

높이를 입력받아 비 온 후 얼마나 많은 물이 쌓일 수 있는지 계산하라.

Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]

Output: 6

### 1. 브루트 포스

```
def trap(height: List[int]) -> int:
```

```
    volume = 0
```

```
    for i, h in enumerate(height):
```

```
        left_max = max(height[:i + 1])
```

```
        right_max = max(height[i:])
```

```
        volume += min(left_max, right_max) - h
```

```
    return volume
```

- 각 요소마다 채워질수 있는 양(왼쪽, 오른쪽 벽의 최대의 최소)을 찾음
- $O(n^2)$

### 2. 투포인터 활용

```
def trap(height: List[int]) -> int:
```

```
    if not height:
```

```
        return 0
```

```
    volume = 0
```

```
    left, right = 0, len(height) - 1
```

```
    left_max, right_max = height[left], height[right]
```

```
    while left < right:
```

```
        left_max, right_max = max(left_max, height[left]), max(
            right_max, height[right])
```

```
        if left_max <= right_max:
```

```
            volume += left_max - height[left]
```

```
            left += 1
```

```
        else:
```

```
            volume += right_max - height[right]
```

```
            right -= 1
```

```
    return volume
```

- 투 포인터를 왼쪽, 오른쪽에서 최대 지점으로 이동하면서 채워질수 있는 양을 찾음
- $O(n)$

### 3. 스택 활용

```
def trap(height: List[int]) -> int:
```

```
    stack = []
```

```
    volume = 0
```

```
    for i in range(len(height)):
```

```
        while stack and height[i] > height[stack[-1]]:
```

```
            top = stack.pop()
```

```
            if not len(stack):
```

```
                break
```

```
            distance = i - stack[-1] - 1
```

```
            print('d', distance)
```

```
waters = min(height[i], height[stack[-1]]) - height[top]
print('w', waters)
volume += distance * waters
stack.append(i)
return volume
```

- too difficult solve