

06/30 실습 수업 1분반

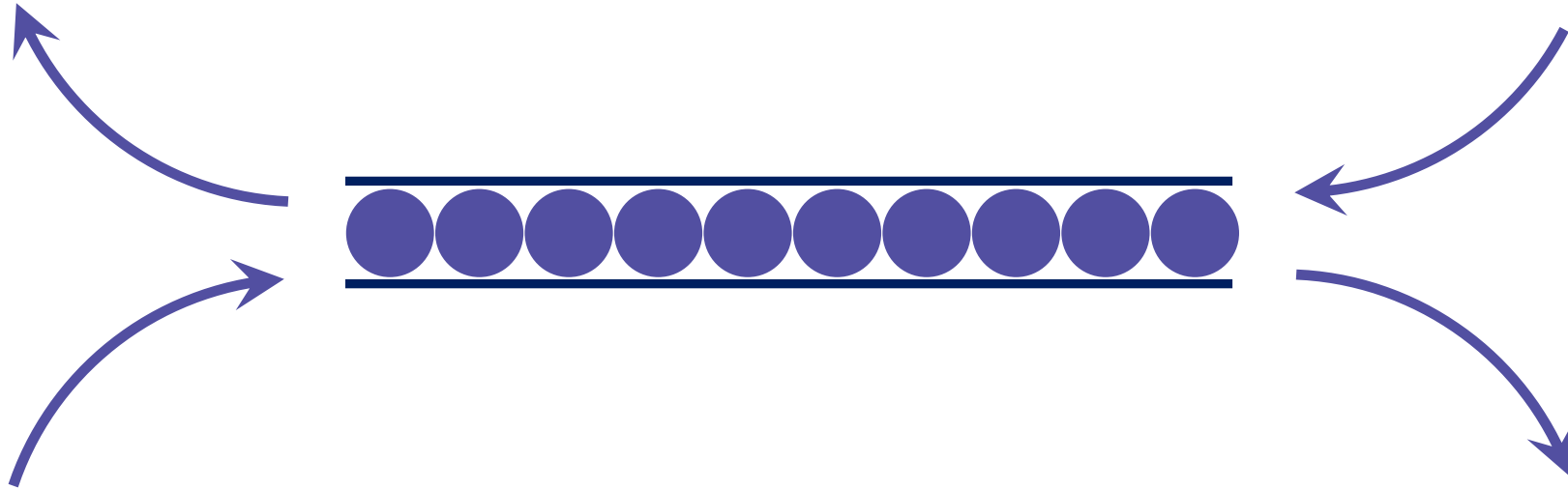
김병철 코치

오늘의 자료를 볼 땐...



두개의 마크에 주목하세요!

도움이 될 만한 자료구조 하나...



Deque

- Queue/Stack의 구조를 합친 구조로, 양쪽에서 삽입/삭제를 모두 할 수 있음.
- 삽입/삭제 연산에 있어 시간복잡도가 $O(1)$ 임.

파이썬에서의 큐와 덱

```
import time
import queue
```

```
time0 = time.time()
q = queue.Queue()
```

```
for i in range(100000):
    q.put(i)
```

```
for i in range(100000):
    q.get()
```

```
time1 = time.time()
```

```
print(time1 - time0)
```

```
import time
from collections import deque
```

```
time0 = time.time()
dq = deque()
```

```
for i in range(100000):
    dq.append(i)
```

```
for i in range(100000):
    dq.popleft()
```

```
time1 = time.time()
```

```
print(time1 - time0)
```

- Deque는 iterable한 데이터를 기반으로 선언 가능하며, 왼쪽에서 진행하는 연산은 left를 붙임.
- Queue는 멀티 스레드 환경에 최적화 되었으며, Deque의 경우 속도 측면에서 더 빠름. (약 100배 이상!)
- 코딩테스트 환경에서는 Deque를 쓰는 것을 권장.

파이썬에서의 덱

```
from collections import deque
```

- Deque를 사용하기 위해서 import 함.

```
dq = deque()  
dq = deque('12345')
```

- Deque의 선언: iterable한 데이터를 받아 생성.

```
dq.append(6) # 1 2 3 4 5 6  
dq.appendleft(0) # 0 1 2 3 4 5 6
```

- 대부분의 연산은 List와 유사하나,
앞에서 진행되는 연산은 left를 붙인다.

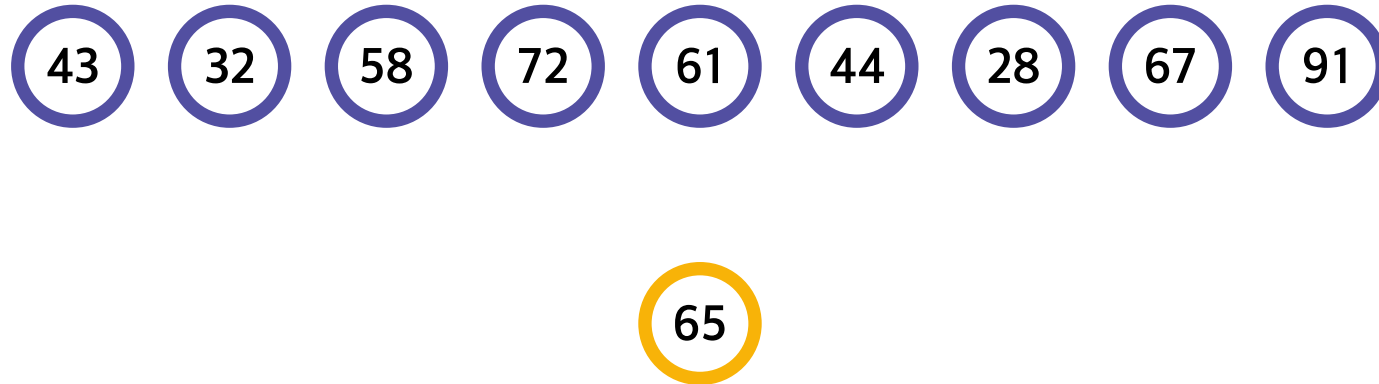
```
removed = dq.pop() # 0 1 2 3 4 5  
print(removed) # 6  
removed = dq.popleft() # 1 2 3 4 5  
print(removed) # 0
```

- pop()/popleft() 메소드는 제거한 값을 리턴함.

```
dq.rotate(1) # 5 1 2 3 4  
dq.rotate(-2) # 2 3 4 5 1
```

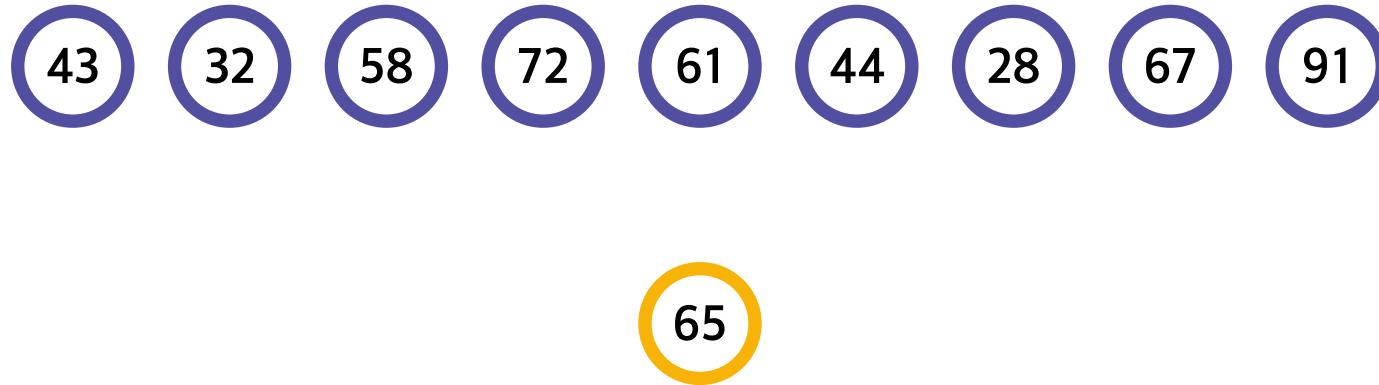
- rotate(n): dq.appendleft(dq.pop()) * n (n > 0)
dq.append(dq.popleft()) * |n| (n < 0)

Quick Sort



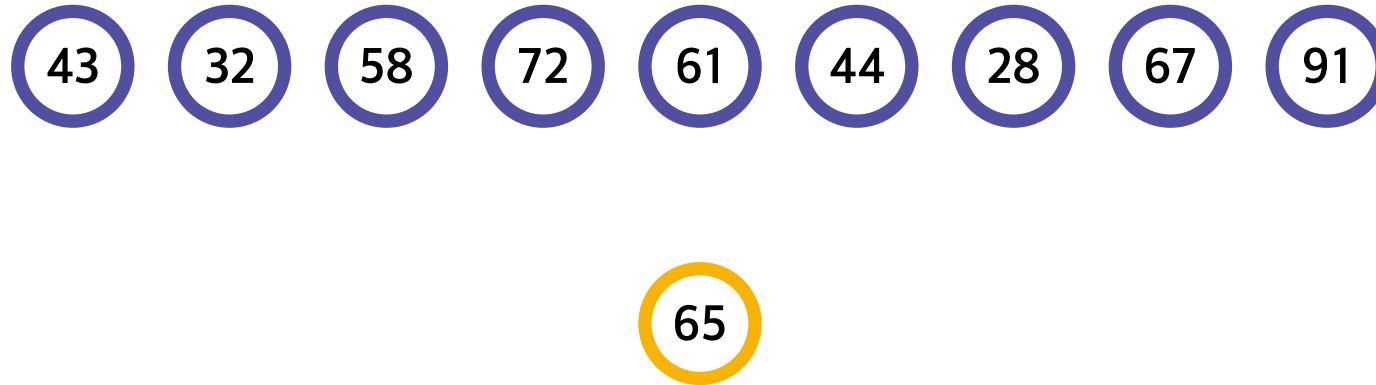
Q. 위쪽 데이터를 노란색 데이터보다 큰 것/작은 것으로 분류할 때, 시간복잡도는 어떻게 될까요?

Quick Sort



Q. 위쪽 데이터를 노란색 데이터보다 큰 것/작은 것으로 분류할 때, 시간복잡도는 어떻게 될까요?
= $O(N)$

Quick Sort



Quick Sort

- 아무거나 비교할 원소를 하나 정한다.
- 이 원소보다 작은 원소, 큰 원소, 같은 원소를 분할한다.
- 이후 작은 원소/큰 원소들에 대해 재귀적으로 퀵 소트를 진행한다.

Quick Sort

- 아무거나 비교할 원소를 하나 정한다.
- 이 원소보다 작은 원소, 큰 원소, 같은 원소를 분할한다.
- 이후 작은 원소/큰 원소들에 대해 재귀적으로 퀵 소트를 진행한다.



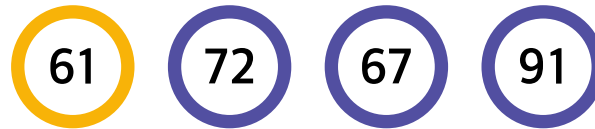
Quick Sort

- 아무거나 비교할 원소를 하나 정한다.
- 이 원소보다 작은 원소, 큰 원소, 같은 원소를 분할한다.
- 이후 작은 원소/큰 원소들에 대해 재귀적으로 퀵 소트를 진행한다.



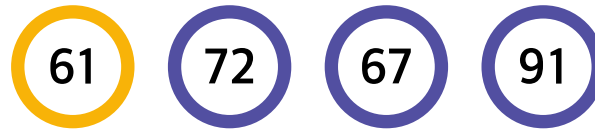
Quick Sort

- 아무거나 비교할 원소를 하나 정한다.
- 이 원소보다 작은 원소, 큰 원소, 같은 원소를 분할한다.
- 이후 작은 원소/큰 원소들에 대해 재귀적으로 퀵 소트를 진행한다.



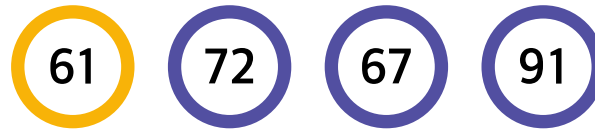
Quick Sort

- 아무거나 비교할 원소를 하나 정한다.
- 이 원소보다 작은 원소, 큰 원소, 같은 원소를 분할한다.
- 이후 작은 원소/큰 원소들에 대해 재귀적으로 퀵 소트를 진행한다.



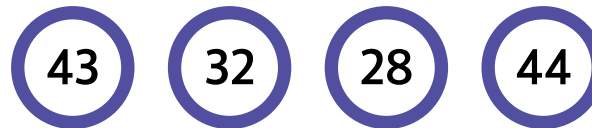
Quick Sort

- 아무거나 비교할 원소를 하나 정한다.
- 이 원소보다 작은 원소, 큰 원소, 같은 원소를 분할한다.
- 이후 작은 원소/큰 원소들에 대해 재귀적으로 퀵 소트를 진행한다.



Quick Sort

- 아무거나 비교할 원소를 하나 정한다.
- 이 원소보다 작은 원소, 큰 원소, 같은 원소를 분할한다.
- 이후 작은 원소/큰 원소들에 대해 재귀적으로 퀵 소트를 진행한다.



Quick Sort

- 아무거나 비교할 원소를 하나 정한다.
- 이 원소보다 작은 원소, 큰 원소, 같은 원소를 분할한다.
- 이후 작은 원소/큰 원소들에 대해 재귀적으로 퀵 소트를 진행한다.



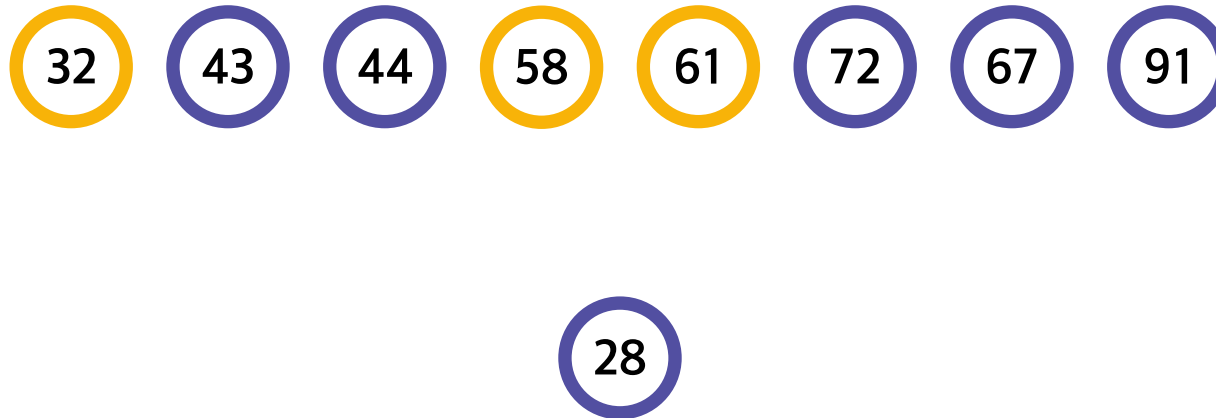
Quick Sort

- 아무거나 비교할 원소를 하나 정한다.
- 이 원소보다 작은 원소, 큰 원소, 같은 원소를 분할한다.
- 이후 작은 원소/큰 원소들에 대해 재귀적으로 퀵 소트를 진행한다.



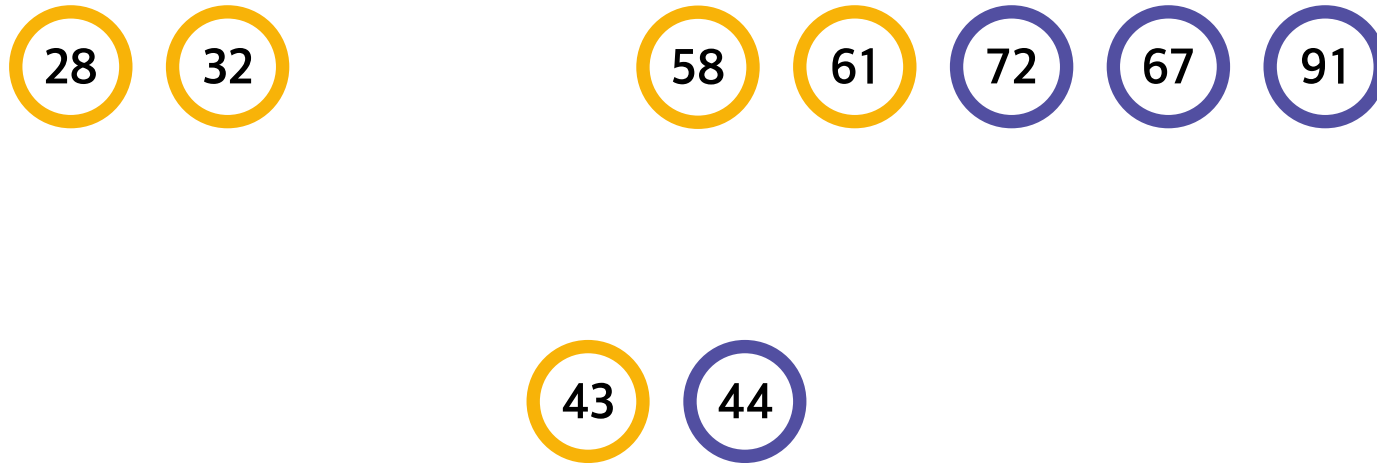
Quick Sort

- 아무거나 비교할 원소를 하나 정한다.
- 이 원소보다 작은 원소, 큰 원소, 같은 원소를 분할한다.
- 이후 작은 원소/큰 원소들에 대해 재귀적으로 퀵 소트를 진행한다.



Quick Sort

- 아무거나 비교할 원소를 하나 정한다.
- 이 원소보다 작은 원소, 큰 원소, 같은 원소를 분할한다.
- 이후 작은 원소/큰 원소들에 대해 재귀적으로 퀵 소트를 진행한다.



Quick Sort

- 아무거나 비교할 원소를 하나 정한다.
- 이 원소보다 작은 원소, 큰 원소, 같은 원소를 분할한다.
- 이후 작은 원소/큰 원소들에 대해 재귀적으로 퀵 소트를 진행한다.



Quick Sort

- 아무거나 비교할 원소를 하나 정한다.
- 이 원소보다 작은 원소, 큰 원소, 같은 원소를 분할한다.
- 이후 작은 원소/큰 원소들에 대해 재귀적으로 퀵 소트를 진행한다.



Quick Sort

- 아무거나 비교할 원소를 하나 정한다.
- 이 원소보다 작은 원소, 큰 원소, 같은 원소를 분할한다.
- 이후 작은 원소/큰 원소들에 대해 재귀적으로 퀵 소트를 진행한다.



Quick Sort

- 아무거나 비교할 원소를 하나 정한다.
- 이 원소보다 작은 원소, 큰 원소, 같은 원소를 분할한다.
- 이후 작은 원소/큰 원소들에 대해 재귀적으로 퀵 소트를 진행한다.



Quick Sort

- 아무거나 비교할 원소를 하나 정한다.
- 이 원소보다 작은 원소, 큰 원소, 같은 원소를 분할한다.
- 이후 작은 원소/큰 원소들에 대해 재귀적으로 퀵 소트를 진행한다.



Quick Sort

- 아무거나 비교할 원소를 하나 정한다.
- 이 원소보다 작은 원소, 큰 원소, 같은 원소를 분할한다.
- 이후 작은 원소/큰 원소들에 대해 재귀적으로 퀵 소트를 진행한다.



Pivot의 중요성



이래도 $O(N \log N)$ 일까??

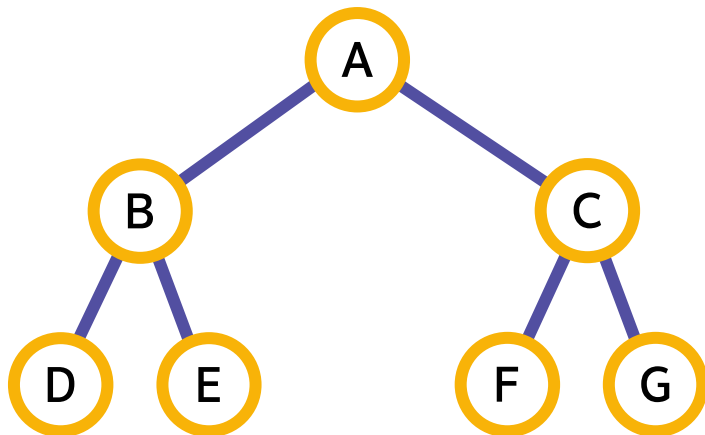
Pivot의 중요성



이래도 $O(N \log N)$ 일까??

이진 트리에서의 순회

 Silver 1 - 트리 순회 (#1991)



- 전위 순회 (Preorder Traversal)

부모 → 왼쪽 자식 → 오른쪽 자식 순으로 탐색

- 중위 순회 (Inorder Traversal)

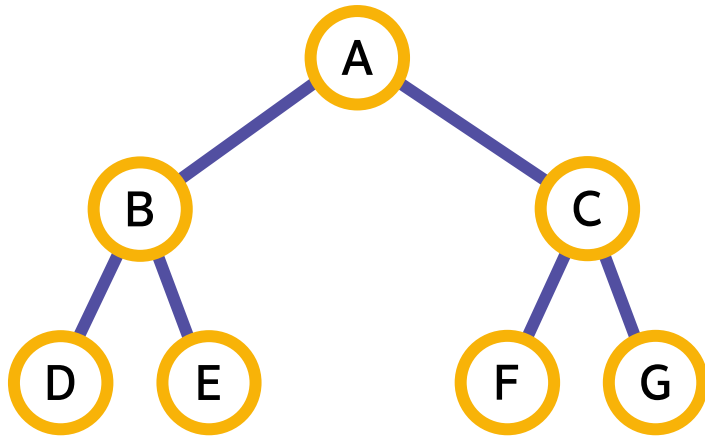
왼쪽 자식 → 부모 → 오른쪽 자식 순으로 탐색

- 후위 순회 (Postorder Traversal)

왼쪽 자식 → 오른쪽 자식 → 부모 순으로 탐색

이진 트리에서의 순회

 Silver 1 - 트리 순회 (#1991)



● 전위 순회 (Preorder Traversal)

부모 → 왼쪽 자식 → 오른쪽 자식 순으로 탐색

A - B - D - E - C - F - G

● 중위 순회 (Inorder Traversal)

왼쪽 자식 → 부모 → 오른쪽 자식 순으로 탐색

D - B - E - A - F - C - G

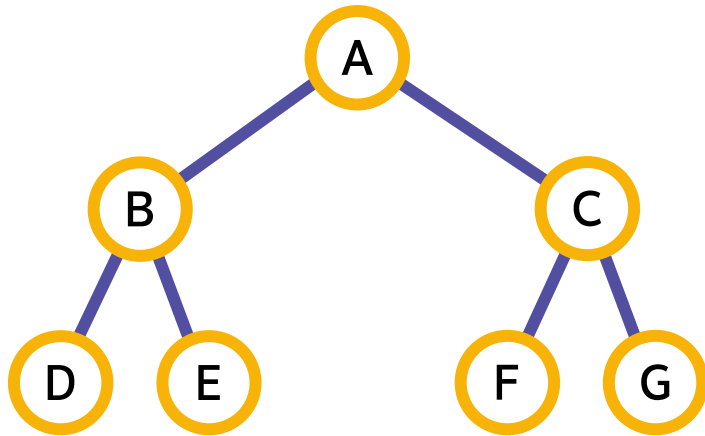
● 후위 순회 (Postorder Traversal)

왼쪽 자식 → 오른쪽 자식 → 부모 순으로 탐색

D - E - B - F - G - C - A

이진 트리에서의 순회

 Silver 1 - 트리 순회 (#1991)



● 전위 순회 (Preorder Traversal)

```
function PreOrder(node)  
  visit(node)  
  PreOrder(node.left)  
  PreOrder(node.right)
```

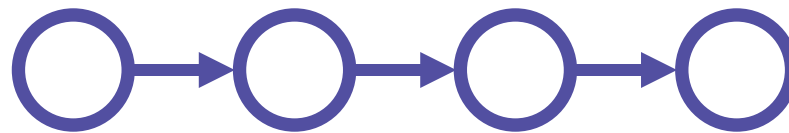
● 중위 순회 (Inorder Traversal)

```
function InOrder(node)  
  InOrder(node.left)  
  visit(node)  
  InOrder(node.right)
```

● 후위 순회 (Postorder Traversal)

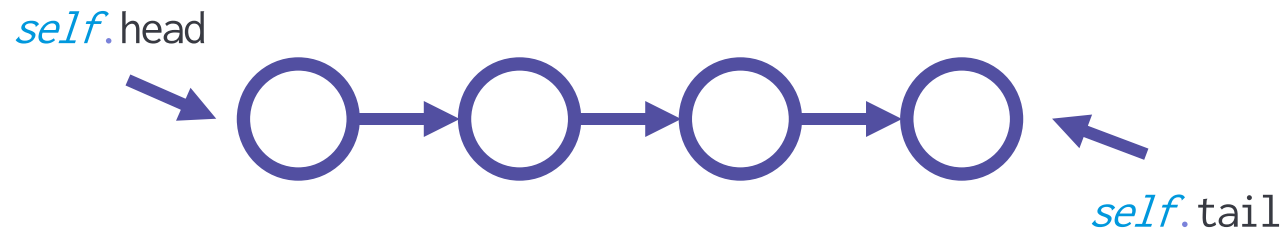
```
function PostOrder(node)  
  PostOrder(node.left)  
  PostOrder(node.right)  
  visit(node)
```

연결 리스트와 배열 변환하기



단일 연결 리스트

연결 리스트와 배열 변환하기




단일 연결 리스트

Head와 tail만 잘 구분하면 됩니다...

올바른 괄호인지 판단하기

 Silver 4 - 괄호 (#9012)

 Level 2 - 올바른 괄호

이미 푸셨지만... 한번 다시 볼까요?

이 문제를 진짜 스택으로 푸는게 의미 있으려면!

 Silver 4 - 균형잡힌 세상 (#4949)

줄 세우기

 Gold 5 - 좁은 미술전시관 (#10476)

 Gold 4 - 개근상 (#1563)

Dynamic Programming

- 특정 범위까지의 값을 구하기 위해 **이전 범위의 값을 활용**하여 효율적으로 값을 얻는 기법.
- 이전 범위의 값을 저장하여, (**Memoization**) 똑같은 문제가 발생했을 때 이를 참조하여 해결함.

줄 세우기

 Gold 5 - 좁은 미술전시관 (#10476)

 Gold 4 - 개근상 (#1563)

어렵다!

줄 세우기

 Gold 5 - 좁은 미술전시관 (#10476)

 Gold 4 - 개근상 (#1563)



맨 끝 쪽에 학생이 줄을 선다고 해보자...

줄 세우기

 Gold 5 - 좁은 미술전시관 (#10476)

 Gold 4 - 개근상 (#1563)

<여학생>



줄 세우기

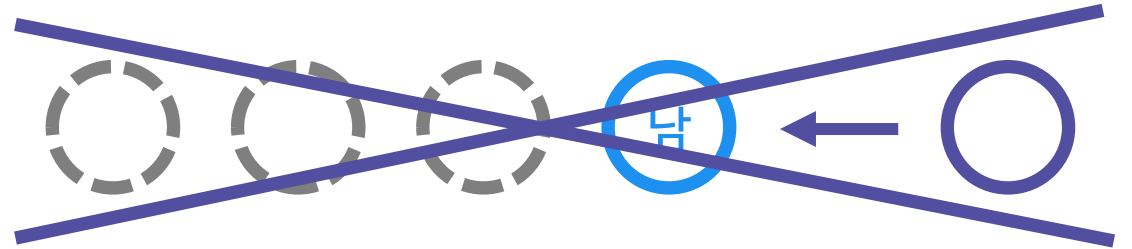
 Gold 5 - 좁은 미술전시관 (#10476)

 Gold 4 - 개근상 (#1563)

<여학생>



<남학생>



줄 세우기

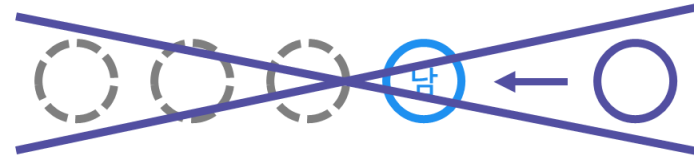
 Gold 5 - 좁은 미술전시관 (#10476)

 Gold 4 - 개근상 (#1563)

<여학생>



<남학생>



$$\begin{aligned}\text{answer}(\text{women}, x) &= \text{answer}(\text{women}, x - 1) + \text{answer}(\text{men}, x - 1) \\ \text{answer}(\text{men}, x) &= \text{answer}(\text{men}, x - 1)\end{aligned}$$

줄 세우기

 Gold 5 - 좁은 미술전시관 (#10476)

 Gold 4 - 개근상 (#1563)

```
answer(women, x) = answer(women, x - 1) + answer(men, x - 1)
answer(men, x) = answer(men, x - 1)
```

```
fibbo(n) = fibbo(n - 1) + fibbo(n - 2)
```

피보나치랑 비슷한 것 같은데...
근데 좀 더 복잡함피보이네 $\pi \pi \dots$

줄 세우기

 Gold 5 - 좁은 미술전시관 (#10476)

 Gold 4 - 개근상 (#1563)

Dynamic Programming

- 특정 범위까지의 값을 구하기 위해 **이전 범위의 값을 활용**하여 효율적으로 값을 얻는 기법.
- 이전 범위의 값을 저장하여, **(Memoization)** 똑같은 문제가 발생했을 때 이를 참조하여 해결함.

줄 세우기

 Gold 5 - 좁은 미술전시관 (#10476)

 Gold 4 - 개근상 (#1563)

DP[N] : 피보나치 수열을 구하기 위해 사용했던 테이블

DP[N][2] : 이 문제를 해결하기 위해 사용하는 테이블

줄 세우기

 Gold 5 - 좁은 미술전시관 (#10476)

 Gold 4 - 개근상 (#1563)

DP[N] : 피보나치 수열을 구하기 위해 사용했던 테이블

DP[N][2] : 이 문제를 해결하기 위해 사용하는 테이블

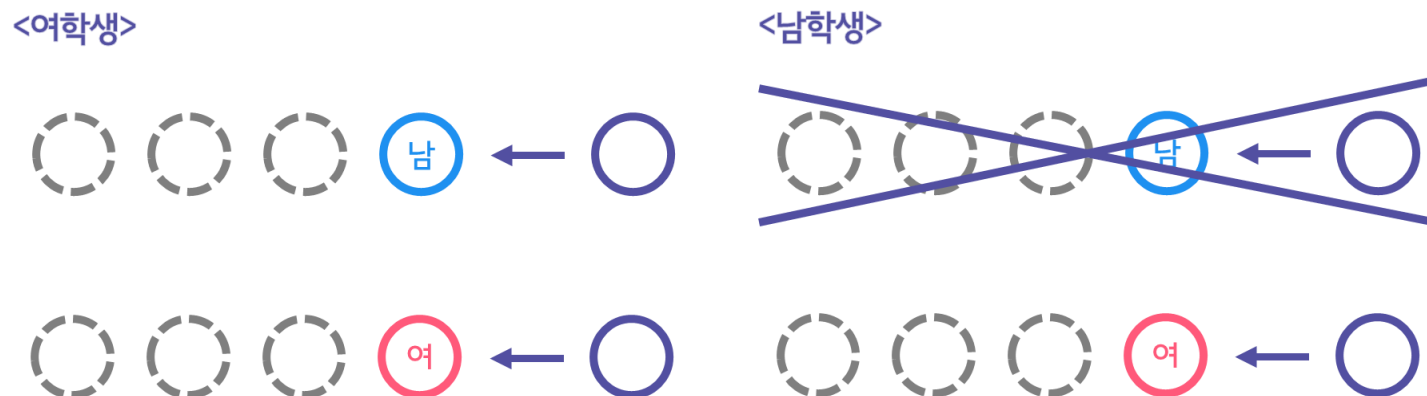
DP[x][0] : 맨 끝이 여학생일 때의 경우의 수!

DP[x][1] : 맨 끝이 남학생일 때의 경우의 수!

줄 세우기

 Gold 5 - 좁은 미술전시관 (#10476)

 Gold 4 - 개근상 (#1563)



$$\begin{aligned} dp[x][0] &= dp[x-1][0] + dp[x-1][1] \\ dp[x][1] &= dp[x-1][0] \end{aligned}$$

List Comprehension

[expression *for* component *in* input_sequence <*if* statement>]

리스트를 쉽게 채워주는 친구입니다!

List Comprehension

[expression *for* component *in* input_sequence <*if* statement>]

리스트를 쉽게 채워주는 친구입니다!

이상한 소문

 Silver 3 - 바이러스 (#2606)

```
import sys
sys.setrecursionlimit(100000)
```

```
def numStudents(n_nodes, myInput) :
    """
```

학생들 사이의 친구관계가 myInput으로 주어질 때, 정원이
가 퍼트린 소문을 듣게되는 학생의 수를 반환합니다.

```
    result = 0
```

```
    return result
```


이상한 소문

 Silver 3 - 바이러스 (#2606)

```
sys.setrecursionlimit(100000)
```



파이썬은 재귀에 너무 약해요...

재귀 함수가 1,000번만 중첩 되어도 프로그램이 터짐!

sys.setrecursionlimit 를 사용하면 재귀 제한을 강제로 늘릴 수 있음!

(주의) 그럼에도 불구하고 다른 언어보다 약합니다...

이상한 소문

 Silver 3 - 바이러스 (#2606)

도와줘! 옛날 자료!

촌수 계산하기

 Silver 2 - 촌수계산 (#2644)

너도... 옛날 자료!