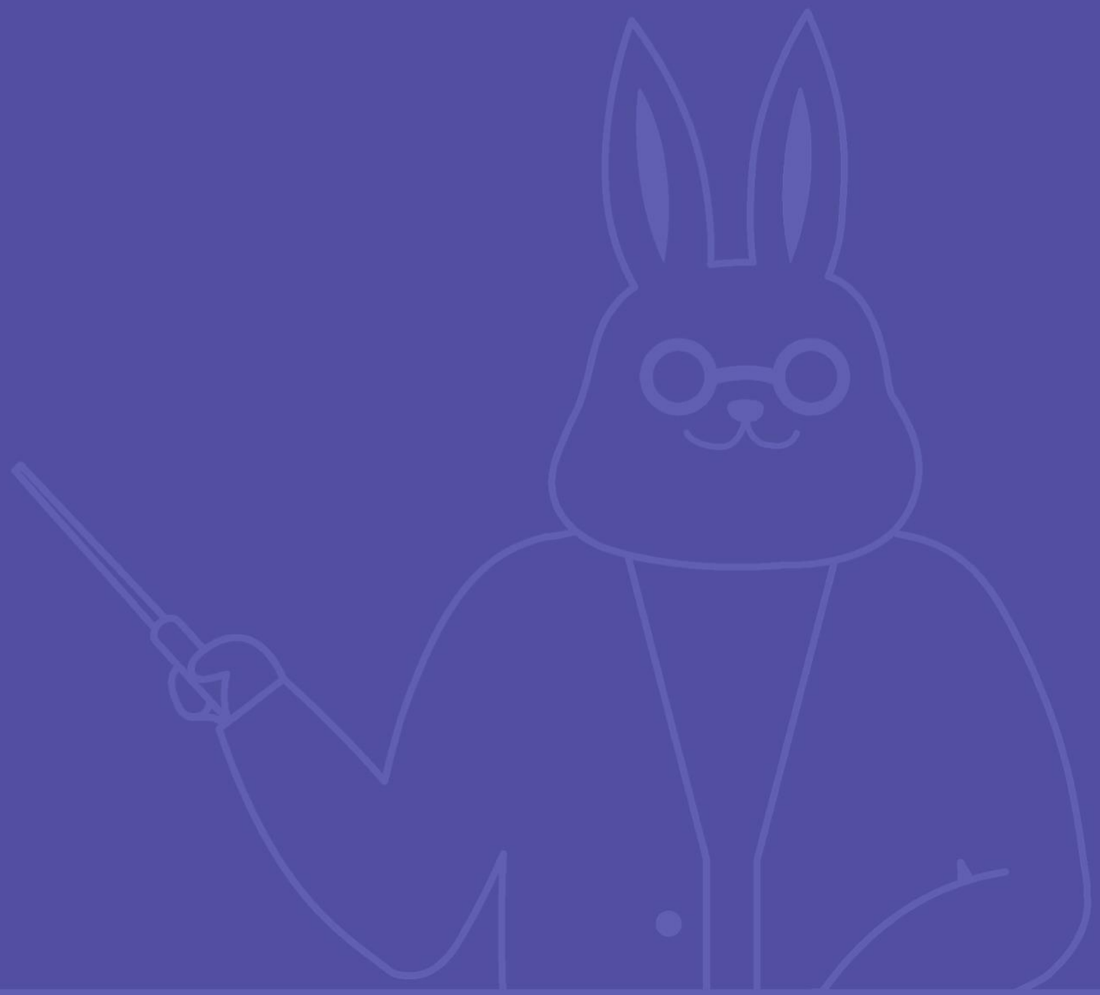
35와 14의 최대공약수?

```python
def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)

print(gcd(4, 10))
print(gcd(3, 10))
```

gcd(35, 14)
- gcd(14, 7)
  - gcd(7, 0) = 7

03

정렬

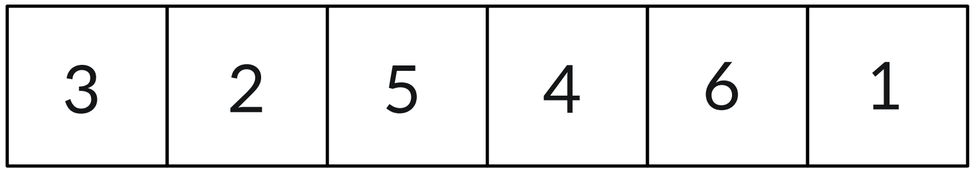


배열을 정렬하는 방법에 대해 알아보자

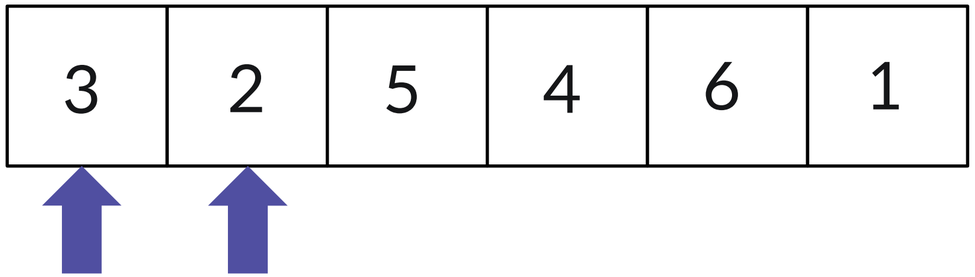
버블 소트

가장 기본적인 배열의 정렬 방법

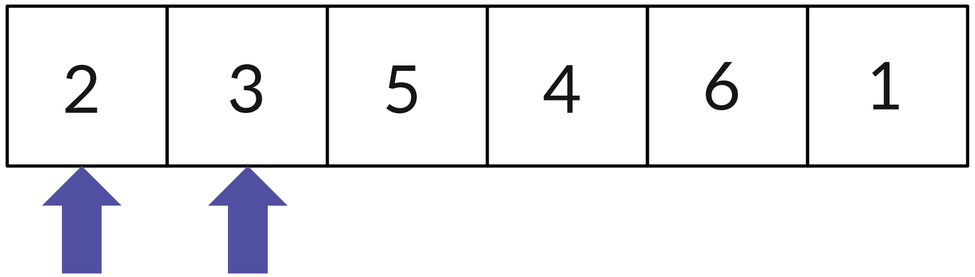
버블 소트



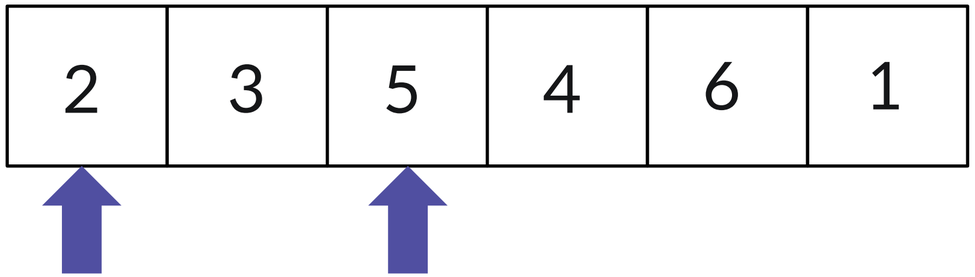
버블 소트



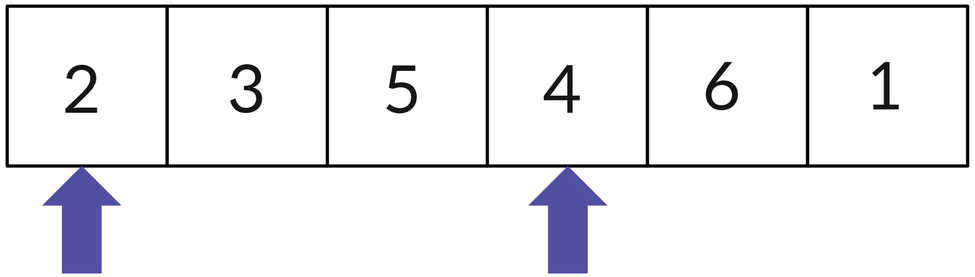
버블 소트



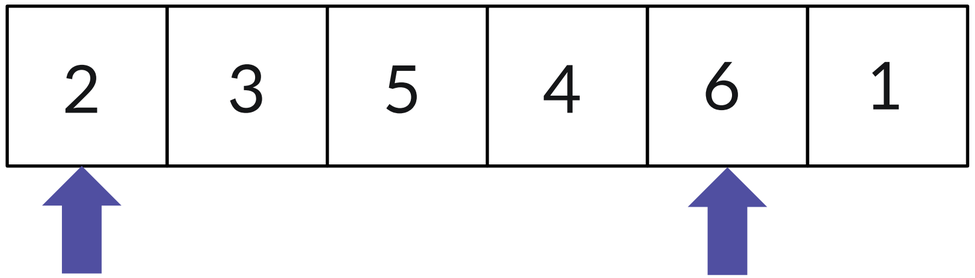
버블 소트



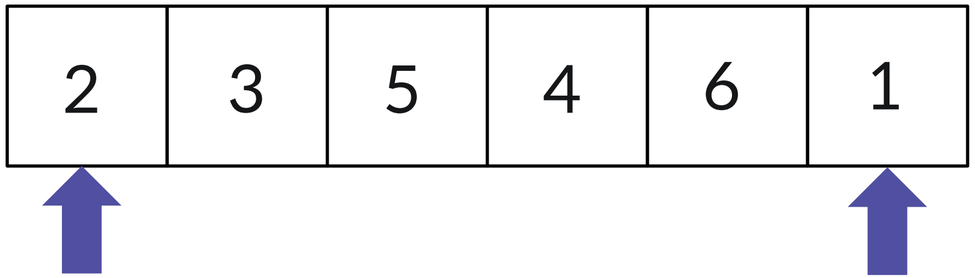
버블 소트



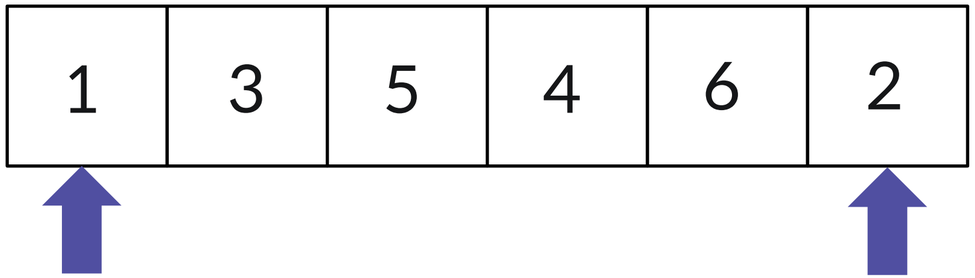
버블 소트



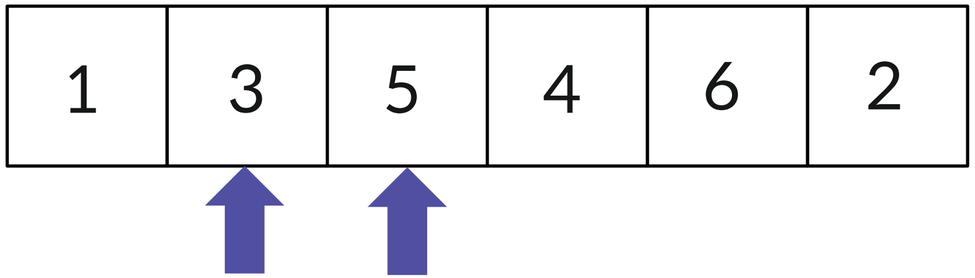
버블 소트



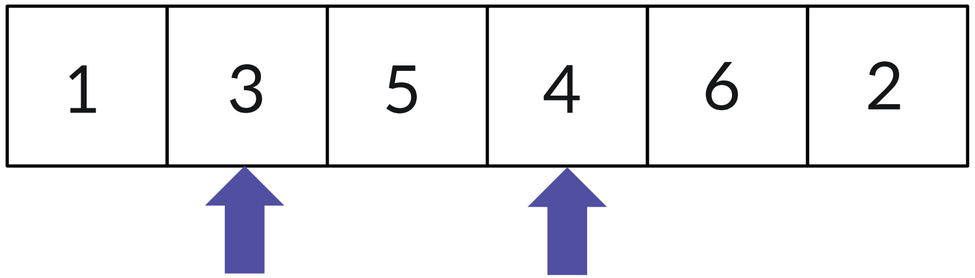
버블 소트



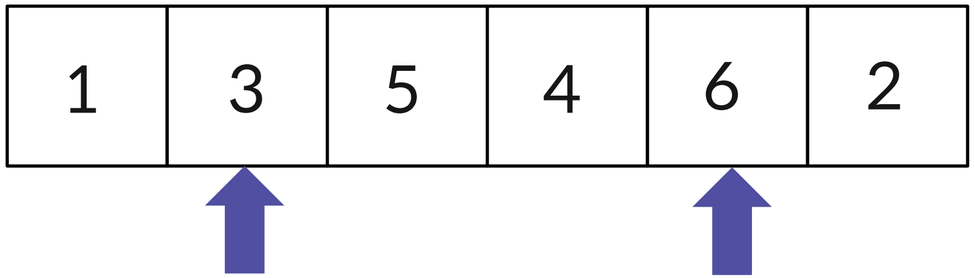
버블 소트



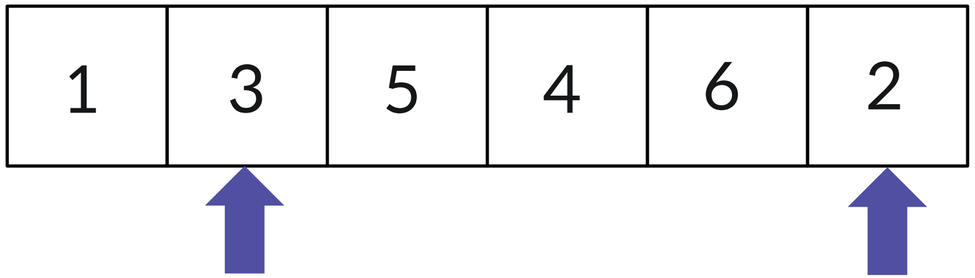
버블 소트



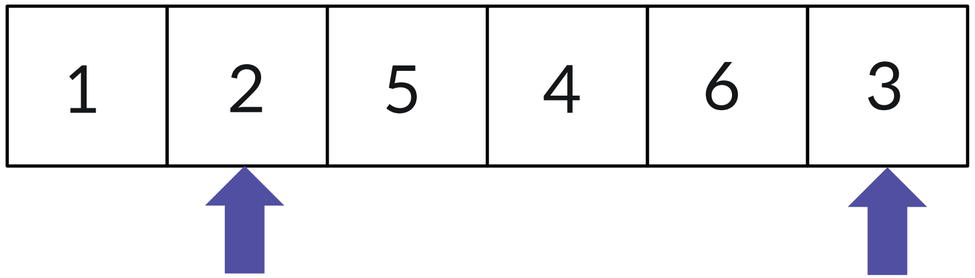
버블 소트



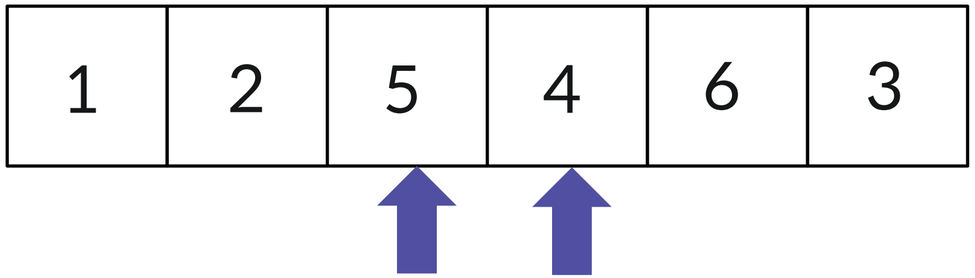
버블 소트



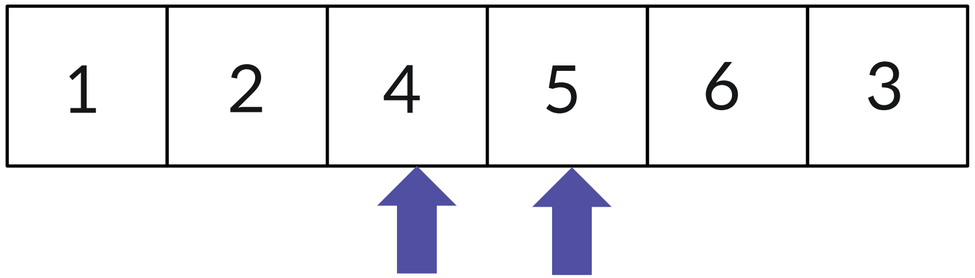
버블 소트



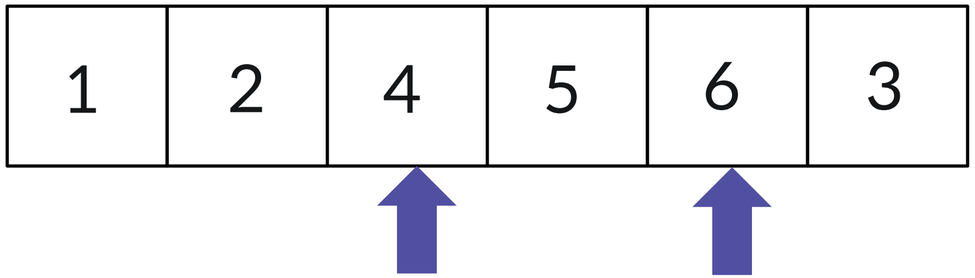
버블 소트



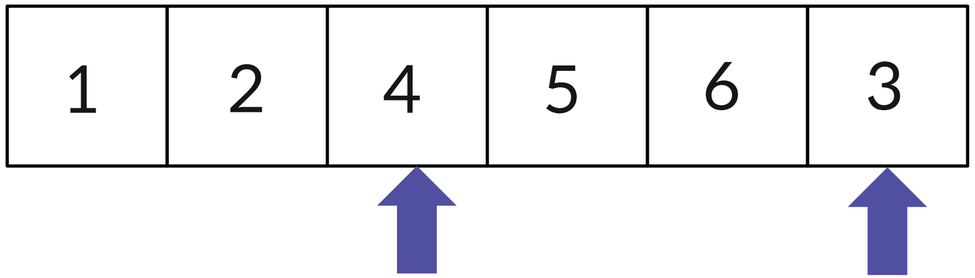
버블 소트



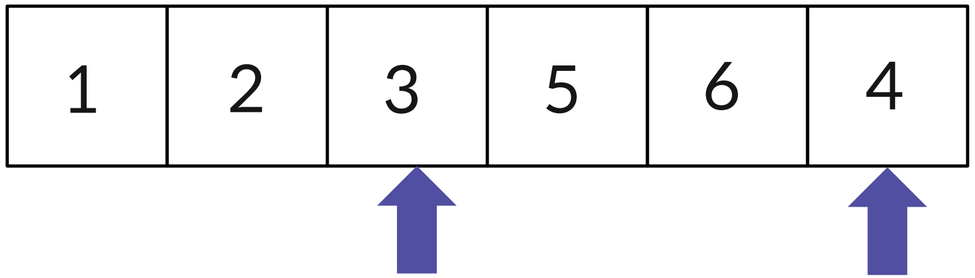
버블 소트



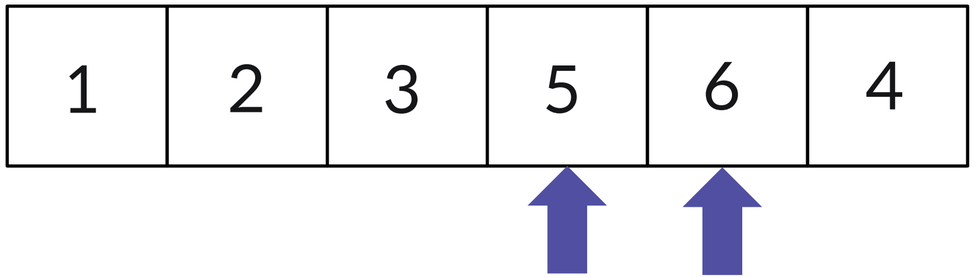
버블 소트



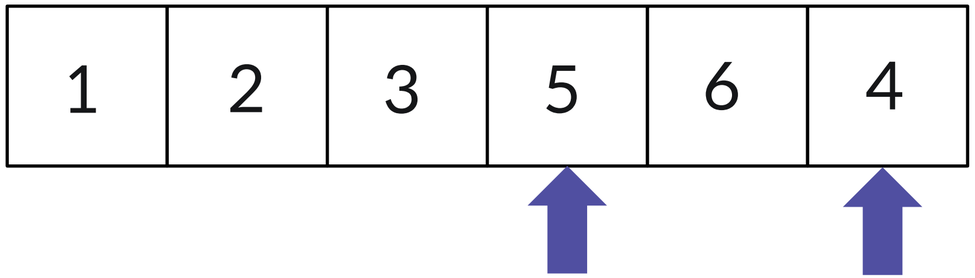
버블 소트



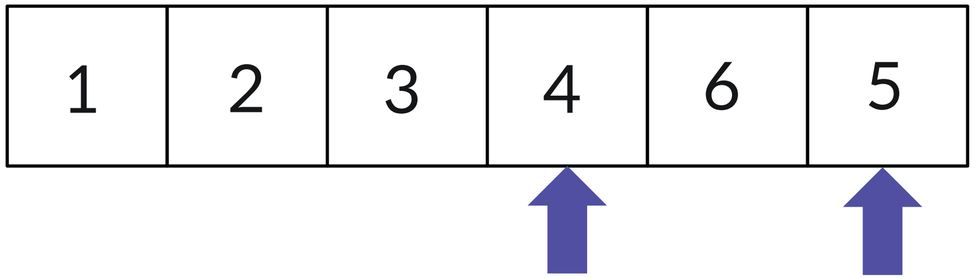
버블 소트



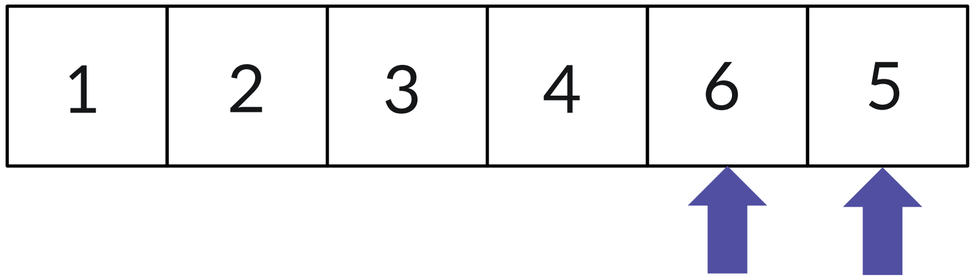
버블 소트



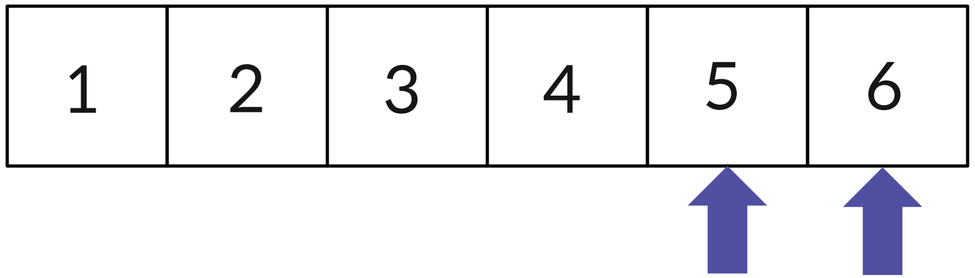
버블 소트



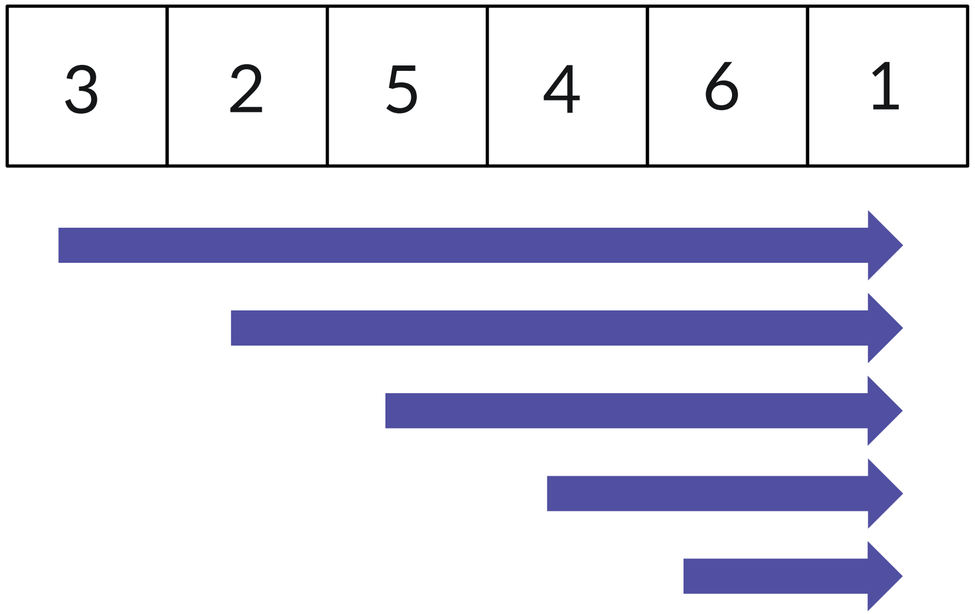
버블 소트



버블 소트



버블 소트

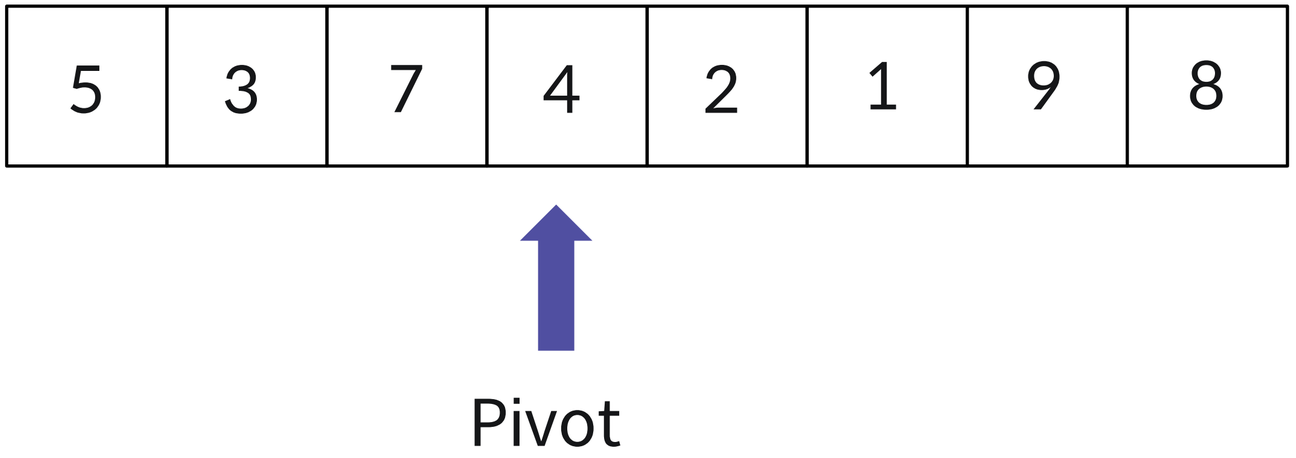


퀵소트

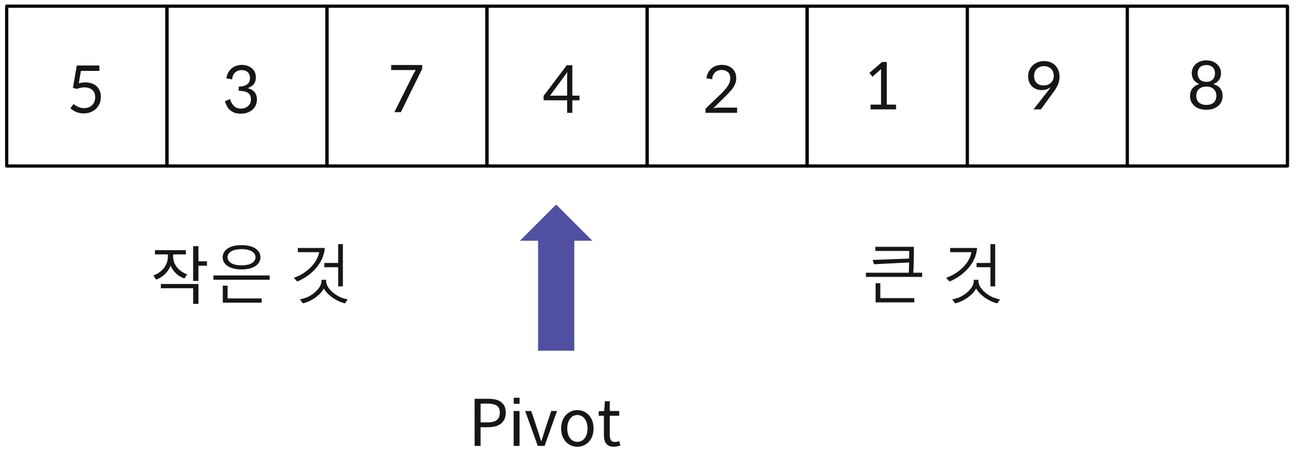
분할 정복을 이용한 정렬 방법

분할 정복이란, 큰 문제를 작은 문제 단위로 쪼개며 문제를 해결하는 방법

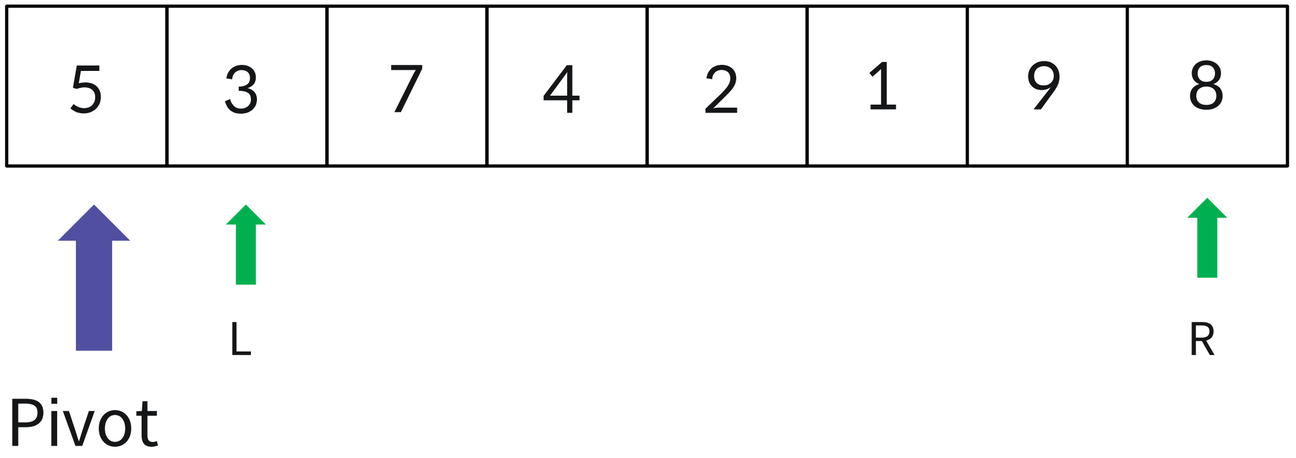
퀵소트



퀵소트

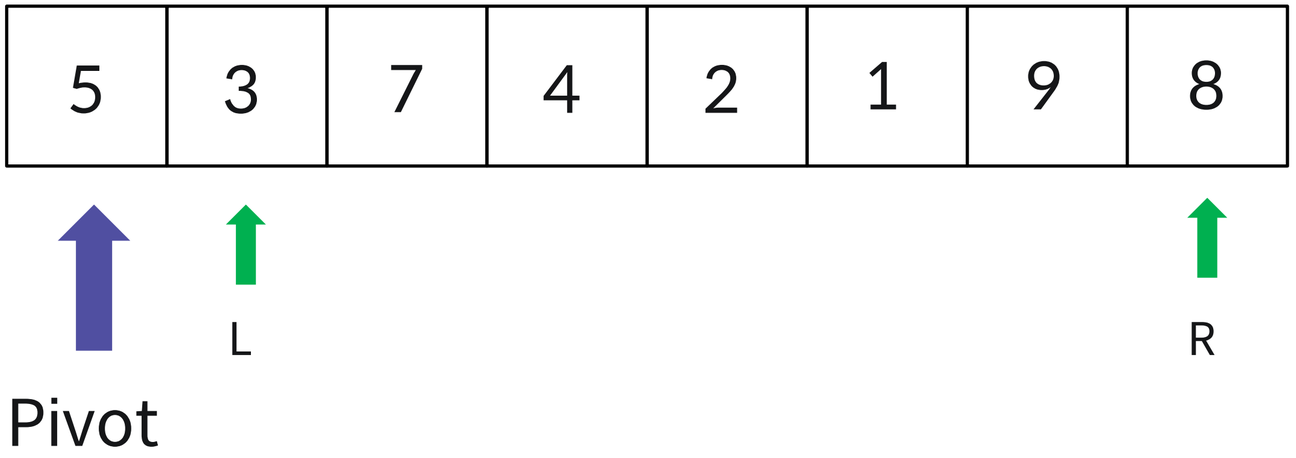


퀵소트



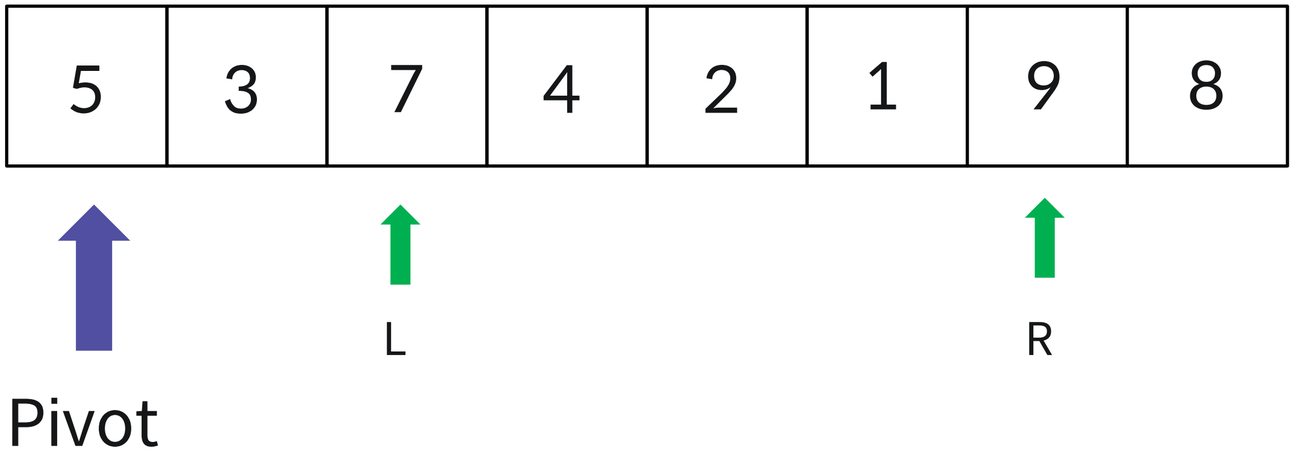
L는 L**번째 값**이 pivot**보다 클 때까지** 움직임
R는 R**번째 값**이 pivot**보다** 작**을 때 까지** 움직임

퀵소트



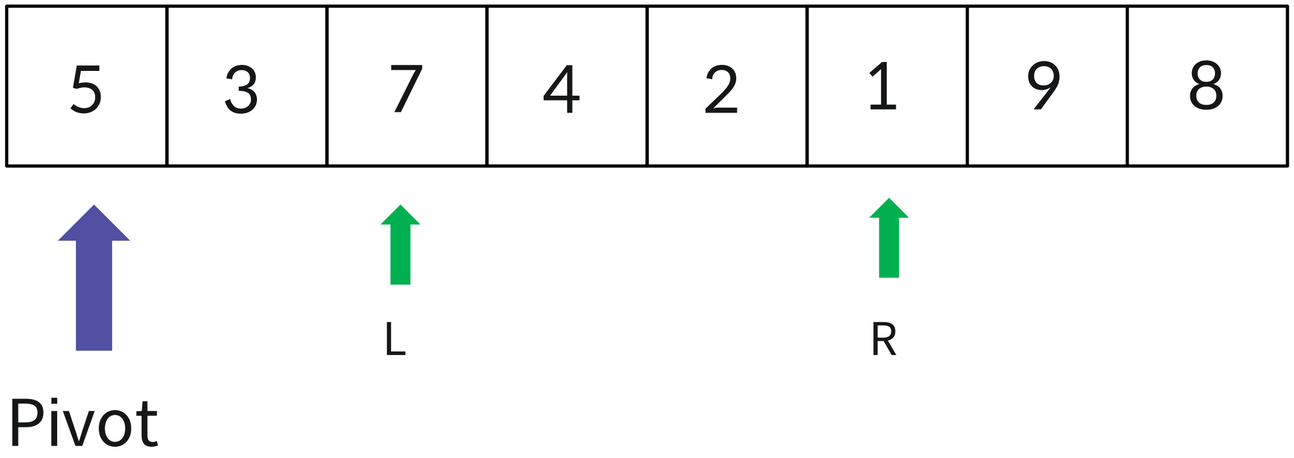
L는 L**번째 값**이 pivot**보다 클 때까지** 움직임
R는 R**번째 값**이 pivot**보다** 작**을 때 까지** 움직임

퀵소트



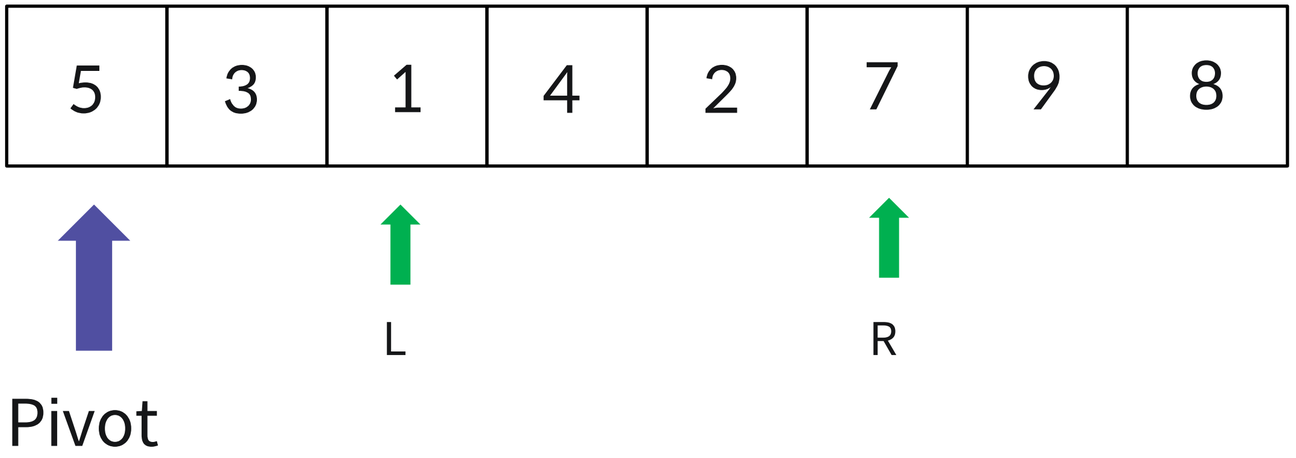
L는 L**번째 값**이 pivot**보다 클 때까지** 움직임
R는 R**번째 값**이 pivot**보다** 작**을 때 까지** 움직임

퀵소트



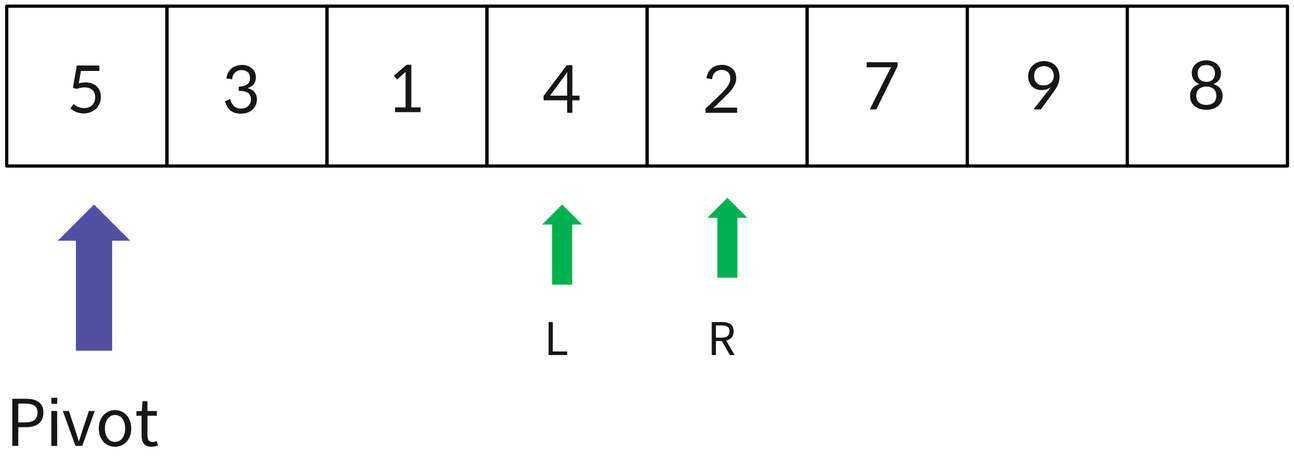
L는 L**번째 값**이 pivot**보다 클 때까지** 움직임
R는 R**번째 값**이 pivot**보다** 작**을 때 까지** 움직임

퀵소트



L과 R을 바꿈

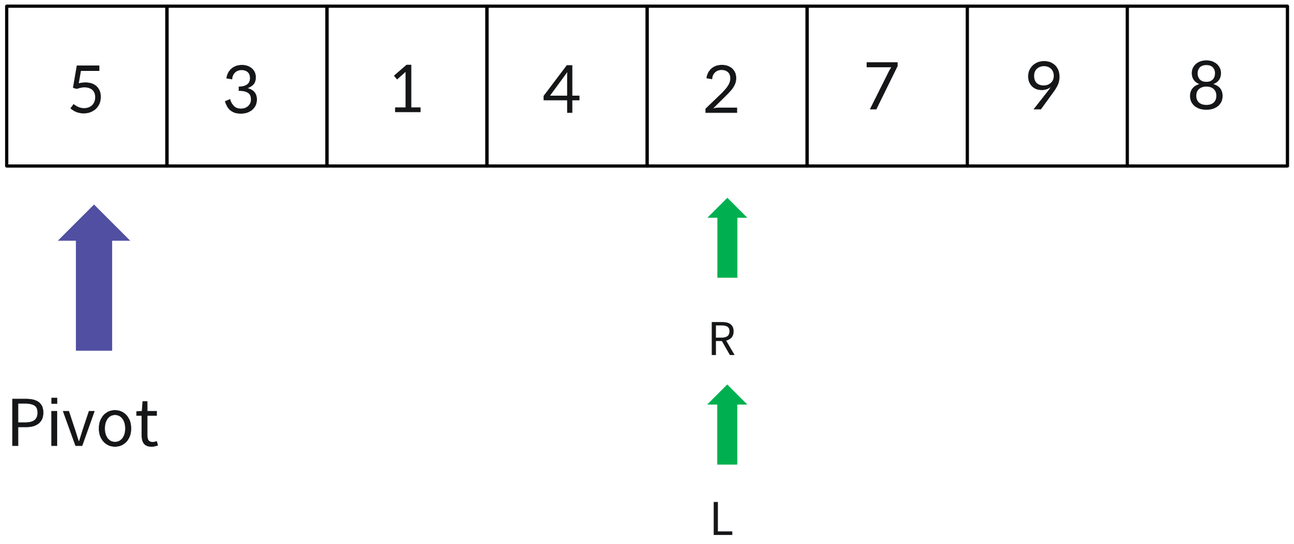
퀵소트



계속 움직임!

L > Pivot, R < Pivot을 찾을 때 까지

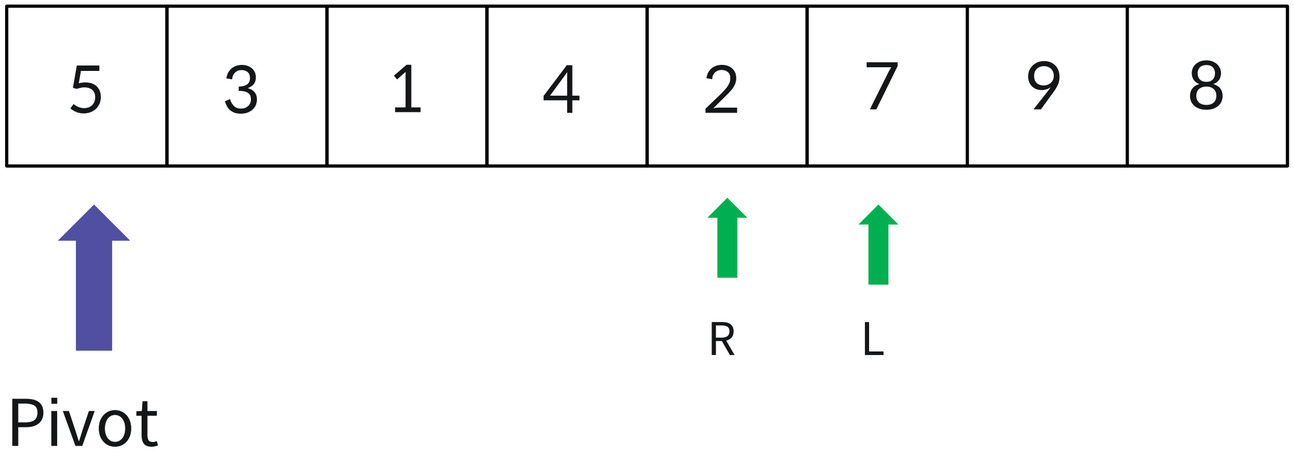
퀵소트



계속 움직임!

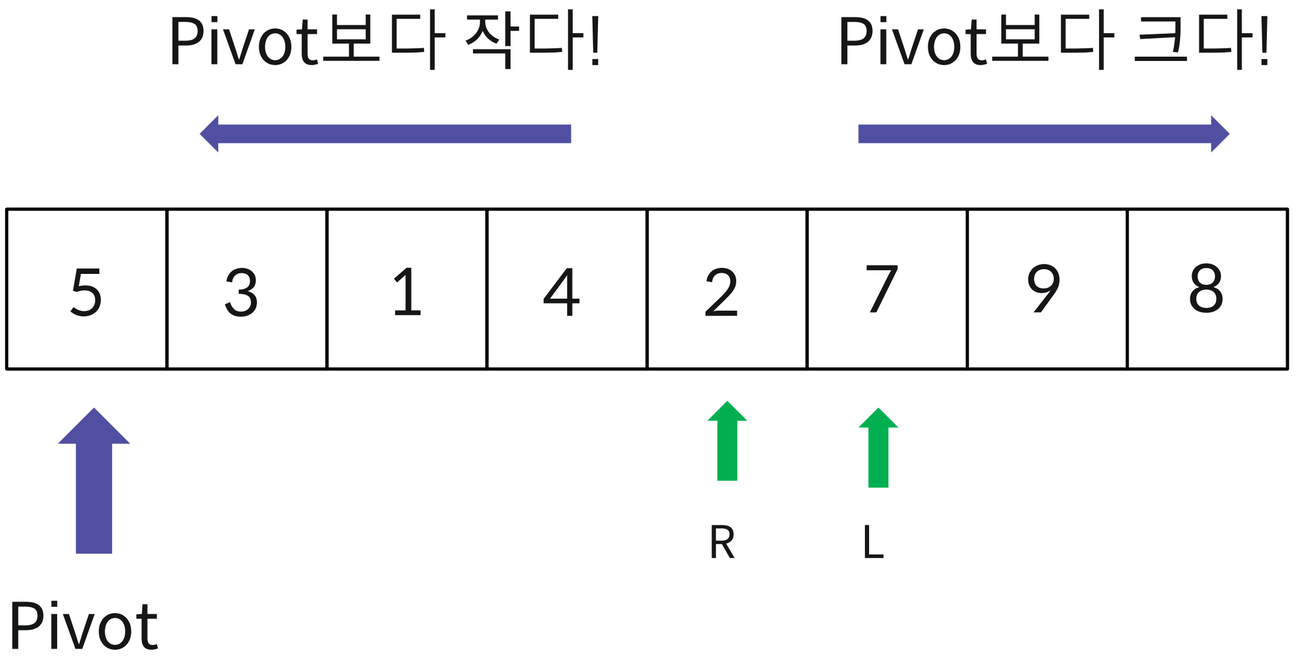
L > Pivot, R < Pivot을 찾을 때 까지

퀵소트



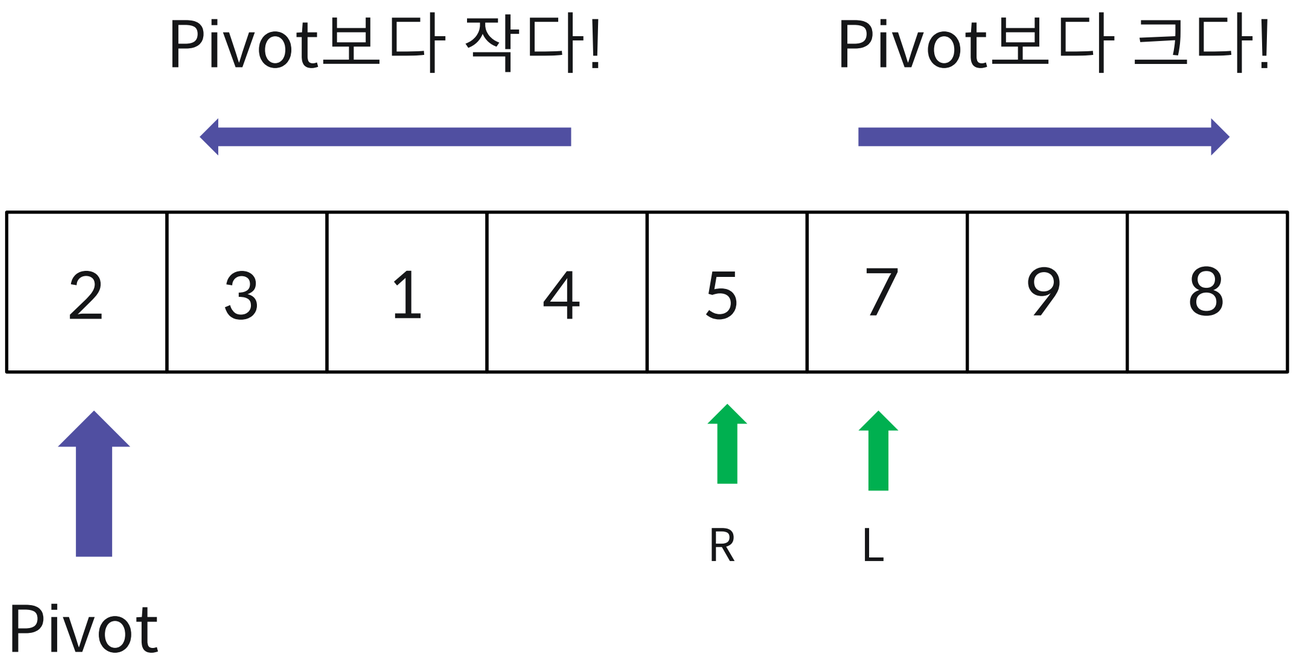
R < L이 되면?

퀵소트



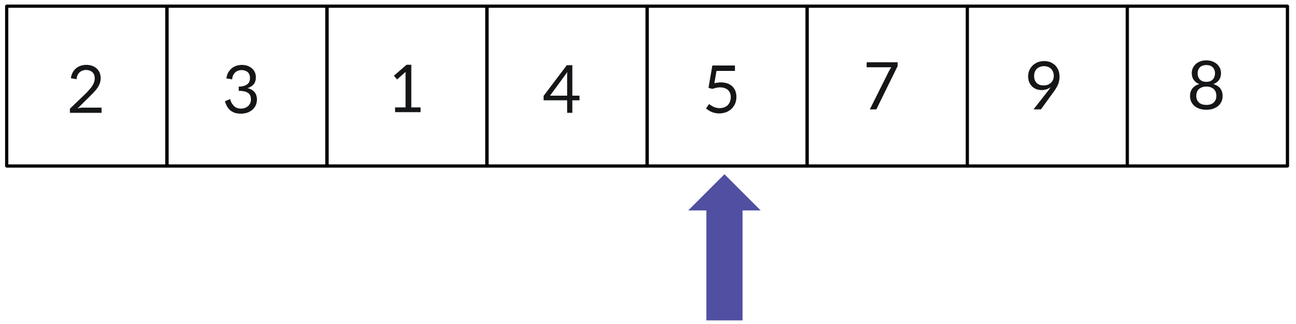
R < L이 되면?
R을 기준으로 왼쪽과 오른쪽이...

퀵소트



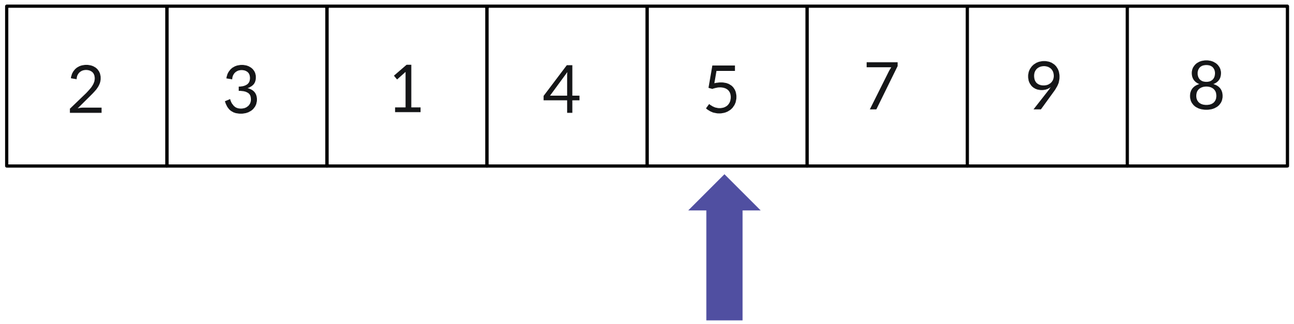
R과 Pivot을 바꿔준다.

퀵소트



왼쪽 배열과 오른쪽 배열에 대해 같은 작업을 반복한다.

퀵소트

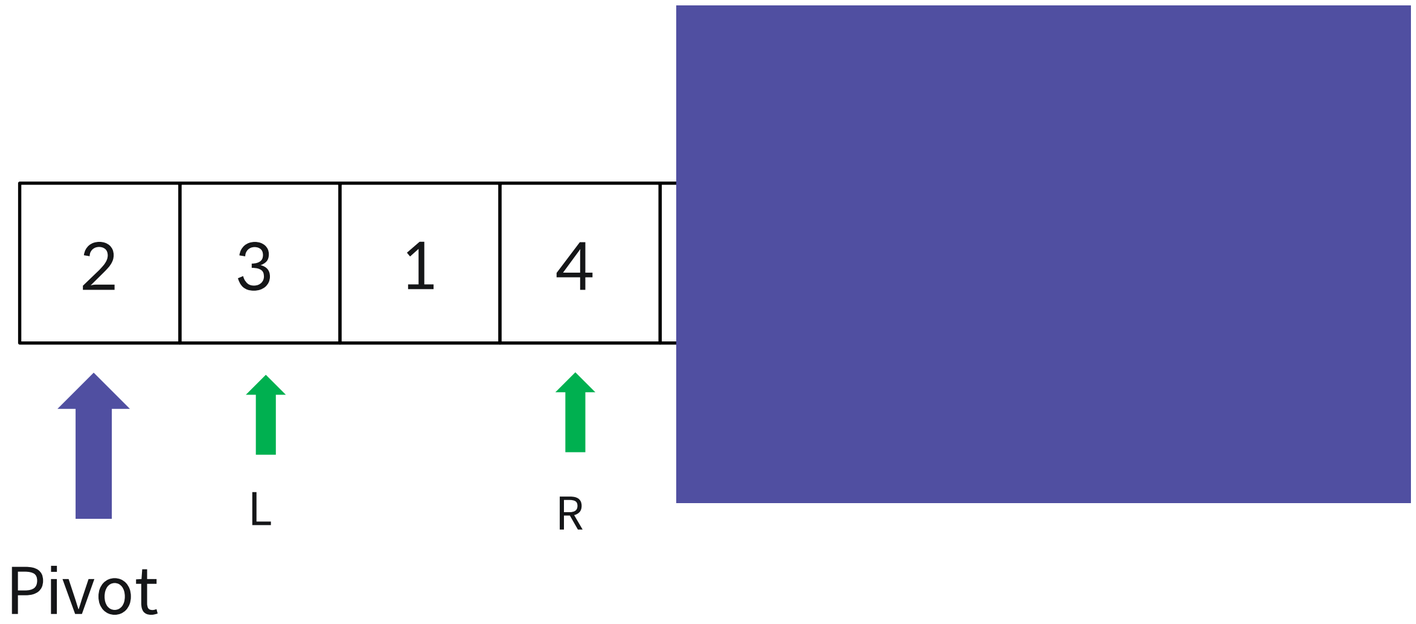


좀 더 정확히 표현하자면

왼쪽 배열 = (L, pivot – 1)

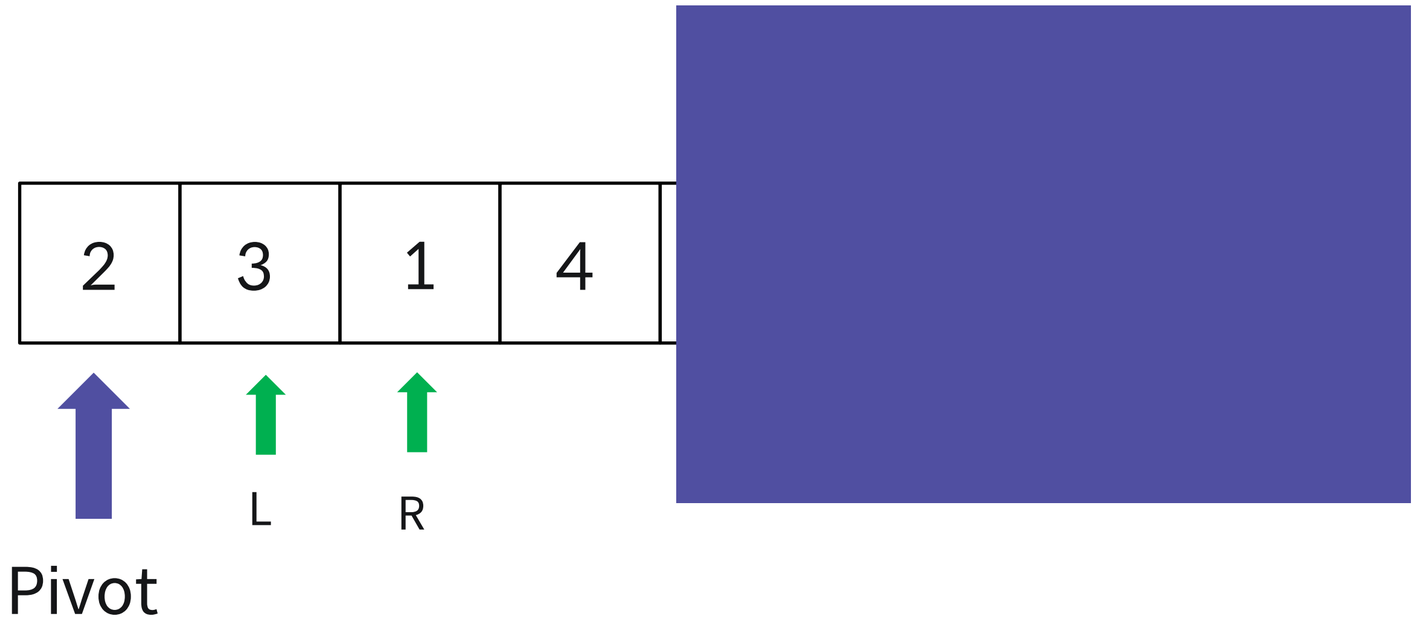
오른쪽 배열 = (pivot + 1, R)

퀵소트



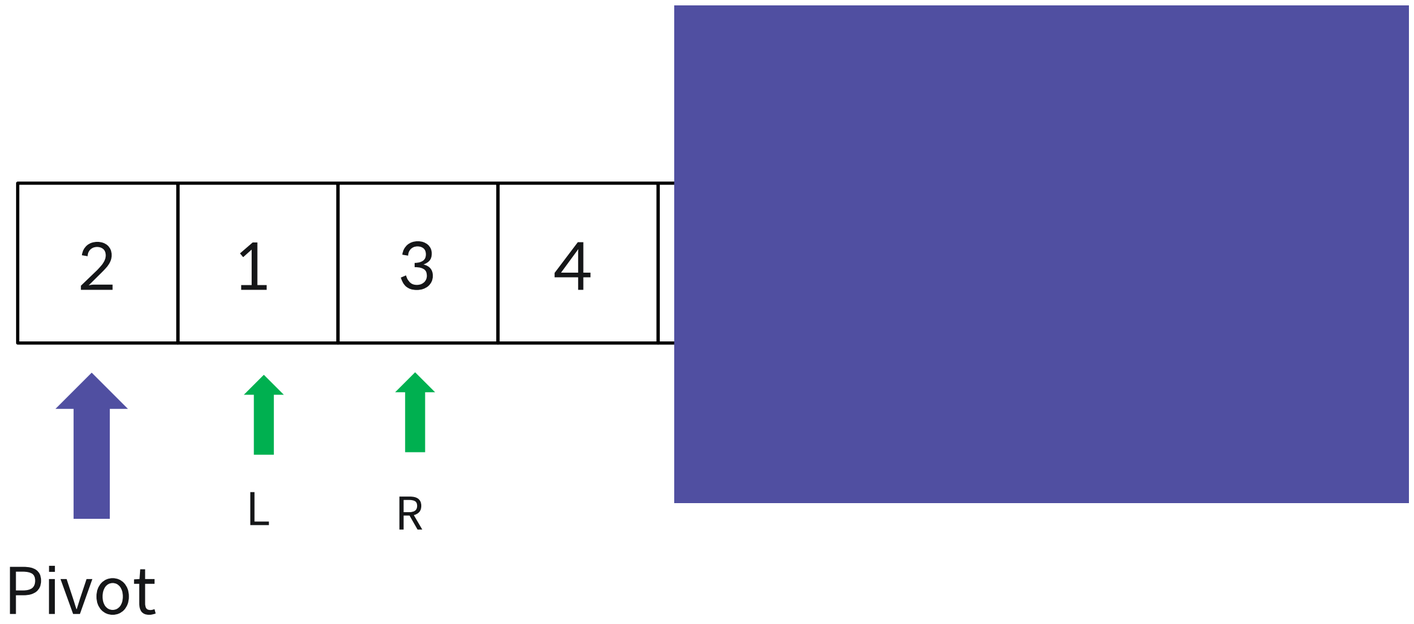
L > Pivot, R < Pivot을 찾으며 움직임

퀵소트



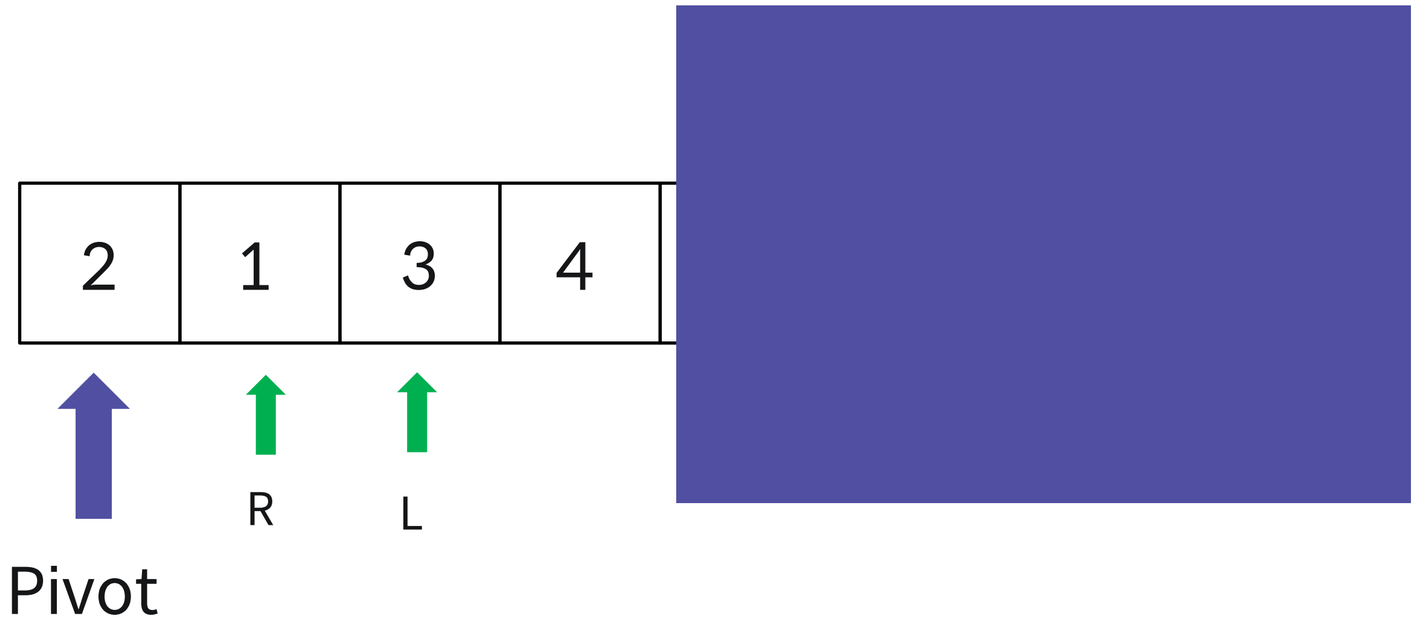
L > Pivot, R < Pivot을 찾으며 움직임

퀵소트



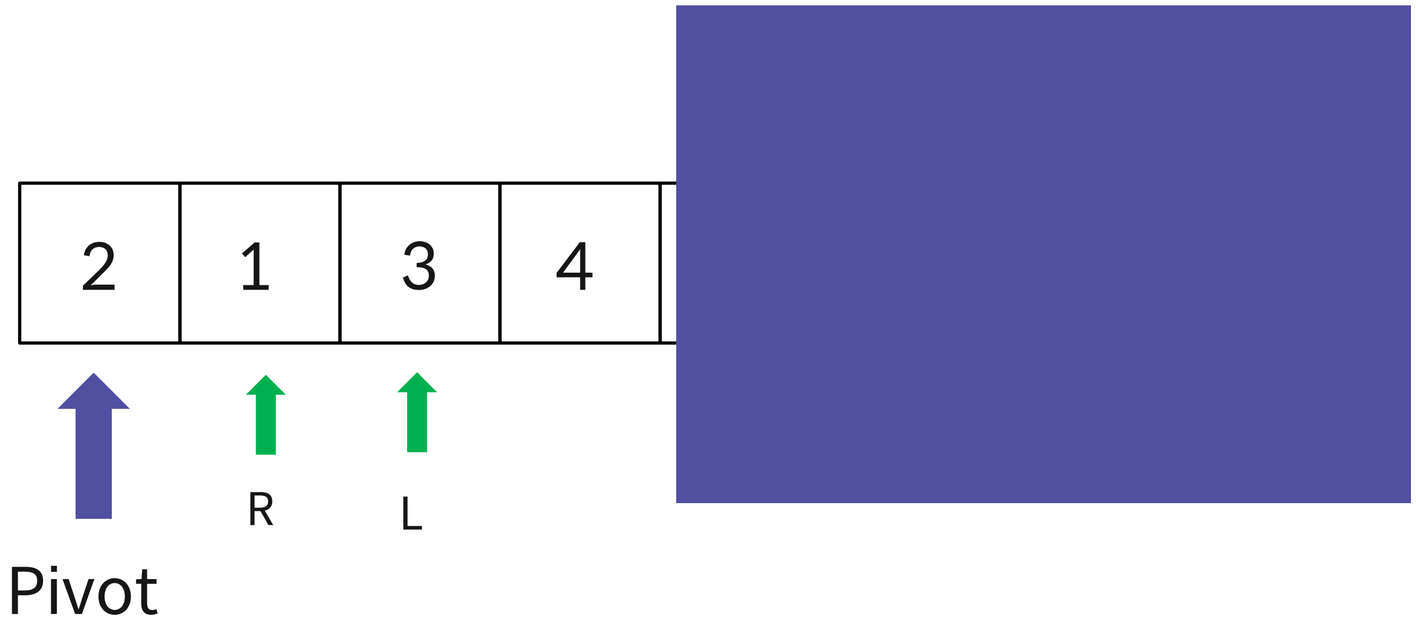
바꾸기!

퀵소트



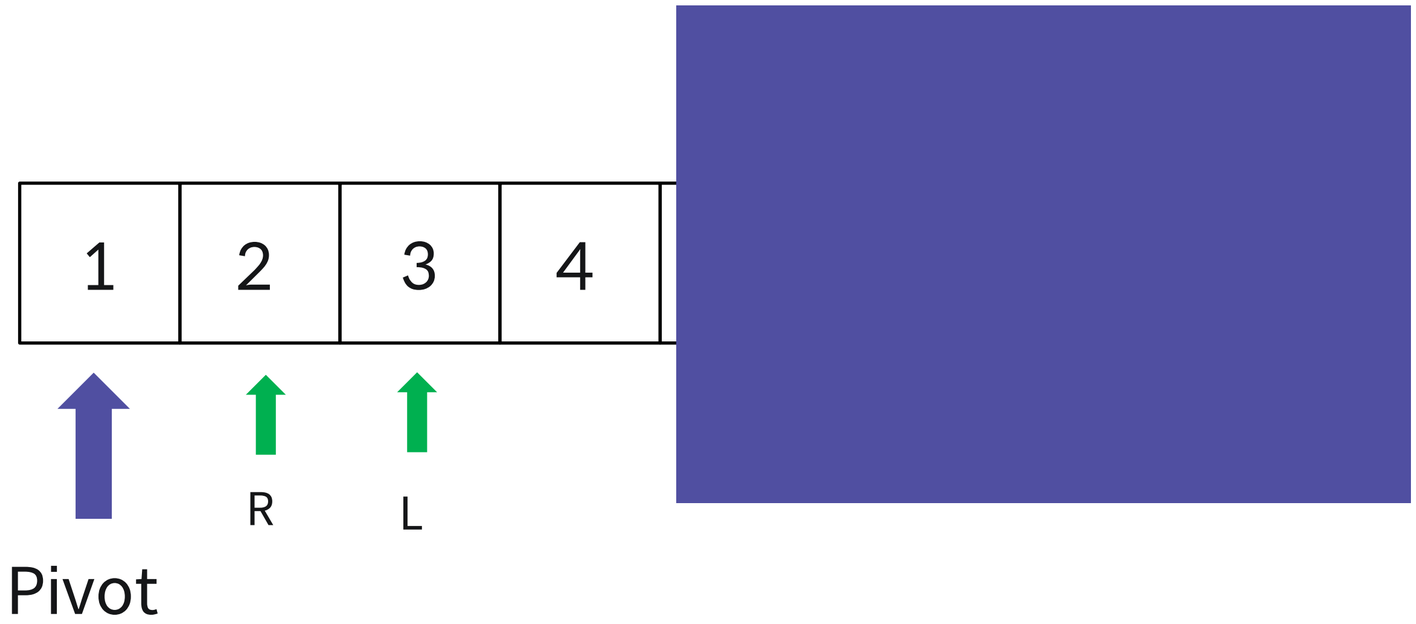
L > Pivot, R < Pivot을 찾으며 움직임

퀵소트



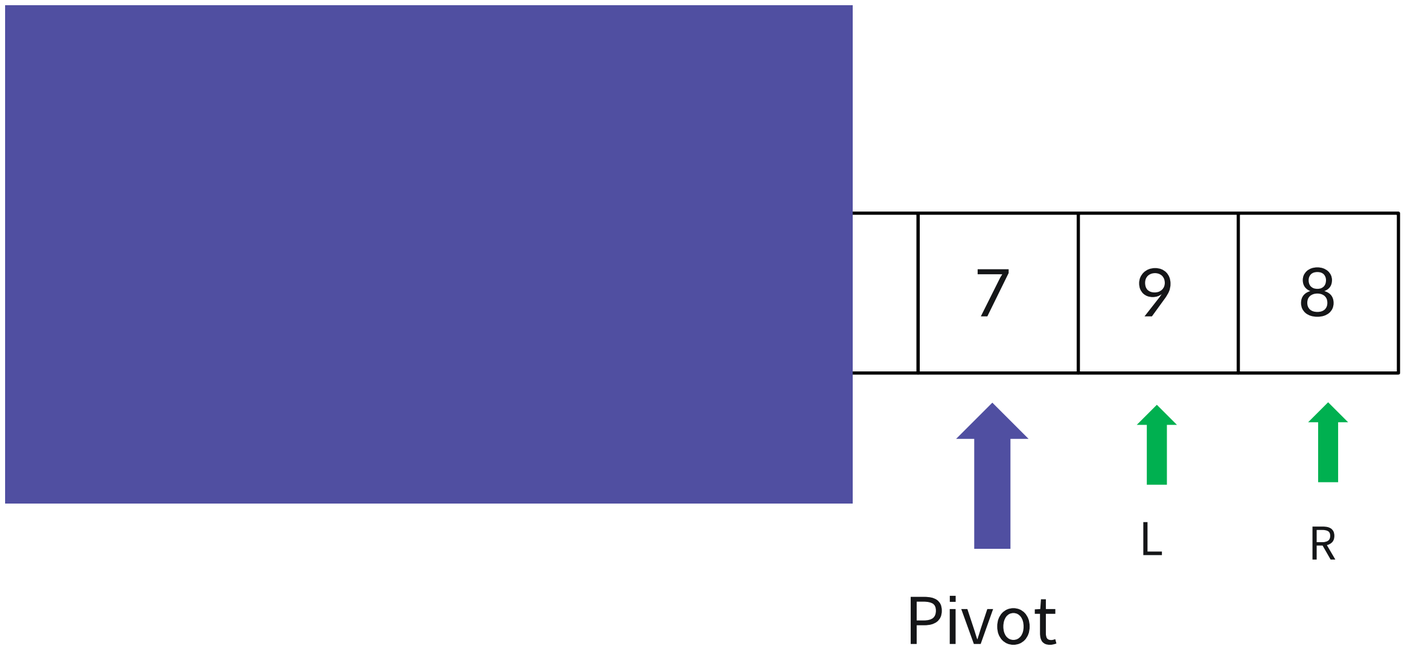
R < L 이므로, Pivot과 R을 바꾼다.

퀵소트



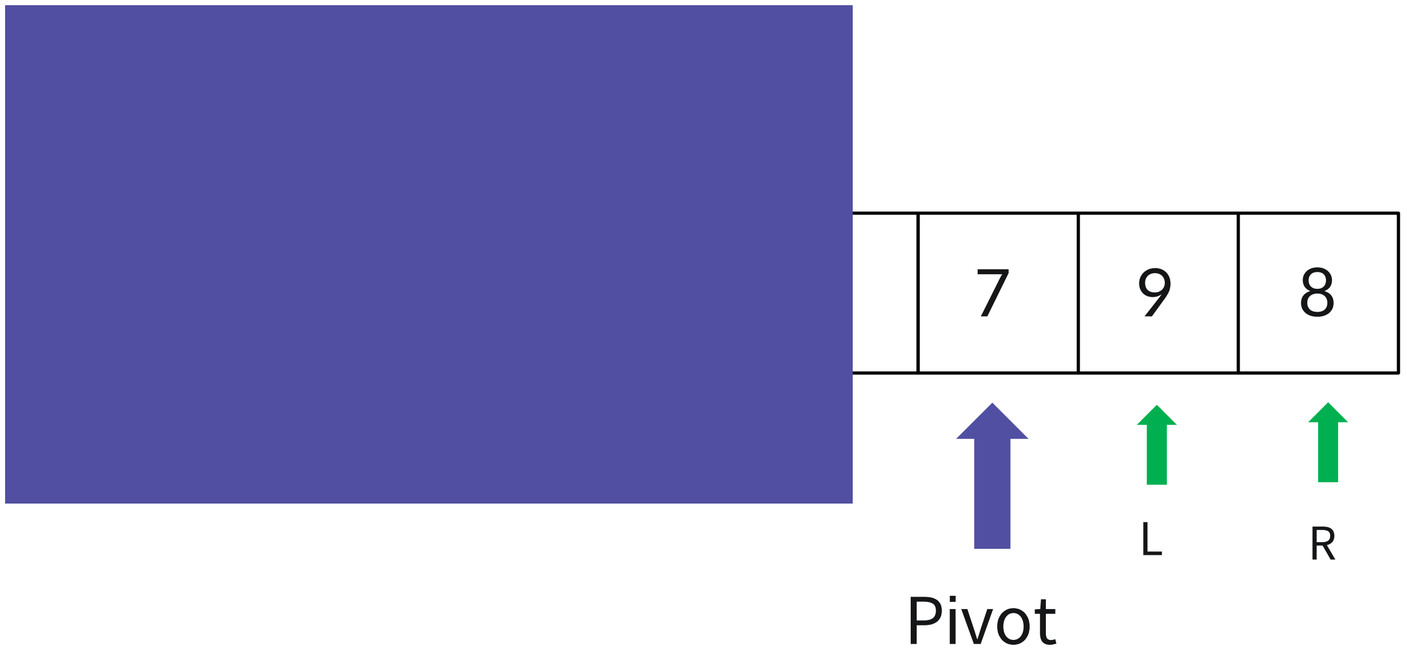
R < L 이므로, Pivot과 R을 바꾼다.

퀵소트



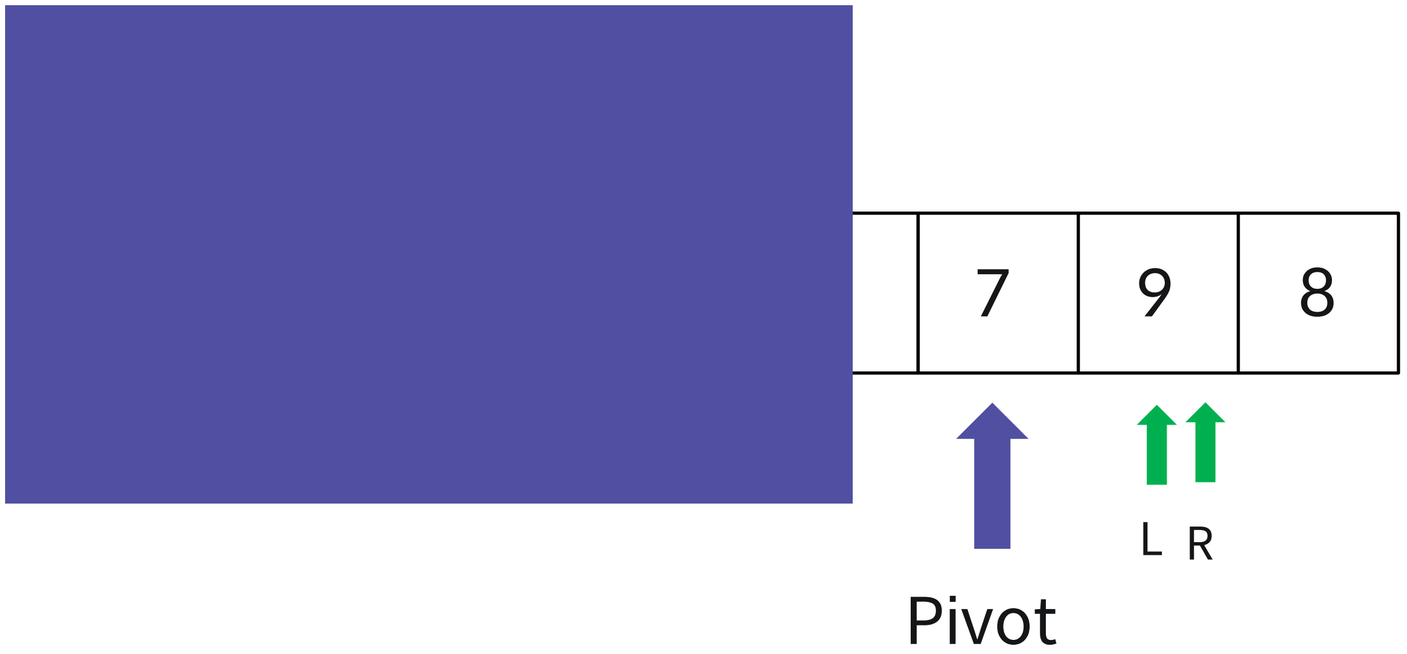
오른쪽 배열도 같은 방법으로 진행

퀵소트



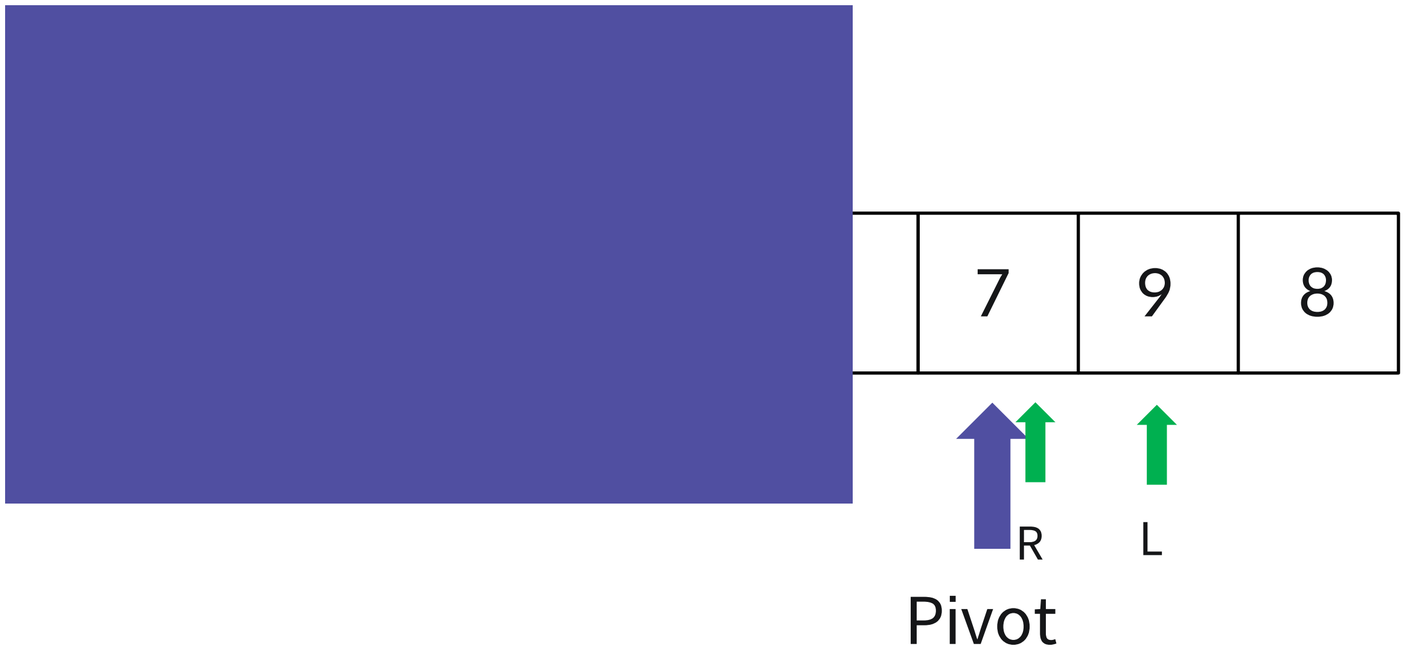
L > Pivot, R < Pivot

퀵소트



L > Pivot, R < Pivot

퀵소트



L > Pivot, R < Pivot

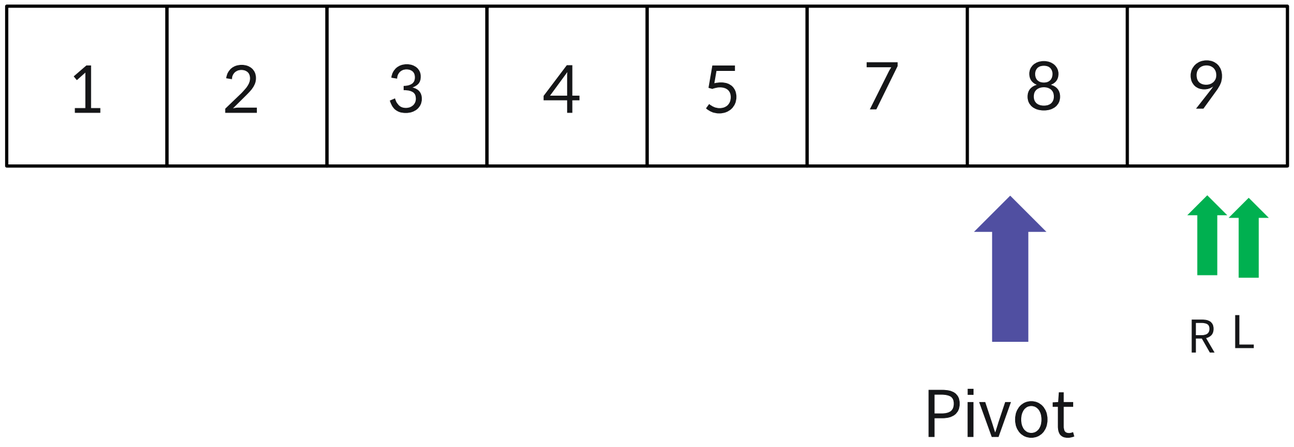
퀵소트

L > Pivot, R < Pivot

퀵소트

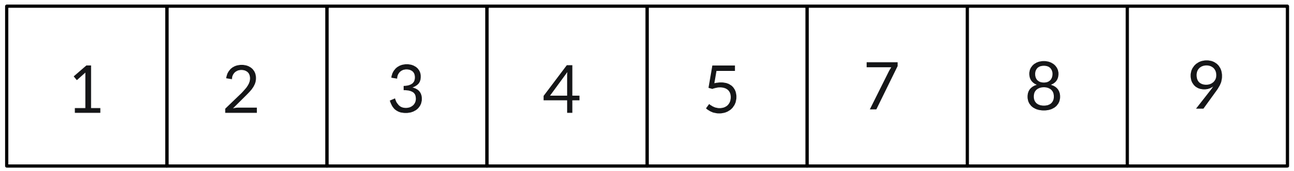
L > Pivot, R < Pivot

퀵소트



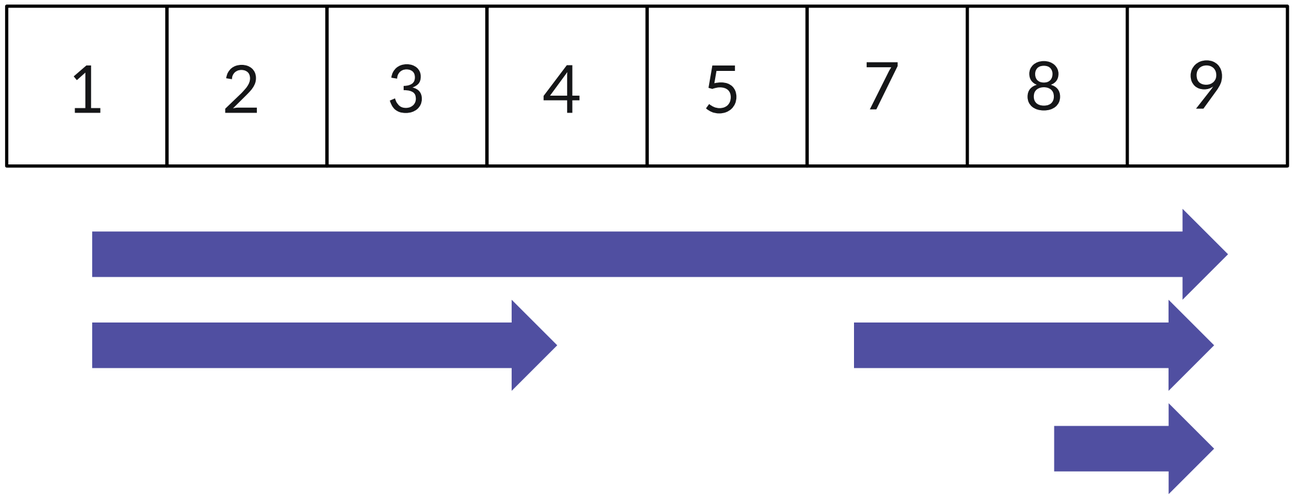
Swap pivot & R

퀵소트



Finish!

퀵소트



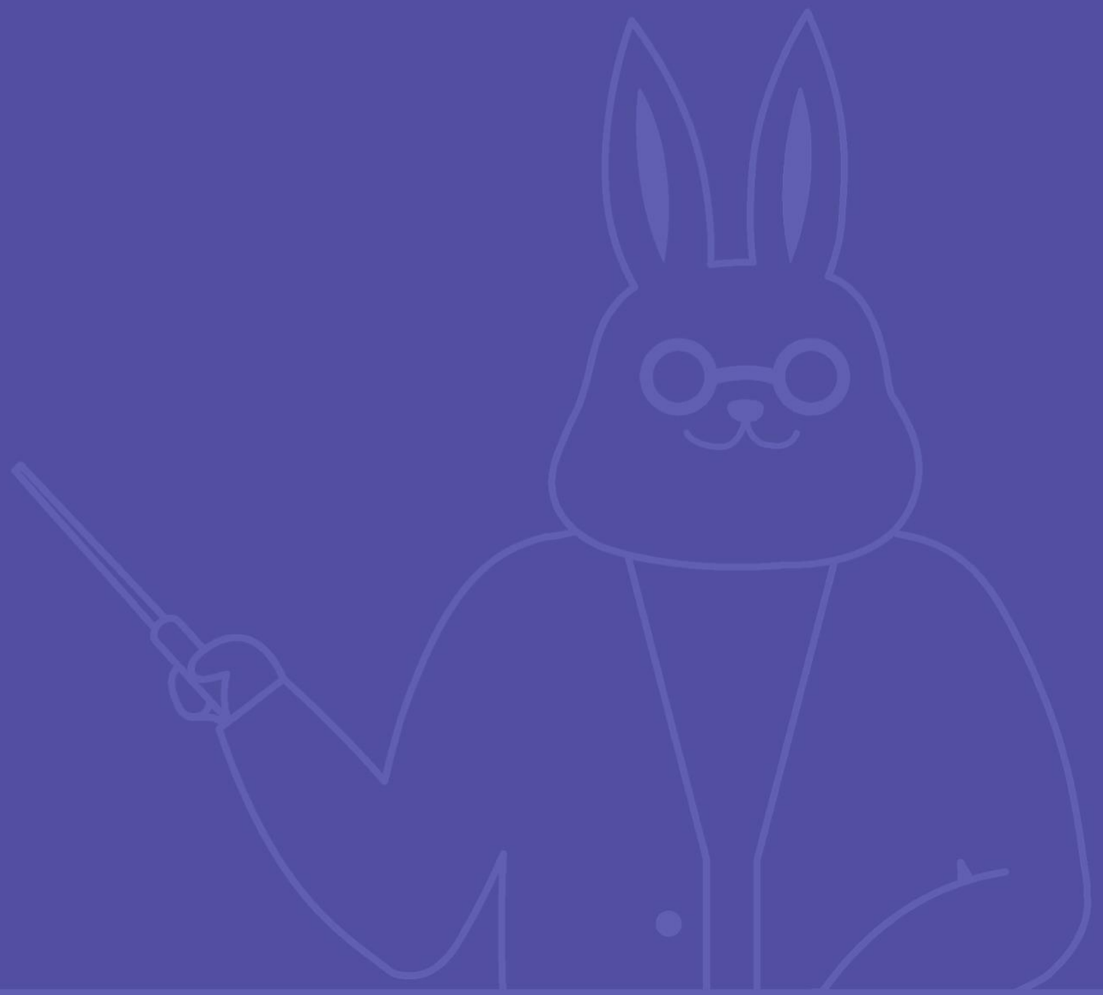
퀵소트

일반적인 경우 O(nlogn), 최악의 경우 O(n^2)

최악의 경우는 보통 역으로 정렬돼있는 경우,
피벗을 어떻게 정하냐에 따라 속도가 달라지는 경우가 있어서
STL의 sort함수는 피벗을 랜덤하게 정하는 기능들이 추가돼있음

04

트리 탐색 (DFS, BFS)



트리를 탐색하는 방법

- 자료구조 시간에 배운 트리를 탐색하는 방법은 크게 2가지가 있는데

트리를 탐색하는 방법

- 자료구조 시간에 배운 트리를 탐색하는 방법은 크게 2가지가 있는데
- **DFS : 깊이 우선 탐색**

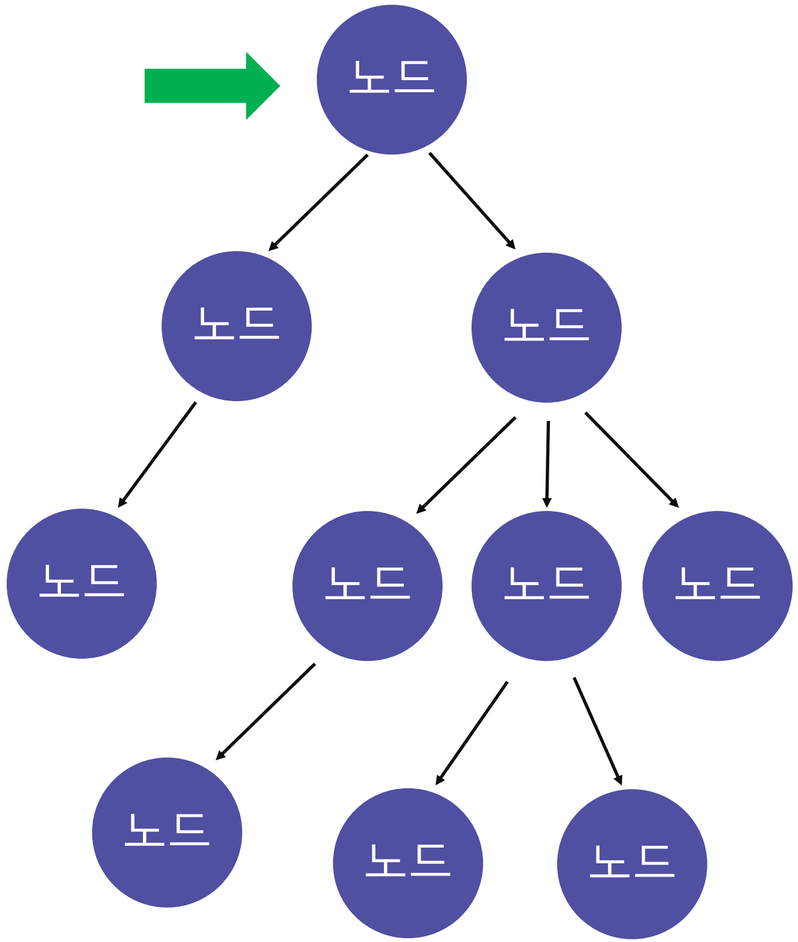
트리를 탐색하는 방법

- 자료구조 시간에 배운 트리를 탐색하는 방법은 크게 2가지가 있는데
- **DFS : 깊이 우선 탐색**
- **BFS : 너비 우선 탐색**

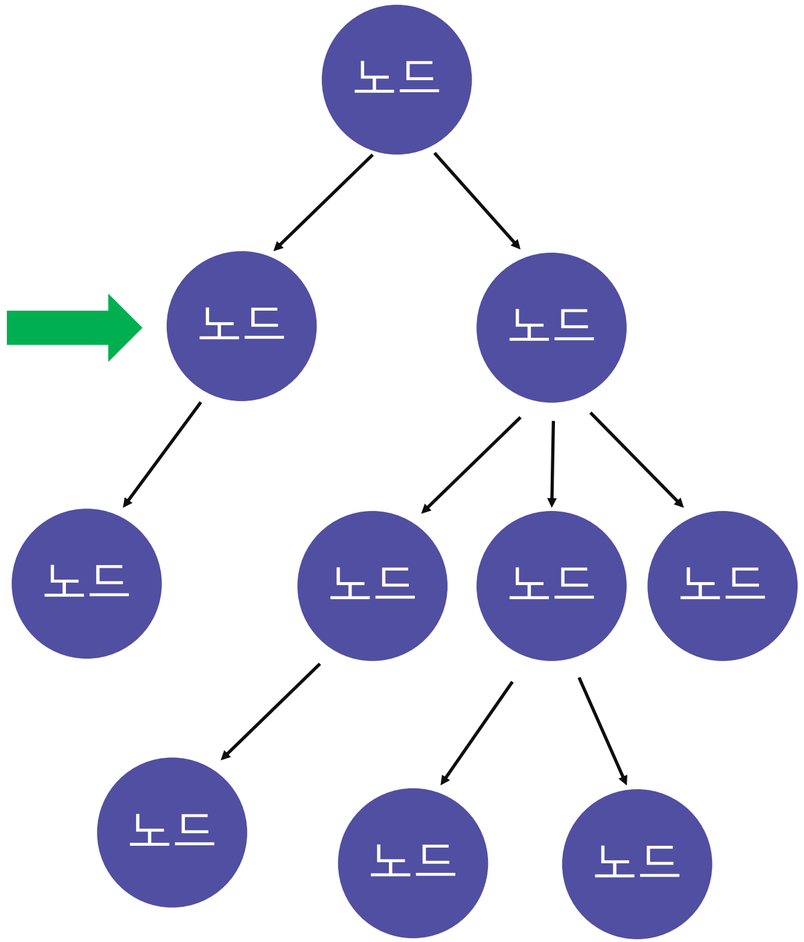
DFS

- 트리의 깊이(Depth)를 우선으로 탐색하는 방법
- 자식 -> 자식 -> 자식 -> …

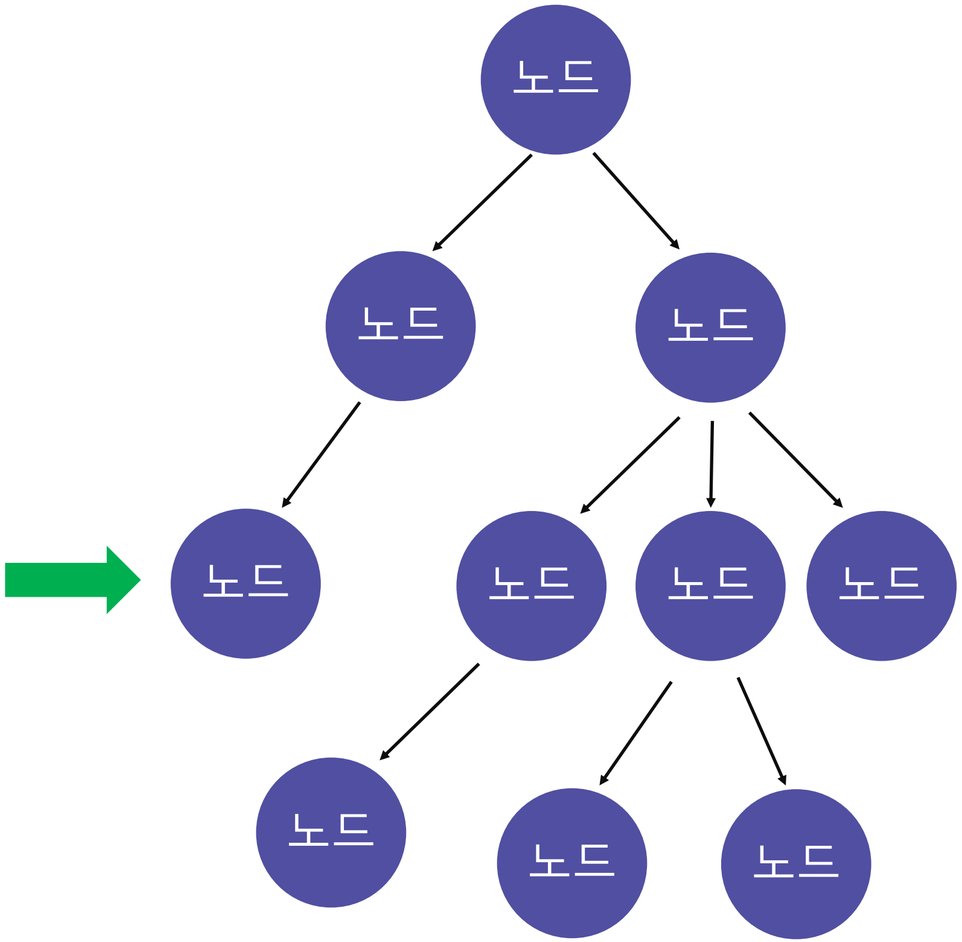
DFS



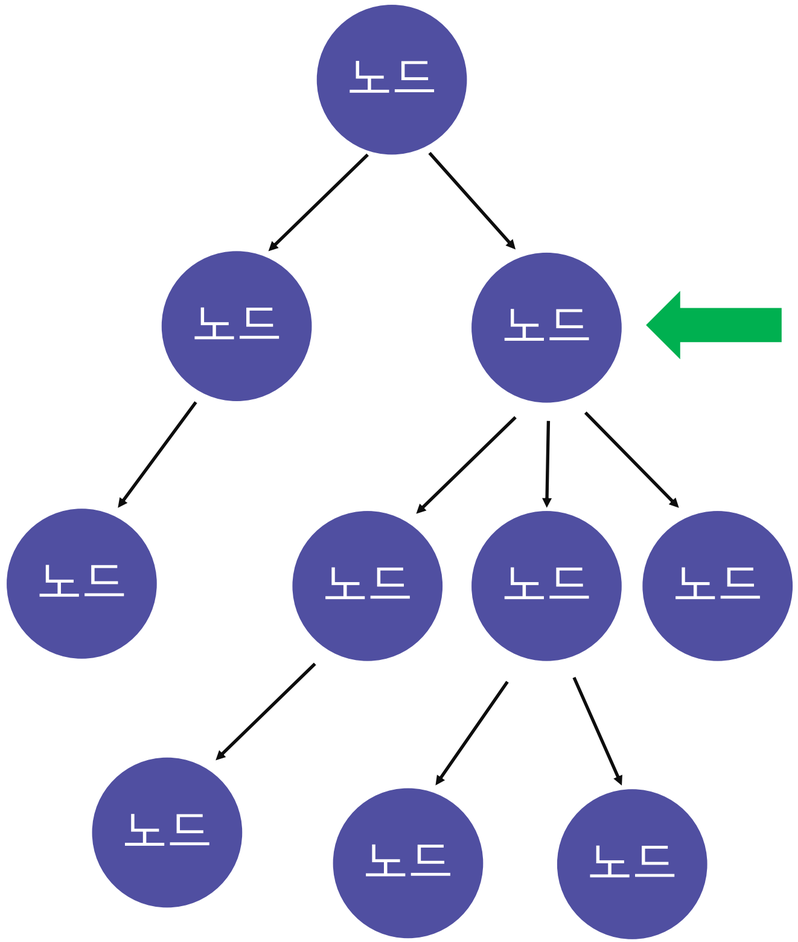
DFS



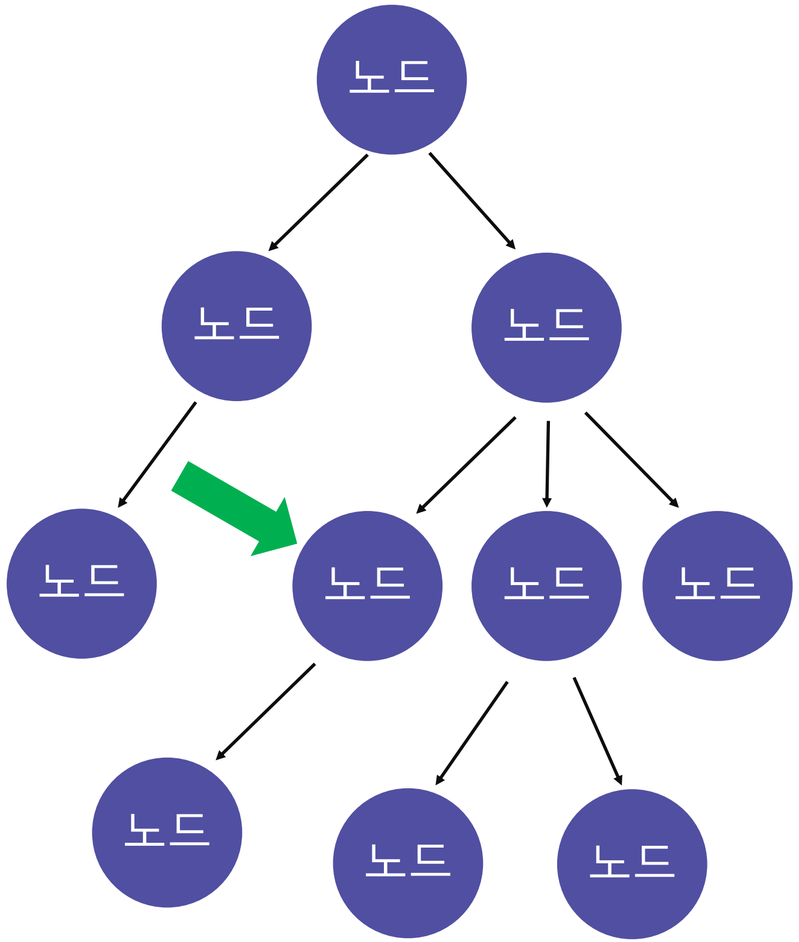
DFS



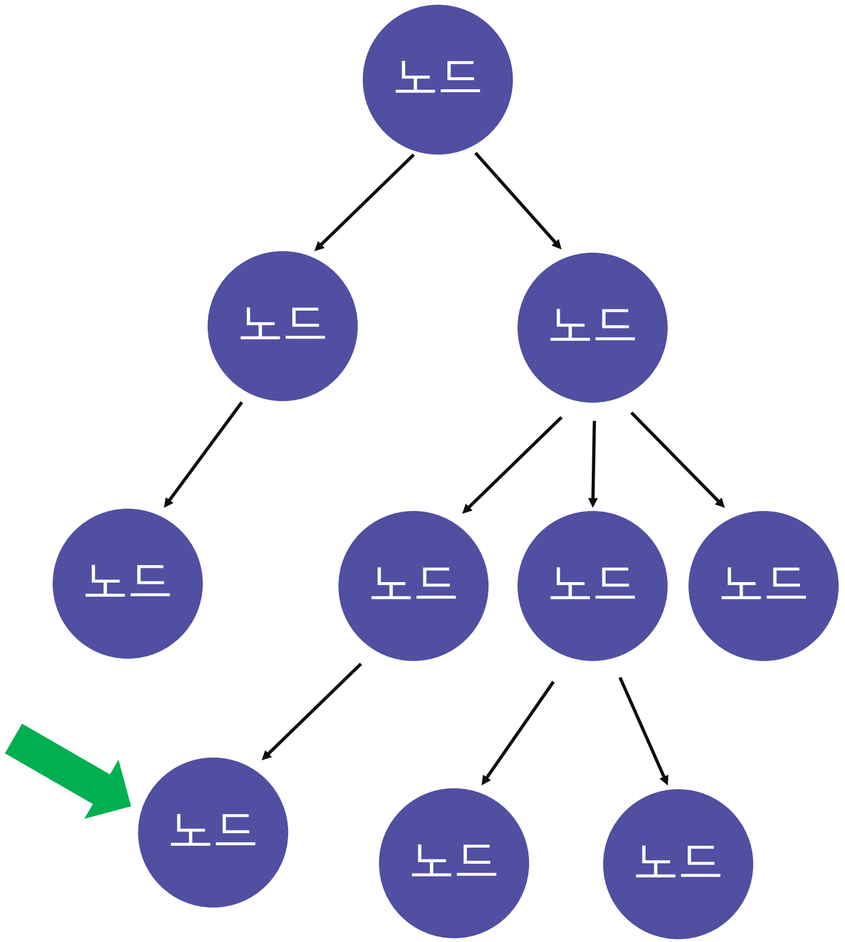
DFS



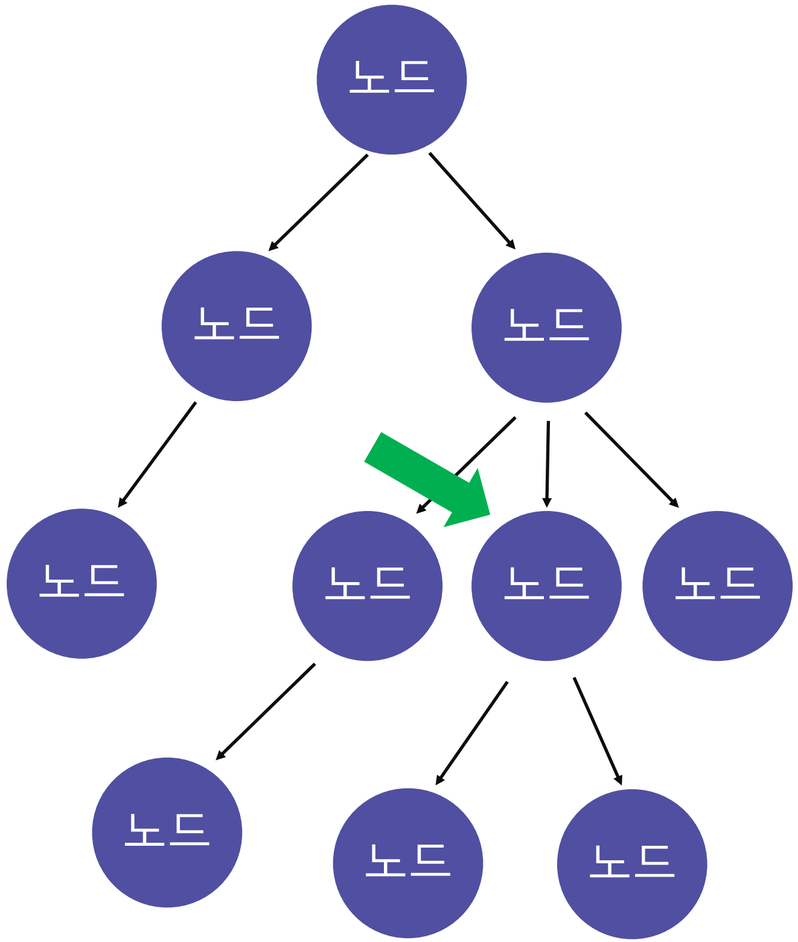
DFS



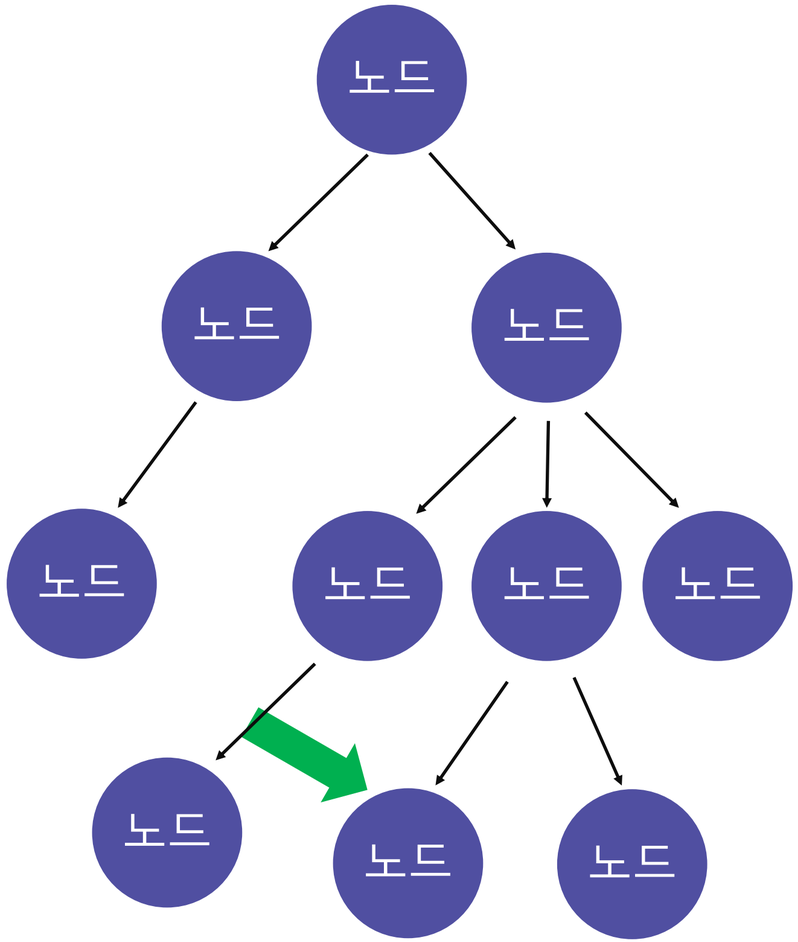
DFS



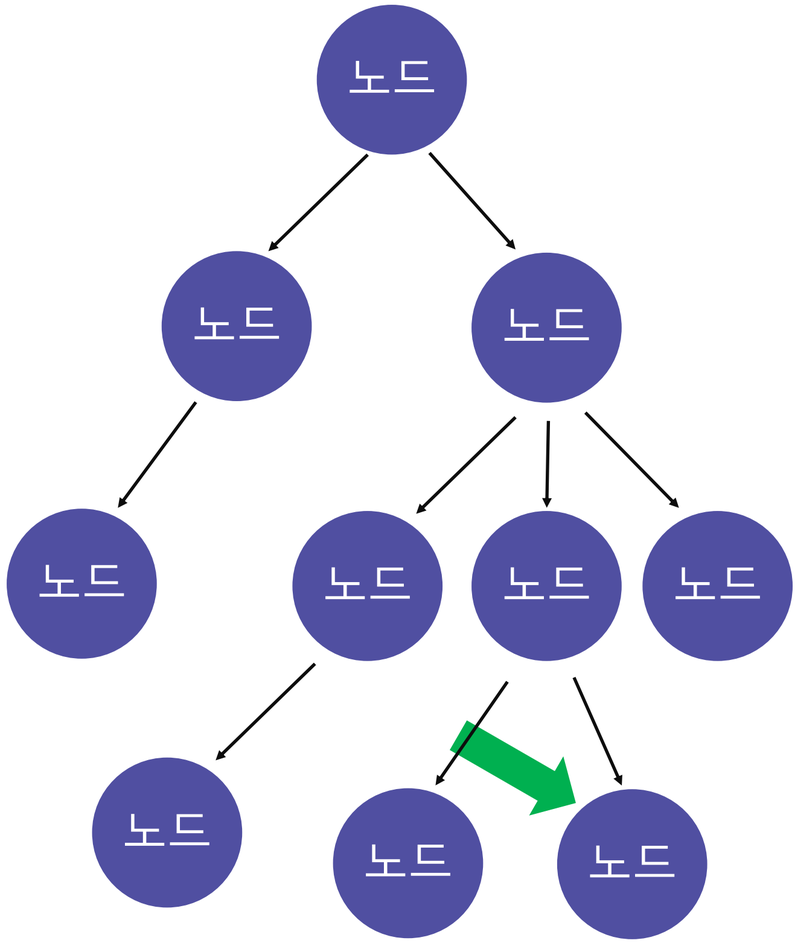
DFS



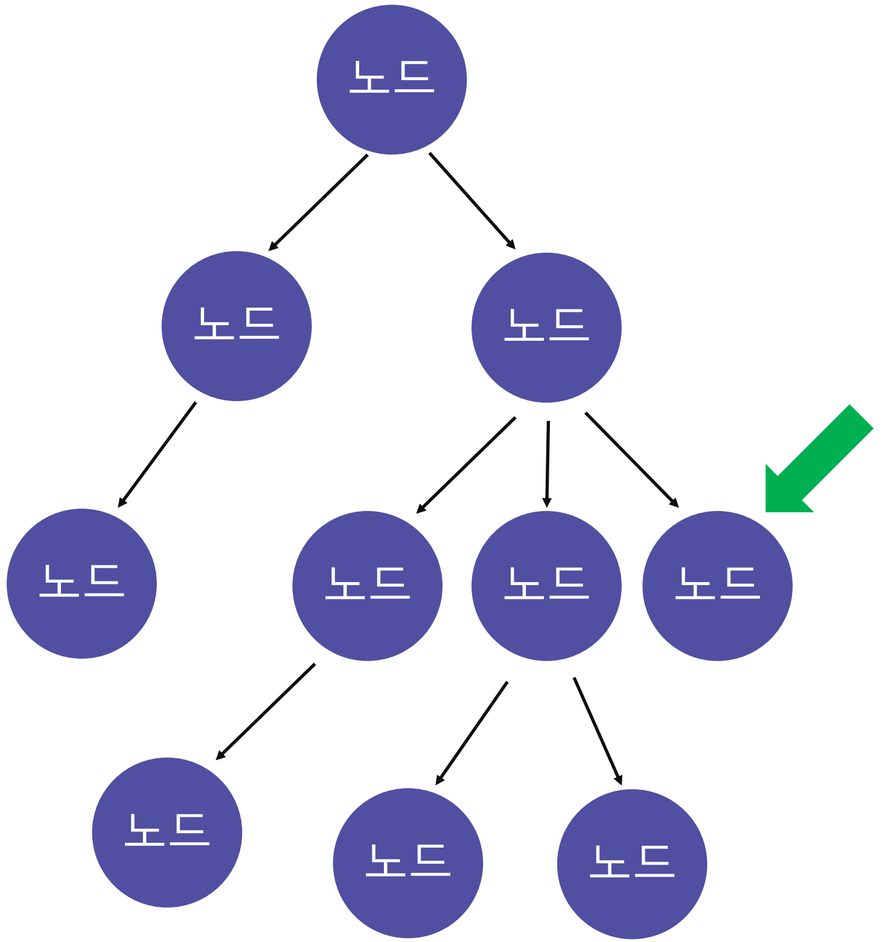
DFS



DFS



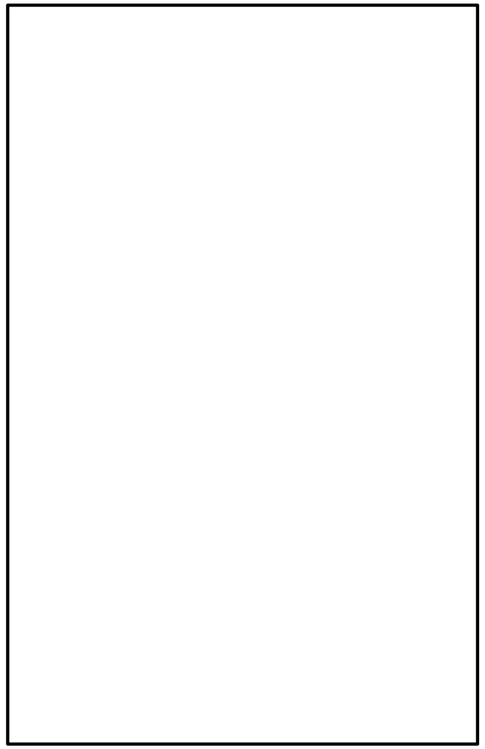
DFS

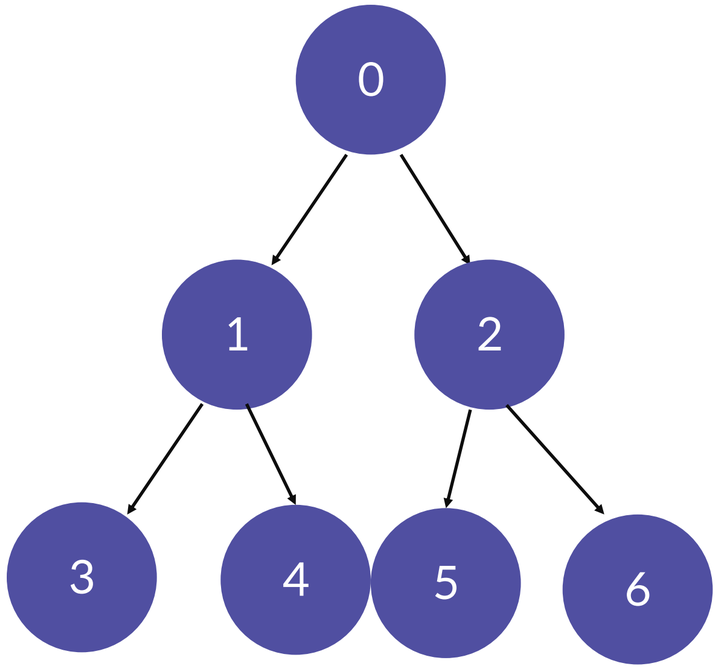


DFS를 구현하는 방법

- 재귀함수 또는 스택을 이용하여 구현
- 여기서는 재귀함수를 이용하여 구현한 DFS 코드를 살펴보자!

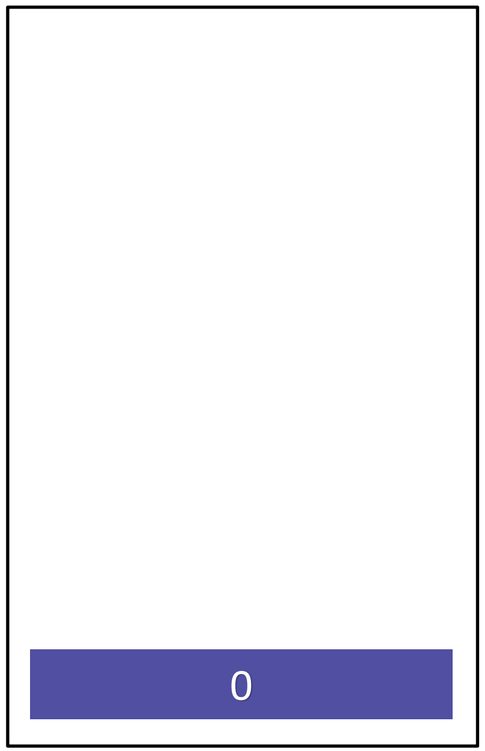
DFS를 구현하는 방법

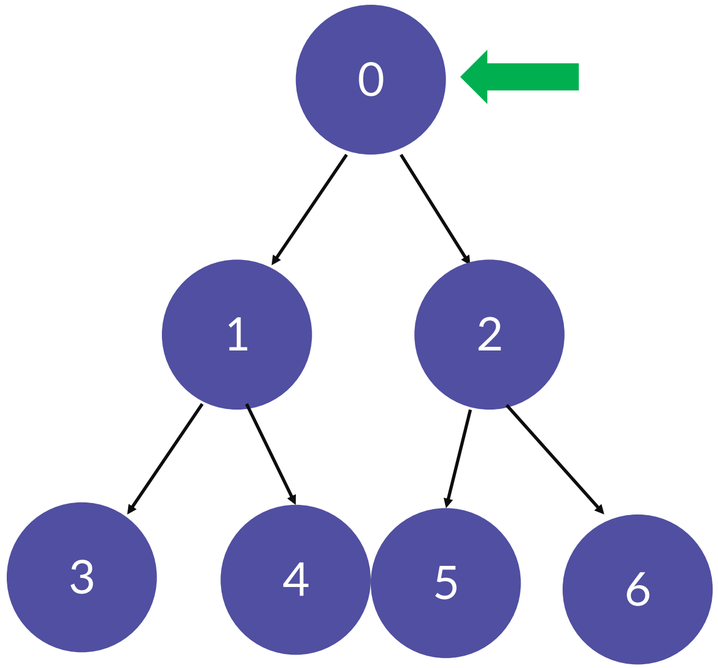





```python
def _dfs(self, index):
    if index >= len(self.tree):
        return

    print(self.tree[index])

    self._dfs(2 * index + 1)
    self._dfs(2 * index + 2)
```

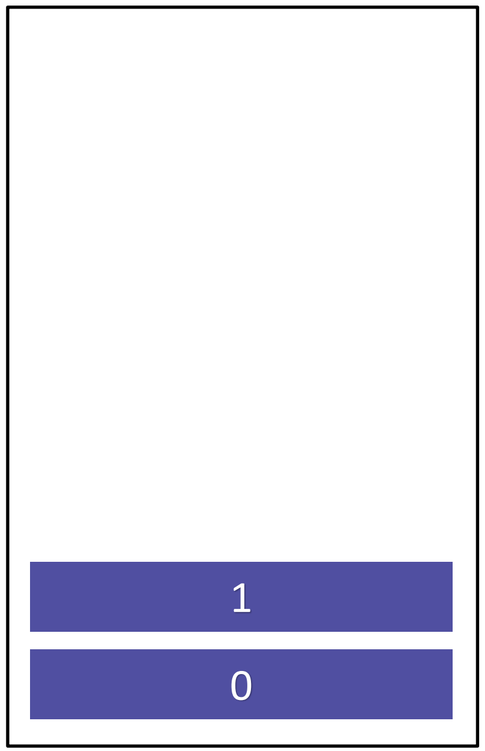
DFS를 구현하는 방법

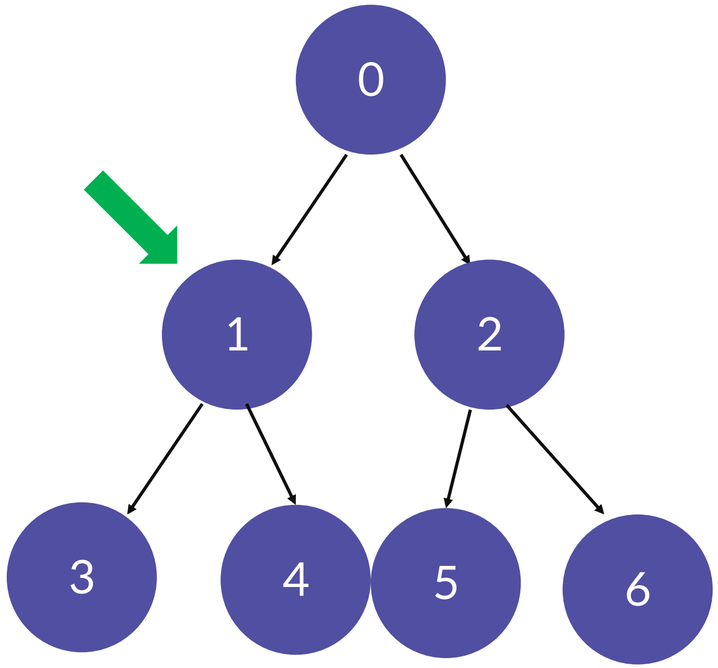





```python
def _dfs(self, index):
    if index >= len(self.tree):
        return

    print(self.tree[index])

    self._dfs(2 * index + 1)
    self._dfs(2 * index + 2)
```

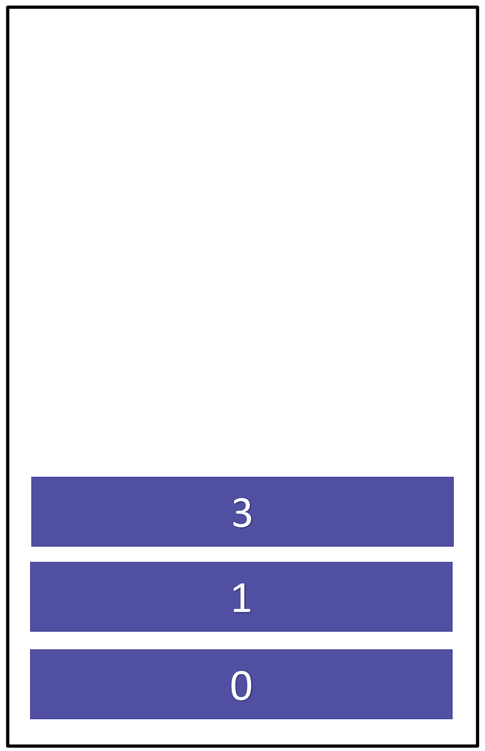
DFS를 구현하는 방법

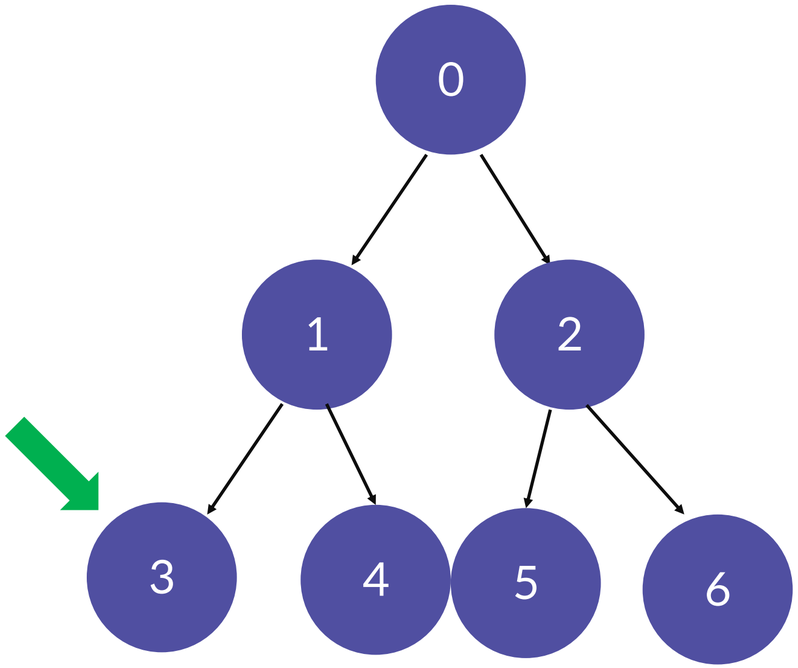





```python
def _dfs(self, index):
    if index >= len(self.tree):
        return

    print(self.tree[index])

    self._dfs(2 * index + 1)
    self._dfs(2 * index + 2)
```

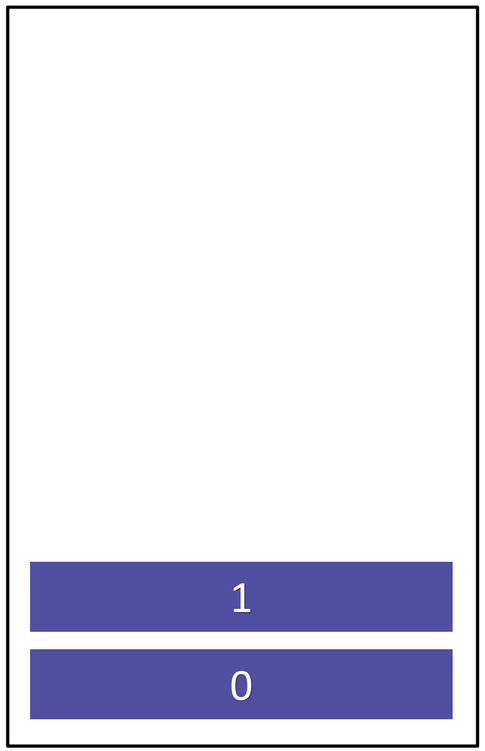
DFS를 구현하는 방법

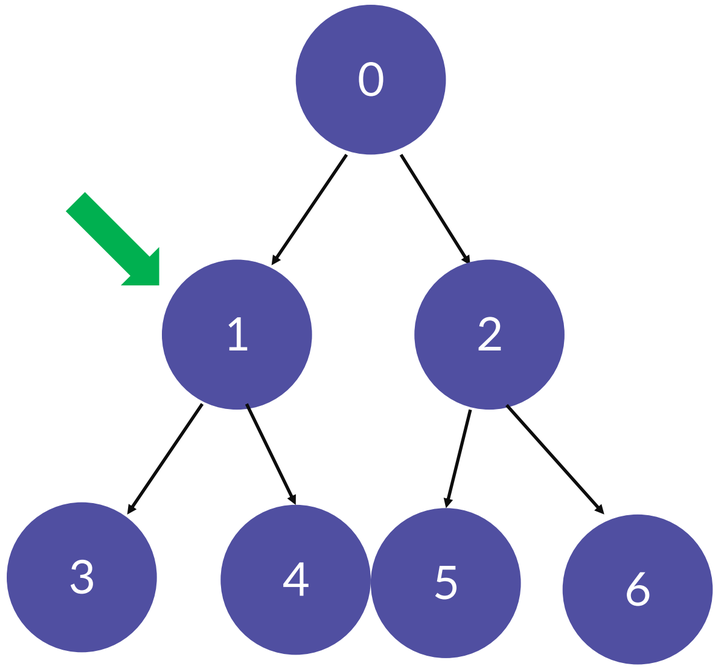





```python
def _dfs(self, index):
    if index >= len(self.tree):
        return

    print(self.tree[index])

    self._dfs(2 * index + 1)
    self._dfs(2 * index + 2)
```

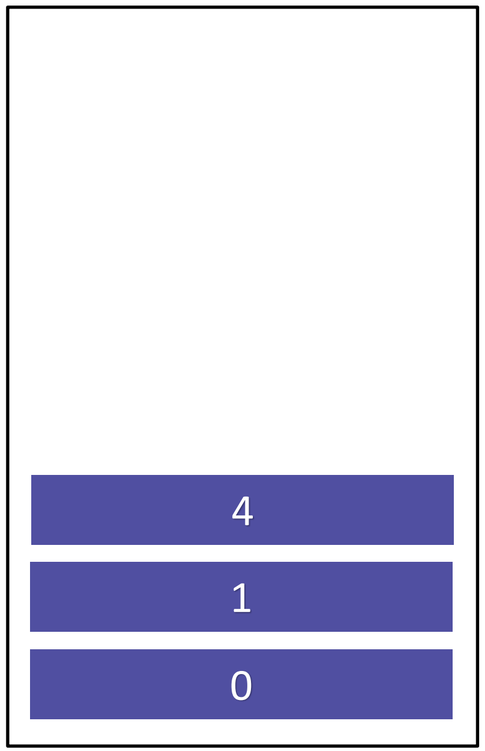
DFS를 구현하는 방법

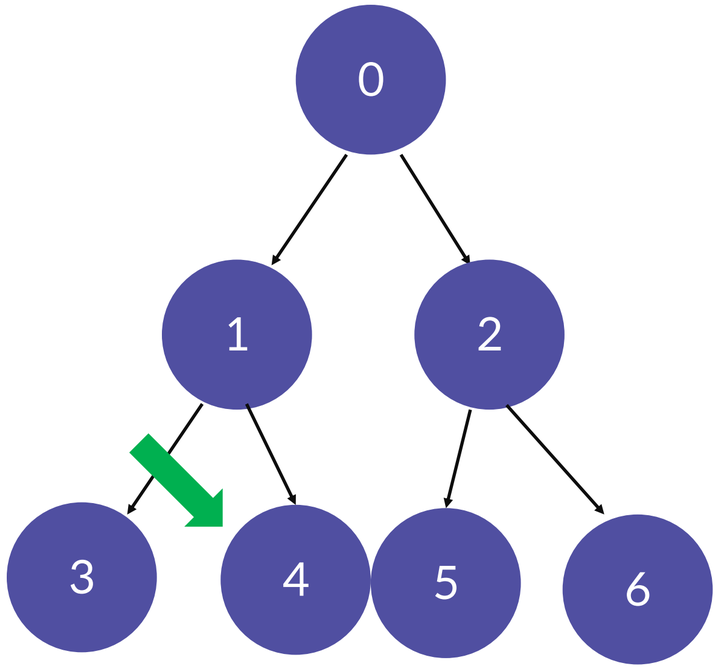





```python
def _dfs(self, index):
    if index >= len(self.tree):
        return

    print(self.tree[index])

    self._dfs(2 * index + 1)
    self._dfs(2 * index + 2)
```

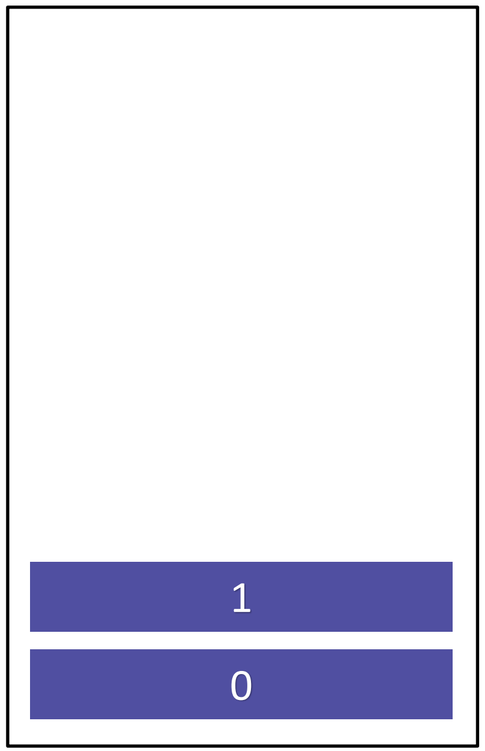
DFS를 구현하는 방법

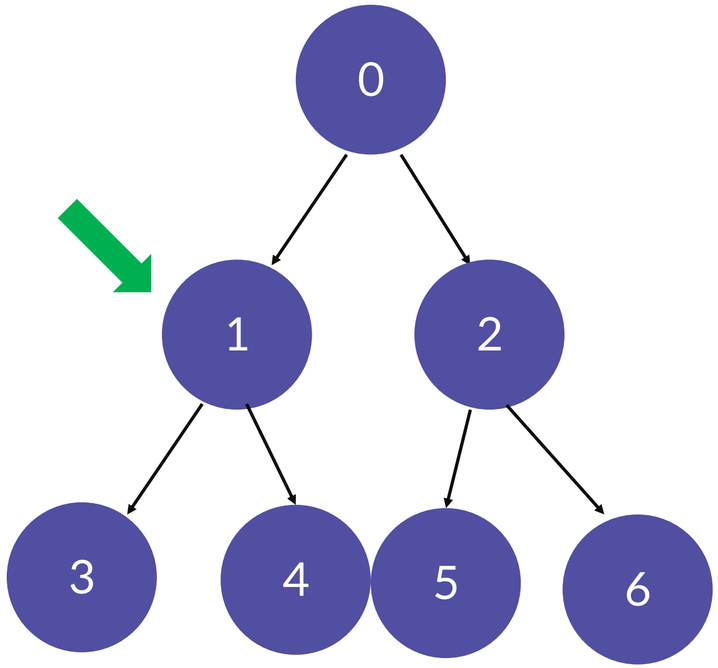





```python
def _dfs(self, index):
    if index >= len(self.tree):
        return

    print(self.tree[index])

    self._dfs(2 * index + 1)
    self._dfs(2 * index + 2)
```

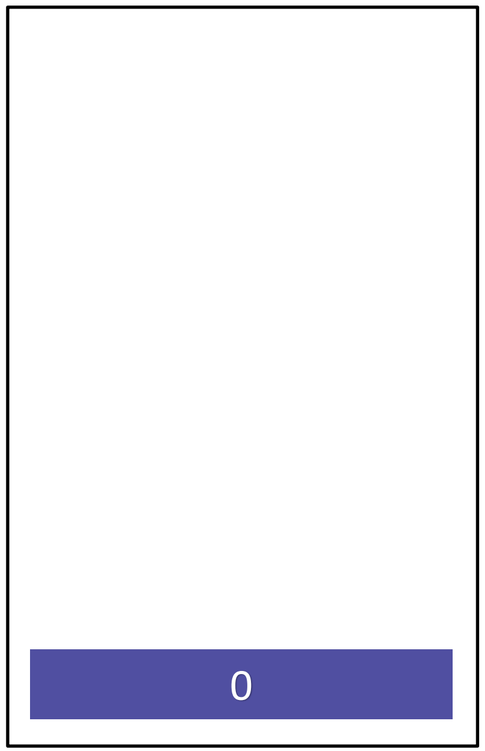
DFS를 구현하는 방법

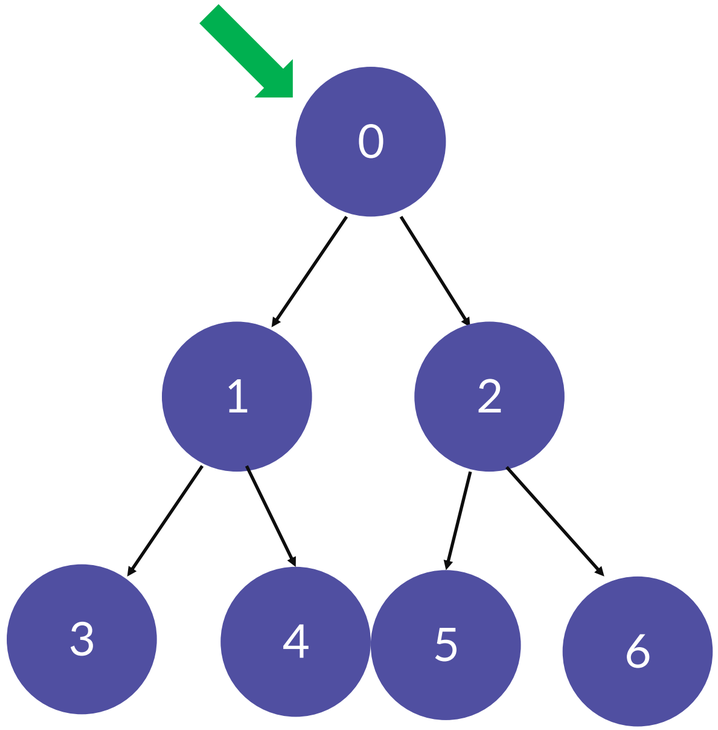





```python
def _dfs(self, index):
    if index >= len(self.tree):
        return

    print(self.tree[index])

    self._dfs(2 * index + 1)
    self._dfs(2 * index + 2)
```

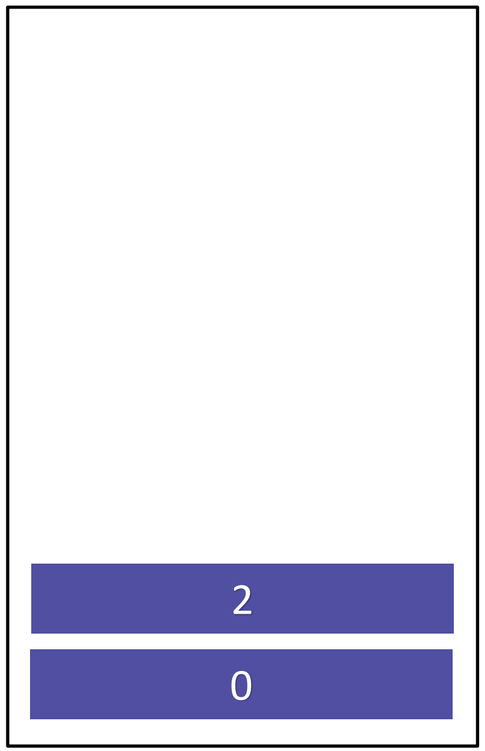
DFS를 구현하는 방법

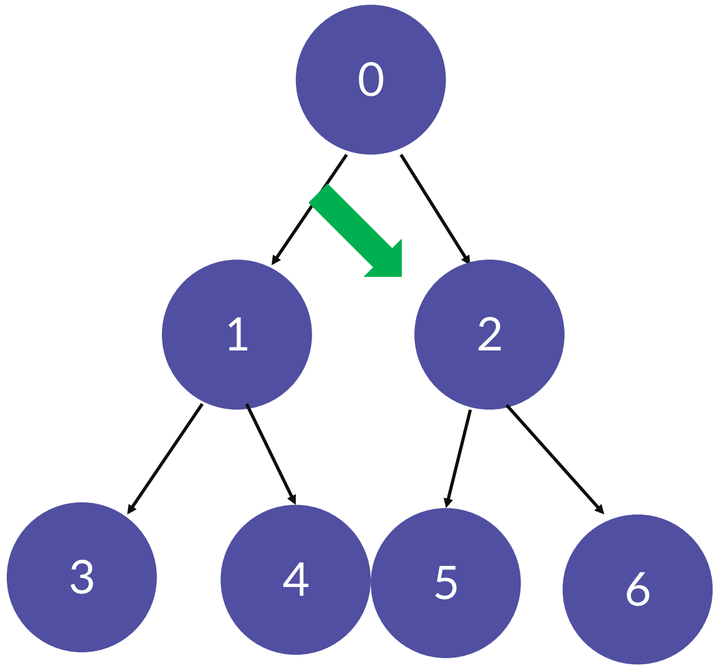





```python
def _dfs(self, index):
    if index >= len(self.tree):
        return

    print(self.tree[index])

    self._dfs(2 * index + 1)
    self._dfs(2 * index + 2)
```

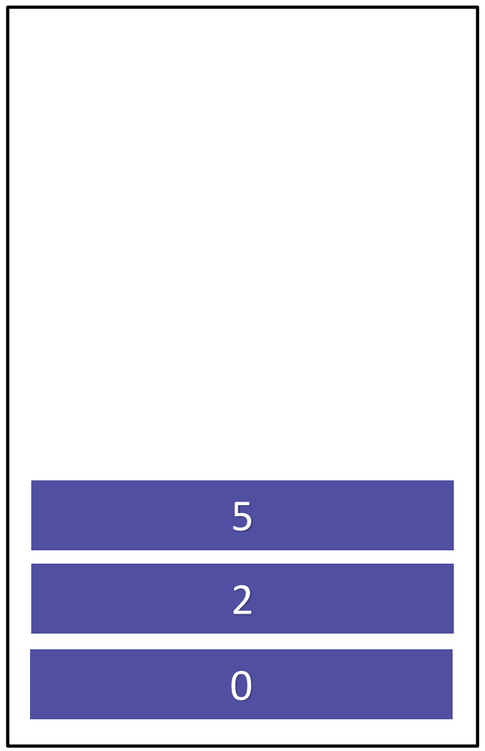
DFS를 구현하는 방법

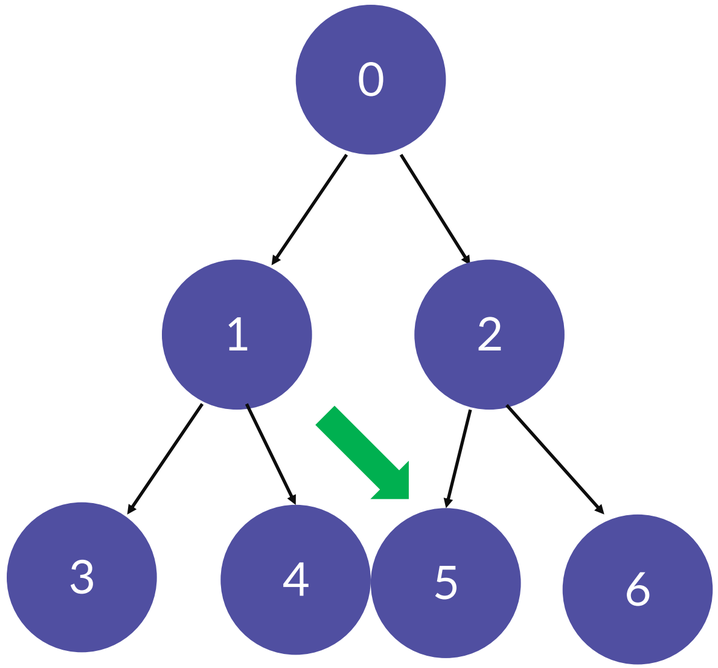





```python
def _dfs(self, index):
    if index >= len(self.tree):
        return

    print(self.tree[index])

    self._dfs(2 * index + 1)
    self._dfs(2 * index + 2)
```

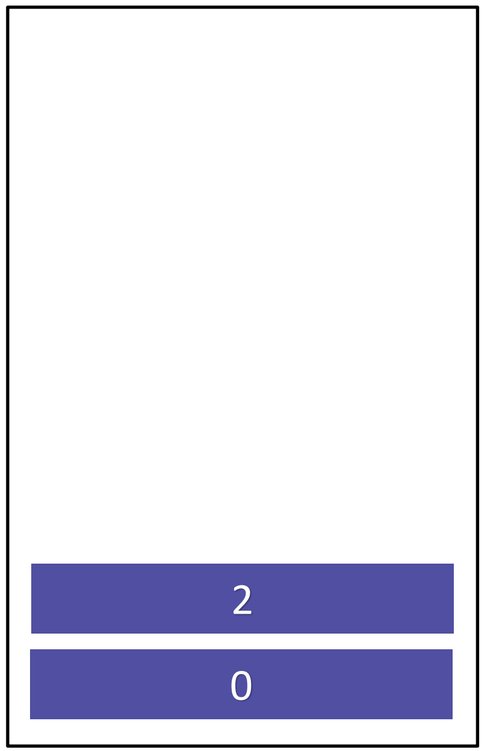
DFS를 구현하는 방법

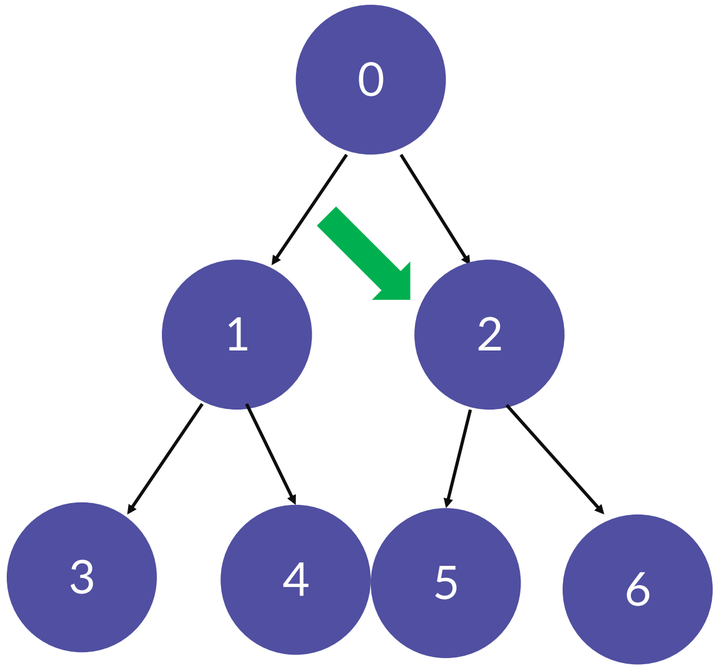





```python
def _dfs(self, index):
    if index >= len(self.tree):
        return

    print(self.tree[index])

    self._dfs(2 * index + 1)
    self._dfs(2 * index + 2)
```

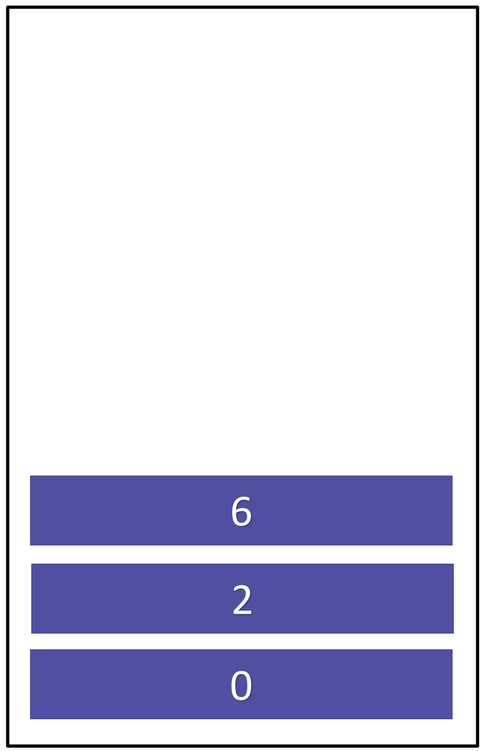
DFS를 구현하는 방법

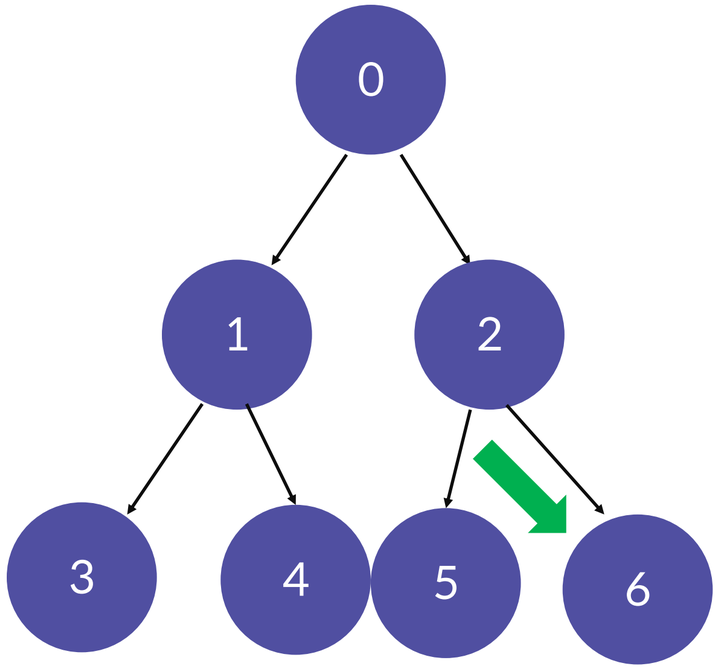





```python
def _dfs(self, index):
    if index >= len(self.tree):
        return

    print(self.tree[index])

    self._dfs(2 * index + 1)
    self._dfs(2 * index + 2)
```

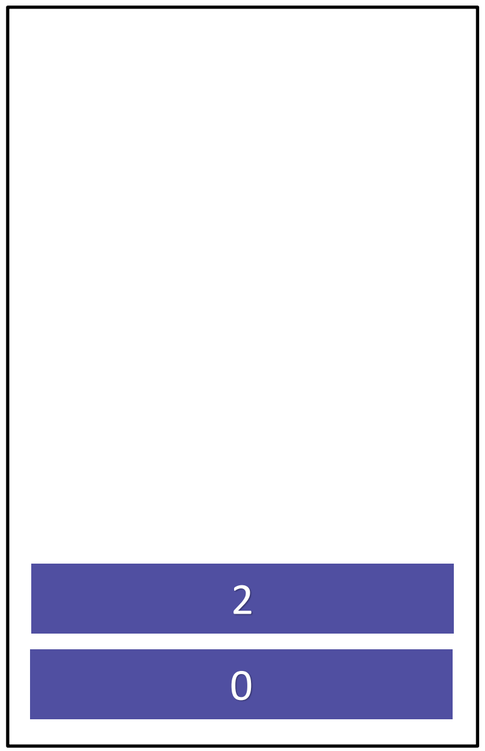
DFS를 구현하는 방법

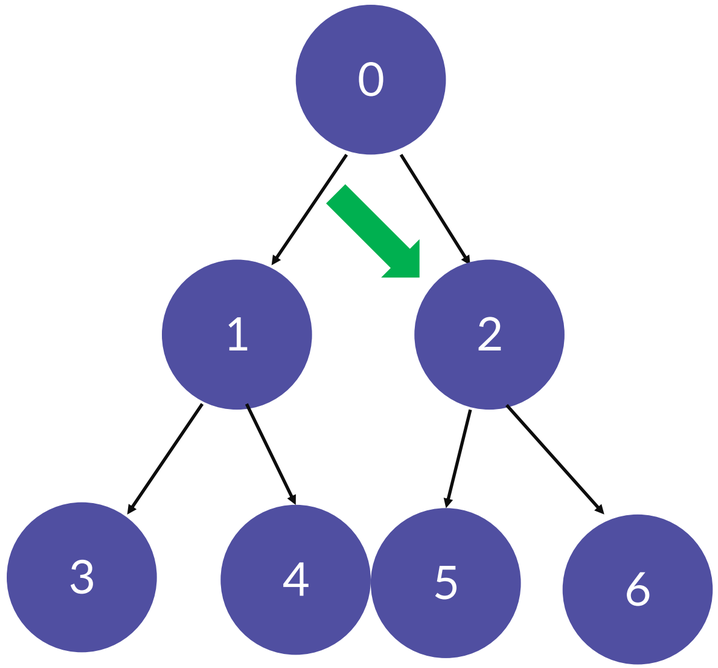





```python
def _dfs(self, index):
    if index >= len(self.tree):
        return

    print(self.tree[index])

    self._dfs(2 * index + 1)
    self._dfs(2 * index + 2)
```

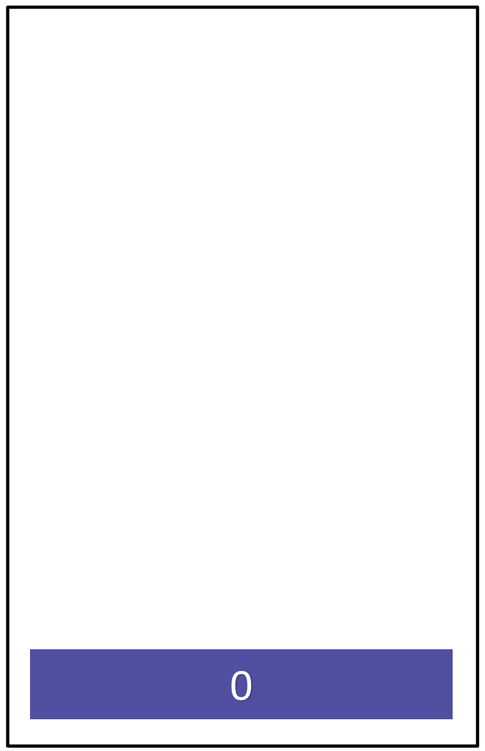
DFS를 구현하는 방법

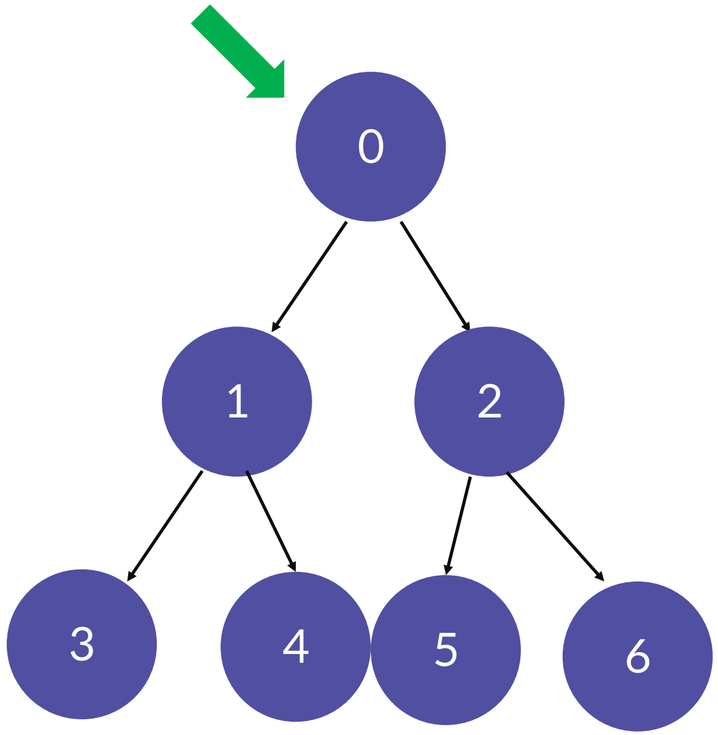





```python
def _dfs(self, index):
    if index >= len(self.tree):
        return

    print(self.tree[index])

    self._dfs(2 * index + 1)
    self._dfs(2 * index + 2)
```

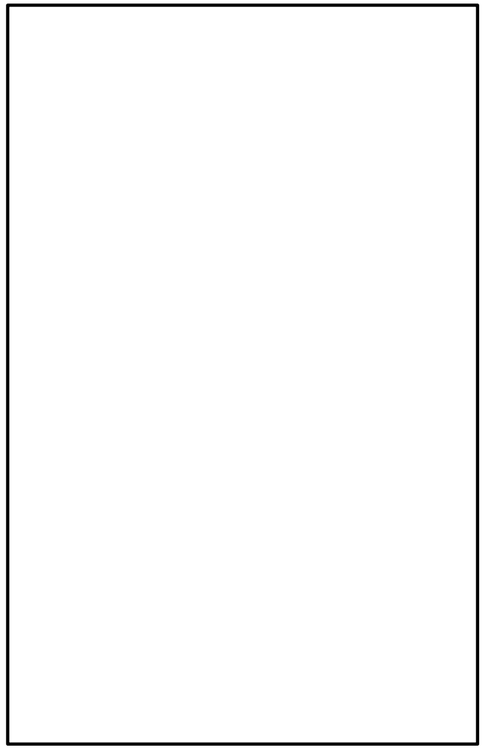
DFS를 구현하는 방법

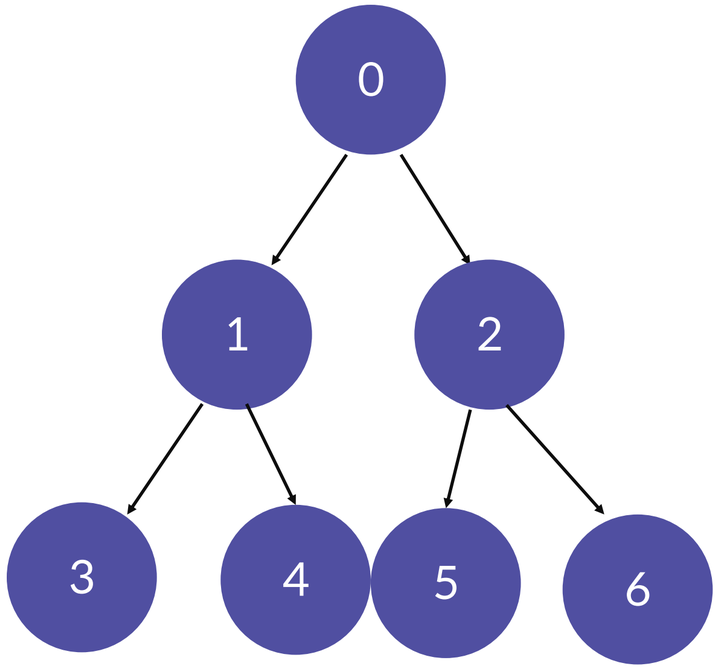





```python
def _dfs(self, index):
    if index >= len(self.tree):
        return

    print(self.tree[index])

    self._dfs(2 * index + 1)
    self._dfs(2 * index + 2)
```

DFS를 구현하는 방법






```python
def _dfs(self, index):
    if index >= len(self.tree):
        return

    print(self.tree[index])

    self._dfs(2 * index + 1)
    self._dfs(2 * index + 2)
```

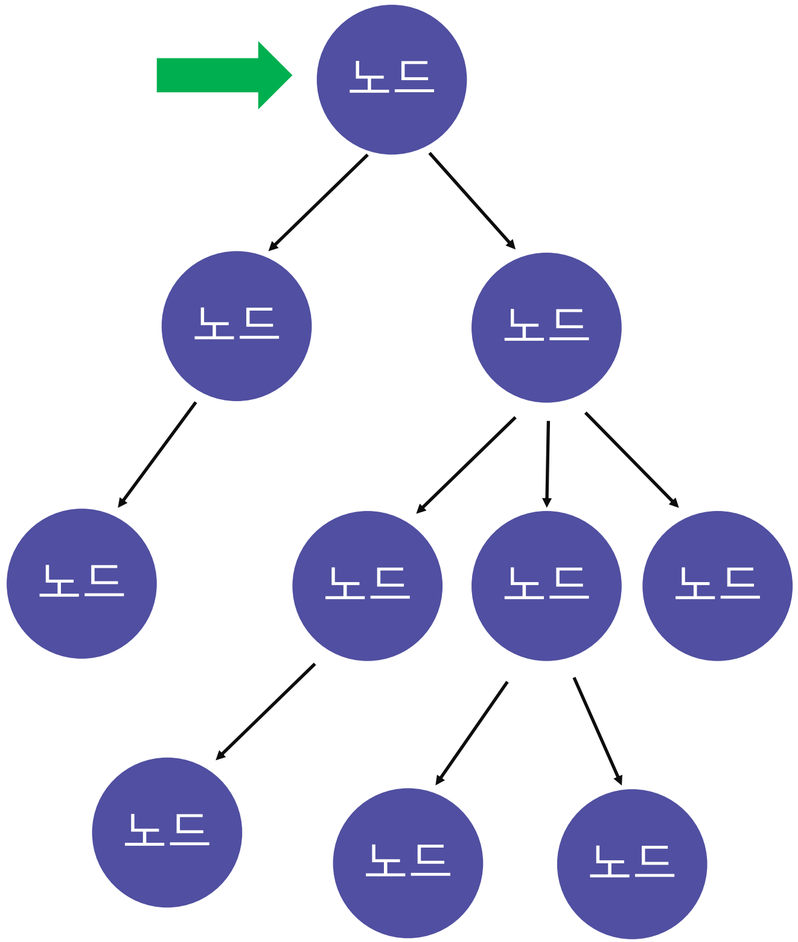
DFS를 구현하는 방법

- 재귀함수 또는 스택을 이용하여 구현
- 여기서는 재귀함수를 이용하여 구현한 DFS 코드를 살펴보자!

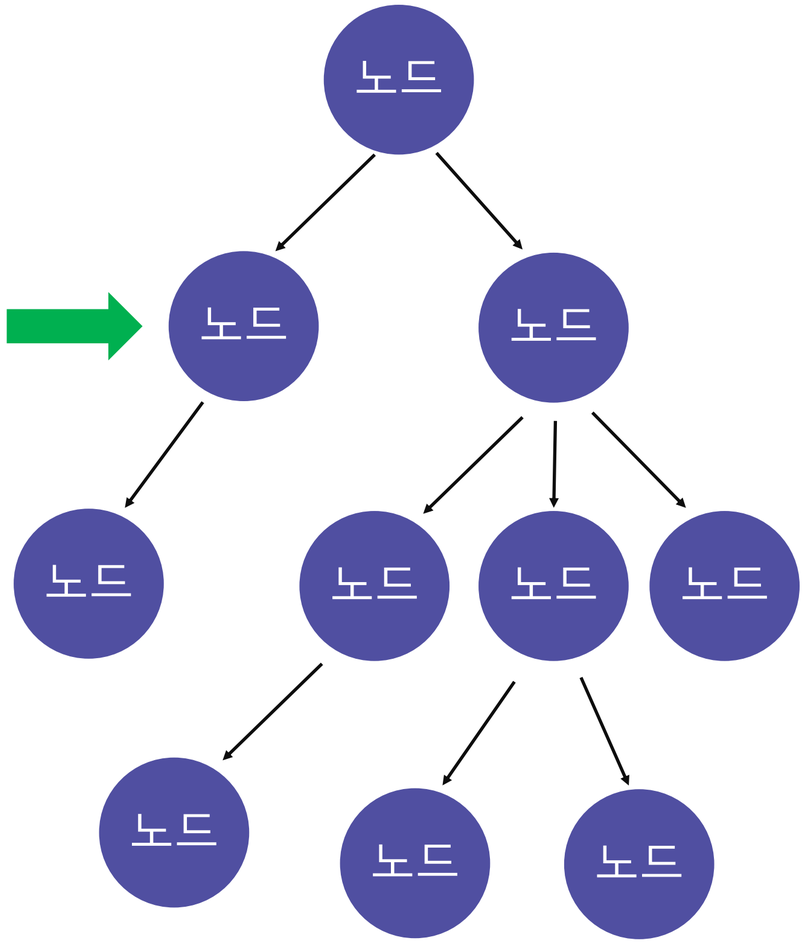
BFS

- 트리의 너비를 우선으로 탐색하는 방법
- 자식1 -> 자식2 -> 자식3 -> 자식1의 자식1 -> 자식1의 자식2 -> …

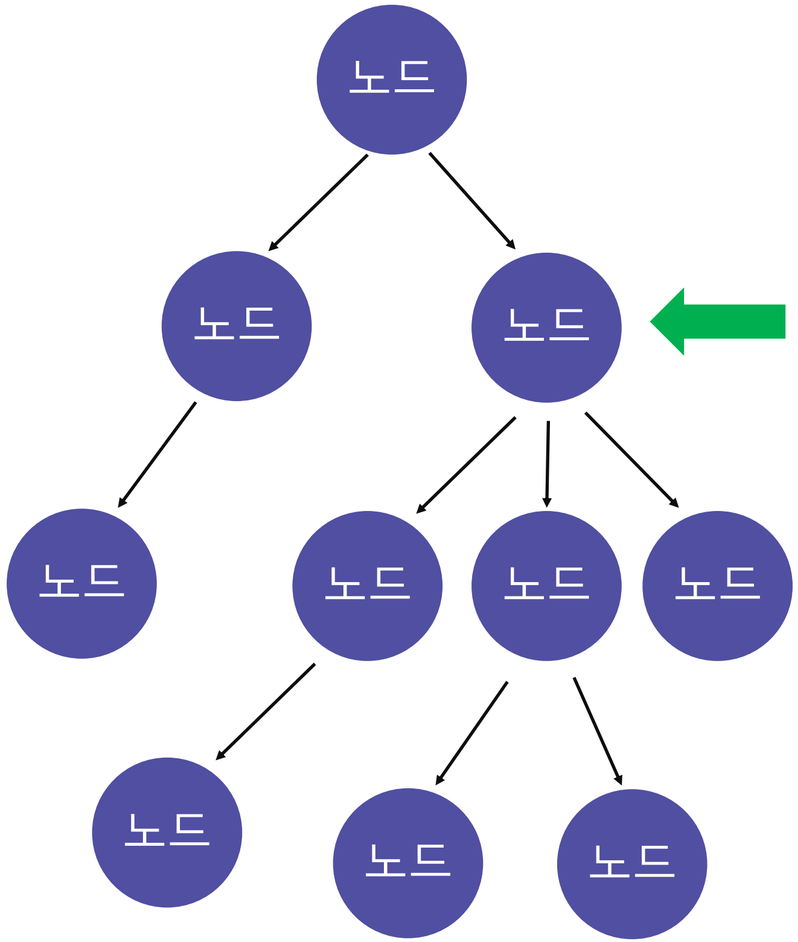
BFS



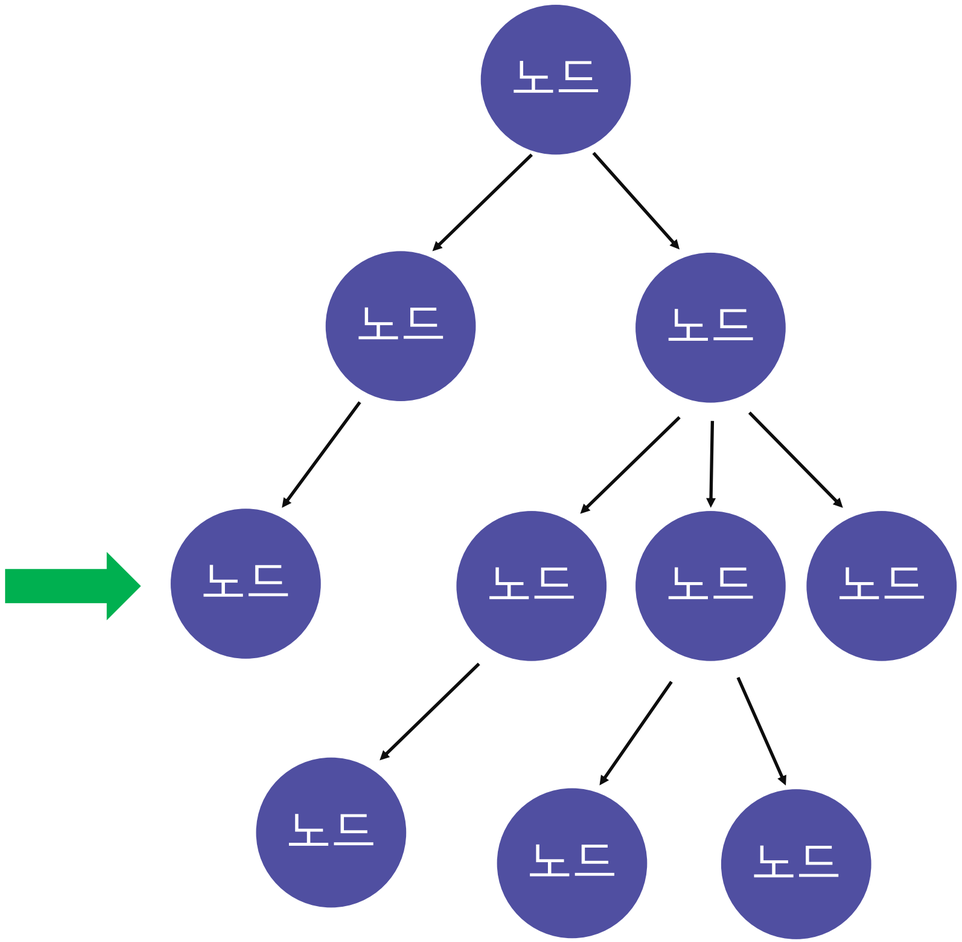
BFS



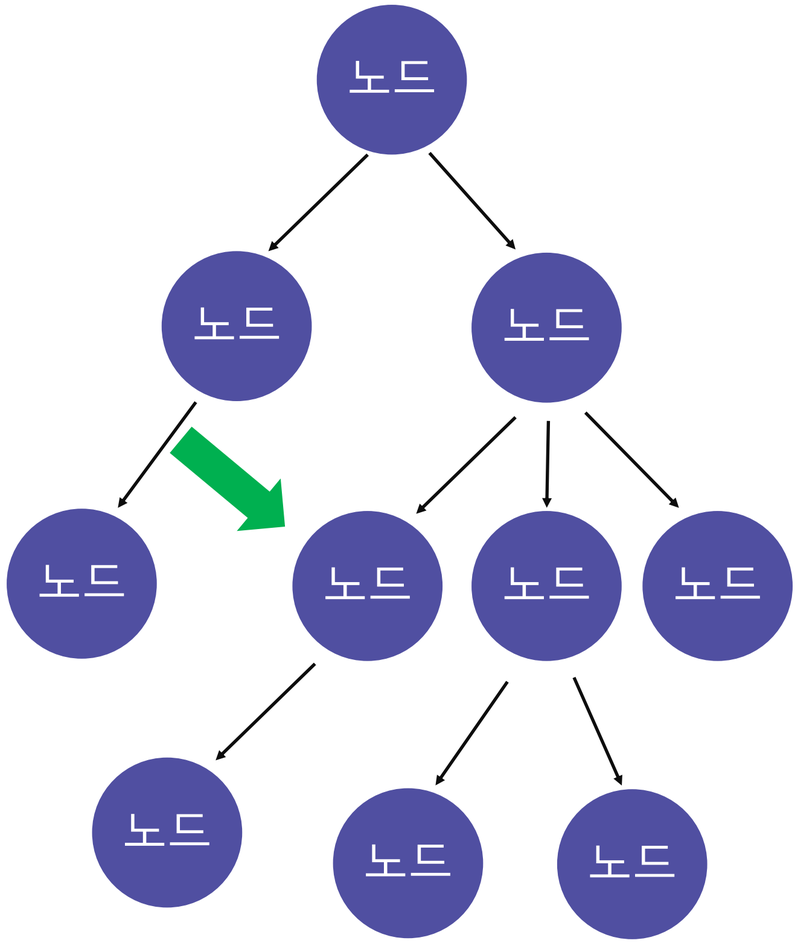
BFS



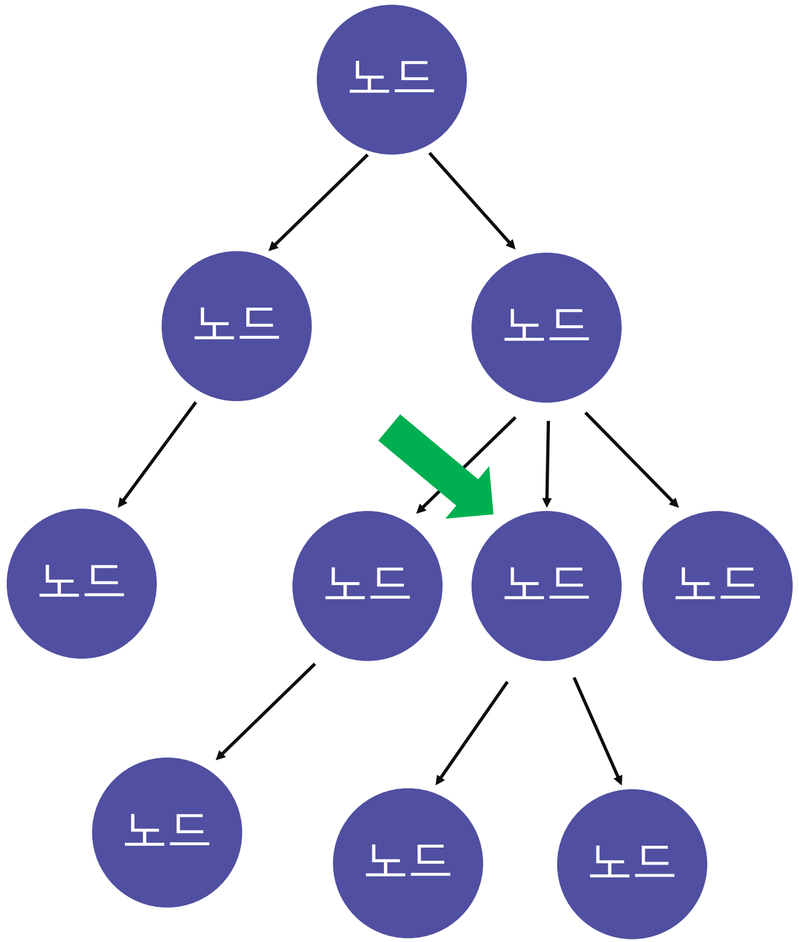
BFS



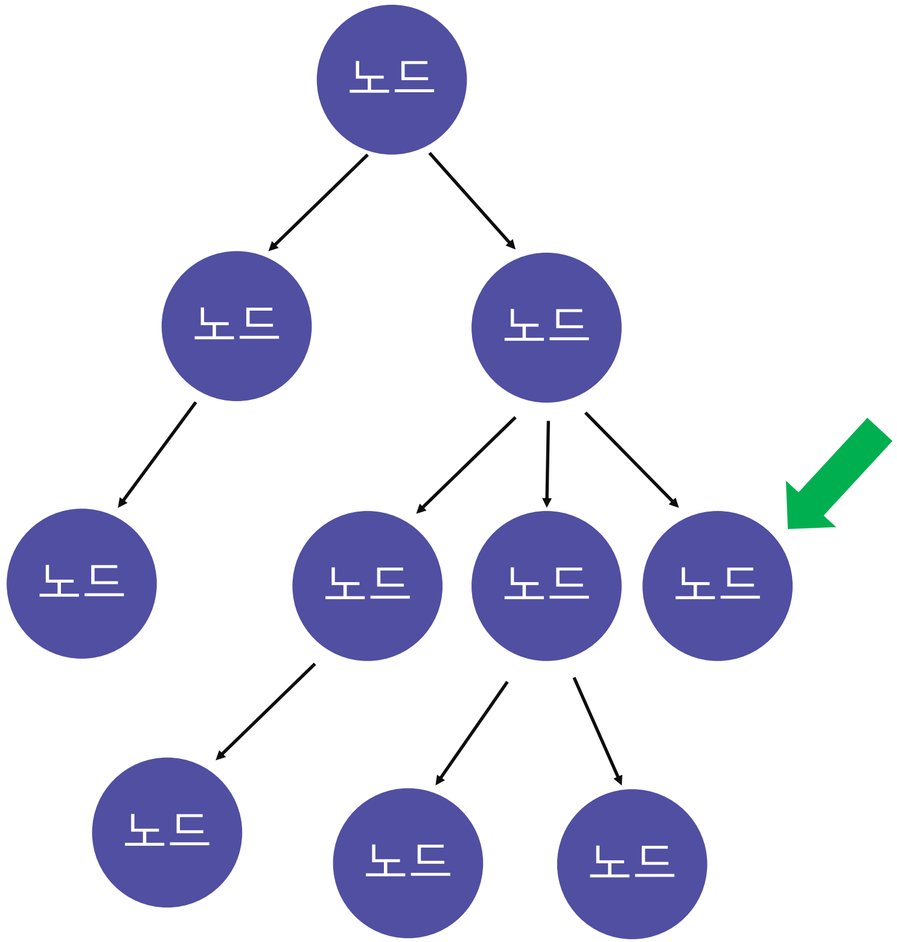
BFS



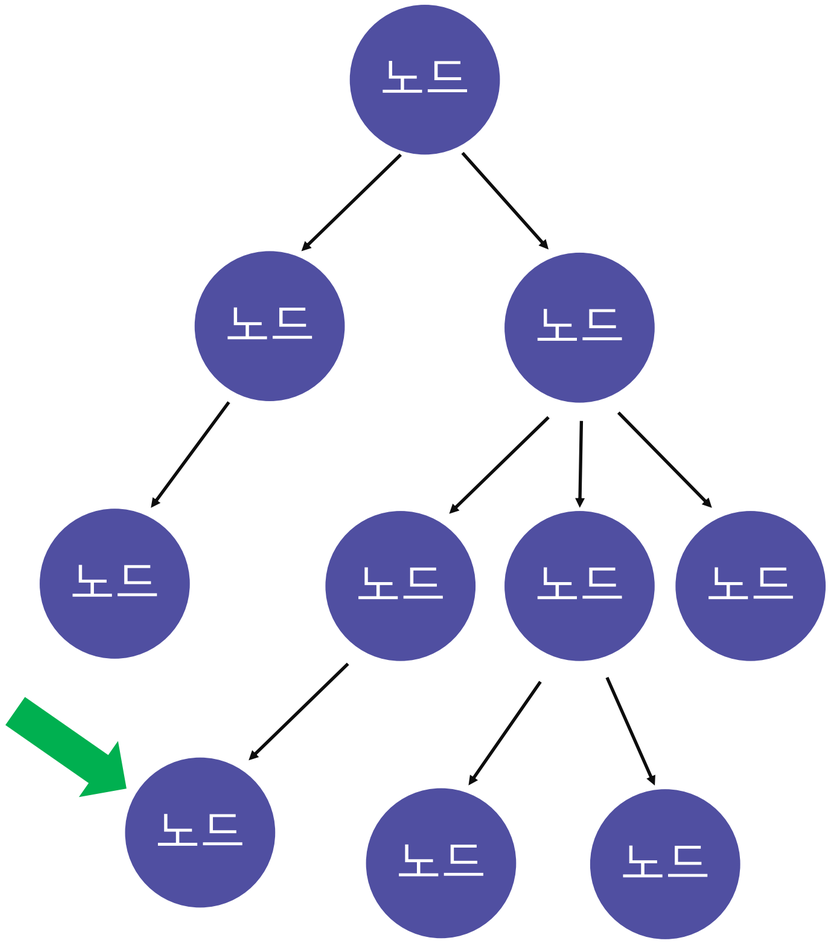
BFS



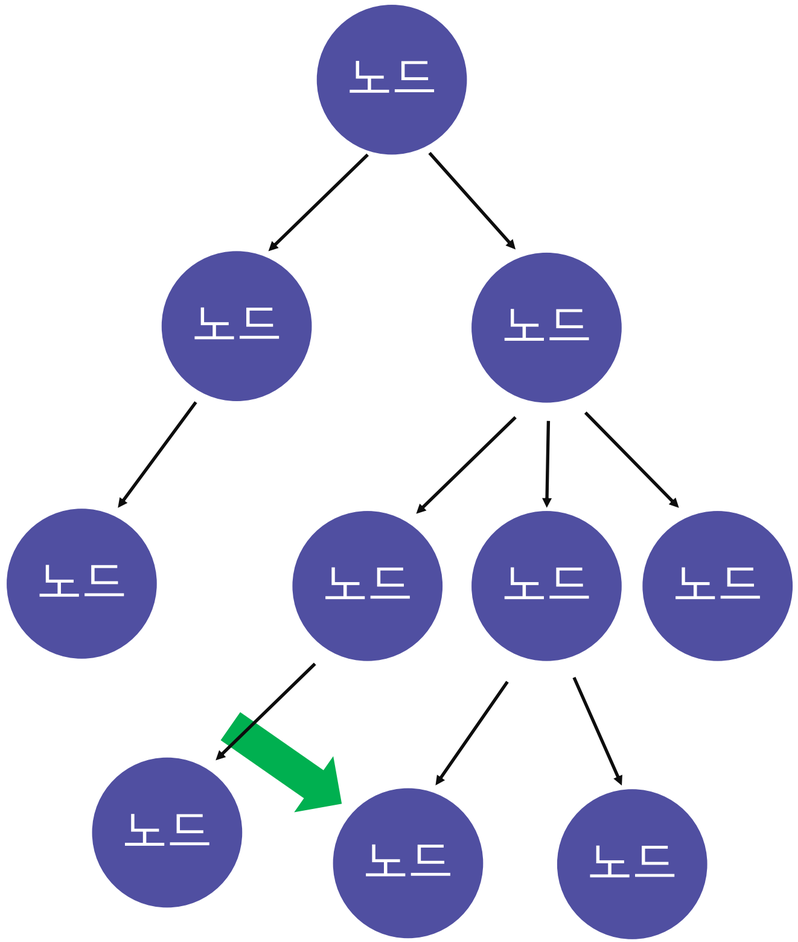
BFS



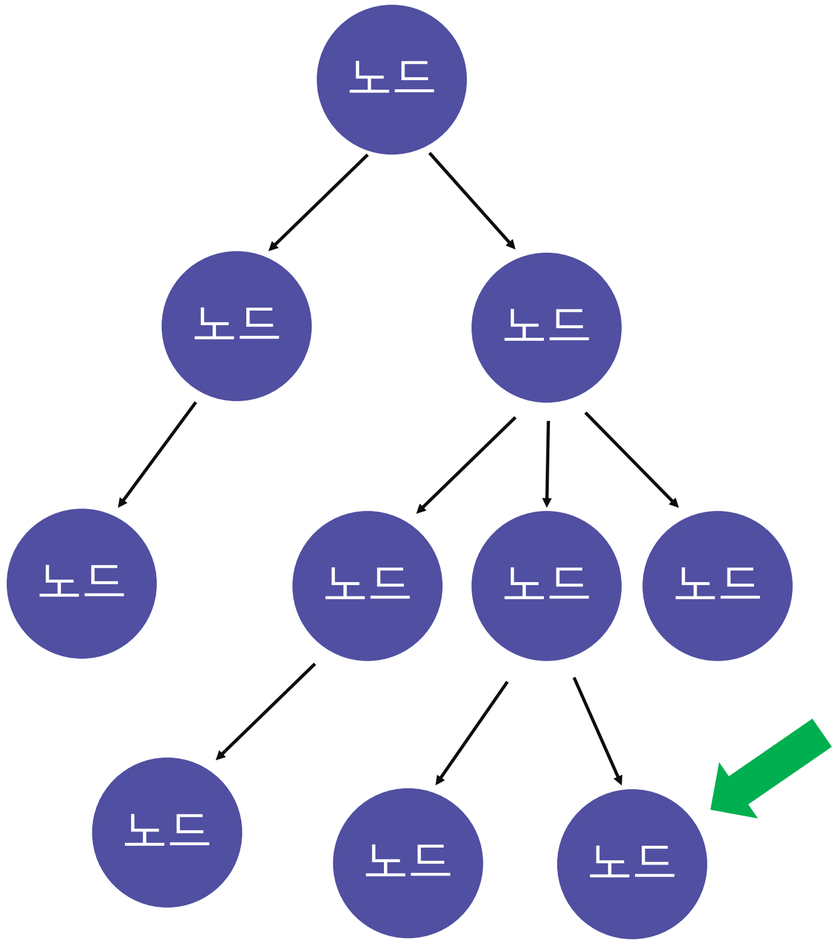
BFS



BFS



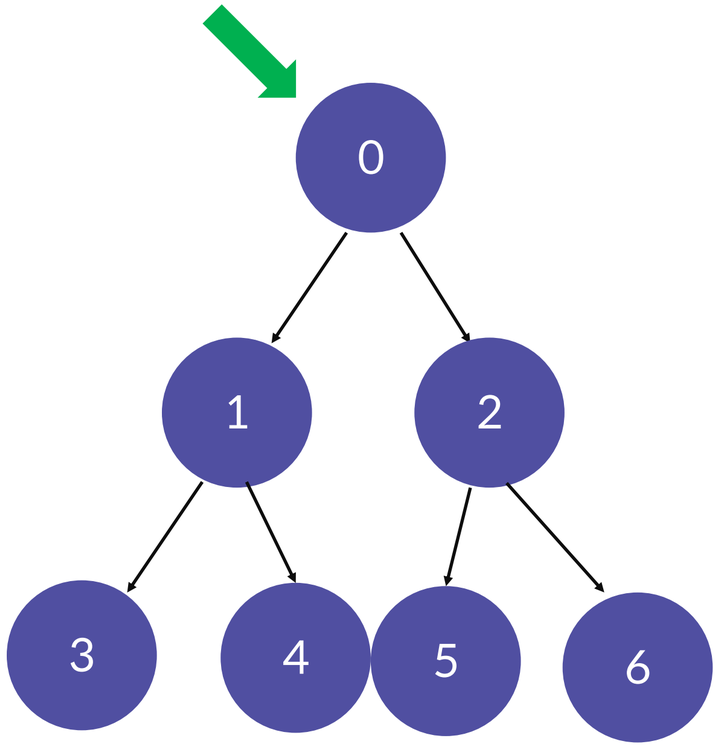
BFS



BFS를 구현하는 방법

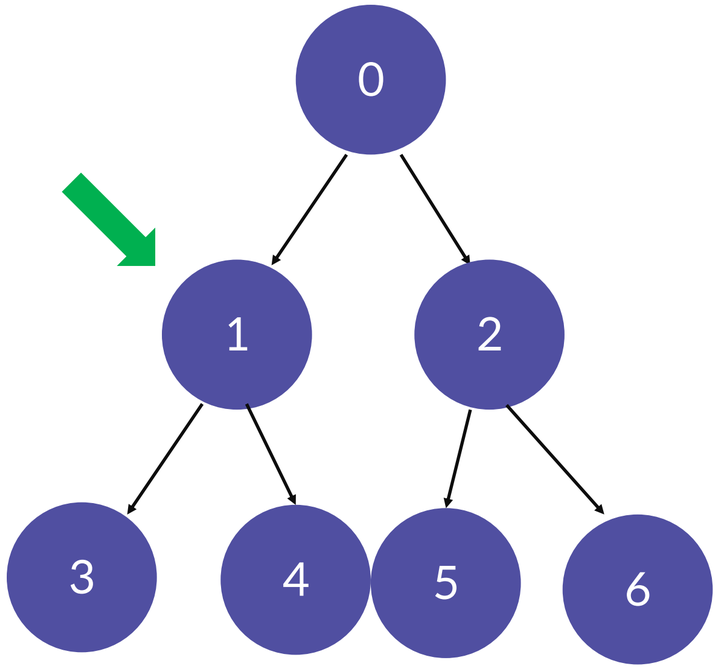
- 큐를 이용하여 구현!
- 자식들을 큐에 다 넣으면서 탐색을 진행한다!

BFS를 구현하는 방법

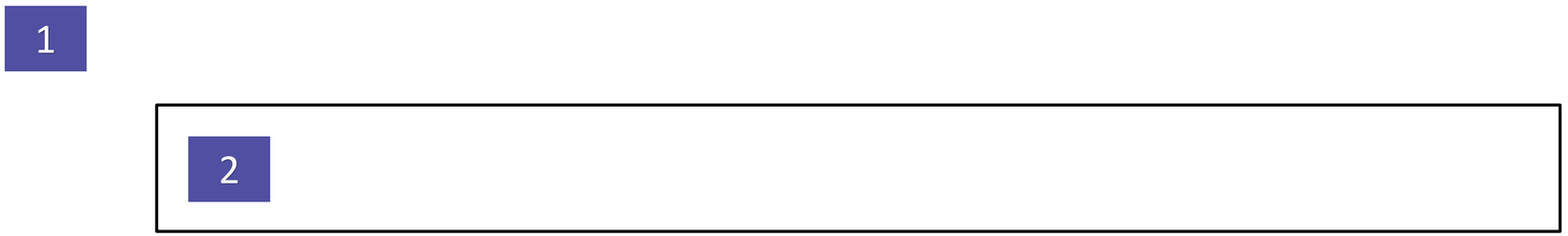


자식들을 큐에 삽입

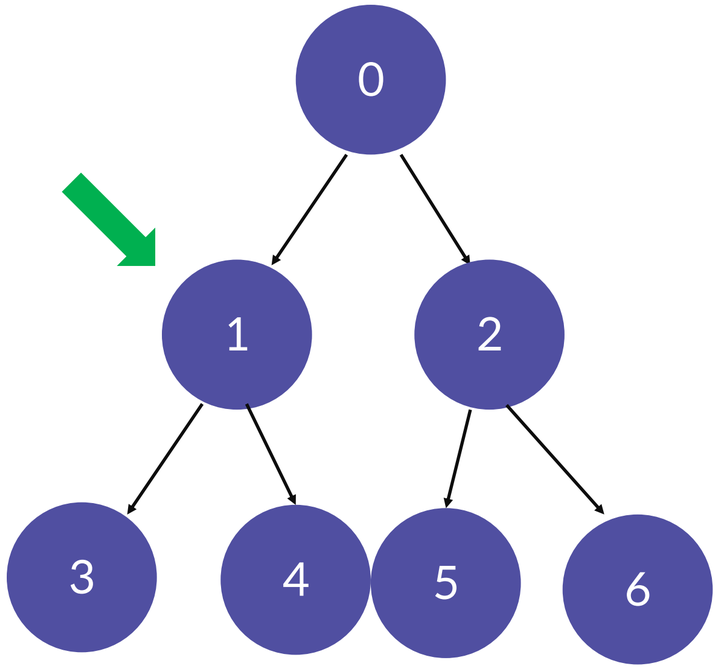
BFS를 구현하는 방법



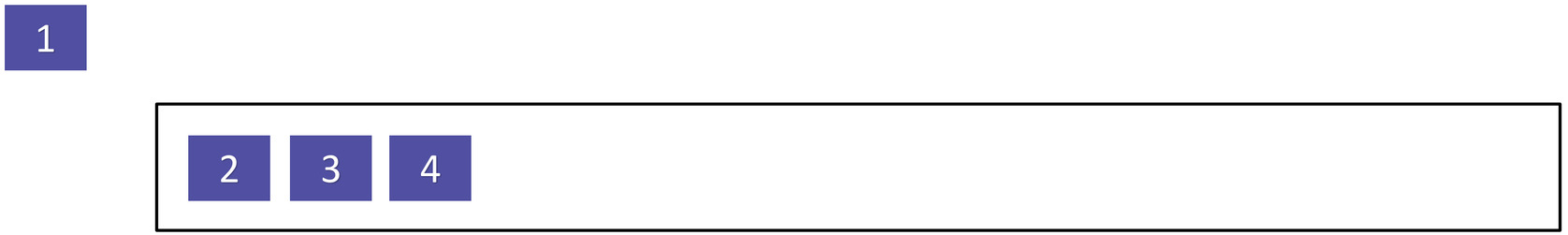
큐에서 원소를 pop하고
그 노드로 이동



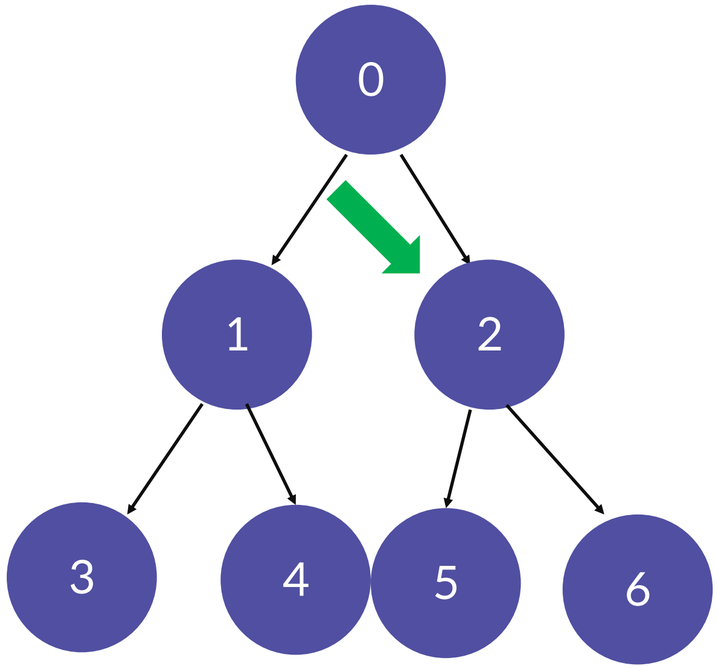
BFS를 구현하는 방법



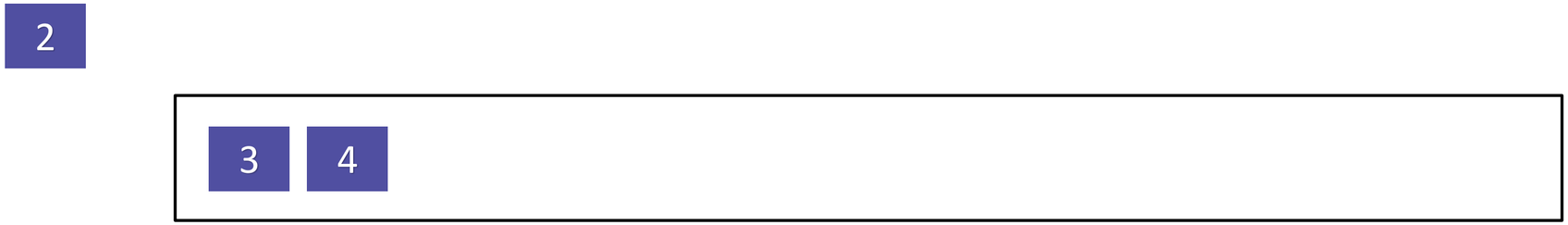
자식들을 큐에 삽입



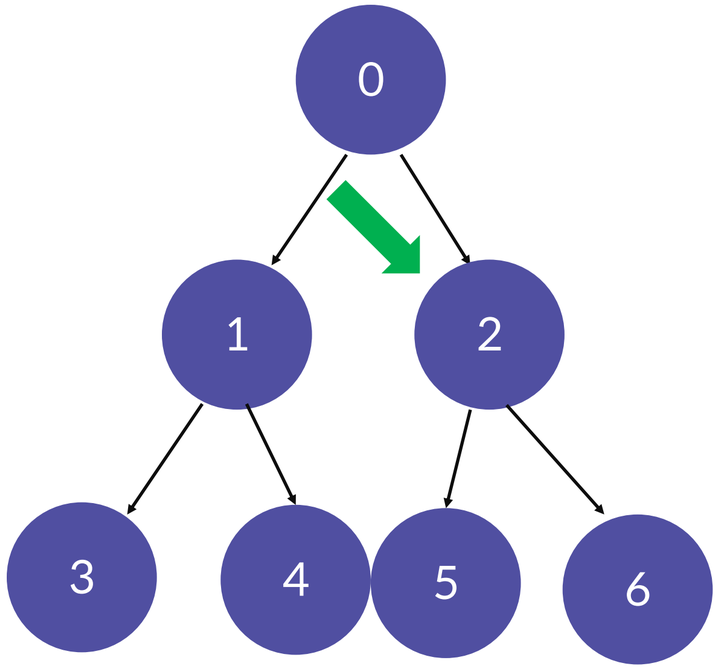
BFS를 구현하는 방법



큐에서 pop



BFS를 구현하는 방법

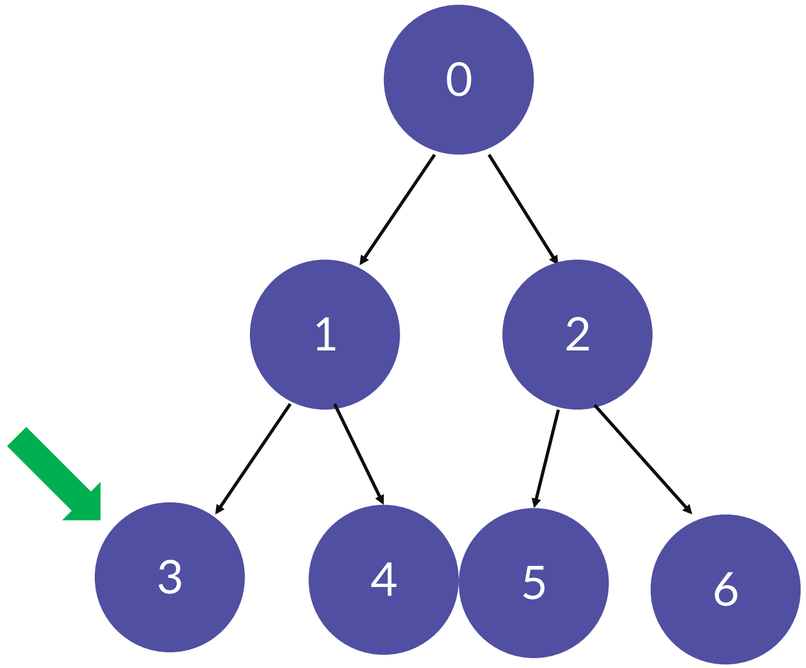


자식들을 큐에 삽입

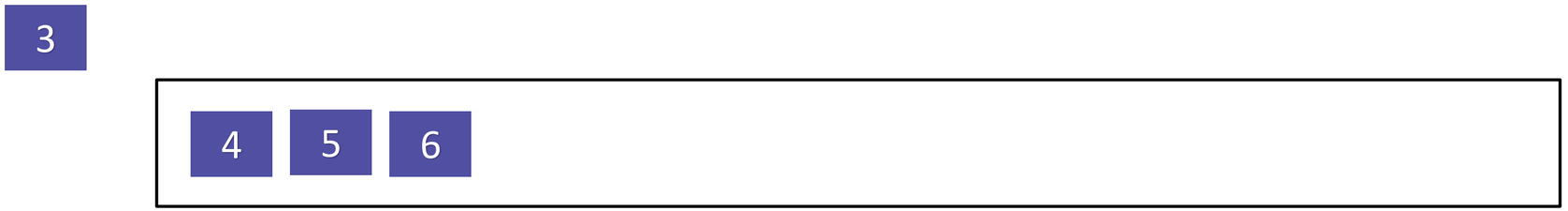
2

3 4 5 6

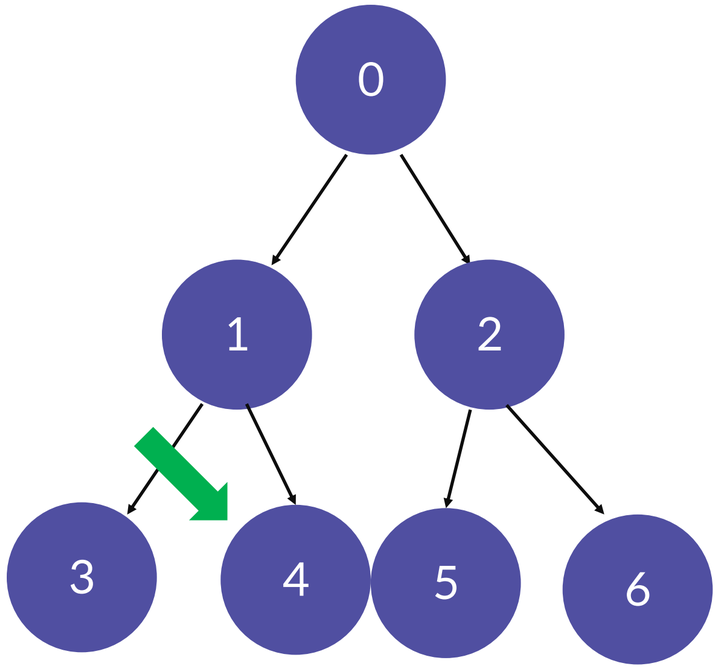
BFS를 구현하는 방법



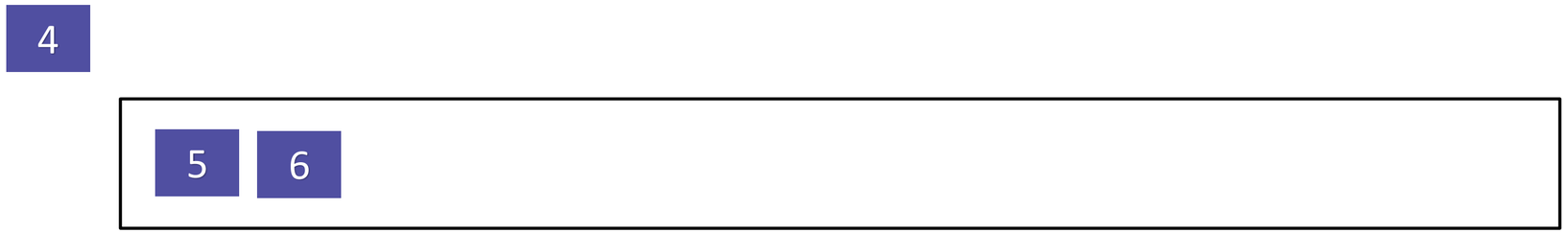
큐에서 pop, 이동



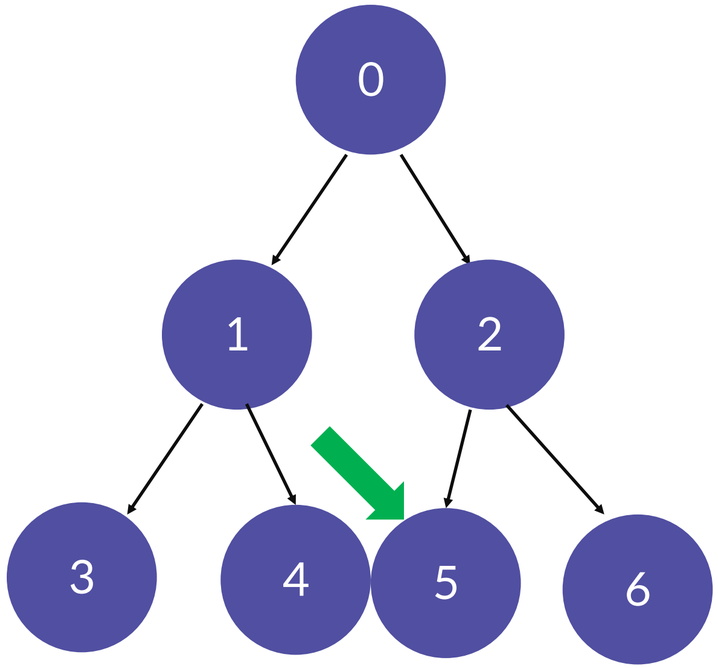
BFS를 구현하는 방법



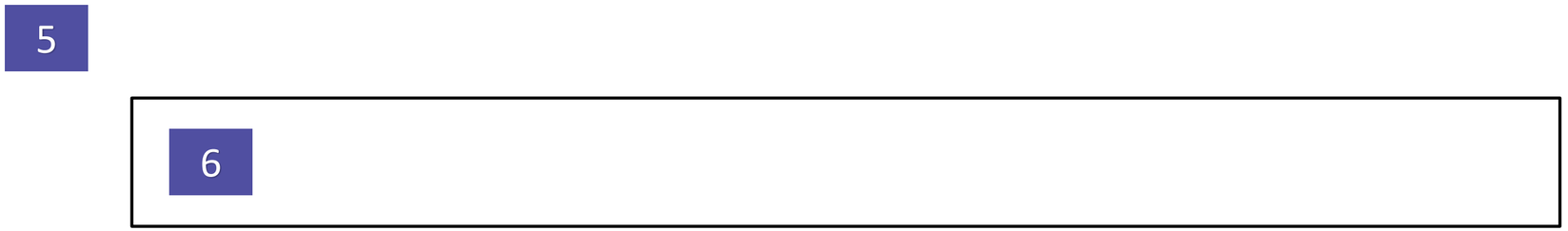
자식이 없기 때문에
바로 pop 후 이동



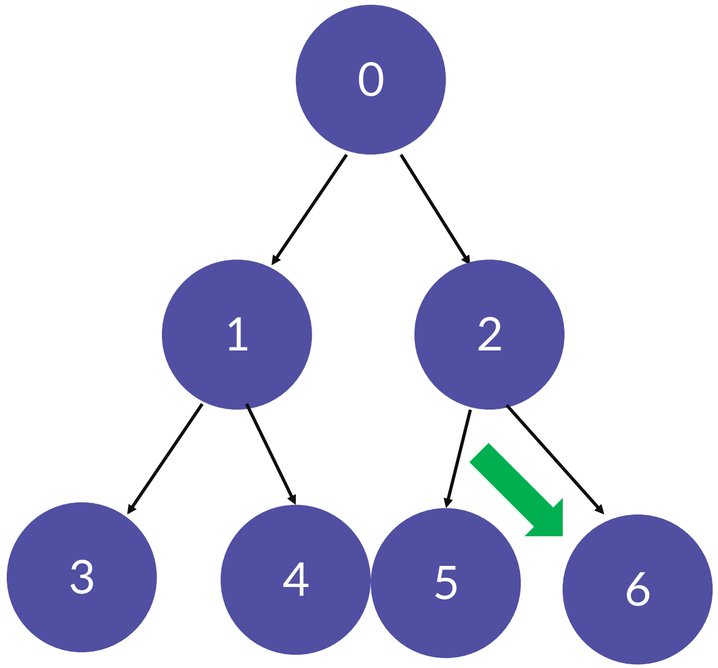
BFS를 구현하는 방법



큐가 빌 때 까지 반복



BFS를 구현하는 방법



큐가 빌 때 까지 반복

6

DFS와 BFS를 구현해보자

- With Python Code!!

DFS와 BFS를 언제 사용하면 좋을까?

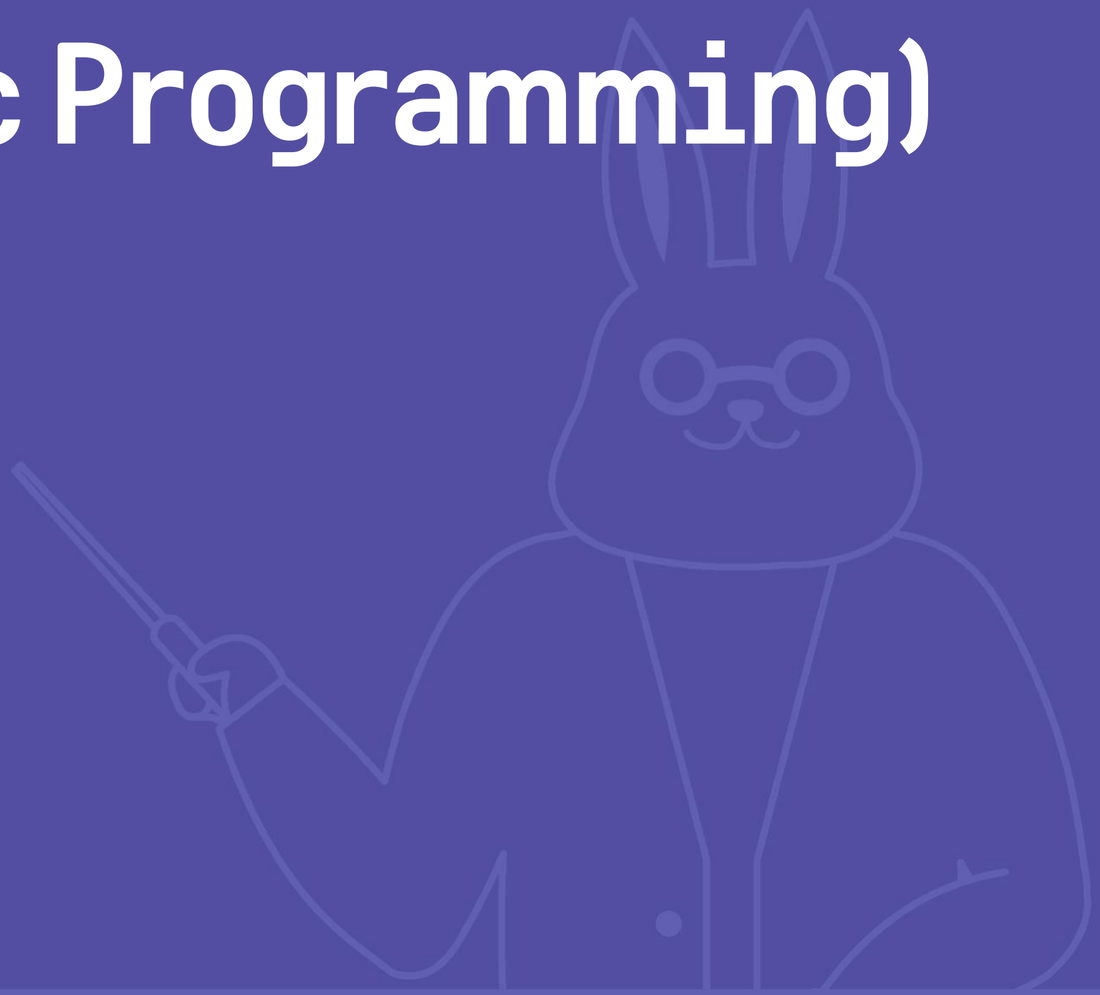
- 보통 길찾기 또는 배열에서 움직임을 구현할 때 많이 사용

- DFS : 재귀를 통해 간단하게 구현할 수 있지만,
  100번 이상 재귀호출이 반복될 경우 Call Stack이 터질 수 있음

- BFS : 큐를 이용해야 하므로 구현이 조금 복잡하지만,
  DFS보다 조금 더 큰 사이즈의 문제를 해결할 수 있다.

05

동적 계획법 (Dynamic Programming)



Dynamic Programming 이란?

- 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = ?

Dynamic Programming 이란?

- 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = ?

- 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = ?

Dynamic Programming 이란?

- 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 8

- 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 8 + 1 = 9

Dynamic Programming 이란?

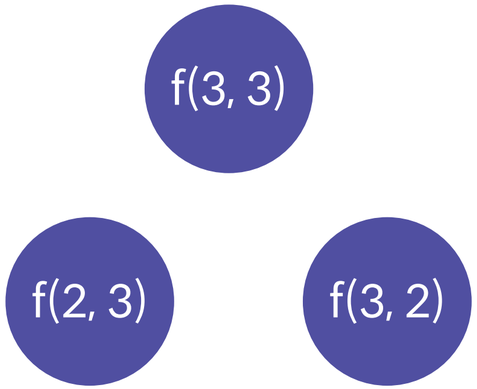
- 어떤 범위까지의 값을 구하기 위해, **그것과 다른 범위의 값**을 이용하는 방법
- 수학에서 점화식!
- “기억하며 풀기"

왜 쓸까?

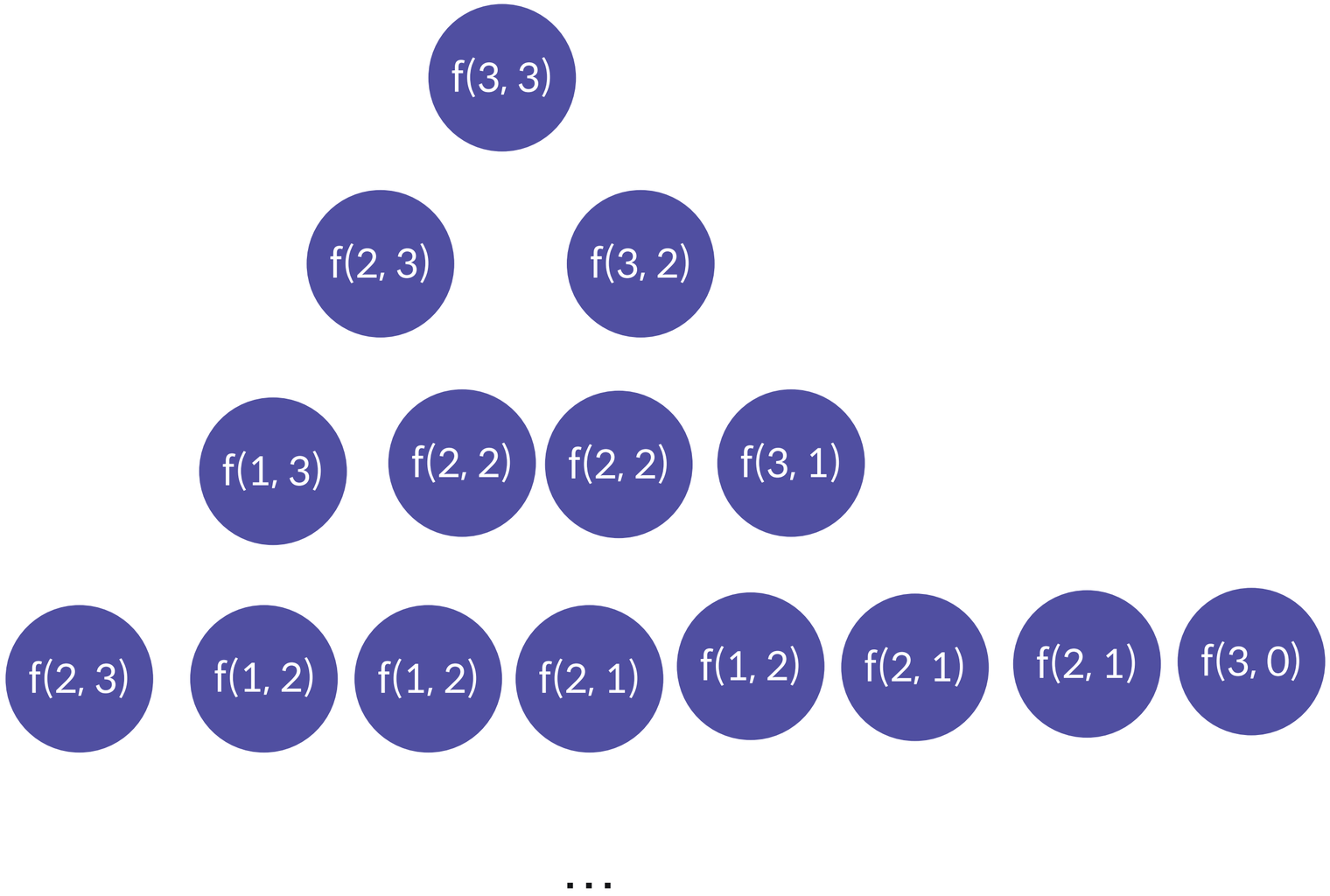
- f(a, b) = f(a-1, b) + f(a, b-1) (a, b >= 1)
- f(0, 0) = 1 이고, 임의의 자연수 n에 대하여 f(n, 0) = f(0, n) = 1 이라고 하자.
- Ex) f(1, 0) = f(0, 1) = 1
- Ex) f(2, 1) = f(1, 1) + f(2, 0) = f(0, 1) + f(1, 0) + 1 = 3

왜 쓸까?

왜 쓸까?



왜 쓸까?



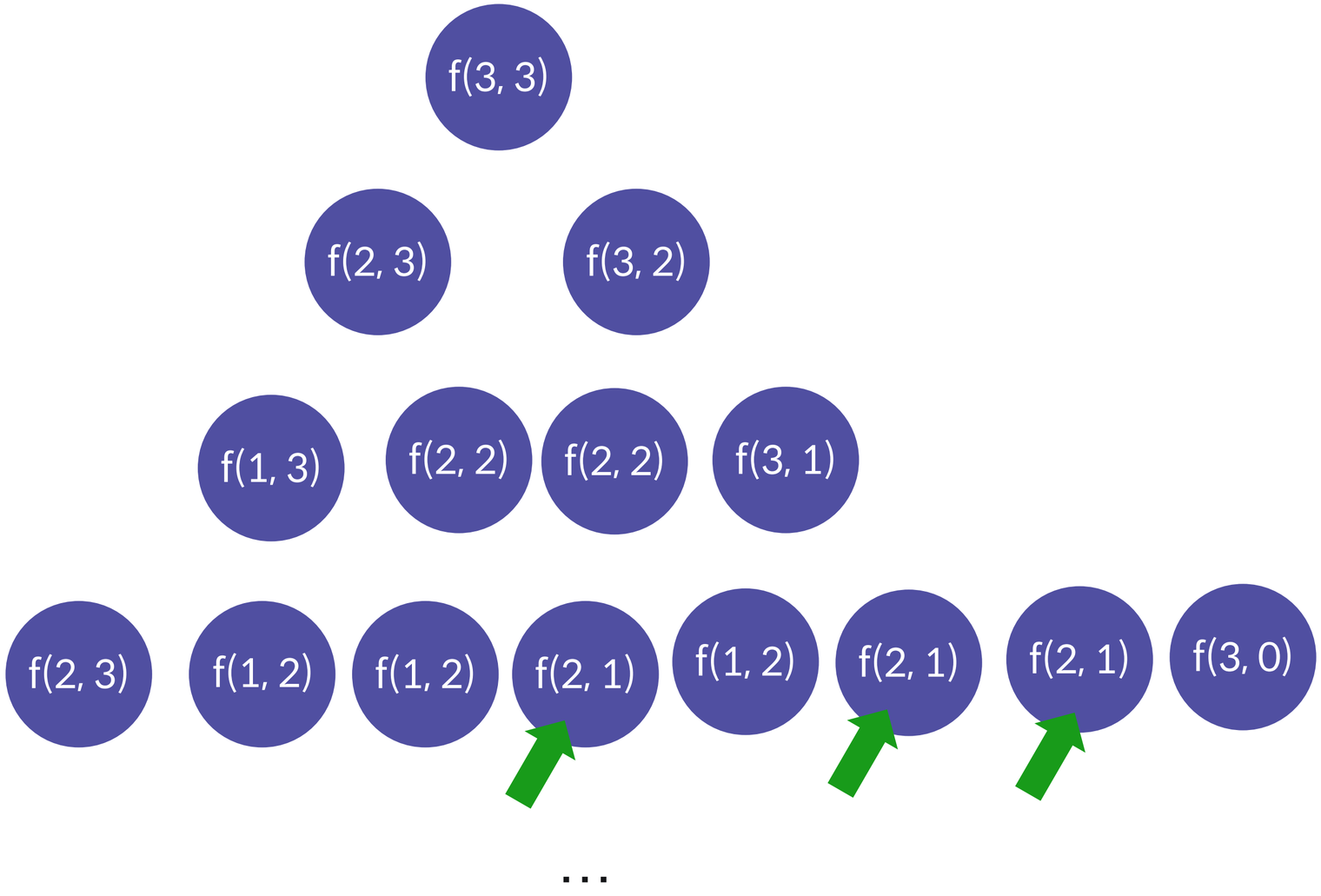
계산 수

- F(2, 2) = 5번
- F(10, 10) = 184755번
- F(20, 20) = ????

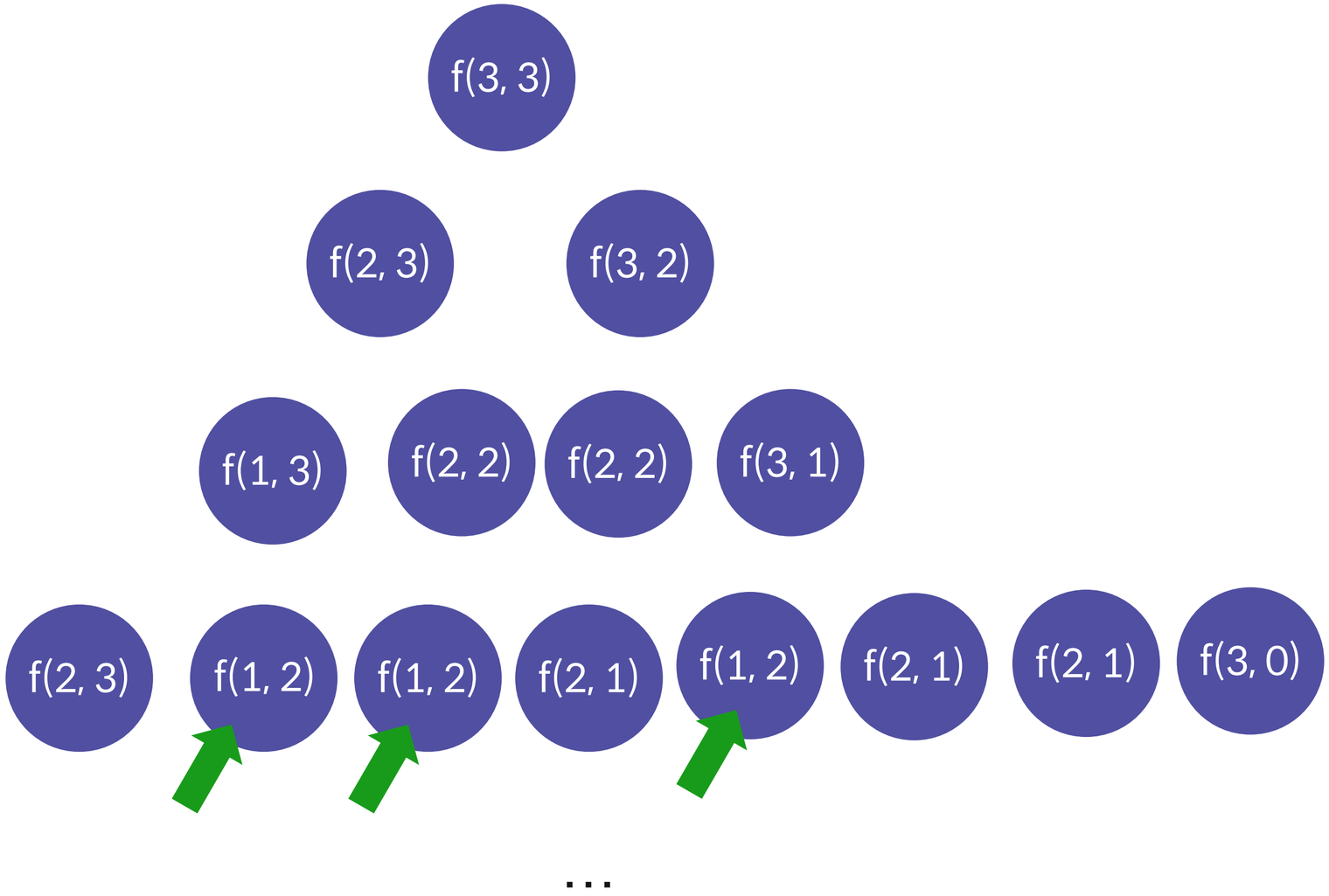
계산 수

- Code를 보면서 연산 시간을 체크해보자!

중복되는 연산



중복되는 연산



중복되는 연산

- 중복되는 연산을 “메모하면서” 계산하면 더 효율적이지 않을까?

중복되는 연산

- 코드를 통해 알아보자!

DP를 이용한 피보나치 수열

- 피보나치 수열은 a(n+1) = a(n) + a(n-1) 이라는 점화식으로 구성 가능
- 따라서 “그 전에 계산해뒀던 결과를 재활용” 할 수 있다.

DP를 이용한 피보나치 수열

- 피보나치 수열은 a(n+1) = a(n) + a(n-1) 이라는 점화식으로 구성 가능
- 따라서 “그 전에 계산해뒀던 결과를 재활용” 할 수 있다.

DP를 이용한 피보나치 수열

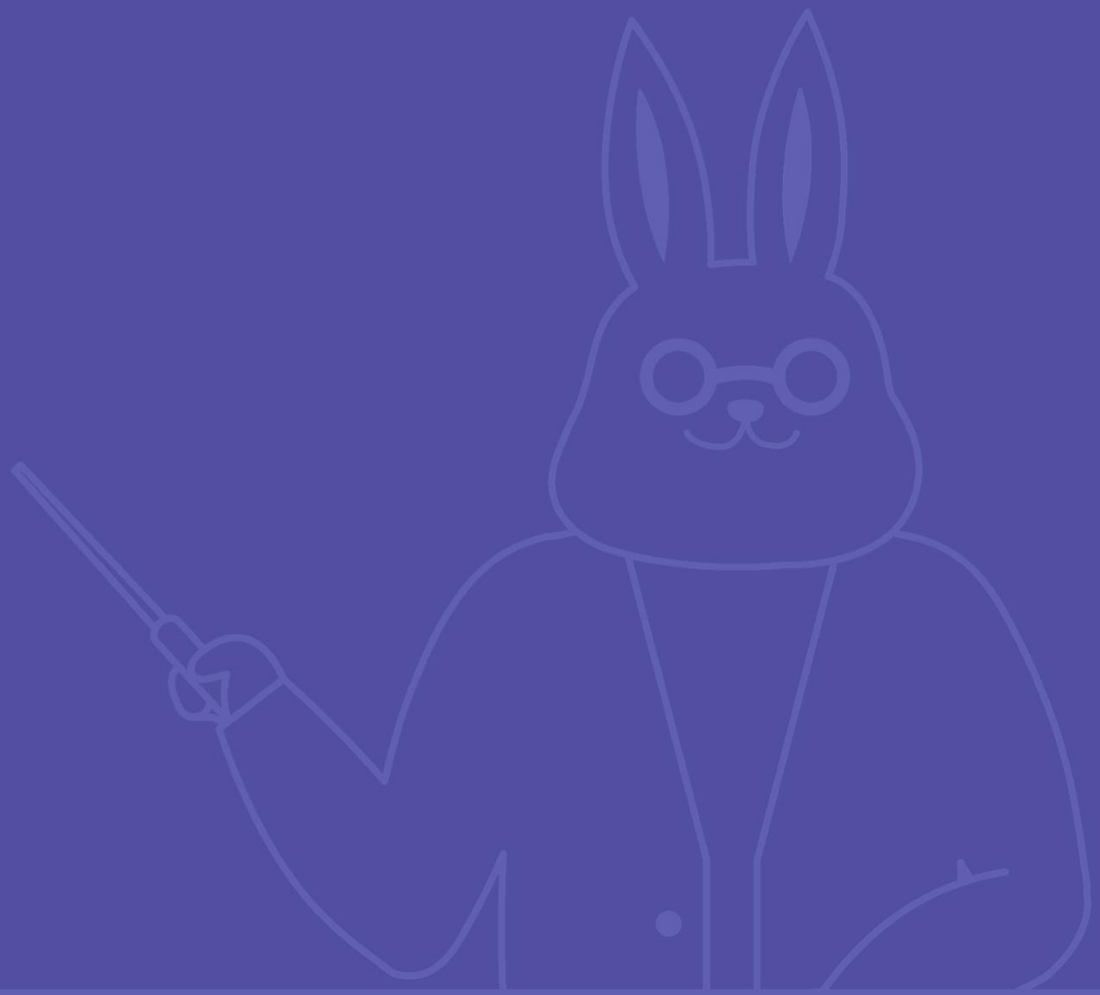
```
arr = [1, 1, ]

for i in range(2, 100):
    arr.append(arr[i-1] + arr[i-2])

print(arr)
```

DP를 이용한 피보나치 수열

- 재귀를 이용한 피보나치 수열과는 얼마나 차이날까?
- 방금 전 재귀를 이용하여 구현한 피보나치 수열을 상상해보자…

06

탐욕 알고리즘



탐욕 알고리즘이란?

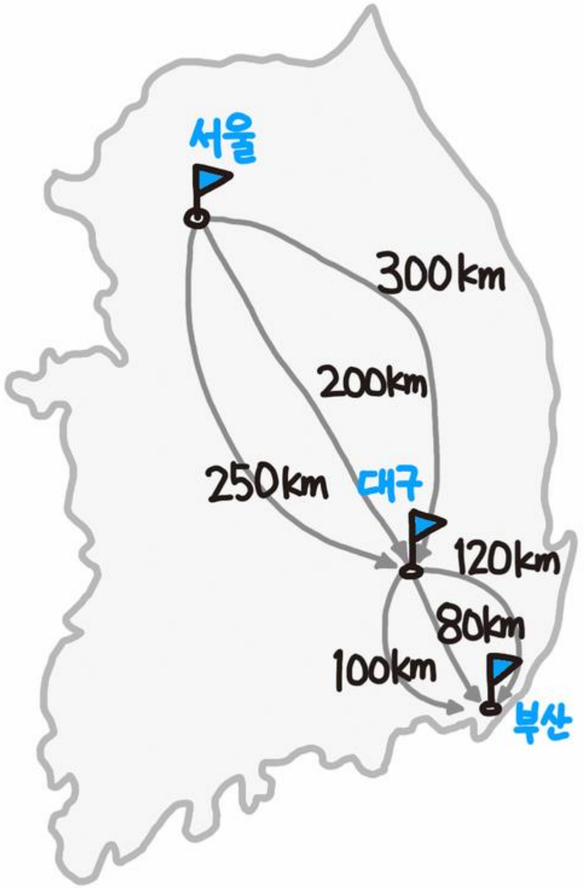
- 지금 이 순간 최적인 답을 찾아가는 알고리즘
- Ad-Hoc이라고도 표현함

탐욕 알고리즘이란?

- 지금 이 순간 최적인 답을 찾아가는 알고리즘
- Ad-Hoc이라고도 표현함
- “최적의 해” 를 보장하지는 않는다!

최적 부분 구조

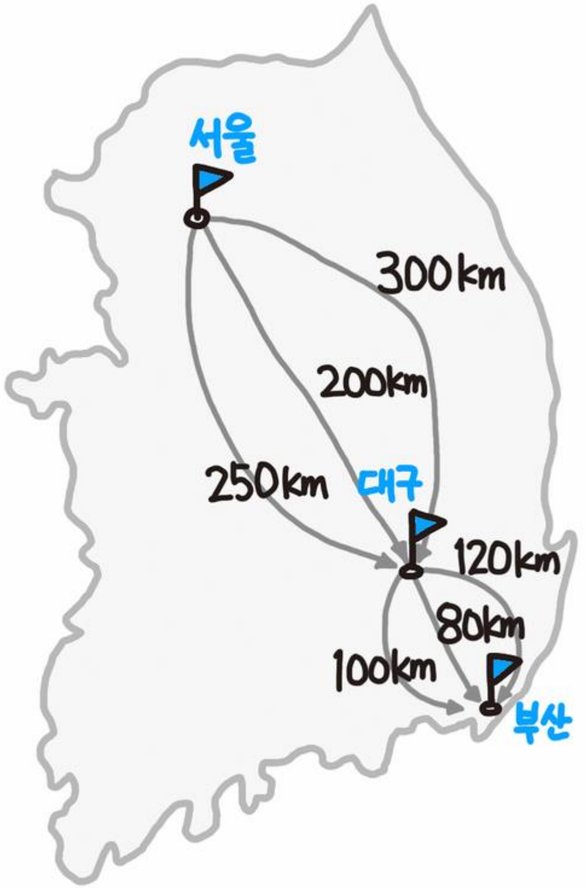
- 서울에서 부산을 가는 최적의 경우?



출저 : 파이썬 알고리즘 인터뷰

최적 부분 구조

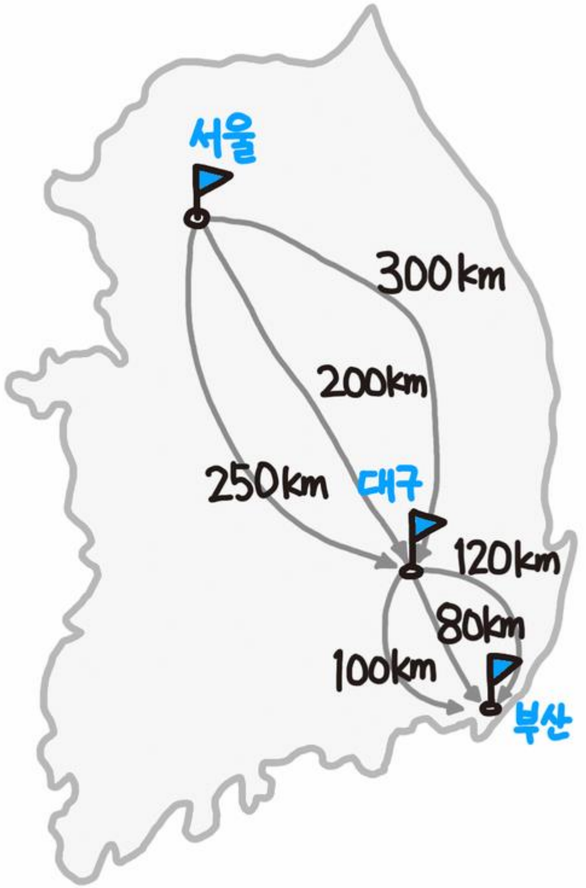
- 서울에서 부산을 가는 최적의 경우?
- 서울 -> 대구로 가는 200km



출저 : 파이썬 알고리즘 인터뷰

최적 부분 구조

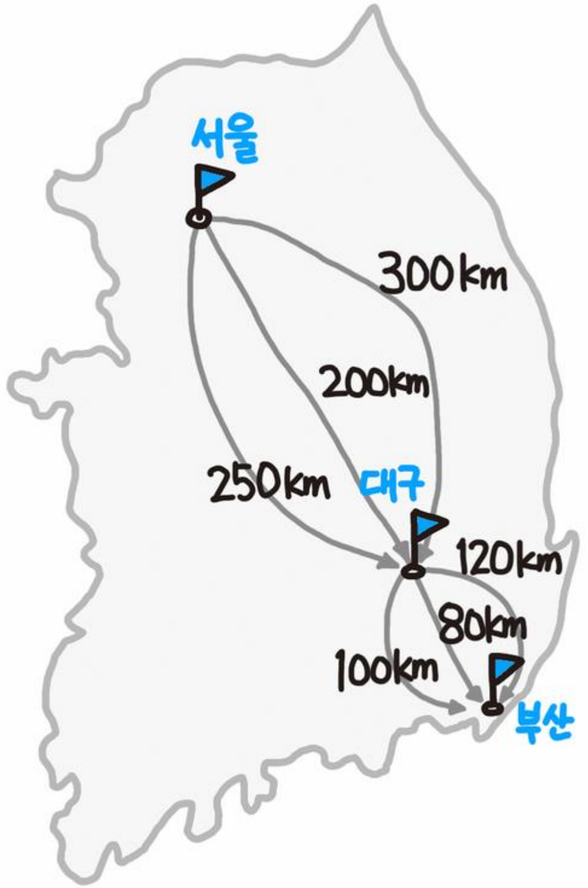
- 서울에서 부산을 가는 최적의 경우?
- 서울 -> 대구로 가는 200km
- 대구 -> 부산으로 가는 80km



출저 : 파이썬 알고리즘 인터뷰

최적 부분 구조

- 서울에서 부산을 가는 최적의 경우?
- 서울 -> 대구로 가는 200km
- 대구 -> 부산으로 가는 80km
- 문제를 부분 문제로 나누어, 부분 문제에 대한 최적 방법을 찾기



출저 : 파이썬 알고리즘 인터뷰

근사 알고리즘

- 적당히 괜찮은 방법을 찾을 때 사용하는 방법

근사 알고리즘

- 적당히 괜찮은 방법을 찾을 때 사용하는 방법
- 최적의 해를 보장하지는 않는다.

근사 알고리즘

- 적당히 괜찮은 방법을 찾을 때 사용하는 방법
- 최적의 해를 보장하지는 않는다.
- 어떤 경우에 쓰면 좋을까?

근사 알고리즘

- 손님이 언제 찾아올지 모르고, 어떤 메뉴를 주문할지 모르며,
  얼마나 식당에 머무를 지 모른다.

- 요리사는 10명이 있고, 각자 한 명의 손님만을 상대할 수 있다.
  또한 요리사의 숙련도가 달라 요리하는 시간에 차이가 있다.

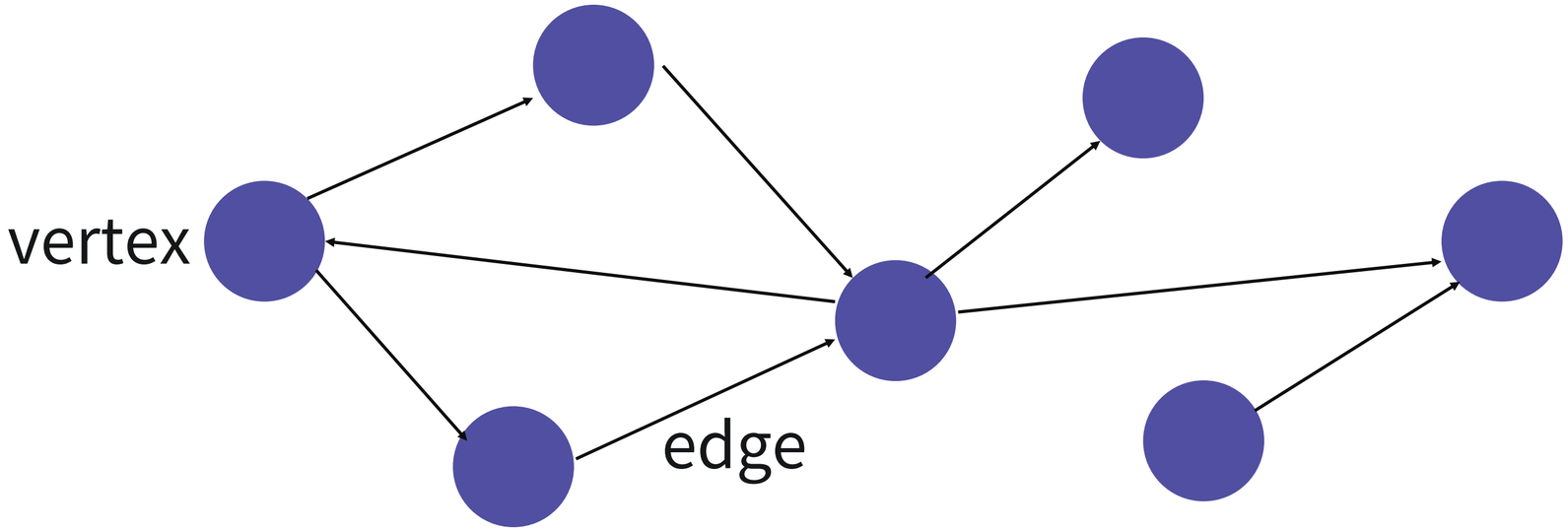
- 이 때, 요리사들을 어떤 손님에 매칭해야 최적의 수익을 얻을 수 있을까?

다익스트라 알고리즘

- 그래프의 한 정점에서 모든 정점까지의 최단거리를 구하는 알고리즘

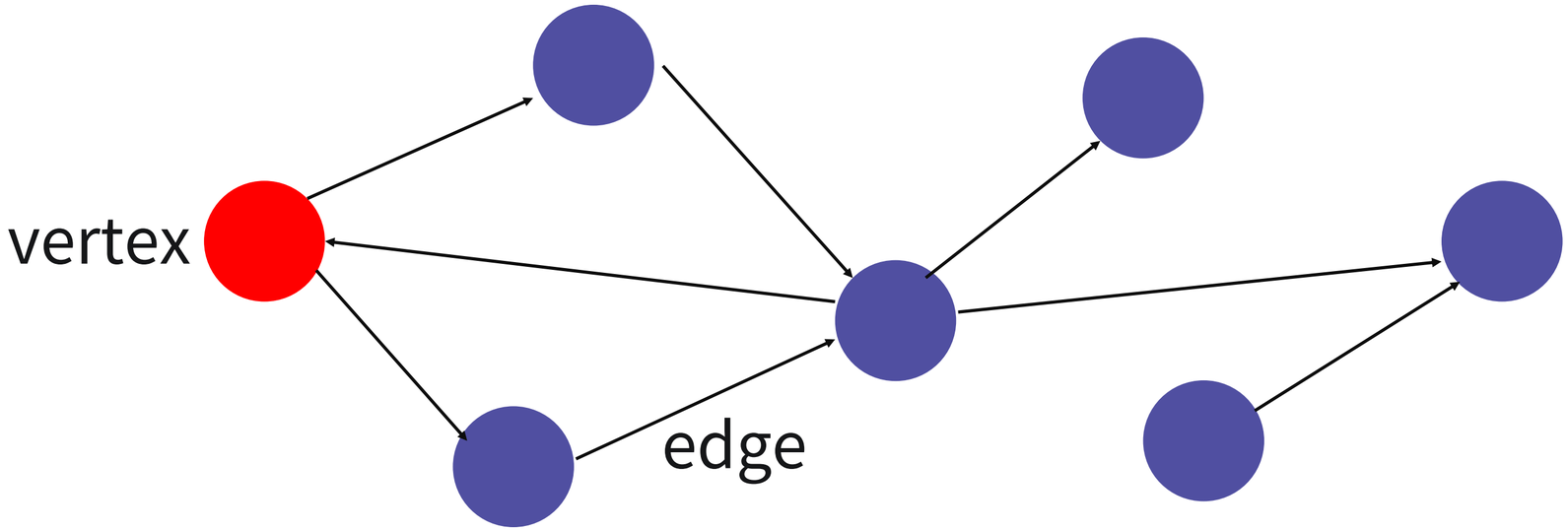
그래프?

- 정점(Vertex)와 변(Edge)로 구성된 자료구조의 한 종류



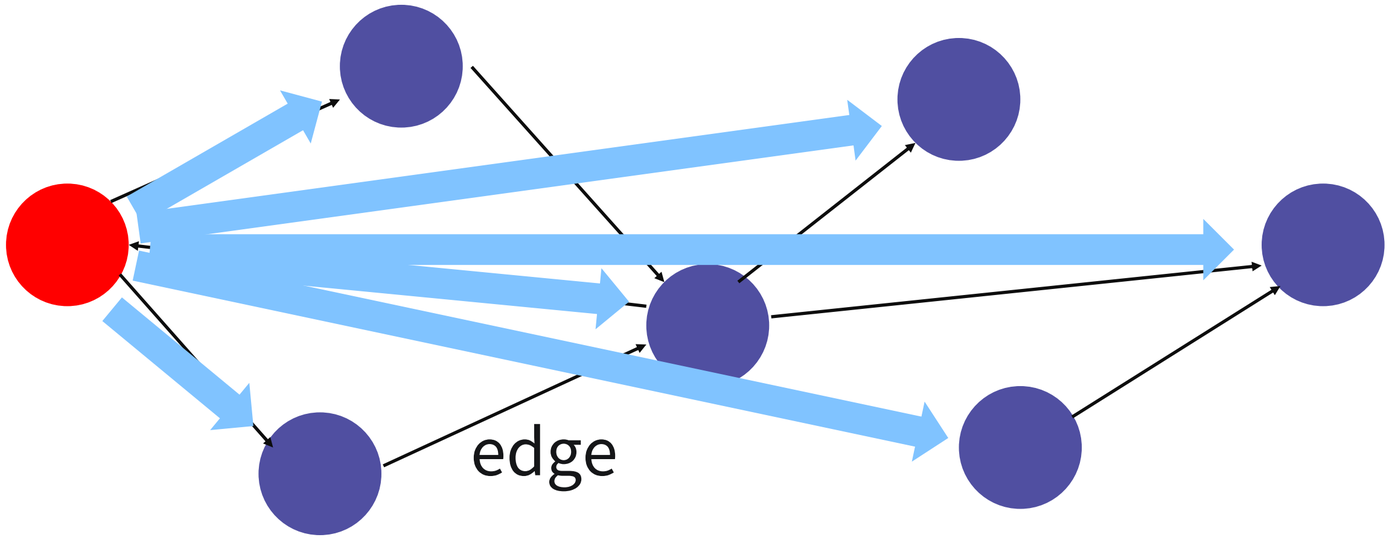
다익스트라 알고리즘이란?

- 한 정점에서 다른 모든 정점까지의 최단거리를 구하는 알고리즘

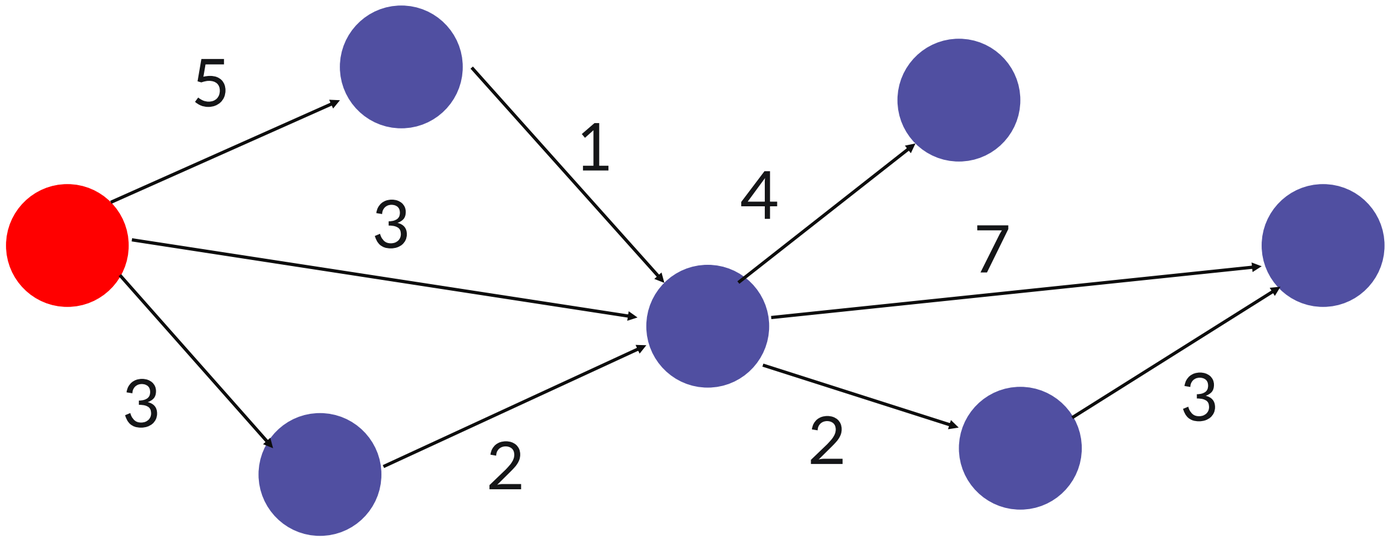


다익스트라 알고리즘이란?

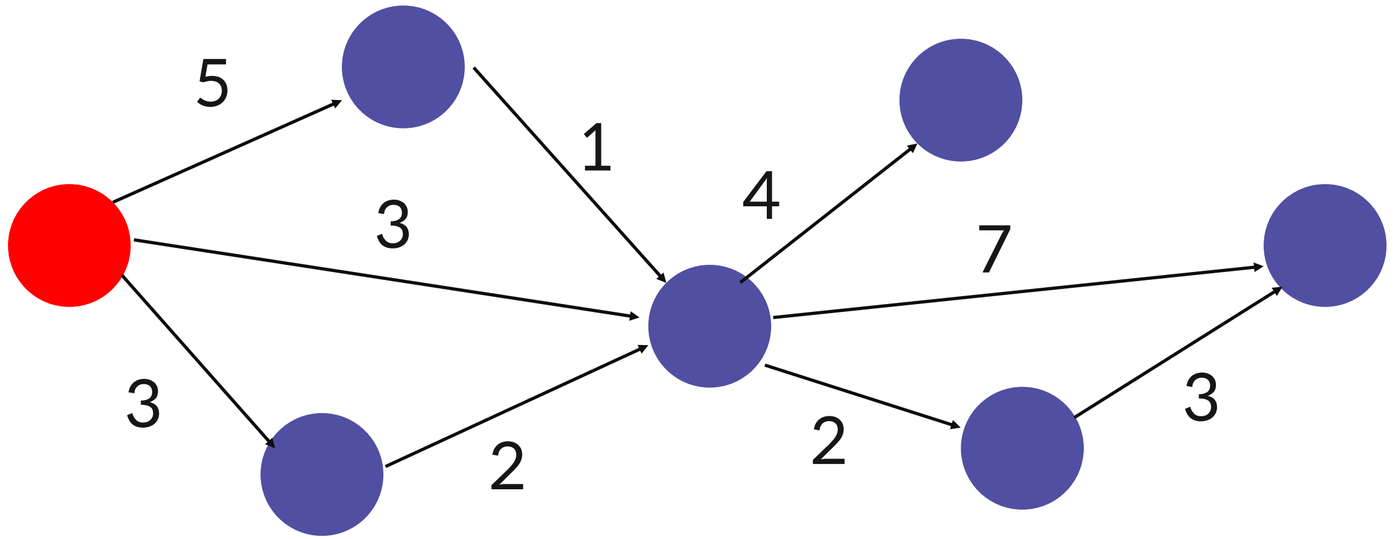
- 한 정점에서 다른 모든 정점까지의 최단거리를 구하는 알고리즘



다익스트라 알고리즘이란?



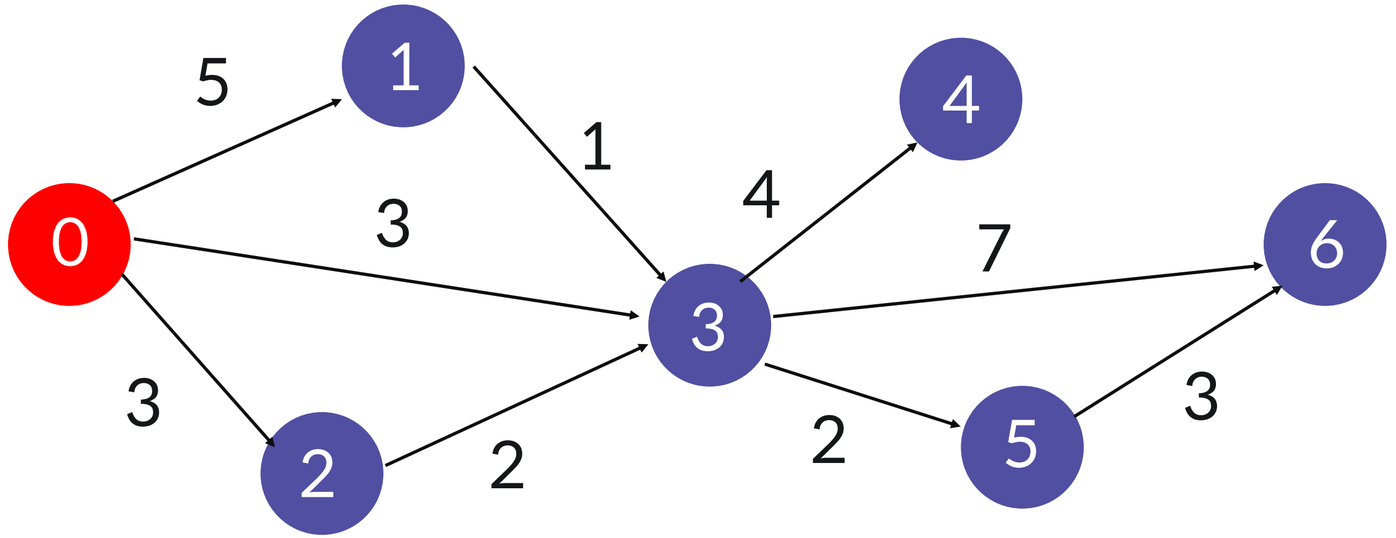
다익스트라 알고리즘이란?



	0	1	2	3	4	5	6	7	8
방문여부									
최단거리									

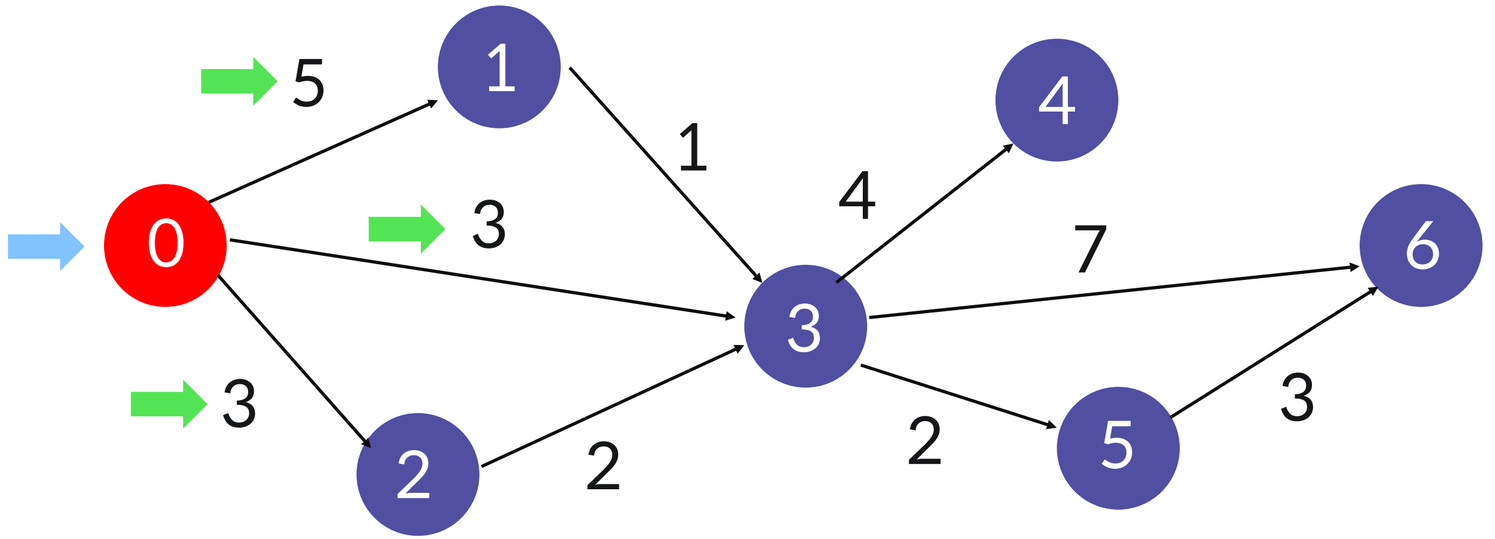
다익스트라 알고리즘이란?

d[A] + p[A][B]와 d[B]를 비교



	0	1	2	3	4	5	6	7	8
방문여부	True								
최단거리	INF	INF	INF	INF	INF	INF	INF	INF	INF

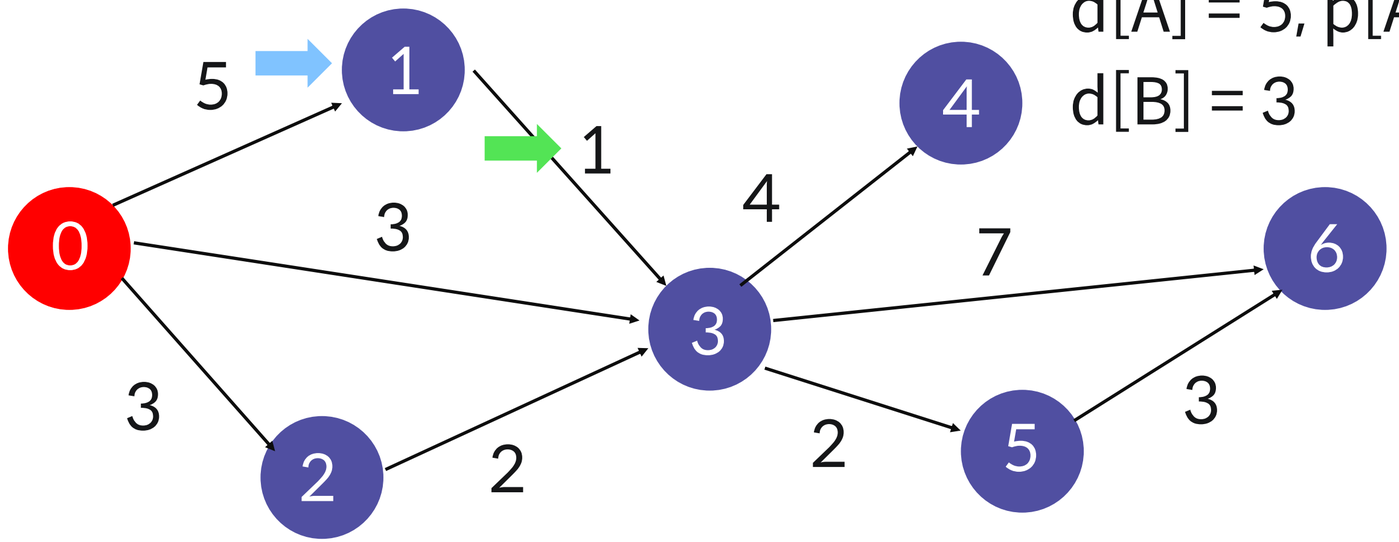
다익스트라 알고리즘이란?



	0	1	2	3	4	5	6	7	8
방문여부	True								
최단거리	INF	5	3	3	INF	INF	INF	INF	INF

다익스트라 알고리즘이란?

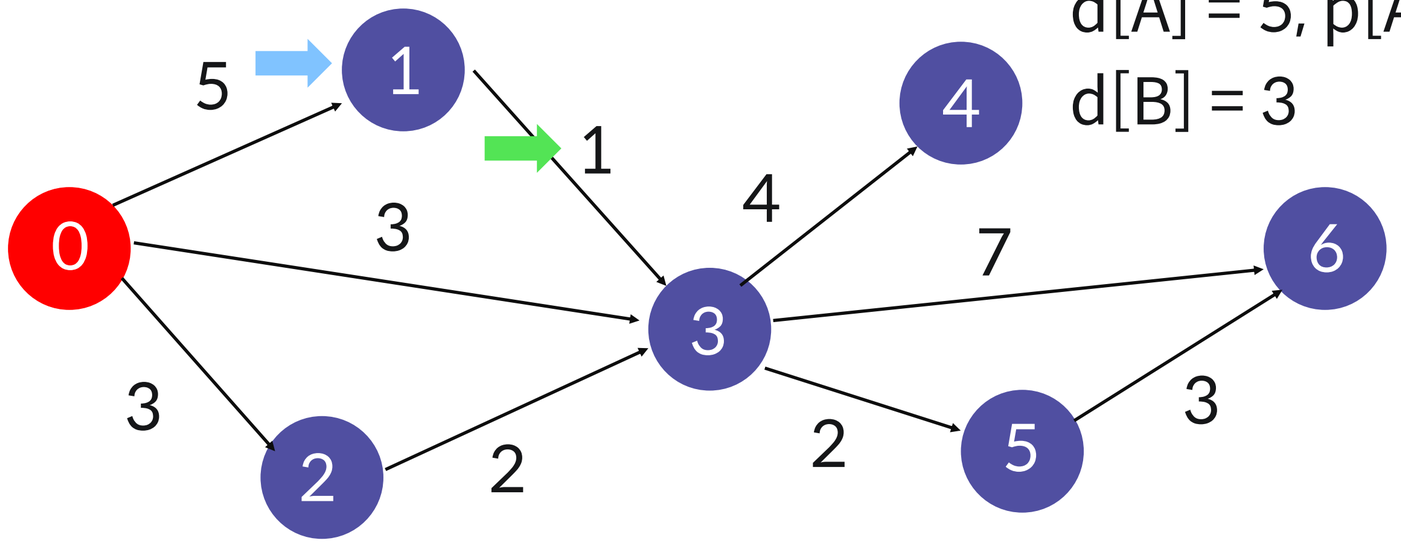
d[A] + p[A][B]와 d[B]를 비교
d[A] = 5, p[A][B] = 1
d[B] = 3



	0	1	2	3	4	5	6	7	8
방문여부	True								
최단거리	INF	5	3	3	INF	INF	INF	INF	INF

다익스트라 알고리즘이란?

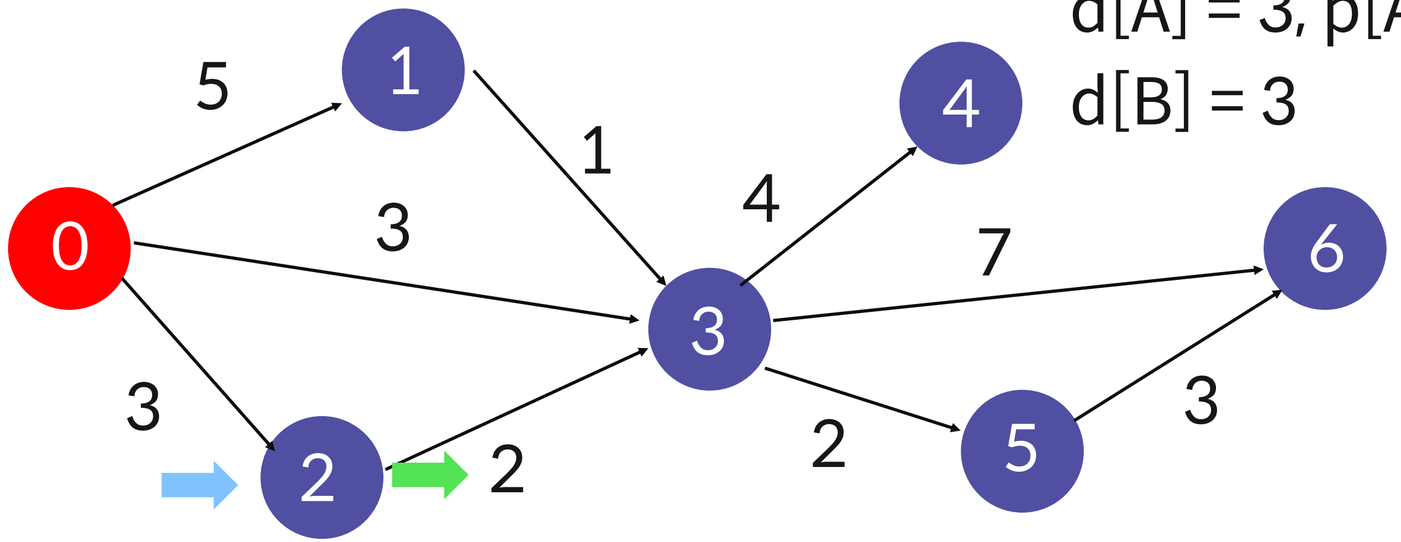
d[A] + p[A][B]와 d[B]를 비교
d[A] = 5, p[A][B] = 1
d[B] = 3



	0	1	2	3	4	5	6	7	8
방문여부	True	True							
최단거리	INF	5	3	3	INF	INF	INF	INF	INF

다익스트라 알고리즘이란?

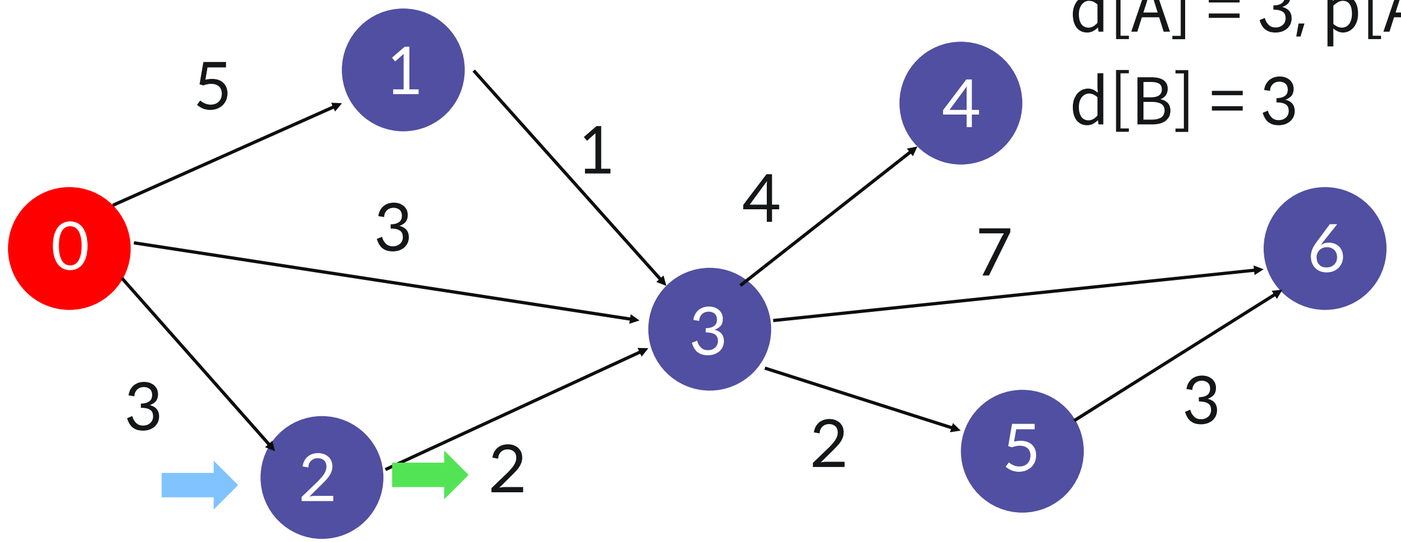
d[A] + p[A][B]와 d[B]를 비교
d[A] = 3, p[A][B] = 2
d[B] = 3



	0	1	2	3	4	5	6	7	8
방문여부	True	True							
최단거리	INF	5	3	3	INF	INF	INF	INF	INF

다익스트라 알고리즘이란?

d[A] + p[A][B]와 d[B]를 비교
d[A] = 3, p[A][B] = 2
d[B] = 3



	0	1	2	3	4	5	6	7	8
방문여부	True	True							
최단거리	INF	5	3	3	INF	INF	INF	INF	INF

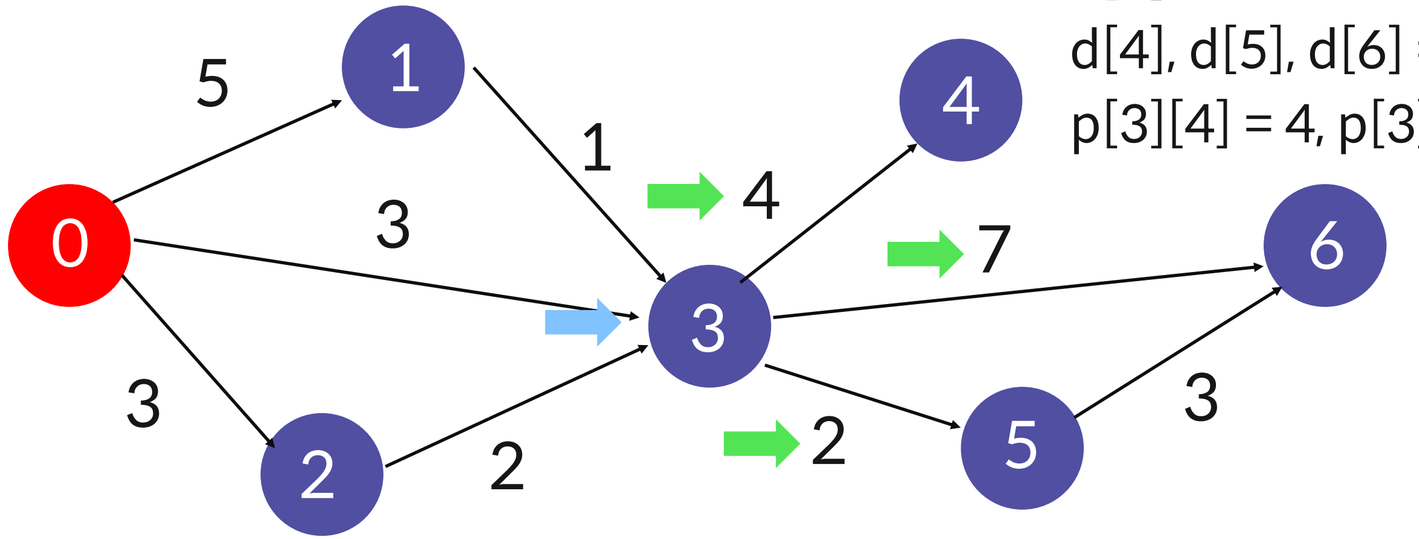
다익스트라 알고리즘이란?

d[A] + p[A][B]와 d[B]를 비교

d[3] = 3

d[4], d[5], d[6] = INF

p[3][4] = 4, p[3][6] = 7, p[3][5] =2



	0	1	2	3	4	5	6	7	8
방문여부	True	True	True						
최단거리	INF	5	3	3	INF	INF	INF	INF	INF

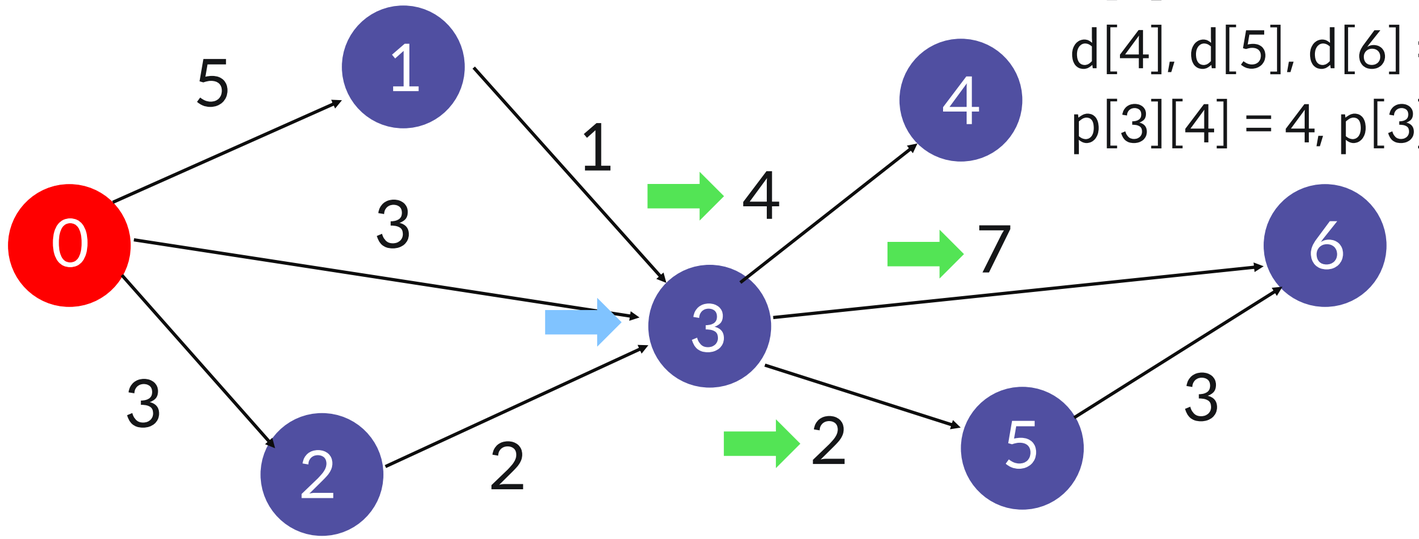
다익스트라 알고리즘이란?

d[A] + p[A][B]와 d[B]를 비교

d[3] = 3

d[4], d[5], d[6] = INF

p[3][4] = 4, p[3][6] = 7, p[3][5] =2



	0	1	2	3	4	5	6	7	8
방문여부	True	True	True						
최단거리	INF	5	3	3	7	5	10	INF	INF

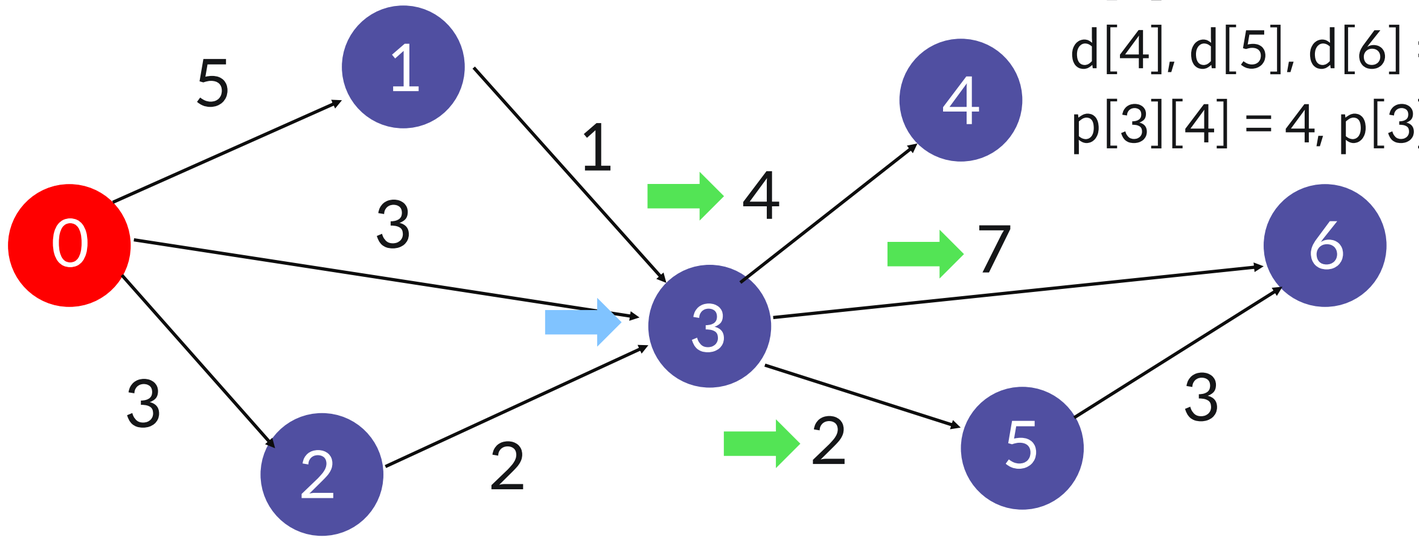
다익스트라 알고리즘이란?

d[A] + p[A][B]와 d[B]를 비교

d[3] = 3

d[4], d[5], d[6] = INF

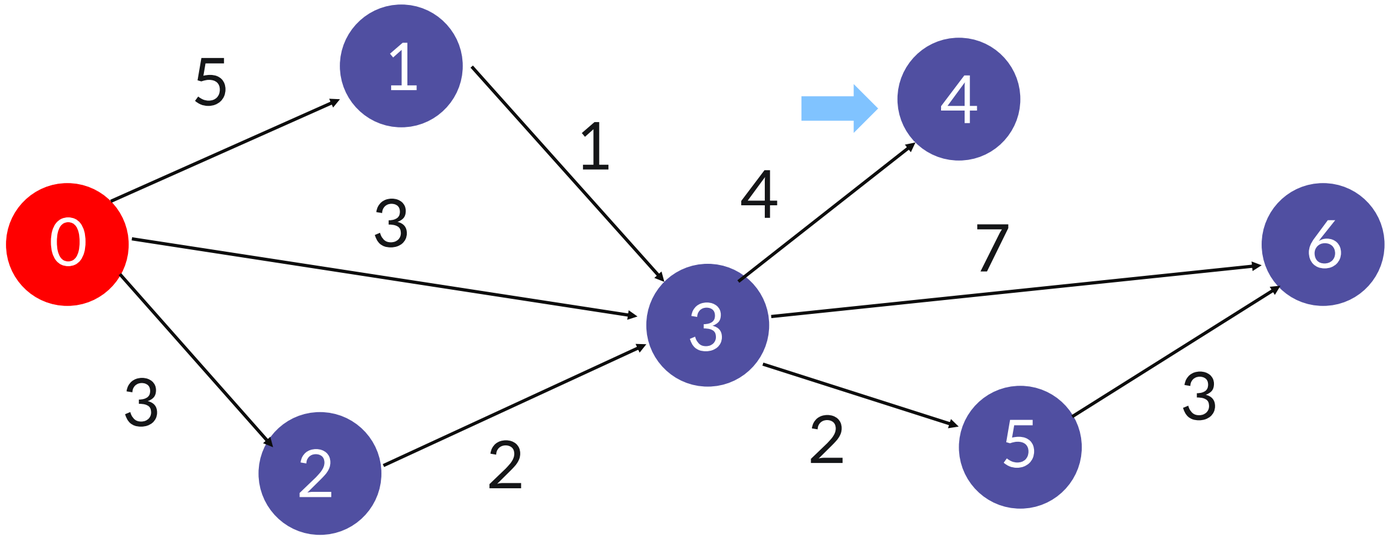
p[3][4] = 4, p[3][6] = 7, p[3][5] =2



	0	1	2	3	4	5	6	7	8
방문여부	True	True	True	True					
최단거리	INF	5	3	3	7	5	10	INF	INF

다익스트라 알고리즘이란?

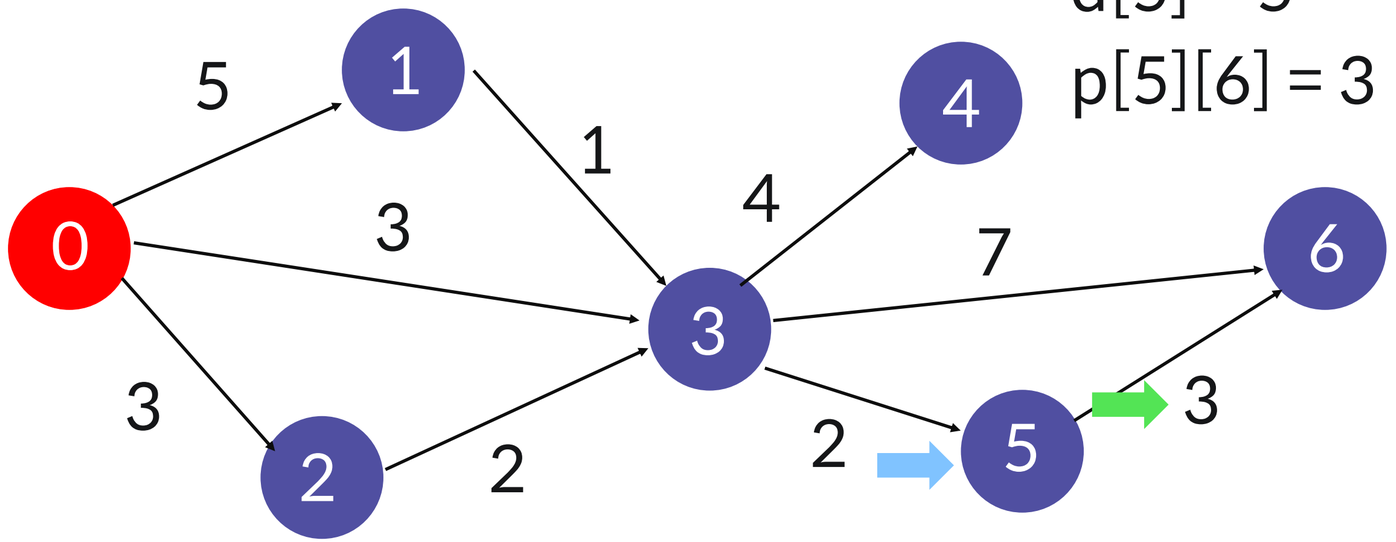
갈 수 있는 곳이 없다!



	0	1	2	3	4	5	6	7	8
방문여부	True	True	True	True	True				
최단거리	INF	5	3	3	7	5	10	INF	INF

다익스트라 알고리즘이란?

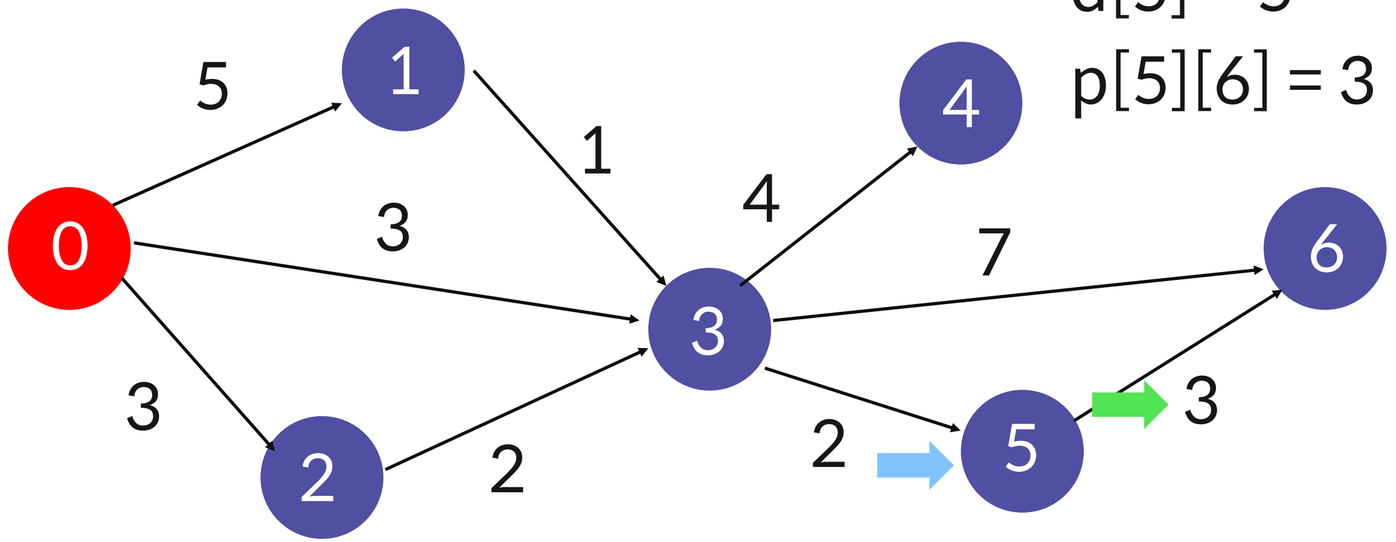
d[6] = 10
d[5] = 5
p[5][6] = 3



	0	1	2	3	4	5	6	7	8
방문여부	True	True	True	True	True				
최단거리	INF	5	3	3	7	5	10	INF	INF

다익스트라 알고리즘이란?

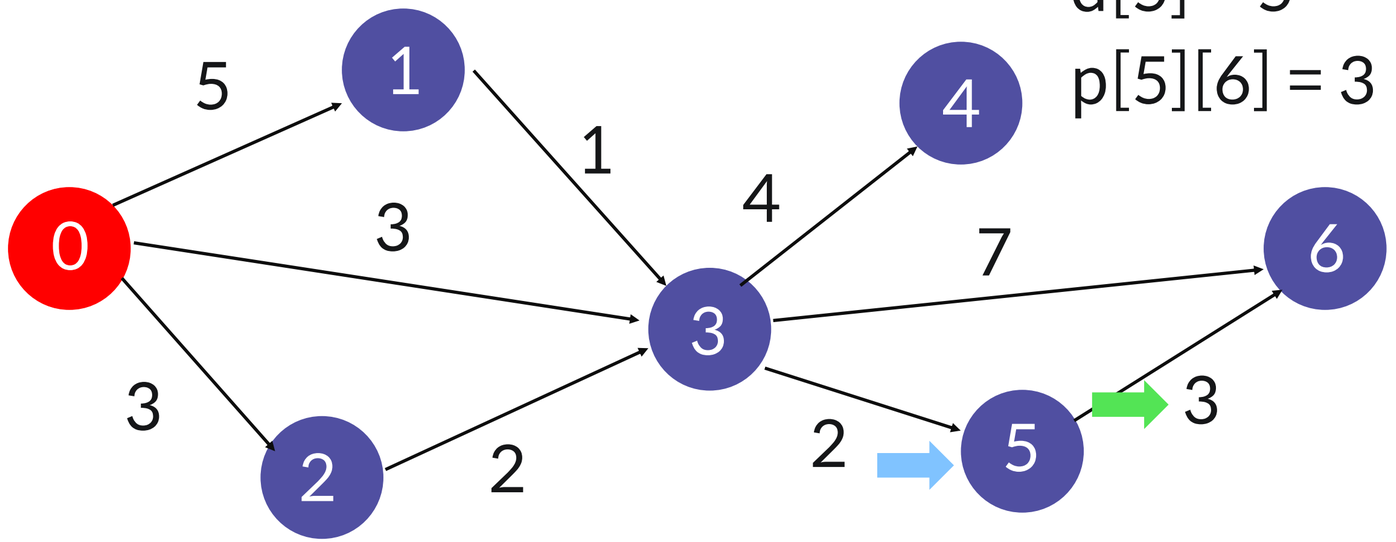
d[6] = 10
d[5] = 5
p[5][6] = 3



	0	1	2	3	4	5	6	7	8
방문여부	True	True	True	True	True				
최단거리	INF	5	3	3	7	5	8	INF	INF

다익스트라 알고리즘이란?

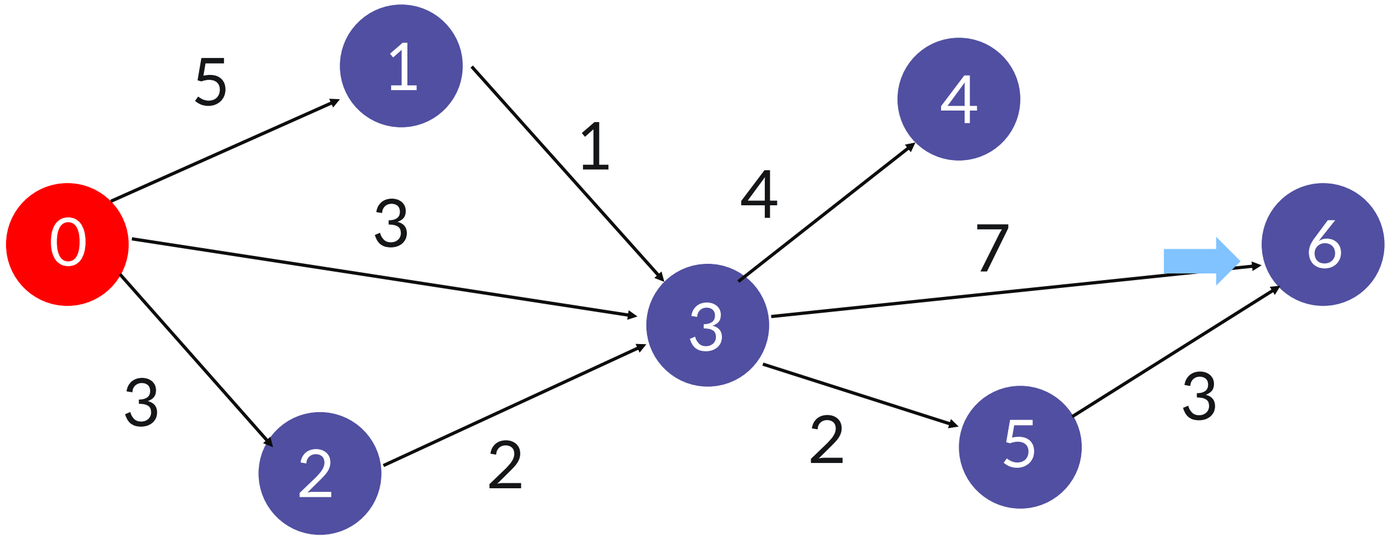
d[6] = 10
d[5] = 5
p[5][6] = 3



	0	1	2	3	4	5	6	7	8
방문여부	True	True	True	True	True	True			
최단거리	INF	5	3	3	7	5	8	INF	INF

다익스트라 알고리즘이란?

갈 수 있는 곳 없음!

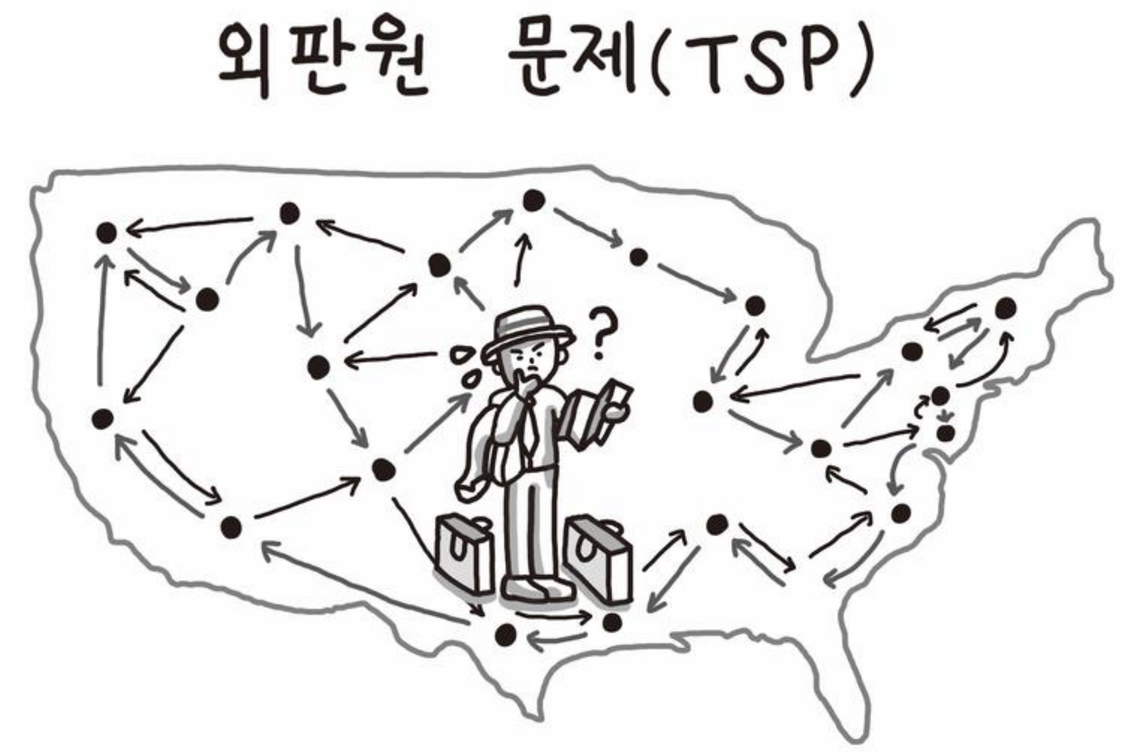


	0	1	2	3	4	5	6	7	8
방문여부	True	True	True	True	True	True	True		
최단거리	INF	5	3	3	7	5	8	INF	INF

최적해를 구하지 못하는 경우

- 배낭 문제
- 외판원 문제

간단하게 알아보는 외판원 문제



외판원이 n개의 도시를 모두 방문하고 원래 도시로 돌아올 때,
어떤 순서로 방문해야 가장 짧은 거리가 될까?

P문제와 NP문제

- P 문제 : 쉬운 문제, 즉 **시간복잡도가 다항식(Polynomial)으로 표현가능**한 문제
- NP 문제 : 어려운 문제, 즉 **시간복잡도가 다항식으로 표현되지 않는** 문제

P-NP 문제

- NP-완전 문제
- NP-난해 문제

NP 완전 문제

- 어떤 문제의 답이 YES일 때, 그 답에 대한 힌트가 주어지면 검산할 수 있는 문제

NP 완전 문제

- 어떤 집합 Z의 부분집합들 중, 원소들의 합이 0이 되는 집합이 존재하는가?
- 이 문제에 대한 일반적인 해답은 없다.
- 그러나 어떤 해답이 주어졌을때, 그 답이 정답인지 아닌지는 다항시간에 판단할 수 있다.

NP 완전 문제

- {-5, 6, 1, 2, -10, -7, 13}
- 이 집합의 부분집합 중 합이 0이되는 부분집합을 구하는 “쉬운” 방법은 없지만
- {6, 1, -7} 이라는 어떤 부분집합의 합이 0이라는 것은 쉽게 알 수 있다.

P-NP 문제