25-1 이니로
알고리즘 멘토링

멘토 - 김수성

스택 수열 / 1874

백준 1874 / https://www.acmicpc.net/problem/1874

문제

스택 (stack)은 기본적인 자료구조 중 하나로, 컴퓨터 프로그램을 작성할 때 자주 이용되는 개념이다. 스택은 자료를 넣는 (push) 입구와 자료를 뽑는 (pop) 입구가 같아 제일 나중에 들어간 자료가 제일 먼저 나오는 (LIFO, Last in First out) 특성을 가지고 있다.

1부터 n까지의 수를 스택에 넣었다가 뽑아 늘어놓음으로써, 하나의 수열을 만들 수 있다. 이때, 스택에 push하는 순서는 반드시 오름차순을 지키도록 한다고 하자. 임의의 수열이 주어졌을 때 스택을 이용해 그 수열을 만들 수 있는지 없는지, 있다면 어떤 순서로 push와 pop 연산을 수행해야 하는지를 알아낼 수 있다. 이를 계산하는 프로그램을 작성하라.

입력

첫 줄에 n (1 ≤ n ≤ 100,000)이 주어진다. 둘째 줄부터 n개의 줄에는 수열을 이루는 1이상 n이하의 정수가 하나씩 순서대로 주어진다. 물론 같은 정수가 두 번 나오는 일은 없다.

출력

입력된 수열을 만들기 위해 필요한 연산을 한 줄에 한 개씩 출력한다. push연산은 +로, pop 연산은 -로 표현하도록 한다. 불가능한 경우 NO를 출력한다.

스택 수열 / 1874

N이 주어지고 N개의 수가 주어짐
+ 연산은 1부터 차례대로 push 연산을 진행
- 연산은 pop 연산 진행 후 값 출력

예제 입력 1 복사

```
8
4
3
6
8
7
5
2
1
```

예제 출력 1 복사

```
+
+
+
+
-
-
+
+
-
+
+
-
-
-
-
-
```

스택 수열 / 1874

예제 입력 1　　예제 출력 1
8
4 3 6 8 7 5 2 1　　+ + + + - - + + - + + - - - - -

정답이 + 면 1부터 N 순서대로 스택에 push
정답이 – 면 현재 스택의 top 값 출력 및 pop

문제 이해가 어려움
출력 -> 예제 설명

스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

예제 출력 1

+ + + + - - + + - + + - - - - -

pop 연산 값

스택

push 연산 값

스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

예제 출력 1

+ + + + - - + + - + + - - - - -

pop 연산 값

스택

push 연산 값

스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

예제 출력 1

+ + + + - - + + - + + - - - -

pop 연산 값

스택

push 연산 값

스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

예제 출력 1

+ + + + - - + + - + + - - - - -

pop 연산 값

스택

push 연산 값

스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

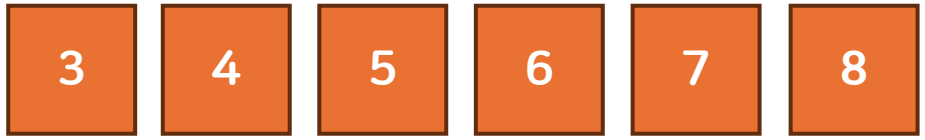
예제 출력 1

+ + + + - - + + - + + - - - - -

pop 연산 값

스택

push 연산 값



스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

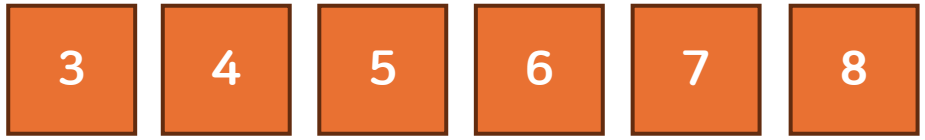
예제 출력 1

+ + + + - - + + - + + - - - - -

pop 연산 값

스택

push 연산 값



스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

예제 출력 1

+ + + + - - + + - + + - - - - -

pop 연산 값

스택

push 연산 값

스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

예제 출력 1

+ + + + - - + + - + + - - - - -

pop 연산 값

스택

push 연산 값

스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

예제 출력 1

+ + + + - - + + - + + - - - - -

pop 연산 값

스택

push 연산 값

스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

예제 출력 1

+ + + + - - + + - + + - - - - -

pop 연산 값

스택

push 연산 값

스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

예제 출력 1

+ + + + - - + + - + + - - - - -

pop 연산 값

스택

push 연산 값

스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

예제 출력 1

+ + + + - - + + - + + - - - - -

pop 연산 값

스택

push 연산 값

스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

예제 출력 1

+ + + + - - + + - + + - - - - -

pop 연산 값

스택

push 연산 값

스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

예제 출력 1

+ + + + - - + + - + + - - - - -

pop 연산 값

스택

push 연산 값

스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

예제 출력 1

+ + + + - - + + - + + - - - - -

pop 연산 값

스택

push 연산 값

스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

예제 출력 1

+ + + + - - + + - + + - - - - -

pop 연산 값

스택

push 연산 값

스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

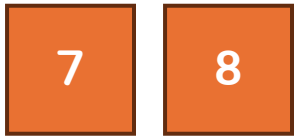
예제 출력 1

+ + + + - - + + - + + - - - - -

pop 연산 값

스택

push 연산 값



스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

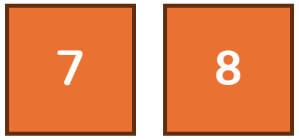
예제 출력 1

+ + + + - - + + - + + - - - - -

pop 연산 값

스택

push 연산 값



스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

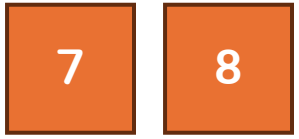
예제 출력 1

+ + + + - - + + - + + - - - - -

pop 연산 값

스택

push 연산 값



스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

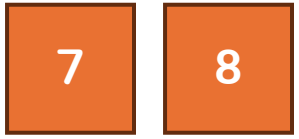
예제 출력 1

+ + + + - - + + - + + - - - - -

pop 연산 값

스택

push 연산 값



스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

예제 출력 1

+ + + + - - + + - + + - - - - -

pop 연산 값

스택

push 연산 값

스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

예제 출력 1

+ + + + - - + + - + + - - - - -

pop 연산 값

스택

push 연산 값

스택 수열 / 1874

예제 입력 1

8

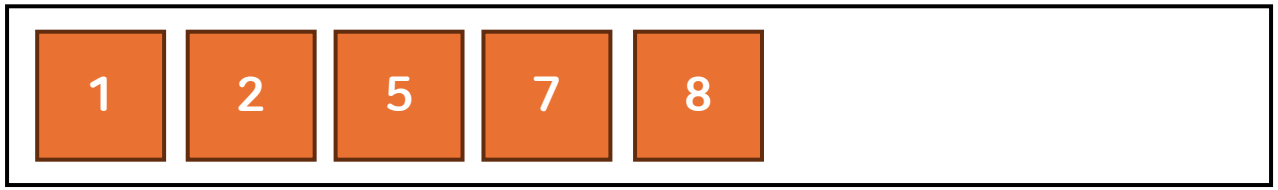
4 3 6 8 7 5 2 1

예제 출력 1

+ + + + - - + + - + + - - - - -

pop 연산 값

스택



push 연산 값

스택 수열 / 1874

예제 입력 1

8

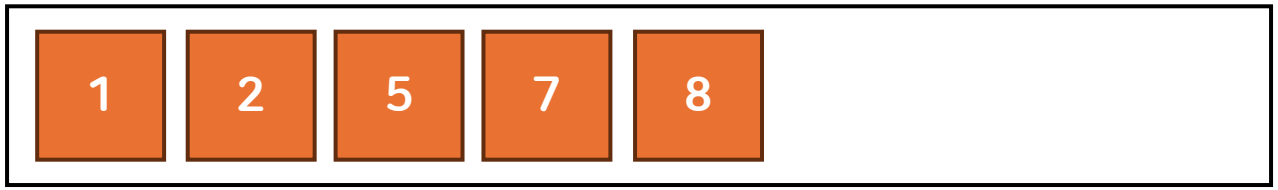
4 3 6 8 7 5 2 1

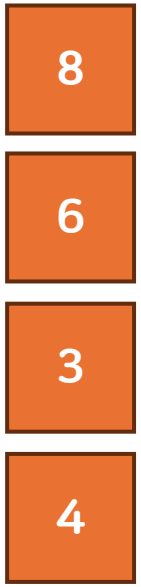
예제 출력 1

+ + + + - - + + - + + - - - - -

pop 연산 값

스택





push 연산 값

스택 수열 / 1874

예제 입력 1

8

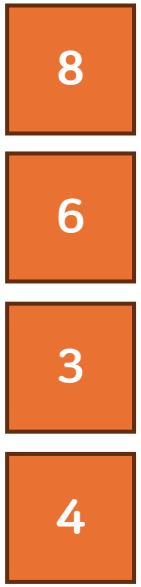
4 3 6 8 7 5 2 1

예제 출력 1

+ + + + - - + + - + + - - - - -

pop 연산 값

스택



push 연산 값

스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

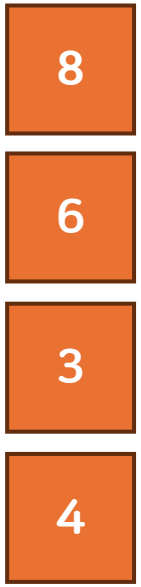
예제 출력 1

+ + + + - - + + - + + - - - -

pop 연산 값

스택

push 연산 값



스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

예제 출력 1

+ + + + - - + + - + + - - - -

pop 연산 값

스택

push 연산 값

스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

예제 출력 1

+ + + + - - + + - + + - - - -

pop 연산 값

스택

push 연산 값

스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

예제 출력 1

+ + + + - - + + - + + - - - -

스택

push 연산 값

pop 연산 값

스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

예제 출력 1

+ + + + - - + + - + + - - - - -

스택

push 연산 값

pop 연산 값

스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

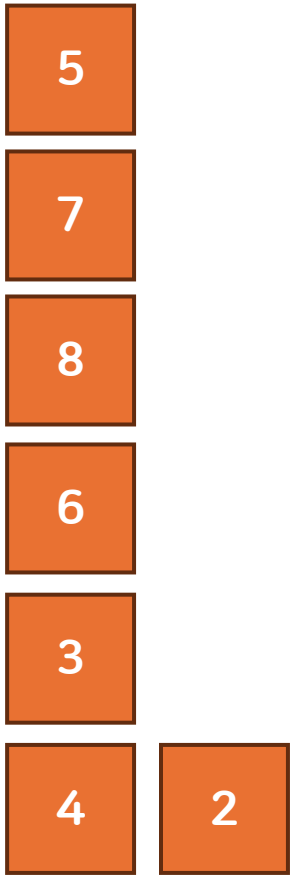
예제 출력 1

+ + + + - - + + - + + - - - - -

스택

push 연산 값

pop 연산 값



스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

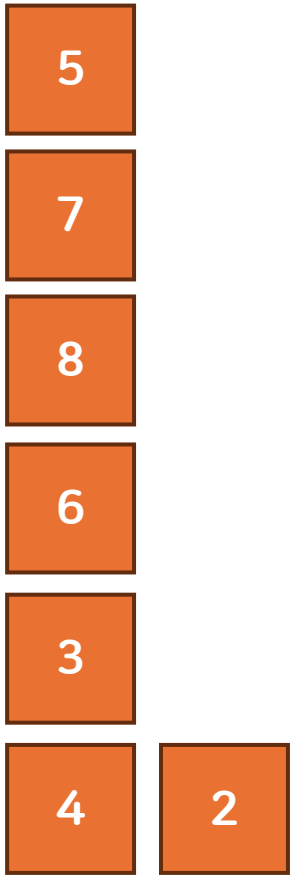
예제 출력 1

+ + + + - - + + - + + - - - -

스택

push 연산 값

pop 연산 값



스택 수열 / 1874

예제 입력 1

8

4 3 6 8 7 5 2 1

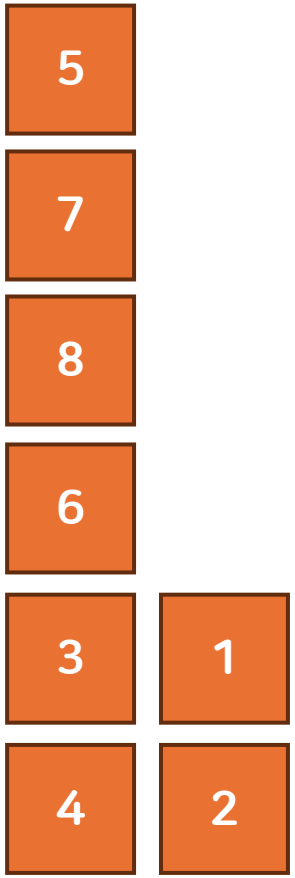
예제 출력 1

+ + + + - - + + - + + - - - -

스택

push 연산 값

pop 연산 값



스택 수열 / 1874

예제 입력 1　　예제 출력 1

8

4 3 6 8 7 5 2 1　　+ + + + - - + + - + + - - - - -

예제 입력은 pop 연산을 했을 때 반환 값의 순서
출력은 push, pop 연산의 순서
+ 는 push - 는 pop 연산

스택 수열 / 1874

예제 입력 1
8
4 3 6 8 7 5 2 1

예제 출력 1
+ + + + - - + + - + + - - - -

x를 출력하기 위해선 x까지의 수를 push 해야 함
이미 x를 넣었다면 pop

x를 스택에 삽입 한 지 판단 하는 방법?
이제 스택에 삽입 할 수가 x 보다 크면
x는 이미 스택에 들어감

스택 수열 / 1874

예제 입력 1　　예제 출력 1
8
4 3 6 8 7 5 2 1　　+ + + + - - + + - + + - - - - -

x를 출력하기 위해선 x까지의 수를 push 해야 함
이미 x를 넣었다면 pop

만약 pop을 했을 때 반환 값이
입력과 다르거나 스택이 비어 있으면 NO

스택 수열 / 1874

8
4 3 6 8 7 5 2 1

스택

push 연산

스택 수열 / 1874

8

4 3 6 8 7 5 2 1

정답

스택

push 연산

스택 수열 / 1874

8

4 3 6 8 7 5 2 1

정답

스택

push 연산

스택 수열 / 1874

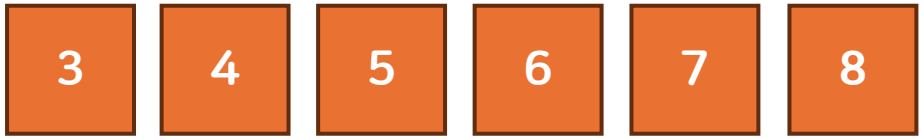
8

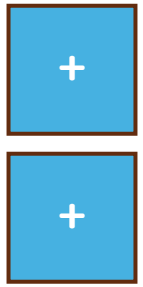
4 3 6 8 7 5 2 1

정답

스택

push 연산





스택 수열 / 1874

8

4 3 6 8 7 5 2 1

정답

스택

push 연산

스택 수열 / 1874

8

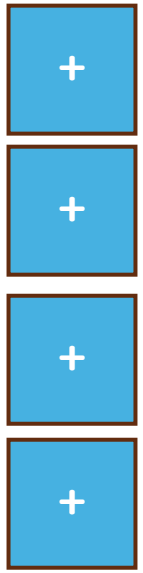
4 3 6 8 7 5 2 1

출력해야 하는 수 까지 넣었으므로 pop

스택

push 연산

정답



스택 수열 / 1874

8

4 3 6 8 7 5 2 1

출력해야 하는 수 까지 넣었으므로 pop

스택

push 연산

정답

스택 수열 / 1874

8

4 3 6 8 7 5 2 1

이미 스택에 3을 넣었으므로 pop

스택

push 연산

정답

스택 수열 / 1874

8

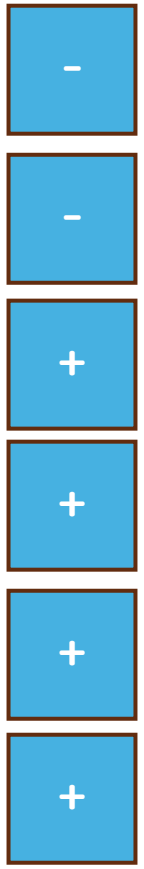
4 3 6 8 7 5 2 1

이미 스택에 3을 넣었으므로 pop

스택

push 연산

정답



스택 수열 / 1874

8

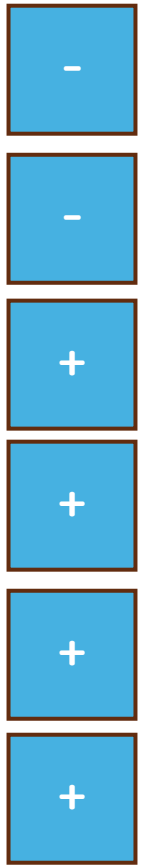
4 3 6 8 7 5 2 1

6을 출력해야 하므로 6까지 push

스택

push 연산

정답



스택 수열 / 1874

8

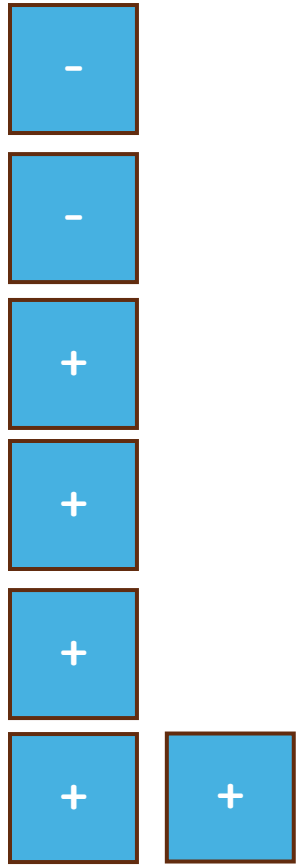
4 3 6 8 7 5 2 1

6을 출력해야 하므로 6까지 push

스택

push 연산

정답



스택 수열 / 1874

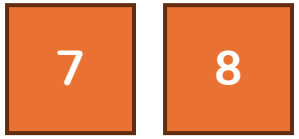
8

4 3 6 8 7 5 2 1

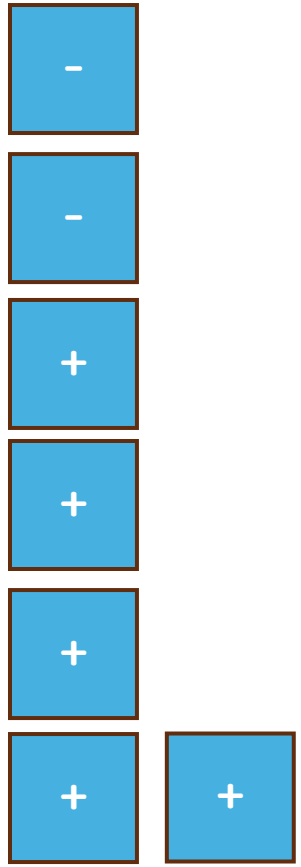
6을 출력해야 하므로 6까지 push

스택

push 연산



정답



스택 수열 / 1874

8

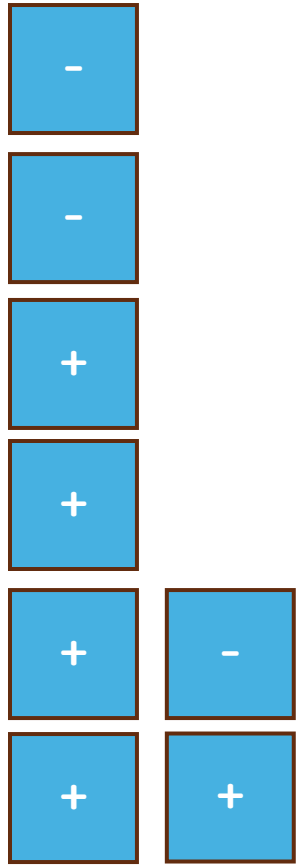
4 3 6 8 7 5 2 1

출력해야 하는 수 까지 넣었으므로 pop

스택

push 연산

정답



스택 수열 / 1874

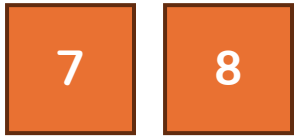
8

4 3 6 8 7 5 2 1

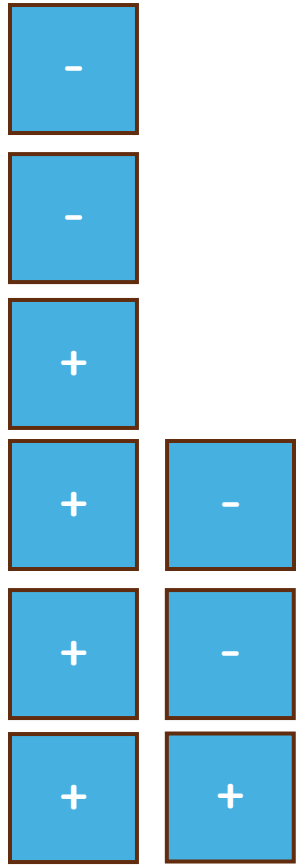
출력해야 하는 수 까지 넣었으므로 pop

스택

push 연산



정답



스택 수열 / 1874

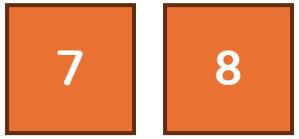
8

4 3 6 8 7 5 2 1

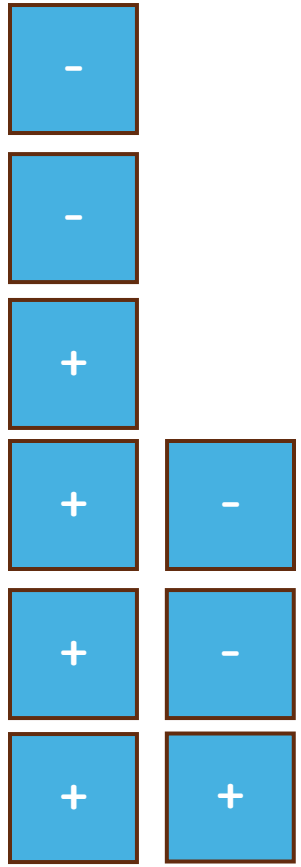
8을 출력해야 하므로 8 까지 push

스택

push 연산



정답



스택 수열 / 1874

8

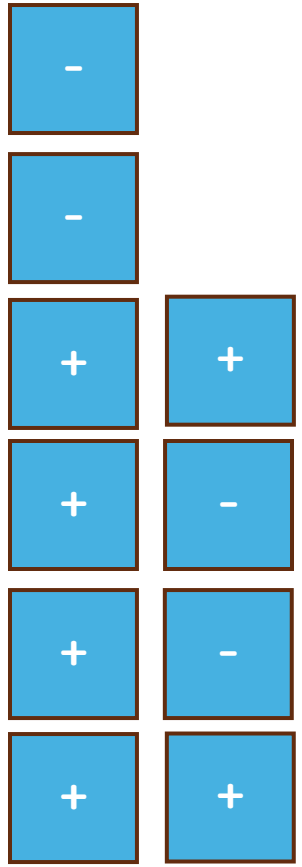
4 3 6 8 7 5 2 1

8을 출력해야 하므로 8 까지 push

스택

push 연산

정답



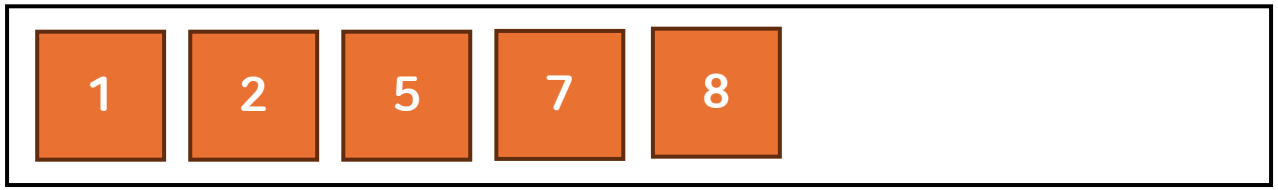
스택 수열 / 1874

8

4 3 6 8 7 5 2 1

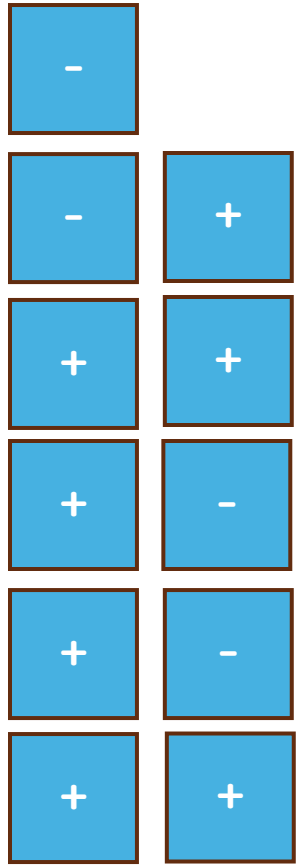
8을 출력해야 하므로 8 까지 push

스택



push 연산

정답



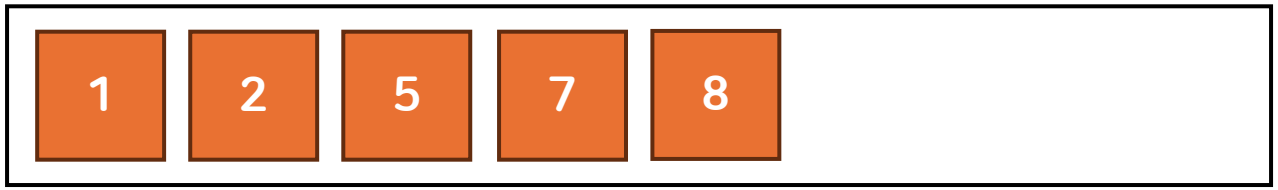
스택 수열 / 1874

8

4 3 6 8 7 5 2 1

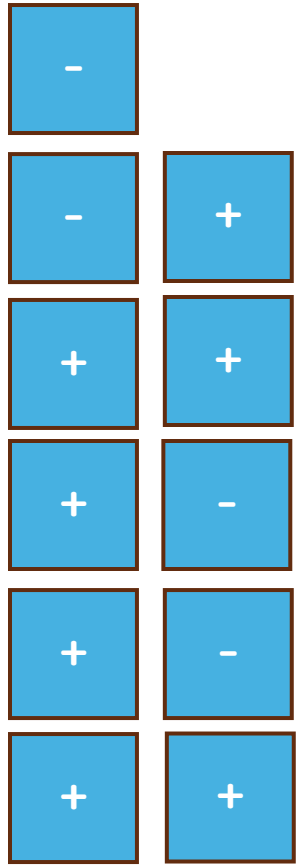
출력 해야 하는 수를 넣었으므로 pop

스택



push 연산

정답



스택 수열 / 1874

8

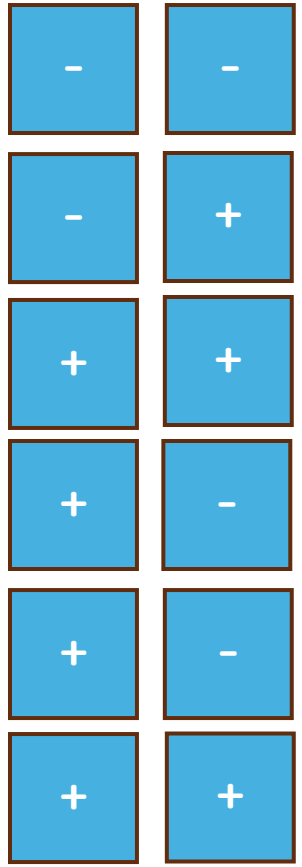
4 3 6 8 7 5 2 1

출력 해야 하는 수를 넣었으므로 pop

스택

push 연산

정답



스택 수열 / 1874

8

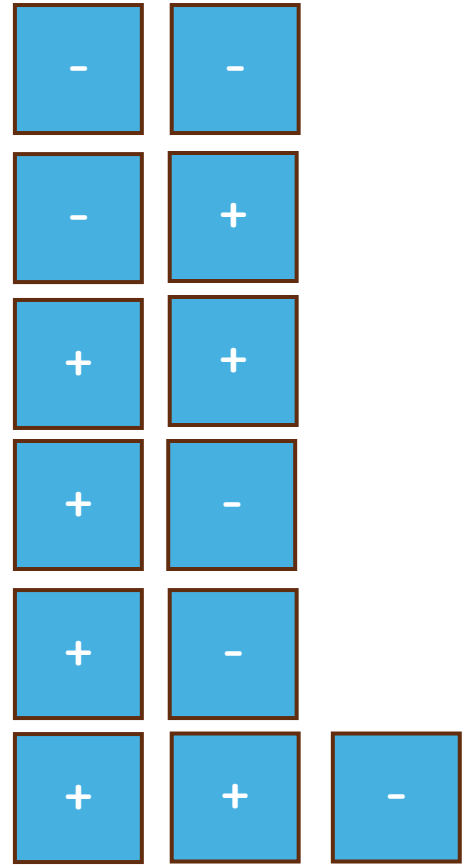
4 3 6 8 7 5 2 1

출력 해야 하는 수를 넣었으므로 pop

스택

push 연산

정답



스택 수열 / 1874

8

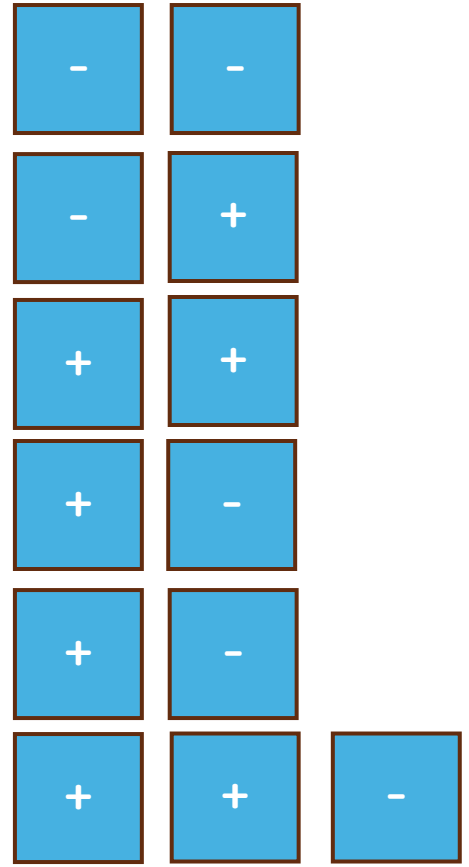
4 3 6 8 7 5 2 1

출력 해야 하는 수를 넣었으므로 pop

스택

push 연산

정답



스택 수열 / 1874

8

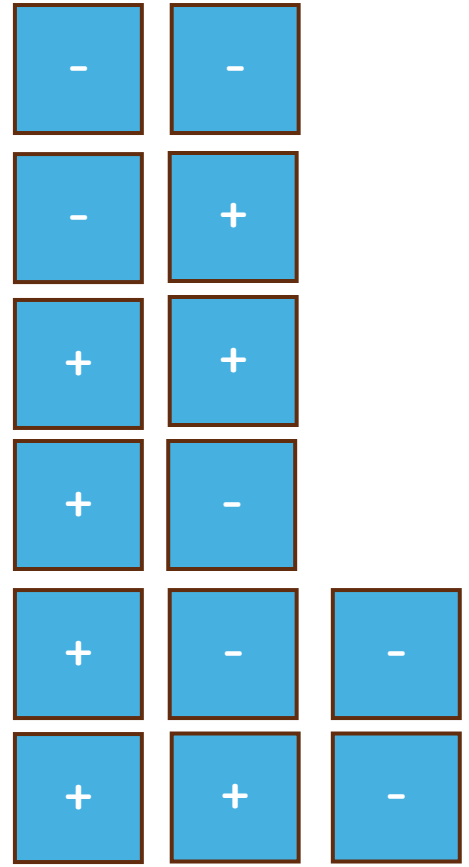
4 3 6 8 7 5 2 1

출력 해야 하는 수를 넣었으므로 pop

스택

push 연산

정답



스택 수열 / 1874

8

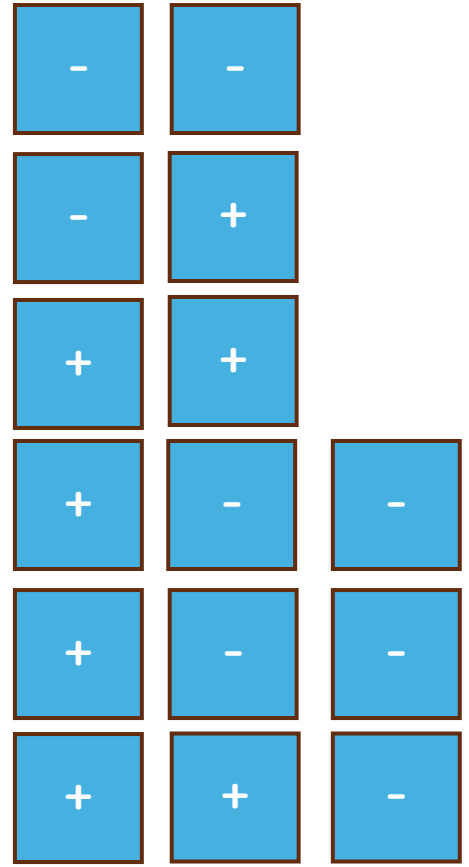
4 3 6 8 7 5 2 1

출력 해야 하는 수를 넣었으므로 pop

스택

push 연산

정답



스택 수열 / 1874

8

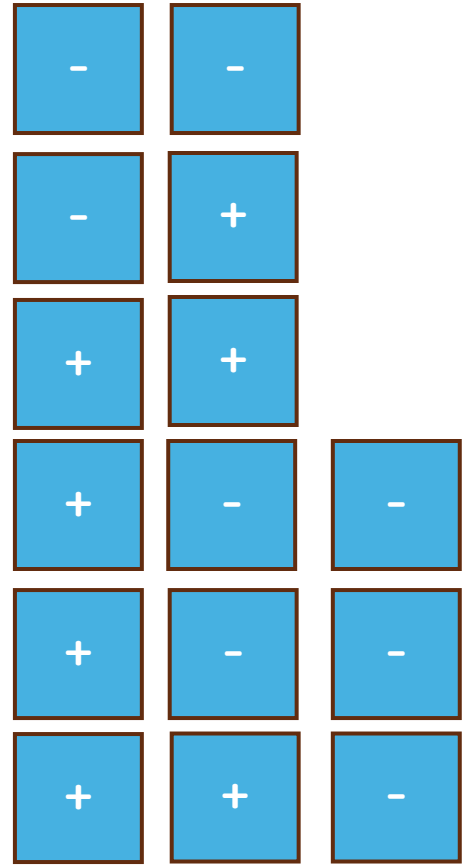
4 3 6 8 7 5 2 1

출력 해야 하는 수를 넣었으므로 pop

스택

push 연산

정답



스택 수열 / 1874

8

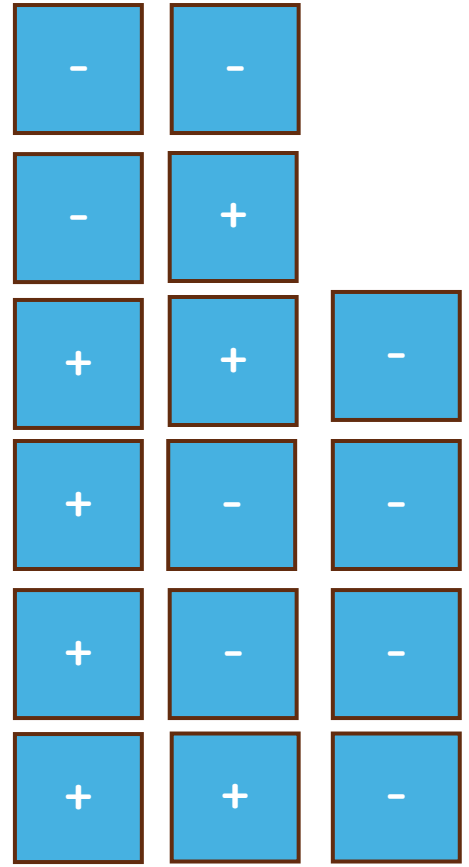
4 3 6 8 7 5 2 1

출력 해야 하는 수를 넣었으므로 pop

스택

push 연산

정답



스택 수열 / 1874

C++
정답을 관리하기 위해
동적 배열 <vector> 사용

#include <vector>
vector <자료형> 변수명;

vector를 처음 본다면
https://dense.tistory.com/entry/cpp-stl-vector

```cpp
#include <iostream>
#include <stack>
#include <vector>
using namespace std;

vector <char> result;
stack <int> st;
int main(){
    int n; cin >> n;
    int use = 1; // 스택에 삽입 할 수
    for(int i = 1;i <= n;i++){
        int x; cin >> x; // 현재 출력해야 하는 수
        while(use <= x){ // x가 삽입이 안되었으면
            st.push(use); use++; // 스택에 수 삽입
            result.push_back('+'); // 정답에 + 추가
        }

        // 스택이 비었거나 스택의 값이 x가 아니면
        if(st.empty() || st.top() != x){
            cout << "NO";
            return 0;
        }

        st.pop();
        result.push_back('-'); // 정답에 - 추가
    }

    for(int i = 0;i < result.size();i++) cout << result[i] << '\n';


    return 0;
}
```

스택 수열 / 1874

Python

```python
n = int(input())

stack = []
result = []
use = 1 # 스택에 삽입 할 수

for i in range(n):
    x = int(input()) # 현재 출력해야 하는 수
    while(use <= x): # x가 삽입이 안 되었으면
        stack.append(use) # 스택에 수 삽입
        use += 1
        result.append('+') # 정답에 + 추가

    # 스택이 비었거나 스택의 값이 x가 아니면
    if(len(stack) == 0 or stack[-1] != x):
        print("NO")
        exit(0)

    stack.pop()
    result.append('-') # 정답에 - 추가

for i in result:
    print(i)
```

질문?

괄호의 값 / 2504

백준 2504 / https://www.acmicpc.net/problem/2504

문제

4개의 기호 '(', ')', '[', ']'를 이용해서 만들어지는 괄호열 중에서 올바른 괄호열이란 다음과 같이 정의된다.

1. 한 쌍의 괄호로만 이루어진 '()'와 '[]'는 올바른 괄호열이다.
2. 만일 X가 올바른 괄호열이면 '(X)'이나 '[X]'도 모두 올바른 괄호열이 된다.
3. X와 Y 모두 올바른 괄호열이라면 이들을 결합한 XY 도 올바른 괄호열이 된다.

예를 들어 '(()[[]])'나 '(())[][]'는 올바른 괄호열이지만 '([)]'나 '(()()[]'은 모두 올바른 괄호열이 아니다. 우리는 어떤 올바른 괄호열 X에 대하여 그 괄호열의 값(괄호값)을 아래와 같이 정의하고 값(X)로 표시한다.

1. '()'인 괄호열의 값은 2이다.
2. '[]'인 괄호열의 값은 3이다.
3. '(X)'의 괄호값은 2×값(X)으로 계산된다.
4. '[X]'의 괄호값은 3×값(X)으로 계산된다.
5. 올바른 괄호열 X와 Y가 결합된 XY의 괄호값은 값(XY)= 값(X)+값(Y)로 계산된다.

예를 들어 '(()[[]])([])'의 괄호값을 구해보자. '()[[]]'의 괄호값이 2 + 3×3=11 이므로 '(()[[]])'의 괄호값은 2×11=22 이다. 그리고 '([])'의 값은 2×3=6 이므로 전체 괄호열의 값은 22 + 6 = 28 이다.

여러분이 풀어야 할 문제는 주어진 괄호열을 읽고 그 괄호값을 앞에서 정의한대로 계산하여 출력하는 것이다.

입력

첫째 줄에 괄호열을 나타내는 문자열(스트링)이 주어진다. 단 그 길이는 1 이상, 30 이하이다.

출력

첫째 줄에 그 괄호열의 값을 나타내는 정수를 출력한다. 만일 입력이 올바르지 못한 괄호열이면 반드시 0을 출력해야 한다.

괄호의 값 / 2504

예제 입력 1
(()[[]])([]) = (2 + [3])(3) = (2 + 3 * 3) + 2 * 3
= 2 * 2 + 2 * (3 * 3) + 2 * 3
= 4 + 18 + 6 = 28
예제 입력 2는 올바른 괄호열이 아님 / 0 출력

예제 입력 1 복사

```
(()[[]])([])
```

예제 출력 1 복사

```
28
```

예제 입력 2 복사

```
[][](([])
```

예제 출력 2 복사

```
0
```

괄호의 값 / 2504

우선 올바른 괄호열인지 판단해보자
(()[[]])([])

닫는 괄호는 아직 매칭 되지 않은
여는 괄호 중에서 가장 마지막 괄호랑 매칭 됨

-> 여는 괄호일 때는 스택에 넣고
    닫는 괄호일 때는 스택의 값이랑 매칭

괄호의 값 / 2504

(()[[]])([])

스택

괄호의 값 / 2504

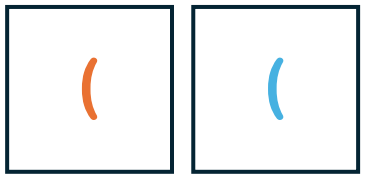
(()[[]])([])

스택

괄호의 값 / 2504

((()[[]])([]))

스택



괄호의 값 / 2504

(()[[]])([])

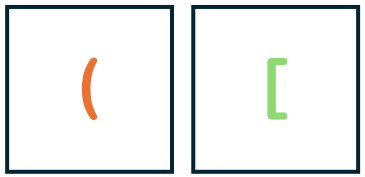
스택

여는 괄호이므로
스택의 마지막 값이랑 매칭

괄호의 값 / 2504

(()[[]])([])

스택



괄호의 값 / 2504

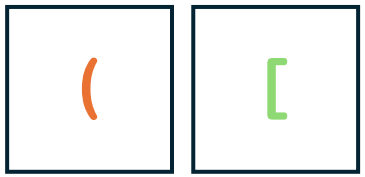
(()[[]])([])

스택

괄호의 값 / 2504

((()[[]])([]))

스택



괄호의 값 / 2504

(()[[]])([])

스택

괄호의 값 / 2504

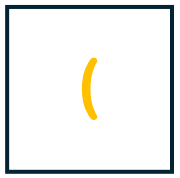
(()[[]])([])

스택

괄호의 값 / 2504

((()[[]])([])

스택



괄호의 값 / 2504

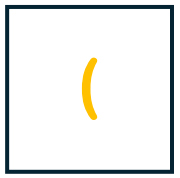
((()[[]])([]))

스택

괄호의 값 / 2504

((()[[]])([]))

스택



괄호의 값 / 2504

(()[[]])([])

스택

괄호의 값 / 2504

()(
모든 괄호 매칭이 끝났을 때
매칭이 안된 값이 있으면 올바른 값이 아님

())
닫는 괄호를 여는 괄호랑 매칭 시켜야 할 때
스택에 값이 없으면 올바른 값이 아님

괄호의 값 / 2504

C++

```cpp
stack <char> st;
bool chk(string& s){
    // 올바른 괄호열이면 1 아니면 0 반환
    for(int i = 0;i < s.size();i++){
        // 여는 괄호면 스택에 추가
        if(s[i] == '(' || s[i] == '[') st.push(s[i]);
        else{
            // 매칭할 여는 괄호가 없으면 틀린 괄호열
            if(st.empty()) return 0;

            // 매칭할 여는 괄호가 다른 종류의 괄호면 틀린 괄호열
            if(s[i] == ')' && st.top() == '[') return 0;
            else if(s[i] == ']' && st.top() == '(') return 0;

            // 여는 괄호랑 매칭 했으므로 여는 괄호를 스택에서 제거
            st.pop();
        }
    }

    // 매칭되지 않은 괄호가 있으면 틀린 괄호열
    if(st.size()) return 0;

    // 끝까지 다 순회했는데 틀린 괄호열이 아니면
    // 올바른 괄호열
    return 1;
}
```

괄호의 값 / 2504

Python

```python
def chk(s):
    # 올바른 괄호열이면 True 아니면 False 반환
    st = []
    for i in s:
        # 여는 괄호면 스택에 추가
        if i == '(' or i == '[':
            st.append(i)
        else:
            # 매칭할 여는 괄호가 없으면 틀린 괄호열
            if len(st) == 0:
                return False

            # 매칭할 여는 괄호가 다른 종류의 괄호면 틀린 괄호열
            if i == ')' and st[-1] == '[':
                return False
            elif i == ']' and st[-1] == '(':
                return False

            # 여는 괄호랑 매칭 했으므로 여는 괄호를 스택에서 제거
            st.pop()

    # 매칭 되지 않은 괄호가 있으면 틀린 괄호열
    # 아니면 올바른 괄호열
    return len(st) == 0
```

괄호의 값 / 2504

예제 입력 1

(()[[]])([]) = (2 + [3])(3) = (2 + 3 * 3) + 2 * 3
= 2 * 2 + 2 * (3 * 3) + 2 * 3
= 4 + 18 + 6 = 28

예제 입력 1 복사

```
(()[[]])([])
```

예제 출력 1 복사

```
28
```

예제 입력 2 복사

```
[][](([])
```

예제 출력 2 복사

```
0
```

괄호의 값 / 2504

예제 입력 1
(()[[]])([]) = (2 + [3])(3)
= 2 * 11 + 2 * 3
= 22 + 6 = 28

괄호의 깊이가 깊은 값부터 계산해야 함
-> 여는 괄호가 들어 올 때마다 깊이가 깊어짐

닫는 괄호가 들어오면 안의 값들을 곱함

괄호의 값 / 2504

예제 입력 1
(()[[]])([])

우선 깊이를 계산해보자
여는 괄호가 들어오면 +1
닫는 괄호가 들어오면 -1

괄호의 값 / 2504

예제 입력 1

(( ) [[ ]]) ([ ])

1 2 1 1 2 2 3 2 1 0 1 2 1 0

깊이가 같은 값들은 서로 더해주면 됨
닫는 괄호가 나왔을 때는 이전 값을 곱함

깊이가 같은 값들은 서로 더해야 함
-> 각 값들을 깊이에 대해서 관리

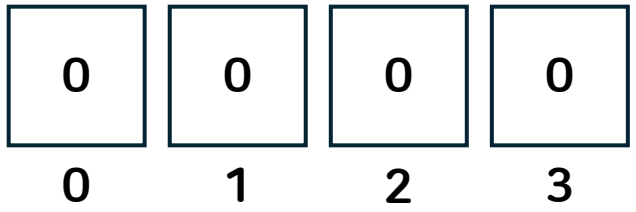
괄호의 값 / 2504

예제 입력 1

(

1

깊이가 같은 값들은 서로 더해주면 됨
닫는 괄호가 나왔을 때는 이전 값을 곱함



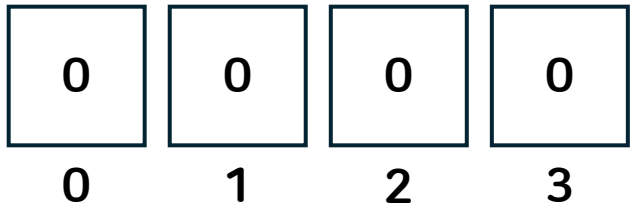
괄호의 값 / 2504

예제 입력 1

((

12

깊이가 같은 값들은 서로 더해주면 됨
닫는 괄호가 나왔을 때는 이전 값을 곱함



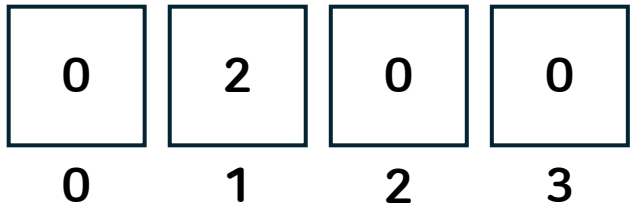
괄호의 값 / 2504

예제 입력 1

(( )

121

닫는 괄호가 나왔는데 이전 값이 0이므로
1로 취급해주고 2를 곱함



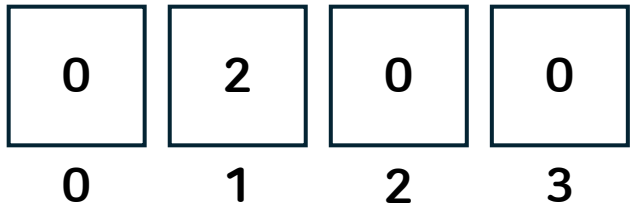
괄호의 값 / 2504

예제 입력 1

(( ) [

1 2 1 2

깊이가 같은 값들은 서로 더해주면 됨
닫는 괄호가 나왔을 때는 이전 값을 곱함

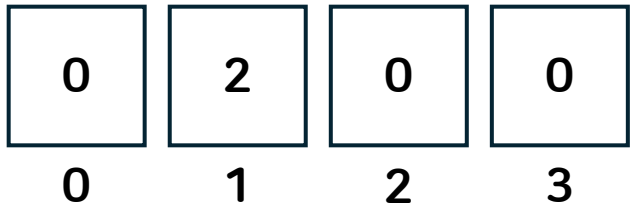


괄호의 값 / 2504

예제 입력 1

(( ) [[
1 2 1 2 3

깊이가 같은 값들은 서로 더해주면 됨
닫는 괄호가 나왔을 때는 이전 값을 곱함



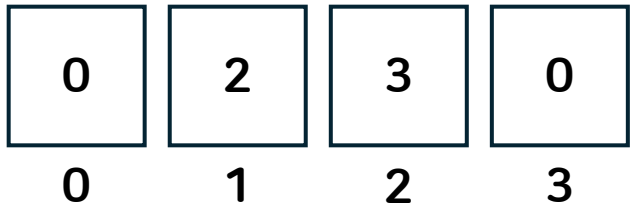
괄호의 값 / 2504

예제 입력 1

(( ) [[ ]]

1 2 1 1 2 2 3 2

마찬가지로 이전 값이 없으므로
1로 취급해주고 3을 곱함



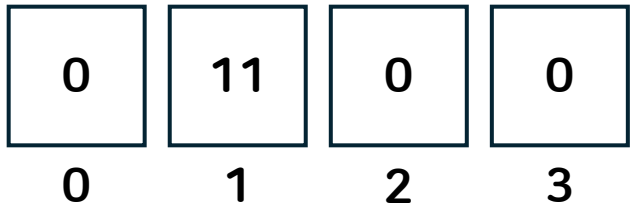
괄호의 값 / 2504

예제 입력 1

(( ) [[ ]]

1 2 1 1 2 3 2 1

이전 값이 3, 3을 곱해줘서 더함
이전 값은 0으로 초기화



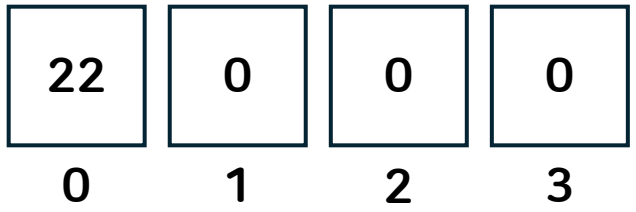
괄호의 값 / 2504

예제 입력 1

(( ) [[ ]])

1 2 1 2 3 2 1 0

이전 값이 2, 2를 곱해줘서 더함
마찬가지로 이전 값은 초기화



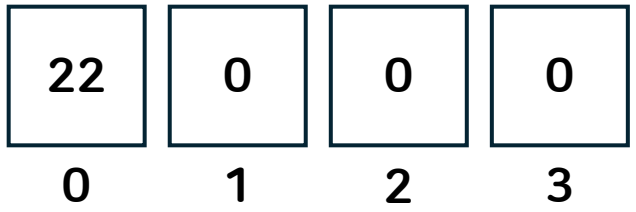
괄호의 값 / 2504

예제 입력 1

(( ) [[ ]]) (

1 2 1 1 2 3 2 1 0 1

깊이가 같은 값들은 서로 더해주면 됨
닫는 괄호가 나왔을 때는 이전 값을 곱함



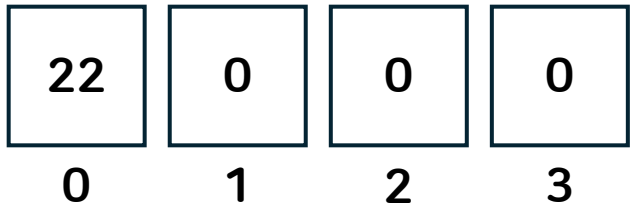
괄호의 값 / 2504

예제 입력 1

(( ) [[ ]]) ([

1 2 1 2 3 2 1 0 1 2

깊이가 같은 값들은 서로 더해주면 됨
닫는 괄호가 나왔을 때는 이전 값을 곱함



괄호의 값 / 2504

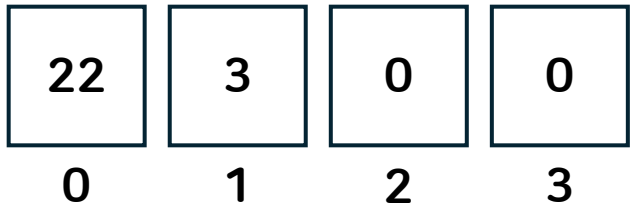
예제 입력 1

(( ) [[ ]]) ([ ]

1 2 1 1 2 2 3 2 1 0 1 2 1

이전 값이 없으므로 1로 취급

3을 더함



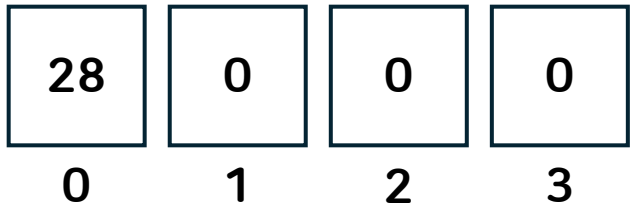
괄호의 값 / 2504

예제 입력 1

(( ) [[ ]]) ([ ])

1 2 1 2 3 2 1 0 1 2 1 0

이전 값이 3, 2를 곱해줘서 더함
마찬가지로 이전 값은 초기화



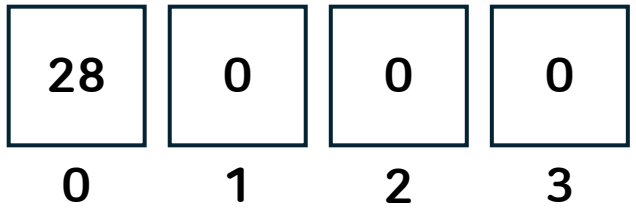
괄호의 값 / 2504

예제 입력 1

(( ) [[ ]]) ([ ])

1 2 1 2 3 2 1 0 1 2 1 0

정답은 인덱스 0에 저장됨



괄호의 값 / 2504

C++

```cpp
stack <char> st;
bool chk(string& s){
    // 올바른 괄호열이면 1 아니면 0 반환
    for(int i = 0;i < s.size();i++){
        // 여는 괄호면 스택에 추가
        if(s[i] == '(' || s[i] == '[') st.push(s[i]);
        else{
            // 매칭할 여는 괄호가 없으면 틀린 괄호열
            if(st.empty()) return 0;

            // 매칭할 여는 괄호가 다른 종류의 괄호면 틀린 괄호열
            if(s[i] == ')' && st.top() == '[') return 0;
            else if(s[i] == ']' && st.top() == '(') return 0;

            // 여는 괄호랑 매칭 했으므로 여는 괄호를 스택에서 제거
            st.pop();
        }
    }

    // 매칭되지 않은 괄호가 있으면 틀린 괄호열
    if(st.size()) return 0;

    // 끝까지 다 순회했는데 틀린 괄호열이 아니면
    // 올바른 괄호열
    return 1;
}
```

```cpp
int result[31];
int main(){
    fastio;
    string s; cin >> s;
    if(!chk(s)){ // 틀린 괄호열이면 0 출력
        cout << 0; return 0;
    }

    int cnt = 0; // 현재 깊이
    for(int i = 0;i < s.size();i++){
        // 여는 괄호면
        if(s[i] == '(' || s[i] == '[') cnt++; // 깊이 증가

        // 닫는 괄호면
        else if(s[i] == ')'){
            cnt--; // 깊이 감소
            // 이전 값에 2를 곱해서 더함
            // 0 이면 1로 취급
            result[cnt] += 2 * max(result[cnt + 1], 1);
            result[cnt + 1] = 0; // 이전 값 초기화
        }
        else{
            cnt--; // 깊이 감소
            // 이전 값에 3을 곱해서 더함
            // 0 이면 1로 취급
            result[cnt] += 3 * max(result[cnt + 1], 1);
            result[cnt + 1] = 0; // 이전 값 초기화
        }
    }

    // 정답은 인덱스 0에 있음
    cout << result[0];
    return 0;
}
```

괄호의 값 / 2504

Python

```python
def chk(s):
    # 올바른 괄호열이면 True 아니면 False 반환
    st = []
    for i in s:
        # 여는 괄호면 스택에 추가
        if i == '(' or i == '[':
            st.append(i)
        else:
            # 매칭할 여는 괄호가 없으면 틀린 괄호열
            if len(st) == 0:
                return False

            # 매칭할 여는 괄호가 다른 종류의 괄호면 틀린 괄호열
            if i == ')' and st[-1] == '[':
                return False
            elif i == ']' and st[-1] == '(':
                return False

            # 여는 괄호랑 매칭 했으므로 여는 괄호를 스택에서 제거
            st.pop()

    # 매칭 되지 않은 괄호가 있으면 틀린 괄호열
    # 아니면 올바른 괄호열
    return len(st) == 0
```

```python
s = input().rstrip()
if not chk(s): # 틀린 괄호열이면 0 출력
    print(0)
    exit(0)

result = [0] * 31
cnt = 0 # 현재 깊이

for i in s:
    if i == '(' or i == '[': # 여는 괄호면
        cnt += 1 # 깊이 증가
    elif i == ')': # 닫는 괄호면
        cnt -= 1 # 깊이 감소
        # 이전 값에 2를 곱해서 더함
        # 0 이면 1로 취급
        result[cnt] += 2 * max(result[cnt + 1], 1)
        result[cnt + 1] = 0 # 이전 값 초기화
    else:
        cnt -= 1 # 깊이 감소
        # 이전 값에 3을 곱해서 더함
        # 0 이면 1로 취급
        result[cnt] += 3 * max(result[cnt + 1], 1)
        result[cnt + 1] = 0 # 이전 값 초기화

# 정답은 인덱스 0에 있음
print(result[0])
```

질문?

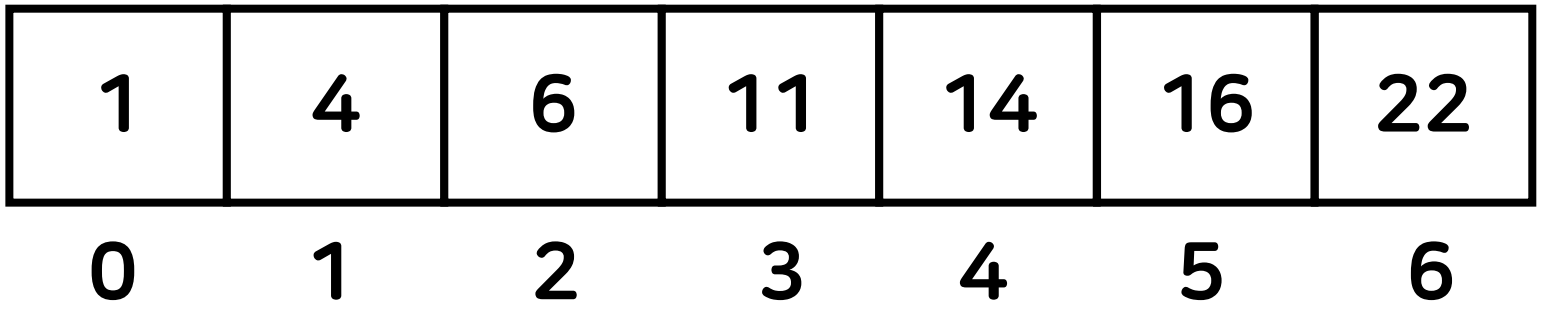
2주차 - 이분탐색 / 매개변수탐색

선형 탐색

정렬 되어 있는 배열 A에서 특정한 수 X를 찾는 방법

나이브하게 구현하면 배열 A를 다 돌아야 함

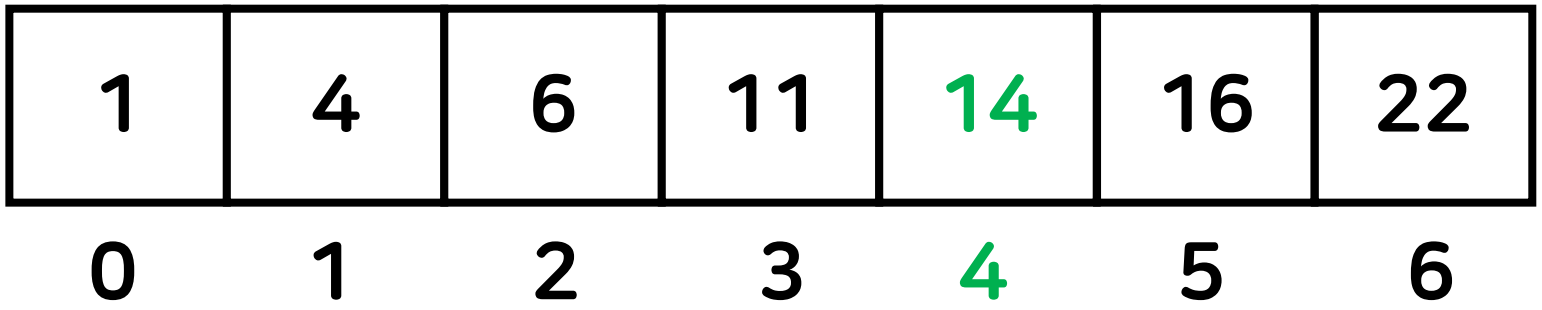
A의 길이가 N일 때 시간 복잡도 O(N)



이분 탐색

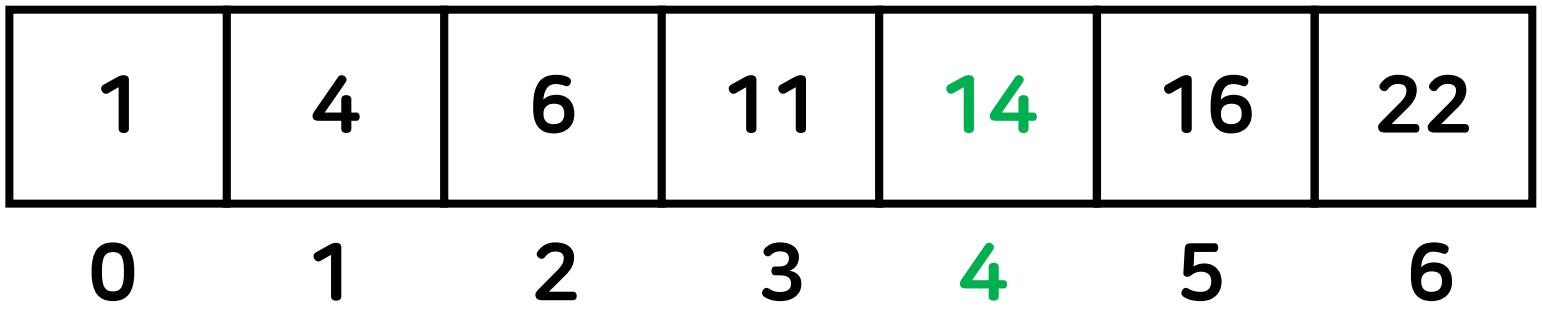
각 수에서 추가적인 정보를 얻을 수는 없을까?

배열 A는 정렬 되어 있으므로
인덱스 0 ~ 3 의 값들은 14 이하
인덱스 5 ~ 6 의 값들은 14 이상



이분 탐색

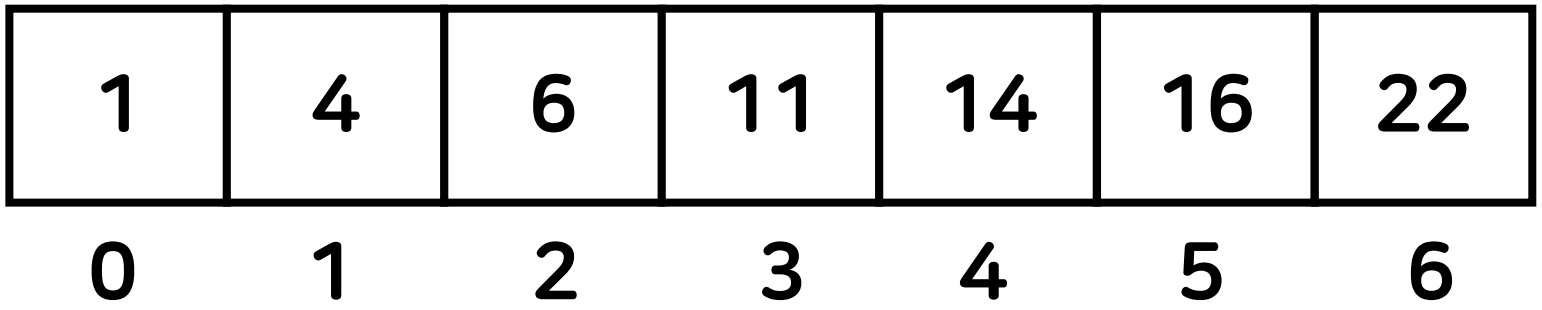
배열 A에서 16을 찾고 있다고 생각할 때
인덱스 4 이전의 값들은 항상 14 이하이므로
인덱스 5 ~ 6의 값만 고려하면 됨



이분 탐색

항상 가운데 수를 탐색하면
탐색 범위를 절반 씩 줄여 나갈 수 있음

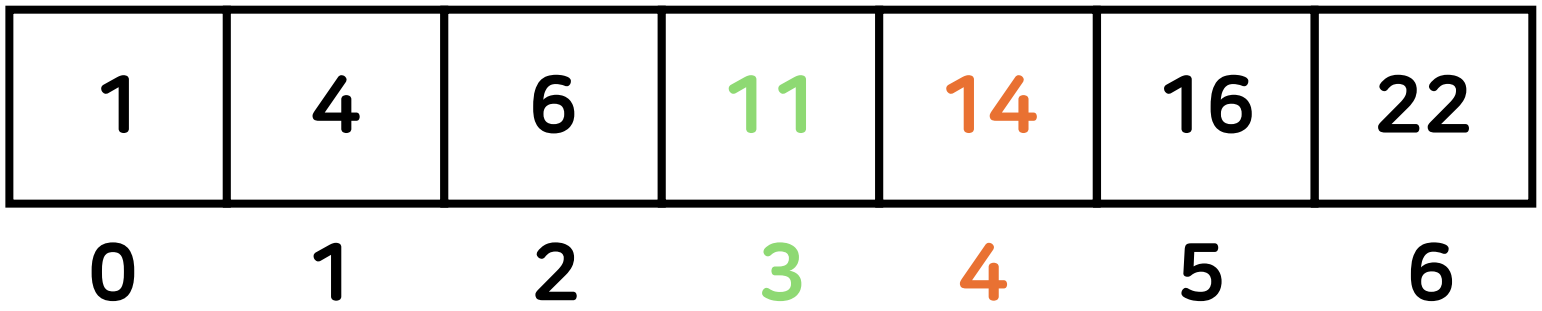
배열 A에서 14를 찾는다고 해보자



이분 탐색

항상 가운데 수를 탐색하면
탐색 범위를 절반 씩 줄여 나갈 수 있음

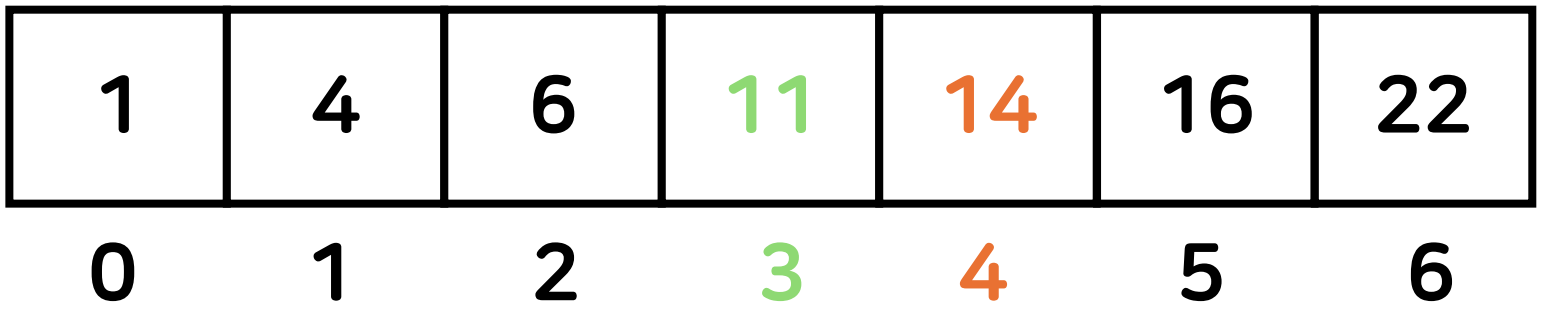
배열 A에서 14를 찾는다고 해보자



이분 탐색

항상 가운데 수를 탐색하면
탐색 범위를 절반 씩 줄여 나갈 수 있음

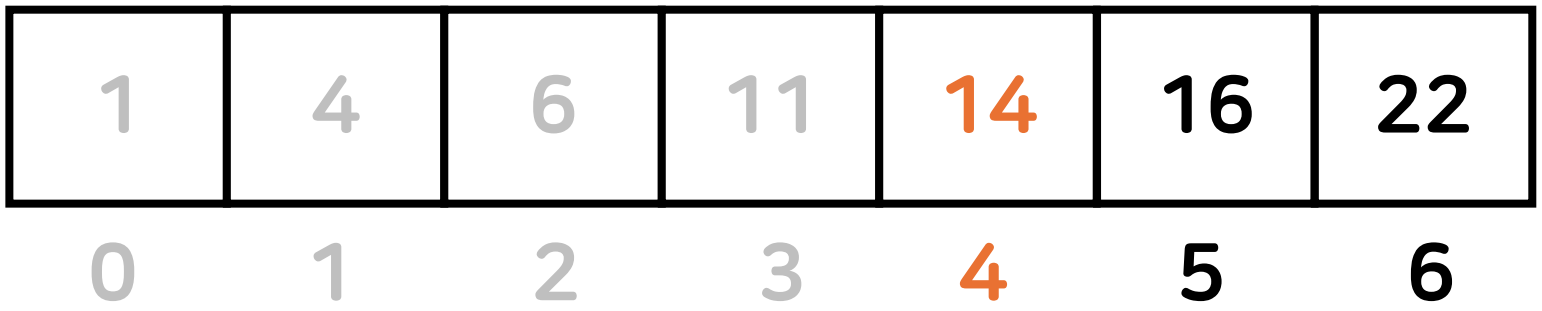
배열 A에서 14를 찾는다고 해보자
A[3] = 11 < 14



이분 탐색

항상 가운데 수를 탐색하면
탐색 범위를 절반 씩 줄여 나갈 수 있음

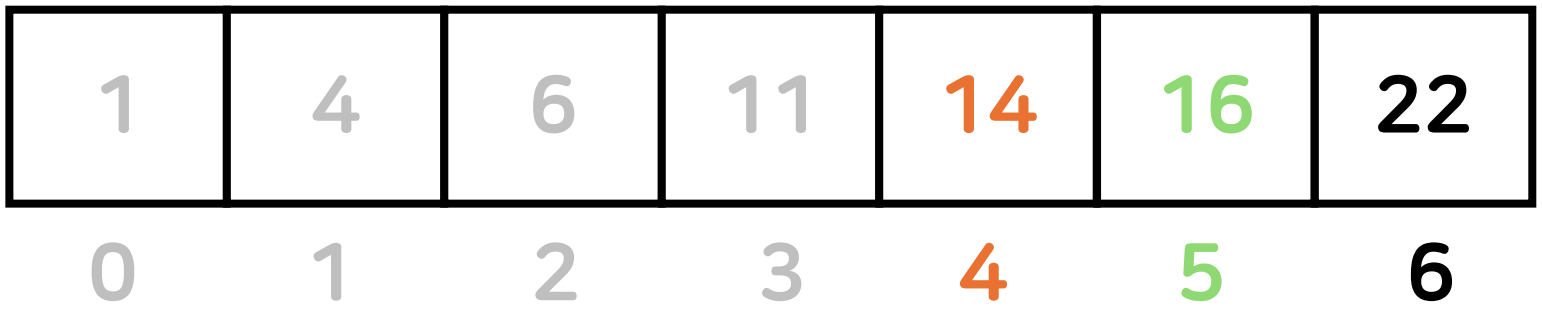
배열 A에서 14를 찾는다고 해보자
A[3] = 11 < 14



이분 탐색

항상 가운데 수를 탐색하면
탐색 범위를 절반 씩 줄여 나갈 수 있음

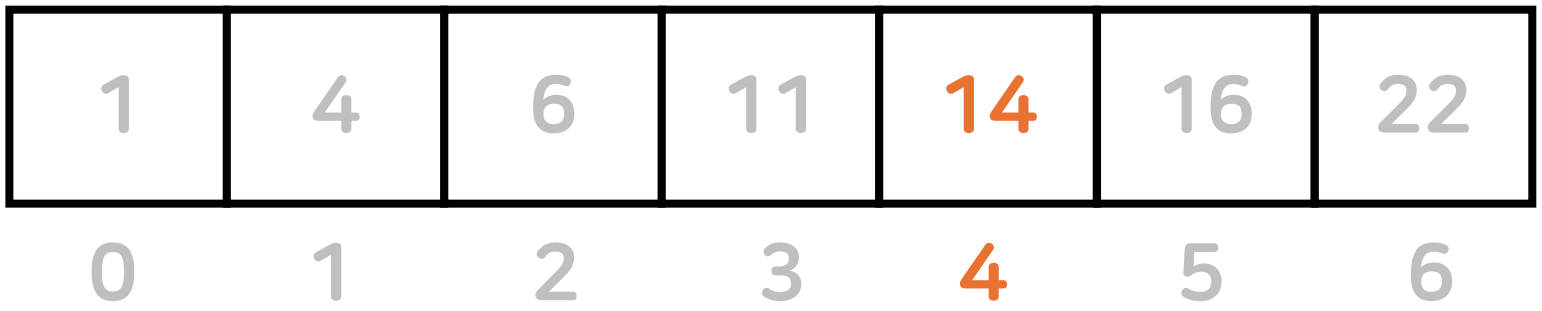
배열 A에서 14를 찾는다고 해보자
A[5] = 16 > 14



이분 탐색

항상 가운데 수를 탐색하면
탐색 범위를 절반 씩 줄여 나갈 수 있음

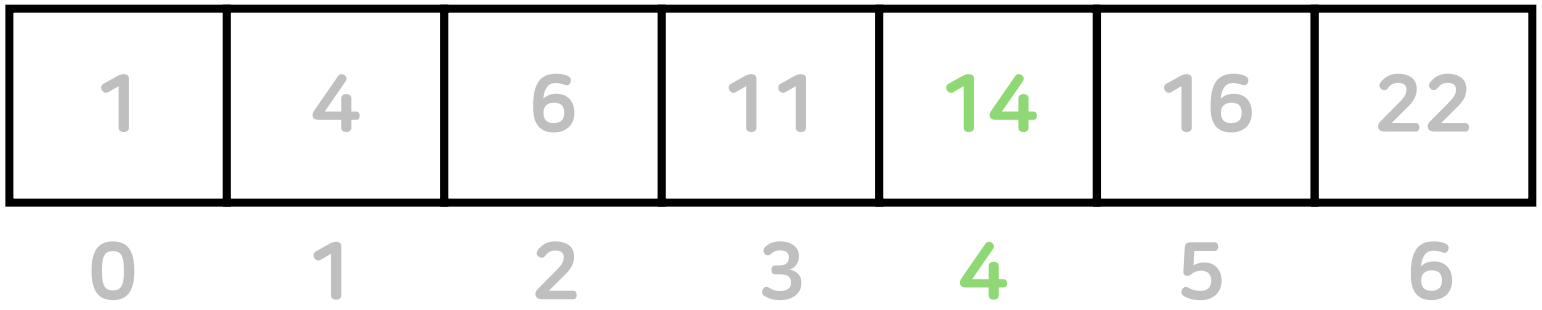
배열 A에서 14를 찾는다고 해보자
A[5] = 16 > 14



이분 탐색

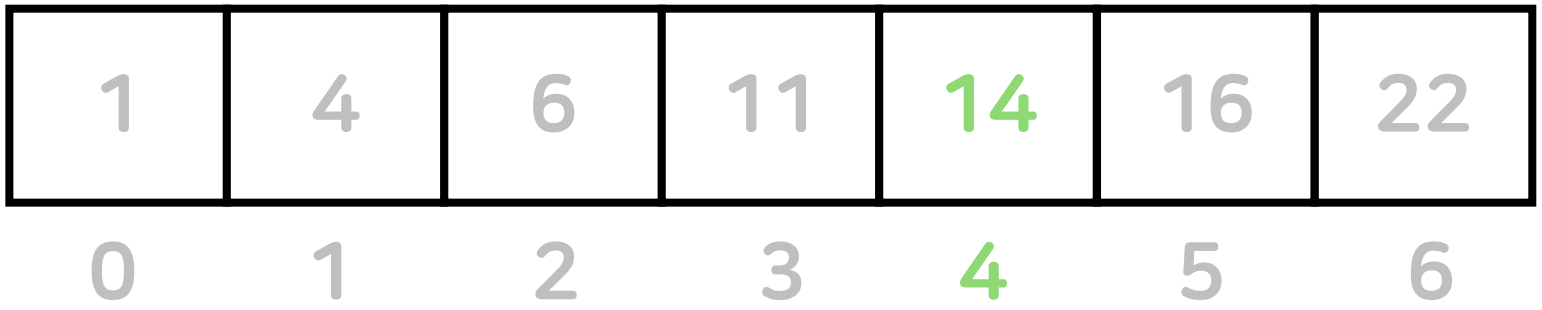
항상 가운데 수를 탐색하면
탐색 범위를 절반 씩 줄여 나갈 수 있음

배열 A에서 14를 찾는다고 해보자
A[4] = 14 == 14



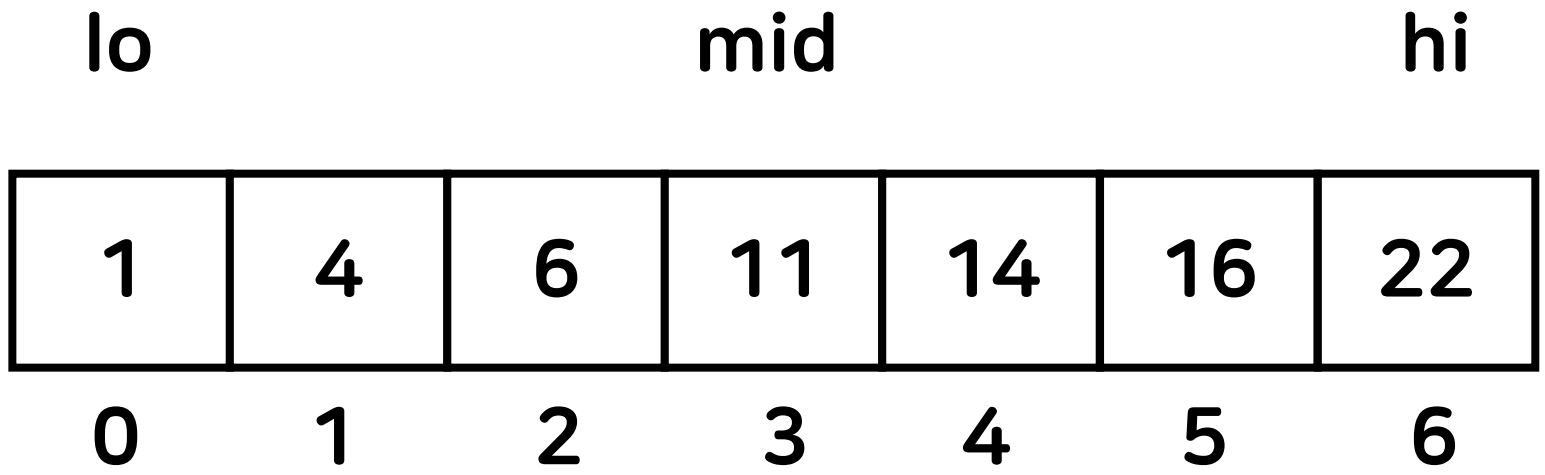
이분 탐색

항상 탐색 범위가 절반 씩 줄음
시간 복잡도는 O(log N)



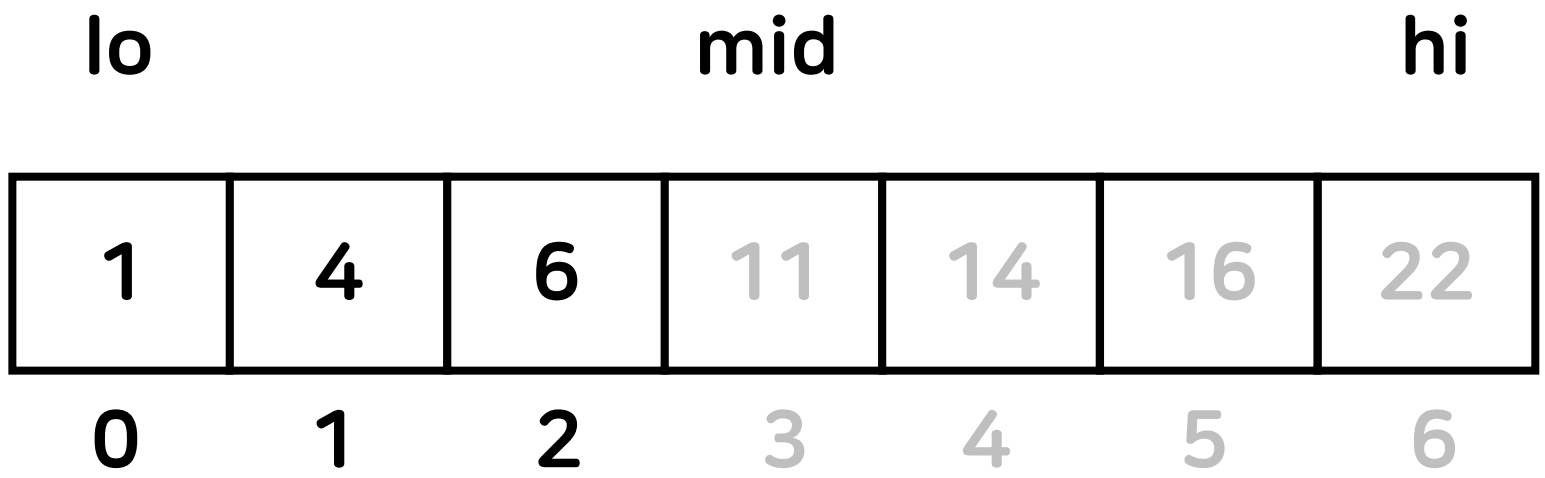
이분 탐색

탐색 범위의 시작점을 lo 끝점을 hi
lo 와 hi 의 중간을 mid 라고 두자



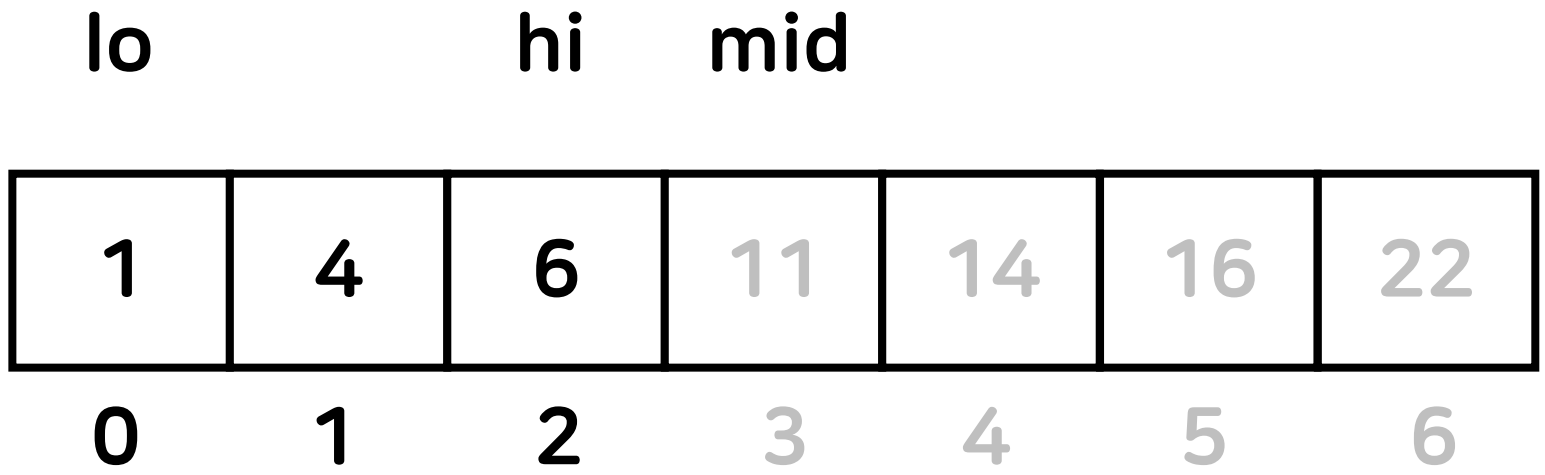
이분 탐색

mid의 값이 X의 값 초과이면 hi = mid - 1
X = 4 < mid = 11



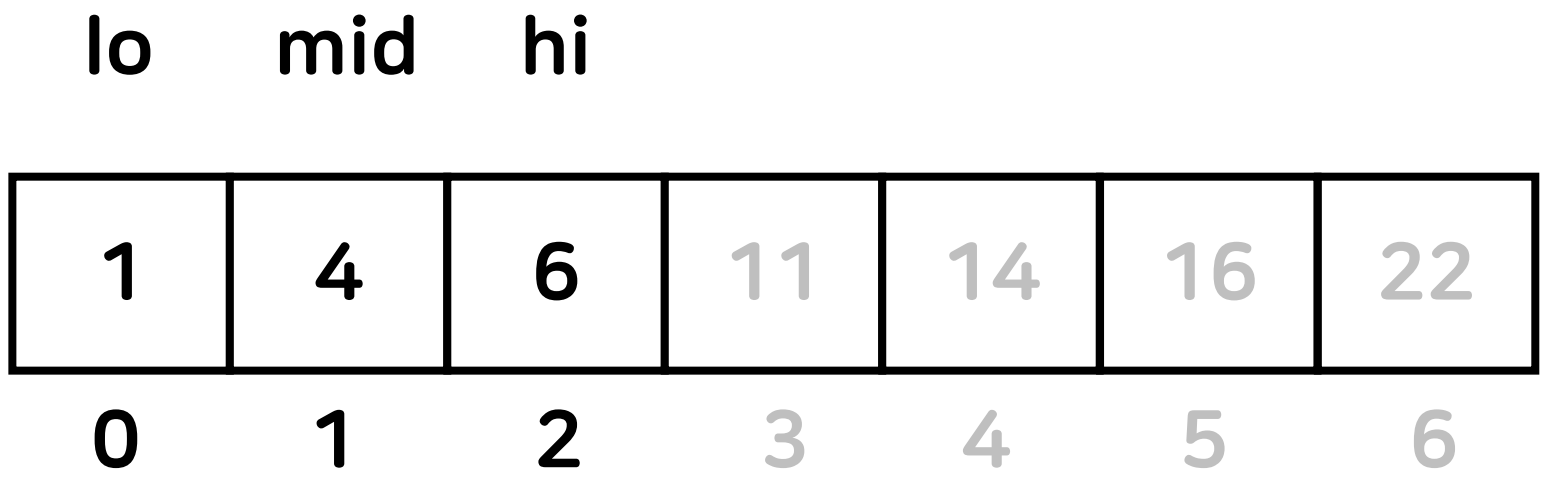
이분 탐색

mid의 값이 X의 값 초과이면 hi = mid - 1
X = 4 < mid = 11



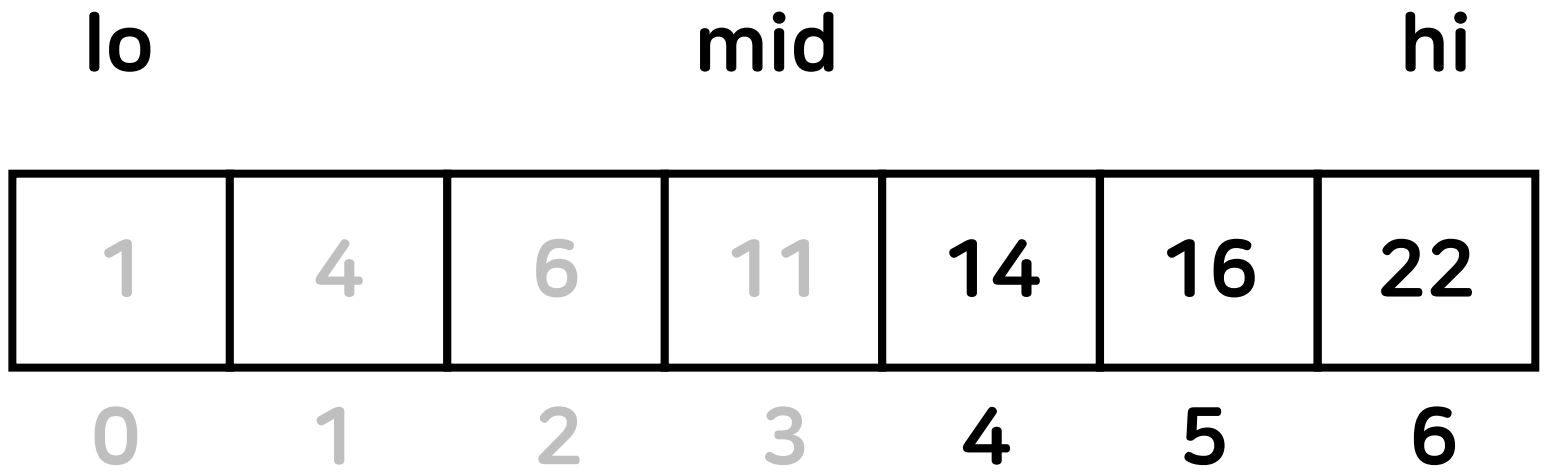
이분 탐색

mid의 값이 X의 값 초과이면 hi = mid - 1
X = 4 < mid = 11



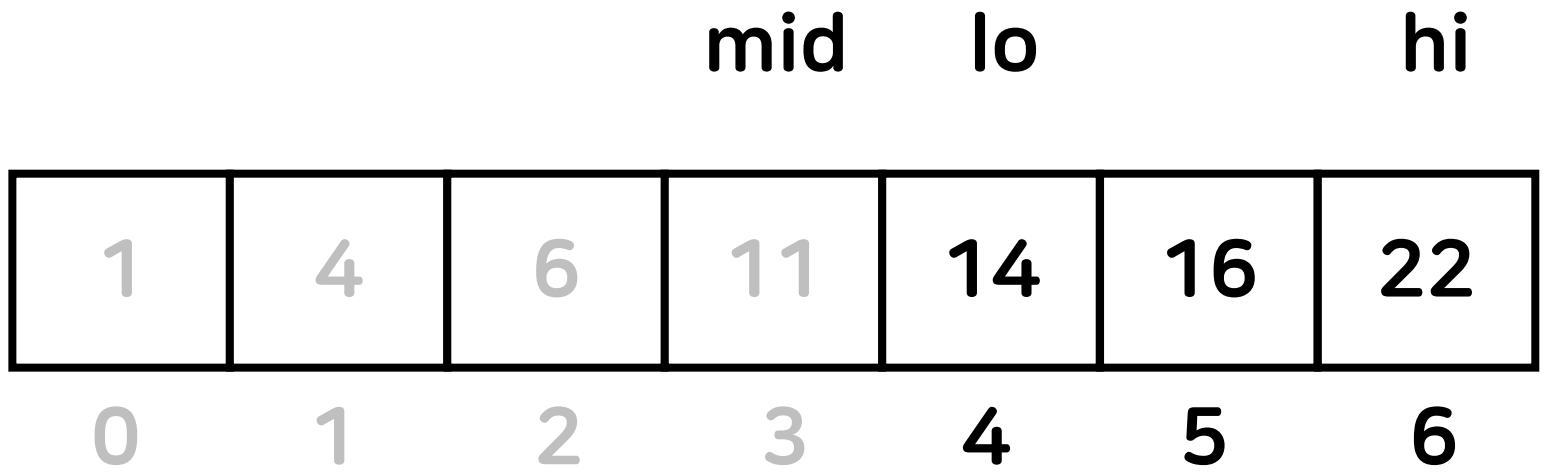
이분 탐색

mid의 값이 X의 값 미만이면 lo = mid + 1
X = 16 > mid = 11



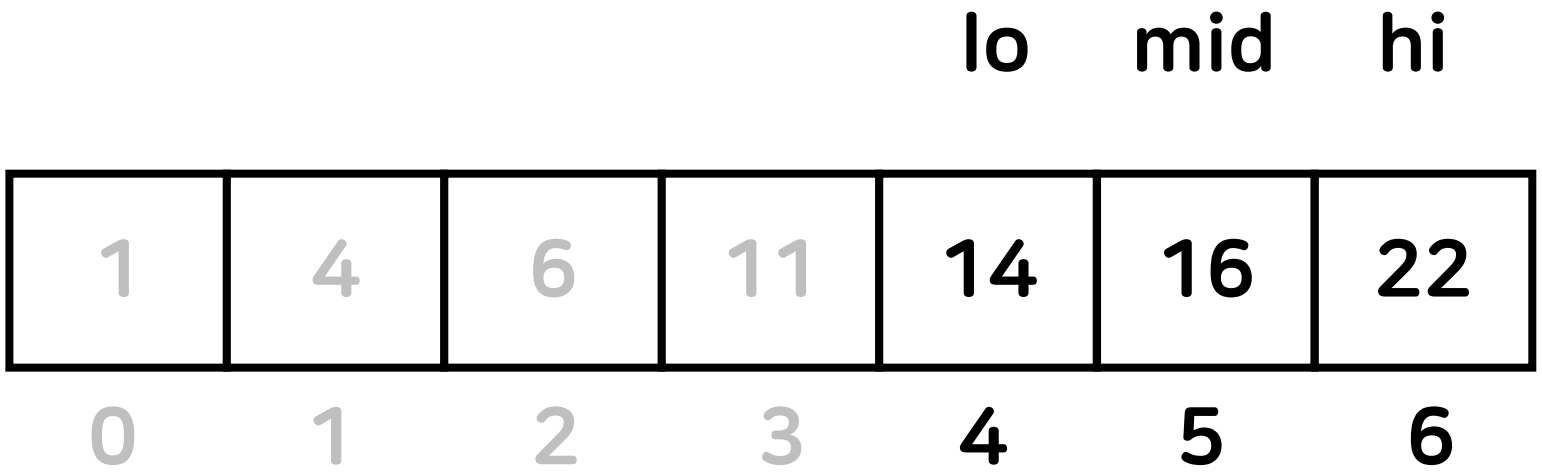
이분 탐색

mid의 값이 X의 값 미만이면 lo = mid + 1
X = 16 > mid = 11



이분 탐색

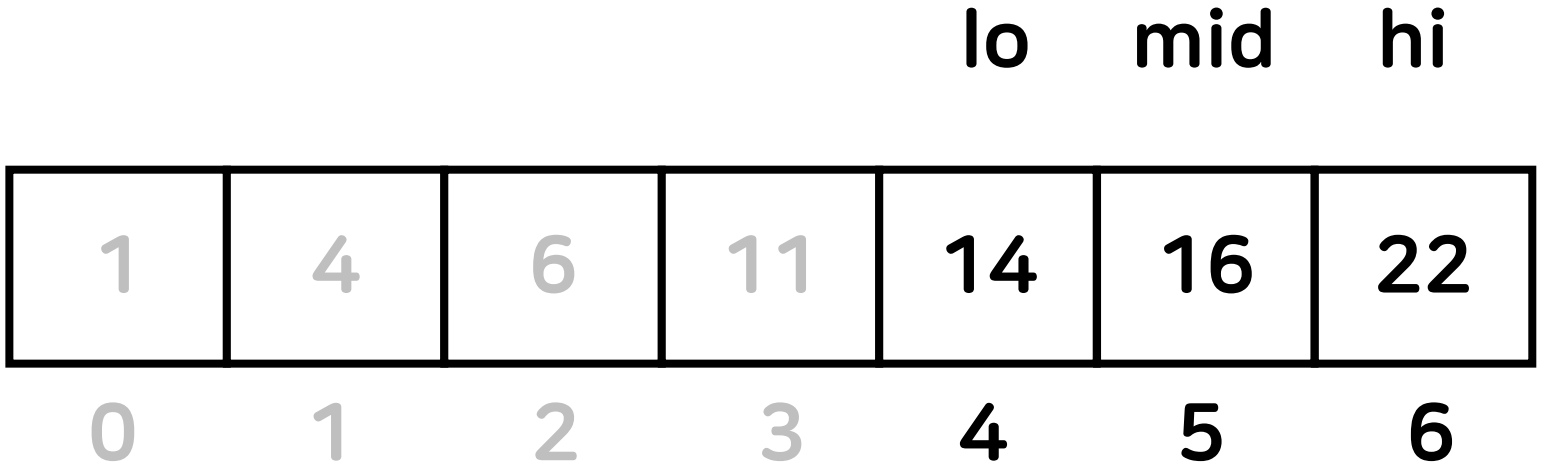
mid의 값이 X의 값 미만이면 lo = mid + 1
X = 16 > mid = 11



이분 탐색

이 과정을 hi가 lo보다 작아지거나
a[mid] == x 일 때 까지 반복

hi가 lo보다 작아질 때 까지 찾지 못하면
그 값은 배열에 없음



이분 탐색

C++

```cpp
#include <iostream>
using namespace std;
int a[7] = { 1, 4, 6, 11, 14, 16, 22};

int binary_search(int x){
    int lo = 0, hi = 6; // A의 크기가 7이므로 hi = 6(7 - 1)
    int ret = -1; // A에 x가 존재하지 않을 경우 -1 반환

    while(lo <= hi){ // 시작점이 끝점보다 커지면 종료
        int mid = (hi + lo) / 2; // 중간값
        if(a[mid] == x){ // 현재 값이 탐색하고 있는 값이면
            ret = mid; break; // 정답은 mid, while문 종료
        }
        // 현재 값이 탐색하고 있는 값보다 크면
        if(a[mid] > x) hi = mid - 1; // mid 이상의 인덱스에는 x가 없음
        else lo = mid + 1; // 아니면 mid 이하의 인덱스에는 x가 없음
    }

    return ret;
}


int main(){
    cout << binary_search(4) << "\n"; // 1
    cout << binary_search(14) << "\n"; // 4
    cout << binary_search(15) << "\n"; // -1

    return 0;
}
```

```
Received Output:
1
4
-1
```

이분 탐색

Python

```python
a = [1, 4, 6, 11, 14, 16, 22]

def binary_search(x):
    lo = 0
    hi = 6 # A의 크기가 7이므로 hi = 6(7 - 1)
    ret = -1 # A에 x가 존재하지 않을 경우 -1 반환

    while lo <= hi: # 시작점이 끝점보다 커지면 종료
        mid = (lo + hi) // 2 # 중간값
        if a[mid] == x: # 현재 값이 탐색하고 있는 값이면
            ret = mid # 정답은 mid
            break # while문 종료

        if a[mid] > x: # 현재 값이 탐색하고 있는 값보다 크면
            hi = mid - 1 # mid 이상의 인덱스에는 x가 없음
        else: # 현재 값이 탐색하고 있는 값보다 작으면
            lo = mid + 1 # mid 이하의 인덱스에는 x가 없음

    return ret

print(binary_search(4)) # 1
print(binary_search(14)) # 4
print(binary_search(15)) # -1
```

```
Received Output:
1
4
-1
```

수 찾기 / 1920

백준 1920 / https://www.acmicpc.net/problem/1920

문제

N개의 정수 A[1], A[2], ..., A[N]이 주어져 있을 때, 이 안에 X라는 정수가 존재하는지 알아내는 프로그램을 작성하시오.

입력

첫째 줄에 자연수 N(1 ≤ N ≤ 100,000)이 주어진다. 다음 줄에는 N개의 정수 A[1], A[2], ..., A[N]이 주어진다. 다음 줄에는 M(1 ≤ M ≤ 100,000)이 주어진다. 다음 줄에는 M개의 수들이 주어지는데, 이 수들이 A안에 존재하는지 알아내면 된다. 모든 정수의 범위는 -2^{31} 보다 크거나 같고 2^{31}보다 작다.

출력

M개의 줄에 답을 출력한다. 존재하면 1을, 존재하지 않으면 0을 출력한다.

수 찾기 / 1920

배열 A의 크기 N이 주어지고 A값이 주어짐
A = [4, 1, 5, 2, 3]
자연수 M이 주어지고 M번에 대해서
X값이 주어지고 A에 X가 있으면 1 아니면 0 출력

예제 입력 1 복사

```
5
4 1 5 2 3
5
1 3 7 9 5
```

예제 출력 1 복사

```
1
1
0
0
1
```

수 찾기 / 1920

이분탐색을 사용하기 위해선 정렬을 해야함
각 언어의 기본 라이브러리 사용 / O(N log N)

X가 M번 주어질 때 이분 탐색을 사용해서
값이 있으면 1 아니면 0 출력
이분 탐색을 M번 해야함 / O(M log N)

예제 입력 1 복사

```
5
4 1 5 2 3
5
1 3 7 9 5
```

예제 출력 1 복사

```
1
1
0
0
1
```

수 찾기 / 1920

이분탐색을 사용하기 위해선 정렬을 해야함
각 언어의 기본 라이브러리 사용 / O(N log N)

C++

<algorithm> 헤더 sort

배열을 정렬 할 때는
시작점이 s, 끝점이 e일 때
sort(a + s, a + e + 1);

```cpp
#include <iostream>
#include <algorithm>
using namespace std;

int main(){
    int a[5] = {4, 1, 2, 5, 3};
    sort(a, a + 5);

    for(int i = 0;i < 5;i++) cout << a[i] << " ";
}
```

```
Received Output:
1 2 3 4 5
```

수 찾기 / 1920

C++

<algorithm> 헤더 sort

벡터를 정렬 할 때는
시작점이 s, 끝점이 e일 때
sort(a.begin() + s , a.begin() + e + 1);

전체를 다 정렬 할 때는
sort(a.begin(), a.end());

```
Received Output:
1 2 3 4 5
```

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

int main(){
    vector <int> a = {4, 1, 2, 5, 3};
    sort(a.begin(), a.end());

    for(int i = 0;i < 5;i++) cout << a[i] << " ";
}
```

수 찾기 / 1920

이분탐색을 사용하기 위해선 정렬을 해야함
각 언어의 기본 라이브러리 사용 / O(N log N)

Python

List의 sort 함수 사용

a.sort()

```
a = [4, 1, 2, 5, 3]
a.sort()

for i in a:
    print(i, end = " ")
```

```
Received Output:
1 2 3 4 5
```

수 찾기 / 1920

C++
fastio를 사용하면 printf(), scanf()
사용이 불가능 하지만
입출력 속도가 증가 함

ios::sync_with_stdio(false);
cin.tie(0); cout.tie(0);

Input:
```
5
4 1 5 2 3
5
1 3 7 9 5
```

Received Output:
```
1
1
0
0
1
```

```cpp
#include <iostream>
#include <algorithm>
using namespace std;
const int MAX = 101010;
int a[MAX];

int binary_search(int x, int size){
    int lo = 0, hi = size - 1; // A의 크기가 size이므로 hi = size - 1
    int ret = -1; // A에 x가 존재하지 않을 경우 -1 반환

    while(lo <= hi){ // 시작점이 끝점보다 커지면 종료
        int mid = (hi + lo) / 2; // 중간값
        if(a[mid] == x){ // 현재 값이 탐색하고 있는 값이면
            ret = mid; break; // 정답은 mid, while문 종료
        }
        // 현재 값이 탐색하고 있는 값보다 크면
        if(a[mid] > x) hi = mid - 1; // mid 이상의 인덱스에는 x가 없음
        else lo = mid + 1; // 아니면 mid 이하의 인덱스에는 x가 없음
    }

    return ret;
}

int main(){
    ios::sync_with_stdio(0); // fastio
    cin.tie(0), cout.tie(0); // fastio

    int n; cin >> n;
    for(int i = 0;i < n;i++) cin >> a[i];
    sort(a, a + n); // 이분탐색을 사용하기 위해선 정렬 해야함

    int m; cin >> m;
    while(m--){
        int x; cin >> x;
        int ret = binary_search(x, n); // 이분탐색 값
        if(ret == -1) cout << 0 << "\n"; // -1 이면 A에 x가 없음
        else cout << 1 << "\n"; // 아니면 A에 x가 있음
    }

    return 0;
}
```

수 찾기 / 1920

Python

import sys
input = sys.stdin.readline

fastio를 사용하면
개행 문자까지 입력 받음
rstrip()으로 개행 문자를
제거해 줘야 함

Input:
```
5
4 1 5 2 3
5
1 3 7 9 5
```

Received Output:
```
1
1
0
0
1
```

```python
import sys
input = sys.stdin.readline

def binary_search(x, size):
    lo = 0
    hi = size - 1 # A의 크기가 size 이므로 hi = size - 1
    ret = -1 # A에 x가 존재하지 않을 경우 -1 반환

    while lo <= hi: # 시작점이 끝점보다 커지면 종료
        mid = (lo + hi) // 2 # 중간값
        if a[mid] == x: # 현재 값이 탐색하고 있는 값이면
            ret = mid # 정답은 mid
            break # while문 종료

        if a[mid] > x: # 현재 값이 탐색하고 있는 값보다 크면
            hi = mid - 1 # mid 이상의 인덱스에는 x가 없음
        else: # 현재 값이 탐색하고 있는 값보다 작으면
            lo = mid + 1 # mid 이하의 인덱스에는 x가 없음

    return ret

n = int(input())
a = list(map(int, input().rstrip().split()))
a.sort() # 이분탐색을 사용하기 위해선 정렬 해야함

m = int(input())
x = list(map(int, input().rstrip().split()))

for i in range(m):
    ret = binary_search(x[i], n) # 이분탐색 값
    if ret == -1: # -1 이면 A에 X가 없음
        print(0)
    else: # 아니면 A에 X가 있음
        print(1)
```

매개 변수 탐색

매개 변수 탐색

최적화 문제를 결정 문제로 바꾸어
이분 탐색으로 문제를 해결 하는 것

결정 문제

Yes/No 로 답할 수 있는 문제

최적화 문제

최대값, 최소값을 찾는 문제

매개 변수 탐색

결정 문제

Yes/No 로 답할 수 있는 문제

최적화 문제

최대값, 최소값을 찾는 문제

결정 문제가 최적화 문제보다 항상 쉬움
최적화 문제를 해결 할 수 있으면
결정 문제를 항상 해결 할 수 있음

랜선 자르기 / 1654

백준 1654 / https://www.acmicpc.net/problem/1654

문제

집에서 시간을 보내던 오영식은 박성원의 부름을 받고 급히 달려왔다. 박성원이 캠프 때 쓸 N개의 랜선을 만들어야 하는데 너무 바빠서 영식이에게 도움을 청했다.

이미 오영식은 자체적으로 K개의 랜선을 가지고 있다. 그러나 K개의 랜선은 길이가 제각각이다. 박성원은 랜선을 모두 N개의 같은 길이의 랜선으로 만들고 싶었기 때문에 K개의 랜선을 잘라서 만들어야 한다. 예를 들어 300cm 짜리 랜선에서 140cm 짜리 랜선을 두 개 잘라내면 20cm는 버려야 한다. (이미 자른 랜선은 붙일 수 없다.)

편의를 위해 랜선을 자르거나 만들 때 손실되는 길이는 없다고 가정하며, 기존의 K개의 랜선으로 N개의 랜선을 만들 수 없는 경우는 없다고 가정하자. 그리고 자를 때는 항상 센티미터 단위로 정수길이만큼 자른다고 가정하자. N개보다 많이 만드는 것도 N개를 만드는 것에 포함된다. 이때 만들 수 있는 최대 랜선의 길이를 구하는 프로그램을 작성하시오.

입력

첫째 줄에는 오영식이 이미 가지고 있는 랜선의 개수 K, 그리고 필요한 랜선의 개수 N이 입력된다. K는 1이상 10,000이하의 정수이고, N은 1이상 1,000,000이하의 정수이다. 그리고 항상 K ≤ N 이다. 그 후 K줄에 걸쳐 이미 가지고 있는 각 랜선의 길이가 센티미터 단위의 정수로 입력된다. 랜선의 길이는 2^{31}-1보다 작거나 같은 자연수이다.

출력

첫째 줄에 N개를 만들 수 있는 랜선의 최대 길이를 센티미터 단위의 정수로 출력한다.

랜선 자르기 / 1654

예제 입력 1 복사

```
4 11
802
743
457
539
```

예제 출력 1 복사

```
200
```

힌트

802cm 랜선에서 4개, 743cm 랜선에서 3개, 457cm 랜선에서 2개, 539cm 랜선에서 2개를 잘라내 모두 11개를 만들 수 있다.

랜선 자르기 / 1654

최적화 문제

최대값, 최소값을 찾는 문제

M개의 랜선을 만들 수 있는 랜선의 최대 길이

결정 문제

Yes/No 로 답할 수 있는 문제

랜선의 길이를 K로 잘랐을 때
M개 이상의 랜선을 만들 수 있는가

랜선 자르기 / 1654

결정 문제

랜선의 길이를 K로 잘랐을 때
M개의 랜선을 만들 수 있는가

K = 0, 1, 2 … 에 대해서 결정 문제를 해결
Decision(K) = Yes 인 K의 최댓값을 찾으면 됨

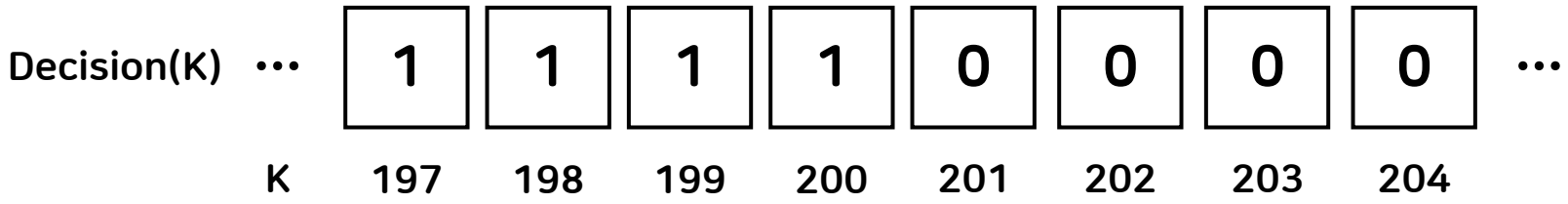
모든 K에 대해서 문제를 해결해야 함 -> 비효율적

랜선 자르기 / 1654

예제 입력 1
4 11
802
743
457
539

예제 출력 1
200

Decision(K)가 1인 K의 최댓값이 정답
결정 문제의 값은 항상 정렬 되어 있음 -> 이분 탐색



랜선 자르기 / 1654

이분 탐색

정답이 존재하는 구간의 [lo, hi] 중간 지점 mid

Decision(mid)가 0이면 정답은 [lo, mid - 1]에 존재
Decision(mid)가 1이면 정답은 [mid, hi]에 존재

이분 탐색의 조건 -> 정렬
결정 문제의 값들이 정렬 되어 있지 않으면
매개 변수 탐색을 사용 할 수 없음

랜선 자르기 / 1654

C++

int를 사용하면 overflow
long long을 사용

mid = (lo + hi + 1) / 2;
올림 값 사용

Input:
```
4 11
802
743
457
539
```

Expected Output:
```
200
```

```cpp
int main(){
    cin >> n >> m;
    for(int i = 1;i <= n;i++) cin >> a[i];
    cout << maximazation(); // 최댓값 출력

    return 0;
}
```

```cpp
#include <iostream>
using namespace std;
int n, m, a[1010101];

bool decision(long long cur){
    long long cnt = 0; // 만들 수 있는 랜선의 개수
    for(int i = 1;i <= n;i++){
        // 랜선의 길이를 cur 만큼 자를 때
        // a[i] / cur 만큼 랜선이 나옴
        cnt += a[i] / cur;
    }

    // 만들 수 있는 랜선의 개수가 M 이상이면 1 아니면 0 반환
    return cnt >= m;
}

int maximazation(){
    // 정답의 범위는 1 ~ 2^31 - 1
    long long lo = 1, hi = (1 << 31) - 1;
    while(lo < hi){
        long long mid = (lo + hi + 1) / 2; // 중간값
        // 결정 문제의 답이 1 이면
        // 정답은 [mid, hi]에 존재
        if(decision(mid)) lo = mid;

        // 결정 문제의 답이 0 이면
        // 정답은 [lo, mid - 1]에 존재
        else hi = mid - 1;
    }

    return lo;
}
```

랜선 자르기 / 1654

Python

mid = (lo + hi + 1) // 2;
올림 값 사용

```
Input:
4 11
802
743
457
539

Expected Output:
200
```

```python
import sys
input = sys.stdin.readline

n, m = list(map(int, input().rstrip().split()))
a = [int(input().rstrip()) for _ in range(n)]

def decision(cur):
    cnt = 0 # 만들 수 있는 랜선의 개수
    for i in a:
        # 랜선의 길이를 cur 만큼 자를 때
        # a[i] // cur 만큼 랜선이 나옴
        cnt += i // cur

    # 만들 수 있는 랜선의 개수가 M 이상이면 1 아니면 0 반환
    return cnt >= m

def maximazation():
    # 정답의 범위는 1 ~ 2^31 - 1
    lo = 1
    hi = 2 ** 31 - 1
    while lo < hi:
        mid = (lo + hi + 1) // 2 # 중간값
        # 결정 문제의 답이 1 이면
        # 정답은 [mid, hi]에 존재
        if(decision(mid)):
            lo = mid

        # 결정 문제의 답이 0 이면
        # 정답은 [lo, mid - 1]에 존재
        else:
            hi = mid - 1
    return lo

print(maximazation())
```

매개 변수 탐색

mid 값을 올림 하는 이유

lo와 hi의 차이가 1 일 때 올림을 하지 않으면

EX) lo = 5, hi = 6, mid = (5 + 6) / 2 = 5

decision(mid)가 1이면 lo = mid

항상 lo = 5, hi = 6, mid = 5로 갱신 됨

올림을 하면 mid = lo = 6으로 갱신되어서 무한루프 X

반대로 최솟값을 구할 때는 mid 값을 내림 해야함

mid = (lo + hi) / 2

매개 변수 탐색

C++

```cpp
int maximazation(){
    // 정답의 범위는 1 ~ N
    long long lo = 1, hi = N;
    while(lo < hi){
        long long mid = (lo + hi + 1) / 2; // 중간값
        // 결정 문제의 답이 1 이면
        // 정답은 [mid, hi]에 존재
        if(decision(mid)) lo = mid;

        // 결정 문제의 답이 0 이면
        // 정답은 [lo, mid - 1]에 존재
        else hi = mid - 1;
    }

    return lo;
}
```

```cpp
int minimazation(){
    // 정답의 범위는 1 ~ N
    long long lo = 1, hi = N;
    while(lo < hi){
        long long mid = (lo + hi) / 2; // 중간값
        // 결정 문제의 답이 1 이면
        // 정답은 [lo, mid]에 존재
        if(decision(mid)) hi = mid;

        // 결정 문제의 답이 0 이면
        // 정답은 [mid + 1, hi]에 존재
        else lo = mid + 1;
    }

    return lo;
}
```

Decision(K) 111111000000 000000111111

매개 변수 탐색

Python

```python
def maximazation():
    # 정답의 범위는 1 ~ N
    lo = 1
    hi = N
    while lo < hi:
        mid = (lo + hi + 1) // 2 # 중간값
        # 결정 문제의 답이 1 이면
        # 정답은 [mid, hi]에 존재
        if(decision(mid)):
            lo = mid

        # 결정 문제의 답이 0 이면
        # 정답은 [lo, mid - 1]에 존재
        else:
            hi = mid - 1
    return lo
```

```python
def minimazation():
    # 정답의 범위는 1 ~ N
    lo = 1
    hi = N
    while lo < hi:
        mid = (lo + hi) // 2 # 중간값
        # 결정 문제의 답이 1 이면
        # 정답은 [lo, mid]에 존재
        if(decision(mid)):
            hi = mid

        # 결정 문제의 답이 0 이면
        # 정답은 [mid + 1, hi]에 존재
        else:
            lo = mid + 1
    return lo
```

Decision(K) 111111000000 000000111111

질문?

휴게소 세우기 / 1477

백준 1477 / https://www.acmicpc.net/problem/1477

문제

다솜이는 유료 고속도로를 가지고 있다. 다솜이는 현재 고속도로에 휴게소를 N개 가지고 있는데, 휴게소의 위치는 고속도로의 시작으로부터 얼만큼 떨어져 있는지로 주어진다. 다솜이는 지금 휴게소를 M개 더 세우려고 한다.

다솜이는 이미 휴게소가 있는 곳에 휴게소를 또 세울 수 없고, 고속도로의 끝에도 휴게소를 세울 수 없다. 휴게소는 정수 위치에만 세울 수 있다.

다솜이는 이 고속도로를 이용할 때, 모든 휴게소를 방문한다. 다솜이는 휴게소를 M개 더 지어서 휴게소가 없는 구간의 길이의 최댓값을 최소로 하려고 한다. (반드시 M개를 모두 지어야 한다.)

예를 들어, 고속도로의 길이가 1000이고, 현재 휴게소가 {200, 701, 800}에 있고, 휴게소를 1개 더 세우려고 한다고 해보자.

일단, 지금 이 고속도로를 타고 달릴 때, 휴게소가 없는 구간의 최댓값은 200~701구간인 501이다. 하지만, 새로운 휴게소를 451구간에 짓게 되면, 최대가 251이 되어서 최소가 된다.

입력

첫째 줄에 현재 휴게소의 개수 N, 더 지으려고 하는 휴게소의 개수 M, 고속도로의 길이 L이 주어진다. 둘째 줄에 현재 휴게소의 위치가 공백을 사이에 두고 주어진다. N = 0인 경우 둘째 줄은 빈 줄이다.

출력

첫째 줄에 M개의 휴게소를 짓고 난 후에 휴게소가 없는 구간의 최댓값의 최솟값을 출력한다.

휴게소 세우기 / 1477

첫째 줄에 M개의 휴게소를 짓고 난 후에 휴게소가 없는 구간의 최댓값의 최솟값을 출력한다.

예제 입력 1 복사

```
6 7 800
622 411 201 555 755 82
```

예제 출력 1 복사

```
70
```

예제 입력 2 복사

```
3 1 1000
200 701 800
```

예제 출력 2 복사

```
251
```

예제 입력 3 복사

```
3 1 1000
300 701 800
```

예제 출력 3 복사

```
300
```

휴게소 세우기 / 1477

최적화 문제

M개의 휴게소를 추가로 지었을 때

휴게소 간의 거리의 최댓값의 최솟값

결정 문제

휴게소 세우기 / 1477

최적화 문제

M개의 휴게소를 추가로 지었을 때
휴게소 간의 거리의 최댓값의 최솟값

결정 문제

휴게소 간의 거리의 최댓값을 K로 만들 때
M개 이하의 휴게소가 추가로 필요한가

휴게소 세우기 / 1477

결정 문제

휴게소 간의 거리의 최댓값을 K로 만들 때
M개 이하의 휴게소가 추가로 필요한가

M개 초과의 휴게소가 필요하면 최댓값을 K로 줄일 수 없음
M개 미만의 휴게소가 필요하면 남은 휴게소를
거리가 1인 휴게소로 배치하면 됨

휴게소 세우기 / 1477

결정 문제

휴게소 간의 거리의 최댓값을 K로 만들 때
M개 이하의 휴게소가 추가로 필요한가

이분 탐색

K가 감소하면 항상 필요한 휴게소 수는 증가함
K가 증가하면 항상 필요한 휴게소 수는 감소함
Decision(K) -> 000001111111
매개 변수 탐색 가능

휴게소 세우기 / 1477

결정 문제

휴게소 간의 거리의 최댓값을 K로 만들 때
M개 이하의 휴게소가 추가로 필요한가

고속도로의 길이가 L이므로 결정 문제의 범위는 1 ~ L
일단 휴게소의 위치들을 정렬해보자

휴게소 세우기 / 1477

예제 입력 1	예제 출력 1
6 7 800 622 411 201 555 755 82	70

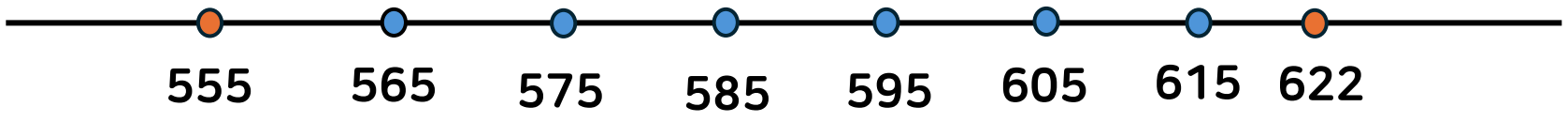
정렬
82 201 411 555 622 755

휴게소 세우기 / 1477

6 7 800 정답
82 201 411 555 622 755 70

휴게소 사이의 거리의 최댓값이 K일 때
각 휴게소 사이에는 (휴게소의 위치 차이 / K) 만큼
추가로 휴게소를 배치 해야 함

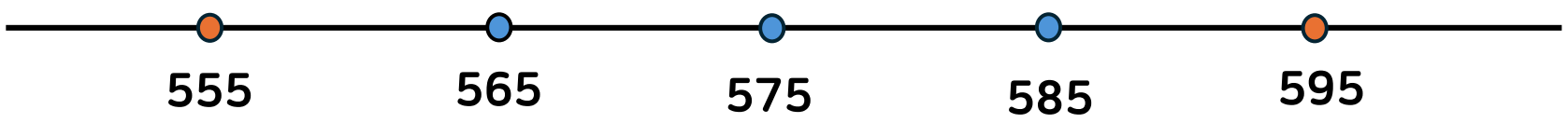
EX) 555 622 -> 차이 67, K = 10 -> 6



휴게소 세우기 / 1477

6 7 800 정답
82 201 411 555 622 755 70

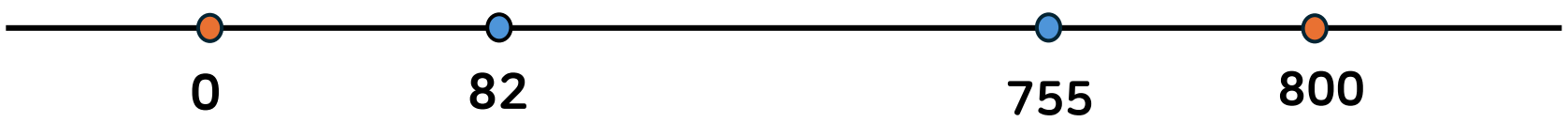
휴게소 간의 거리 차이가 K의 배수일 때는 -1을 해줘야 함
EX) 555 595 -> 차이 40, K = 10 -> 3



휴게소 세우기 / 1477

6 7 800 정답
82 201 411 555 622 755 70

시작점과 휴게소의 거리, 끝점과의 휴게소의 거리도
휴게소가 없는 구간이므로 시작점, 끝점도 휴게소로 지정



휴게소 세우기 / 1477

C++

```
Input:
6 7 800
622 411 201 555 755 82

Expected Output:
70

Received Output:
70
```

```cpp
int main(){
    cin >> n >> m >> l;
    for(int i = 1;i <= n;i++) cin >> a[i];
    a[0] = 0; a[n + 1] = l; // 시작점 0, 끝점 l
    sort(a, a + n + 2); // 정렬

    cout << minimazation();
    return 0;
}
```

```cpp
#include <iostream>
#include <algorithm>
using namespace std;

int a[1010], n, m, l;

bool decision(int cur){
    // 휴게소 사이의 거리의 최댓값을 cur로 만들기 위해
    // 필요한 추가 휴게소의 개수
    int cnt = 0;
    for(int i = 1;i <= n + 1;i++){
        int diff = a[i] - a[i - 1]; // 휴게소의 거리
        cnt += diff / cur; // (차이 / 최댓값) 만큼 추가로 휴게소를 설치
        if(diff % cur == 0) cnt--; // 차이가 최댓값의 배수면 1을 빼줌
    }

    // 설치해야 하는 휴게소의 개수가 M 이하면 1
    // 아니면 0
    return cnt <= m;
}

int minimazation(){
    // 정답의 범위는 1 ~ l
    int lo = 1, hi = l;
    while(lo < hi){
        int mid = (lo + hi) / 2; // 중간값
        // 결정 문제의 답이 1 이면
        // 정답은 [lo, mid]에 존재
        if(decision(mid)) hi = mid;

        // 결정 문제의 답이 0 이면
        // 정답은 [mid + 1, hi]에 존재
        else lo = mid + 1;
    }

    return lo;
}
```

휴게소 세우기 / 1477

Python

```
Input:
6 7 800
622 411 201 555 755 82

Expected Output:
70

Received Output:
70
```

```python
import sys
input = sys.stdin.readline

n, m, l = list(map(int, input().rstrip().split()))
a = list(map(int, input().rstrip().split()))
a.append(0) # 시작점 0
a.append(l) # 끝점 l
a.sort() # 정렬
```

```python
def decision(cur):
    # 휴게소 사이의 거리의 최댓값을 cur로 만들기 위해
    # 필요한 추가 휴게소의 개수
    cnt = 0
    for i in range(1, n + 2):
        diff = a[i] - a[i - 1] # 휴게소의 거리
        cnt += diff // cur # (차이 // 최댓값) 만큼 추가로 휴게소를 설치
        if diff % cur == 0:
            cnt -= 1 # 차이가 최댓값의 배수면 1을 빼줌

    # 설치해야 하는 휴게소의 개수가 M 이하면 1
    # 아니면 0
    return cnt <= m

def minimazation():
    # 정답의 범위는 1 ~ l
    lo = 1
    hi = l
    while(lo < hi):
        mid = (lo + hi) // 2 # 중간값
        # 결정 문제의 답이 1 이면
        # 정답은 [lo, mid]에 존재
        if(decision(mid)):
            hi = mid

        # 결정 문제의 답이 0 이면
        # 정답은 [mid + 1, hi]에 존재
        else:
            lo = mid + 1
    return lo
```

질문?

고생하셨습니다

ABB to BA (Hard) / 32293

백준 32293 / https://www.acmicpc.net/problem/32293

문제

이 문제는 "ABB to BA"의 어려운 버전입니다. 두 버전은 t와 n의 제한을 제외하고 동일합니다.

'A' 와 'B' 만으로 이루어진 문자열 S가 주어집니다. 여러분은 다음 동작을 더 이상 수행할 수 없을 때까지 반복해야 합니다.

- S에서 첫 번째로 부분 문자열 "ABB" 가 등장한 위치를 i라고 할 때, 이 위치의 부분 문자열 "ABB" 를 지우고 "BA" 로 바꿉니다.
- 다시 말해, $S_iS_{i+1}S_{i+2}$가 "ABB" 인 가장 작은 i를 찾아, S_i와 S_{i+1}을 각각 'B' 와 'A' 로 바꾸고 S_{i+2}를 S에서 지웁니다.
- S에 "ABB" 가 부분 문자열로 등장하지 않는다면 동작을 수행할 수 없습니다.

반복이 끝난 후 S의 내용을 출력하는 프로그램을 작성해 주세요.

입력

각 입력은 여러 개의 테스트 케이스로 구성됩니다. 입력의 첫 번째 줄에 테스트 케이스의 개수 t가 주어집니다. ($1 \le t \le 5 \cdot 10^4$)

이후 테스트 케이스의 정보가 주어지며, 각 테스트 케이스의 입력은 다음과 같이 두 줄로 구성됩니다.

- 첫 번째 줄에 S의 길이를 나타내는 정수 n이 주어집니다. ($1 \le n \le 5 \cdot 10^5$)
- 두 번째 줄에 길이 n의 문자열 S가 주어집니다. (S_i는 모두 'A' 또는 'B')

모든 테스트 케이스에 대한 n의 합이 $5 \cdot 10^5$을 초과하지 않습니다.

출력

각 테스트 케이스에 대해 반복이 끝난 후 S의 내용을 한 줄에 출력합니다.

ABB to BA (Hard) / 32293

문자열을 차례대로 보면서 ABB가 있을 때 마다 BA로 바꿔 줌

예제 입력 1 복사

```
3
3
ABB
9
ABABABBBB
12
AAAAAABBBBBB
```

예제 출력 1 복사

```
BA
BAABA
AAAABABA
```

두 번째 테스트 케이스에서 S가 바뀌는 과정은 다음과 같습니다.

- "ABABABBBB" → "ABAB**BA**BB" → "AB**BA**ABB" → "**BA**ABB" → "BAA**BA**"

세 번째 테스트 케이스에서 S가 바뀌는 과정은 다음과 같습니다.

- "AAAAAABBBBBB" → "AAAAA**BA**BBBB" → "AAAAAB**BA**BB" → "AAAA**BA**ABB" → "AAAABA**BA**"

ABB to BA (Hard) / 32293

문자열을 차례대로 보면서 ABB가 있을 때 마다 BA로 바꿔 줌

예제 입력 1 복사

```
3
3
ABB
9
ABABABBBB
12
AAAAAABBBBBB
```

예제 출력 1 복사

```
BA
BAABA
AAAABABA
```

두 번째 테스트 케이스에서 S가 바뀌는 과정은 다음과 같습니다.

- "ABABABBBB" → "ABAB**BA**BB" → "AB**BA**ABB" → "**BA**ABB" → "BAA**BA**"

세 번째 테스트 케이스에서 S가 바뀌는 과정은 다음과 같습니다.

- "AAAAAABBBBBB" → "AAAAA**BA**BBBB" → "AAAAAB**BA**BB" → "AAAA**BA**ABB" → "AAAABA**BA**"

ABB to BA (Hard) / 32293

간단하게 문자열을 계속 순회하면서
ABB가 있을 때마다 BA로 바꿔준다면?

문자열은 중간 삽입 / 삭제가 O(N)
AAAABBBBBBB
AAABABBBBB
AAABBAABB
AABAABA

A,B가 연속으로 나오는 문자열에 대해서는 O(N)번 연산함 / O(N^2)

ABB to BA (Hard) / 32293

그러면 문자열을 한번만 순회하는 방법은 없을까?

AAAABBBBBBB
AAABABBBBB
AAABABBBBB
AAABBAABB

ABB 를 BA로 대체하는 과정에서
B 두개를 1개로 압축하고 A를 B 뒤로 옮긴다고 생각해 보자

ABB to BA (Hard) / 32293

AAAABBBBBBB
AAABABBBBB
AAABABBBBB
AAABBAABB

스택에 차례대로 삽입하면서
마지막 데이터가 ABB가 아닐 때 까지
ABB를 B로 변환, B로 변환한 만큼 스택에 A 추가

ABB to BA (Hard) / 32293

ABB가 되기 위해서는 B가 들어와야 함

하지만 차례대로 스택에 넣기 때문에
ABB가 BA로 변환 될 때는 A 뒤에 B가 없음
-> A는 독립적으로 생각

ABB가 사라질 때 까지 B로 변환한 뒤에
그 뒤에 변환한 만큼 A를 추가해도 괜찮음

ABB to BA (Hard) / 32293

시간 복잡도?

각 연산은 O(1) / 스택의 pop(), push() 연산이 O(1)
각 연산은 ABB -> BA 즉 길이가 1 줄어 들음

문자열의 총 길이가 N 이기 때문에
N번 연산하면 길이는 0이 됨, 즉 연산의 상한 값은 N

즉 시간 복잡도는 O(N) / 각 연산 당 O(1) * 상한 N

ABB to BA (Hard) / 32293

C++

스택의 값 3개를 한번에
확인 해야 하기 때문에

스택 대신 벡터 사용

```cpp
#include <iostream>
#include <vector>
using namespace std;
#define fastio cin.tie(0), cout.tie(0), ios::sync_with_stdio(0);

vector <char> st;
bool chk(){
    int sz = st.size();
    if(sz < 3) return 0;
    if(st[sz - 1] != 'B') return 0;
    if(st[sz - 2] != 'B') return 0;
    if(st[sz - 3] != 'A') return 0;

    return 1;
}

int main(){
    fastio;

    int t; cin >> t;
    while(t--){
        int n; string s;
        cin >> n >> s;
        st.clear(); // 스택 초기화

        for(int i = 0;i < n;i++){
            st.push_back(s[i]);
            int cnt = 0; // 현재 연산 횟수

            while(chk()){ // 스택의 마지막 값이 ABB인지 판별
                for(int i = 0;i < 3;i++) st.pop_back(); // ABB 제거
                cnt++; st.push_back('B'); // ABB를 B로 변환 후 연산 횟수 1 증가
            }

            for(int j = 1;j <= cnt;j++) st.push_back('A'); // 연산 횟수 만큼 A 삽입
        }

        for(int i = 0;i < st.size();i++) cout << st[i];
        cout << "\n";
    }
}
```

ABB to BA (Hard) / 32293

Python

```python
import sys
input = sys.stdin.readline

t = int(input().rstrip())
for _ in range(t):
    n = int(input().rstrip())
    s = input().rstrip()
    st = []

    for i in s:
        st.append(i)
        cnt = 0 # 현재 연산 횟수

        # 스택의 마지막 값이 ABB 이면
        while len(st) >= 3 and st[-3] == 'A' and st[-2] == 'B' and st[-1] == 'B':
            for _ in range(3):
                st.pop() # ABB 제거

            cnt += 1 # 연산 횟수 1 증가
            st.append('B') # B 추가

        for _ in range(cnt):
            st.append('A') # 연산 횟수만큼 A 추가

    print(*st, sep = '')
```

질문?

오아시스 재결합 / 3015

백준 3015 / https://www.acmicpc.net/problem/3015

문제

오아시스의 재결합 공연에 N명이 한 줄로 서서 기다리고 있다.

이 역사적인 순간을 맞이하기 위해 줄에서 기다리고 있던 백준이는 갑자기 자기가 볼 수 있는 사람의 수가 궁금해졌다.

두 사람 A와 B가 서로 볼 수 있으려면, 두 사람 사이에 A 또는 B보다 키가 큰 사람이 없어야 한다.

줄에 서 있는 사람의 키가 주어졌을 때, 서로 볼 수 있는 쌍의 수를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 줄에서 기다리고 있는 사람의 수 N이 주어진다. ($1 \le N \le 500,000$)

둘째 줄부터 N개의 줄에는 각 사람의 키가 나노미터 단위로 주어진다. 모든 사람의 키는 2^{31} 나노미터 보다 작다.

사람들이 서 있는 순서대로 입력이 주어진다.

출력

서로 볼 수 있는 쌍의 수를 출력한다.

오아시스 재결합 / 3015

두 인덱스를 선택 했을 때 그 사이에
그 두 수보다 큰 수가 없는 쌍의 개수

예제 입력 1 복사

```
7
2
4
1
2
2
5
1
```

예제 출력 1 복사

```
10
```

오아시스 재결합 / 3015

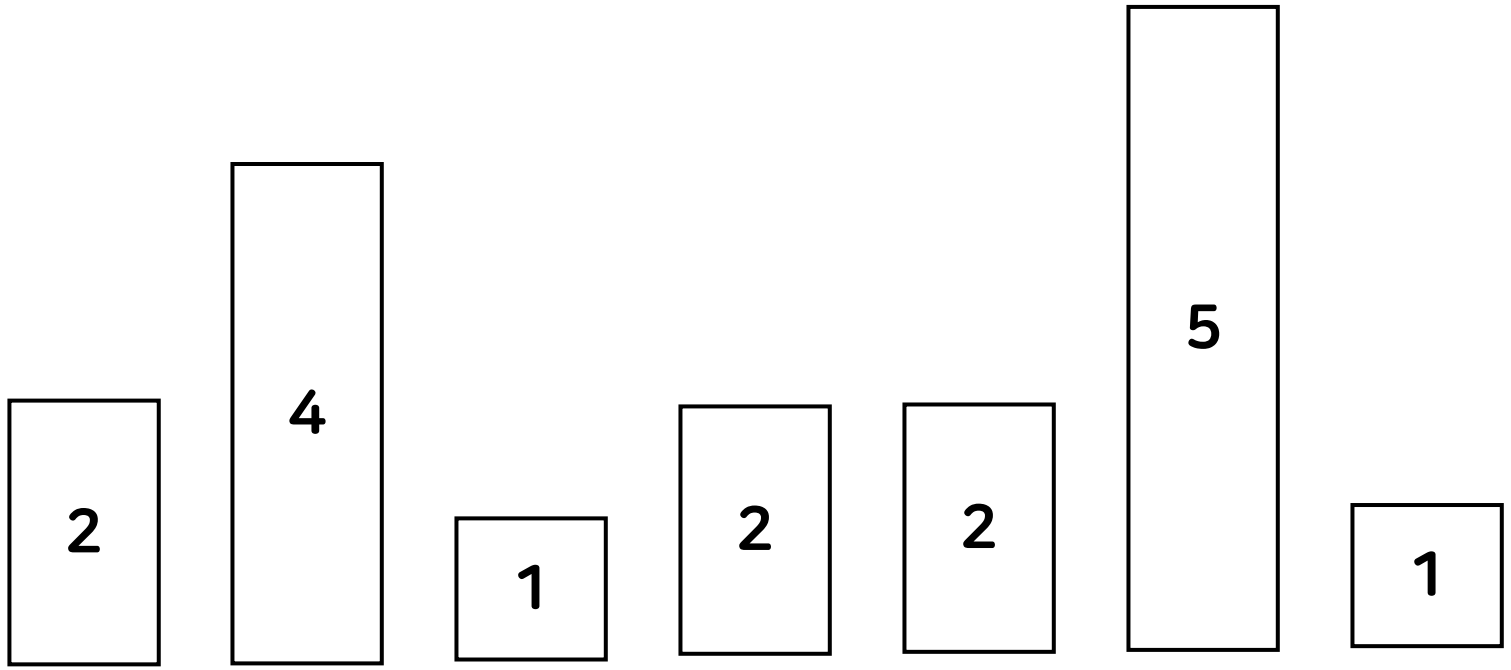
예제 입력 1

7
2 4 1 2 2 5 1

예제 출력 1

10

정답 0



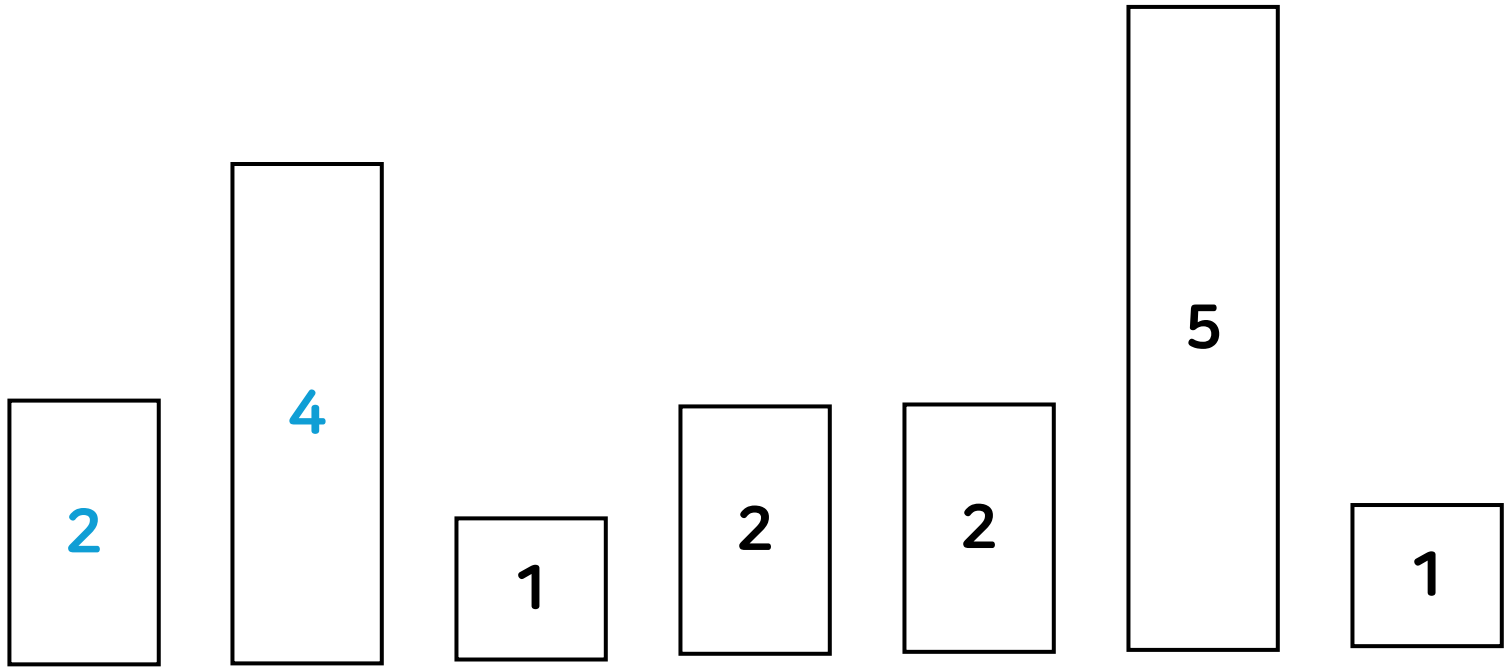
오아시스 재결합 / 3015

예제 입력 1　　**예제 출력 1**

7

2 4 1 2 2 5 1　　10

정답 1



오아시스 재결합 / 3015

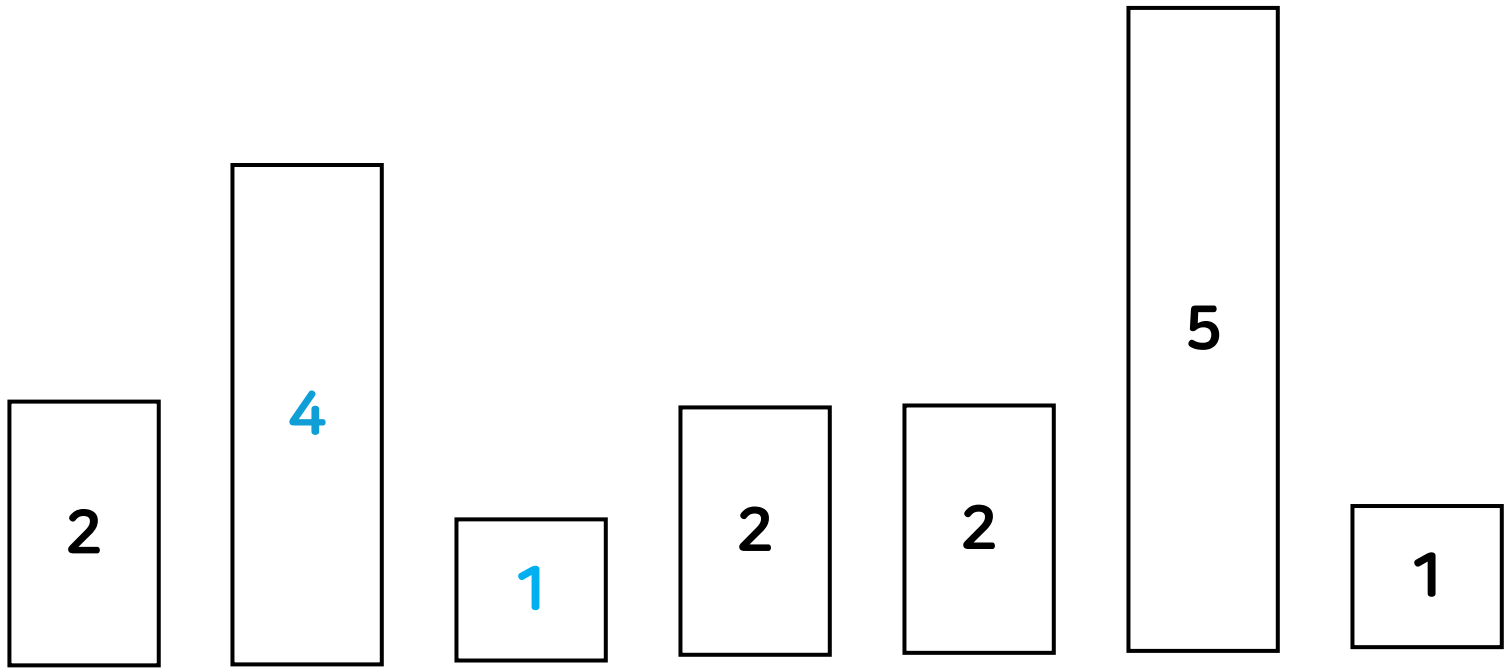
예제 입력 1

7
2 4 1 2 2 5 1

예제 출력 1

10

정답 2



오아시스 재결합 / 3015

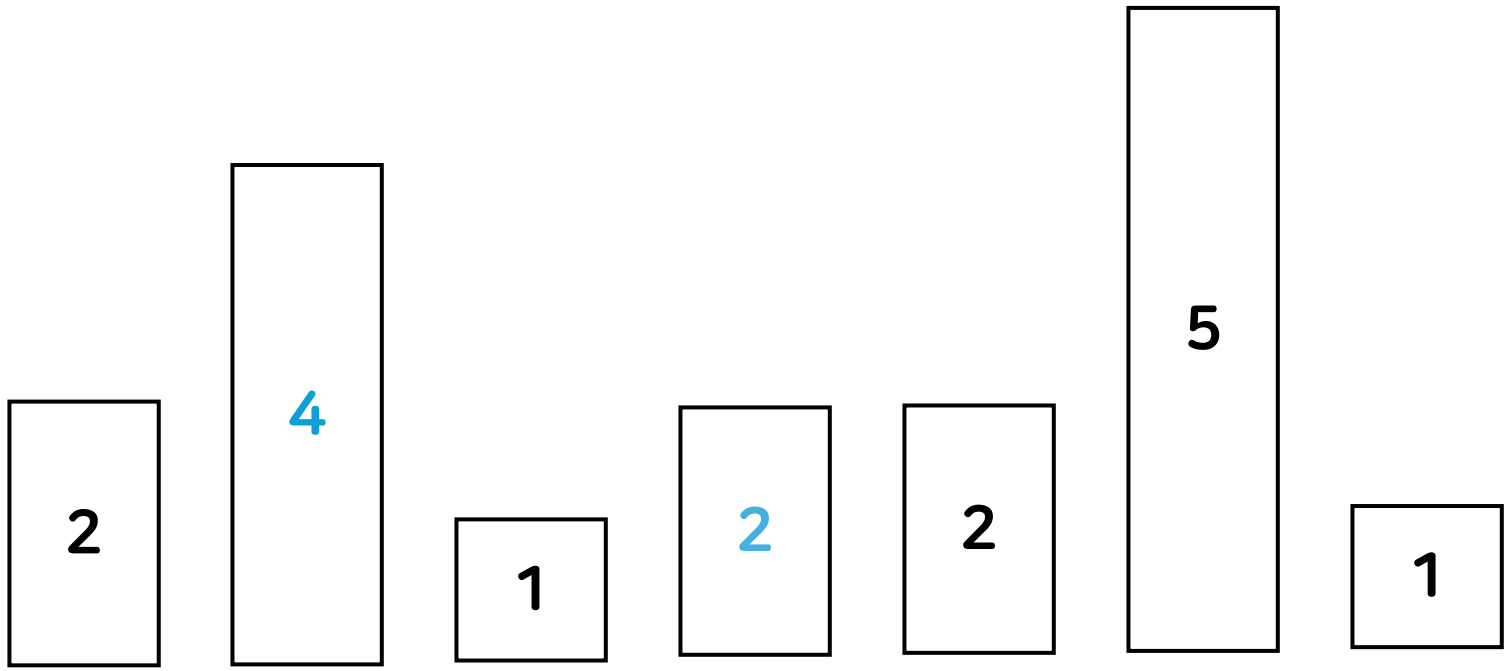
예제 입력 1

7
2 4 1 2 2 5 1

예제 출력 1

10

정답 3



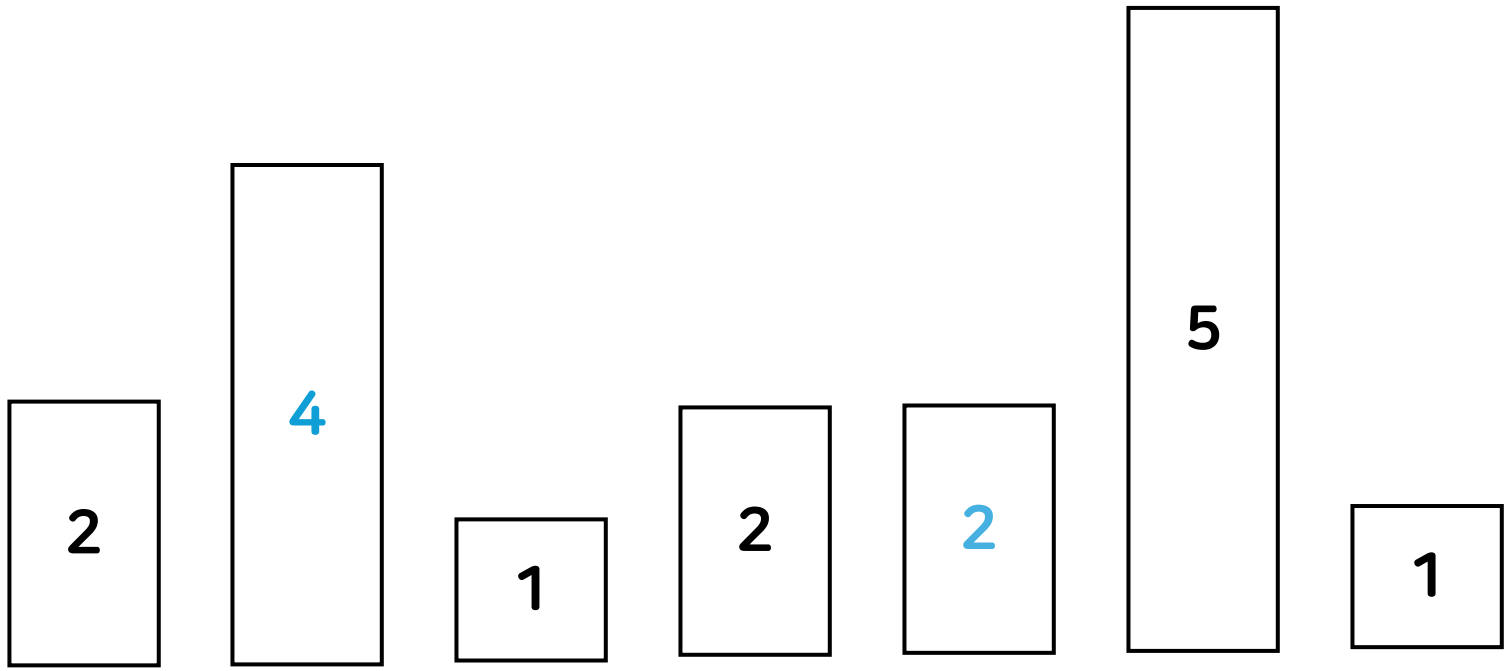
오아시스 재결합 / 3015

예제 입력 1 **예제 출력 1**

7

2 4 1 2 2 5 1 10

정답 4



오아시스 재결합 / 3015

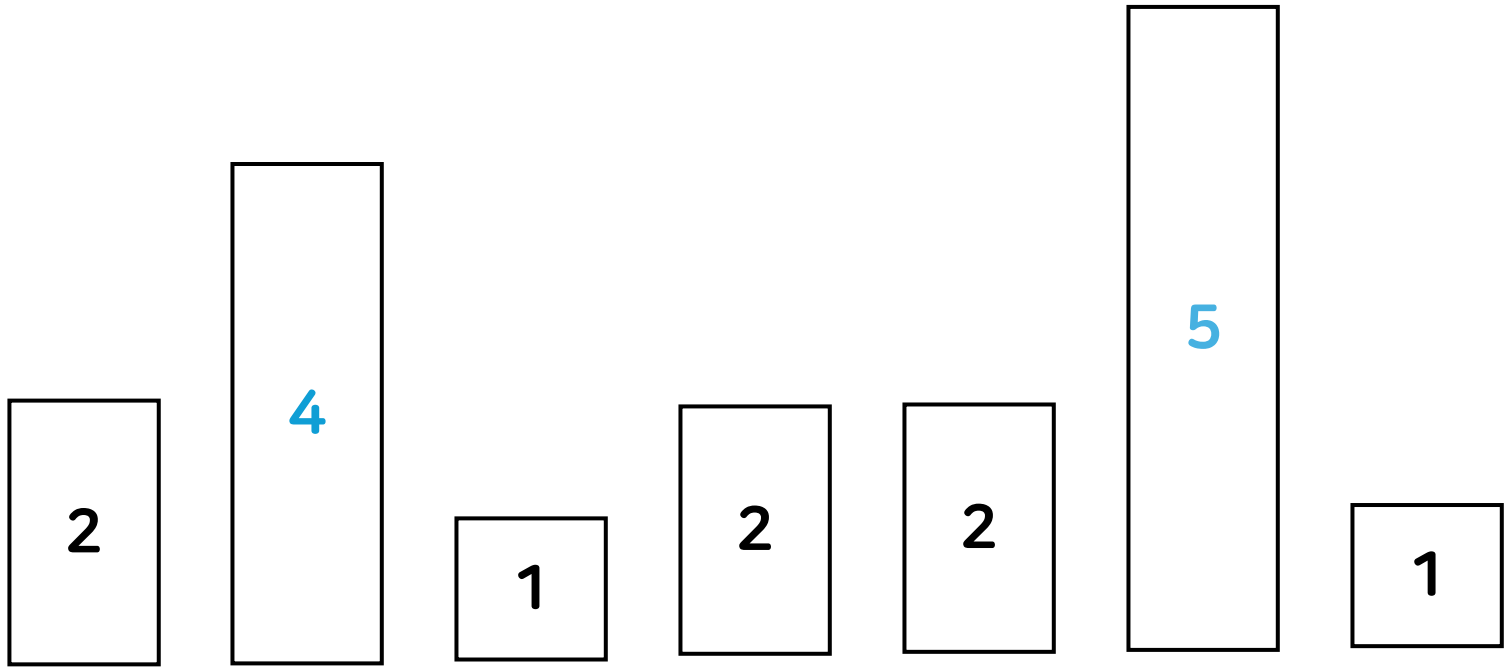
예제 입력 1

7
2 4 1 2 2 5 1

예제 출력 1

10

정답 5



오아시스 재결합 / 3015

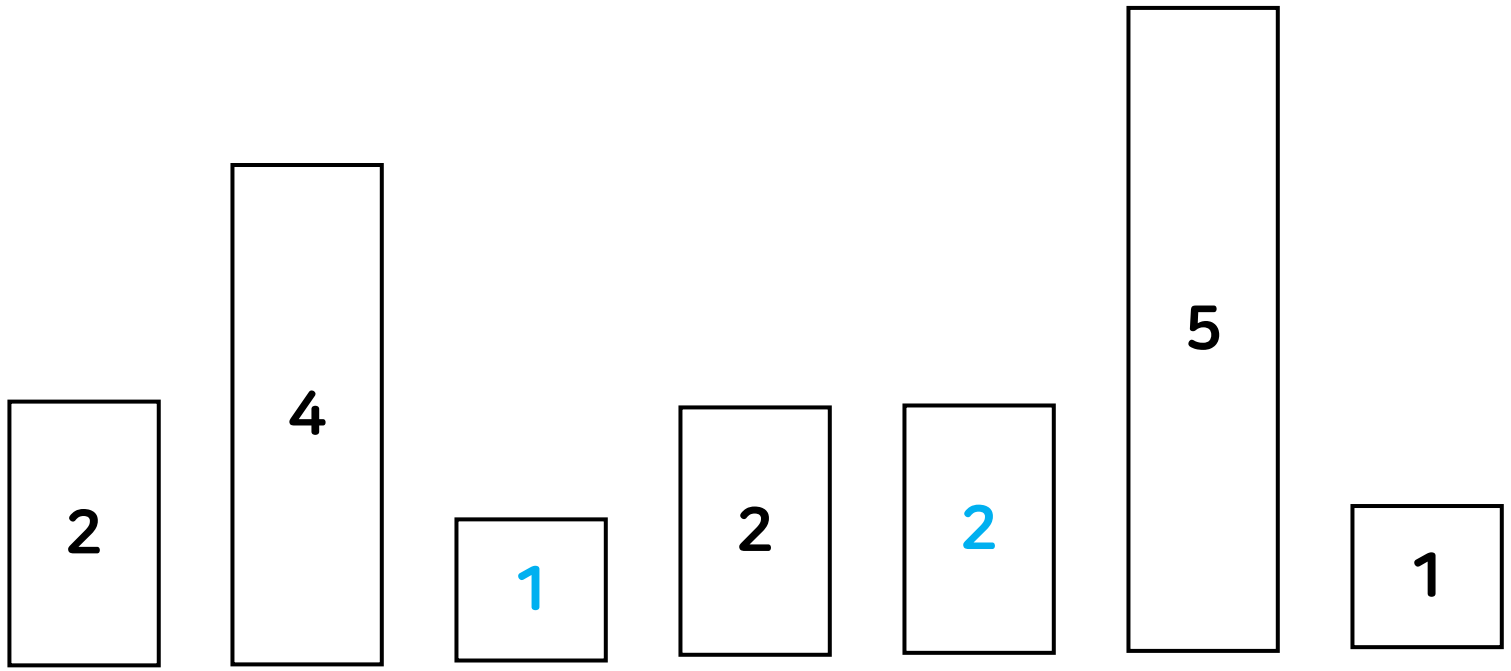
예제 입력 1

7
2 4 1 2 2 5 1

예제 출력 1

10

정답 6



오아시스 재결합 / 3015

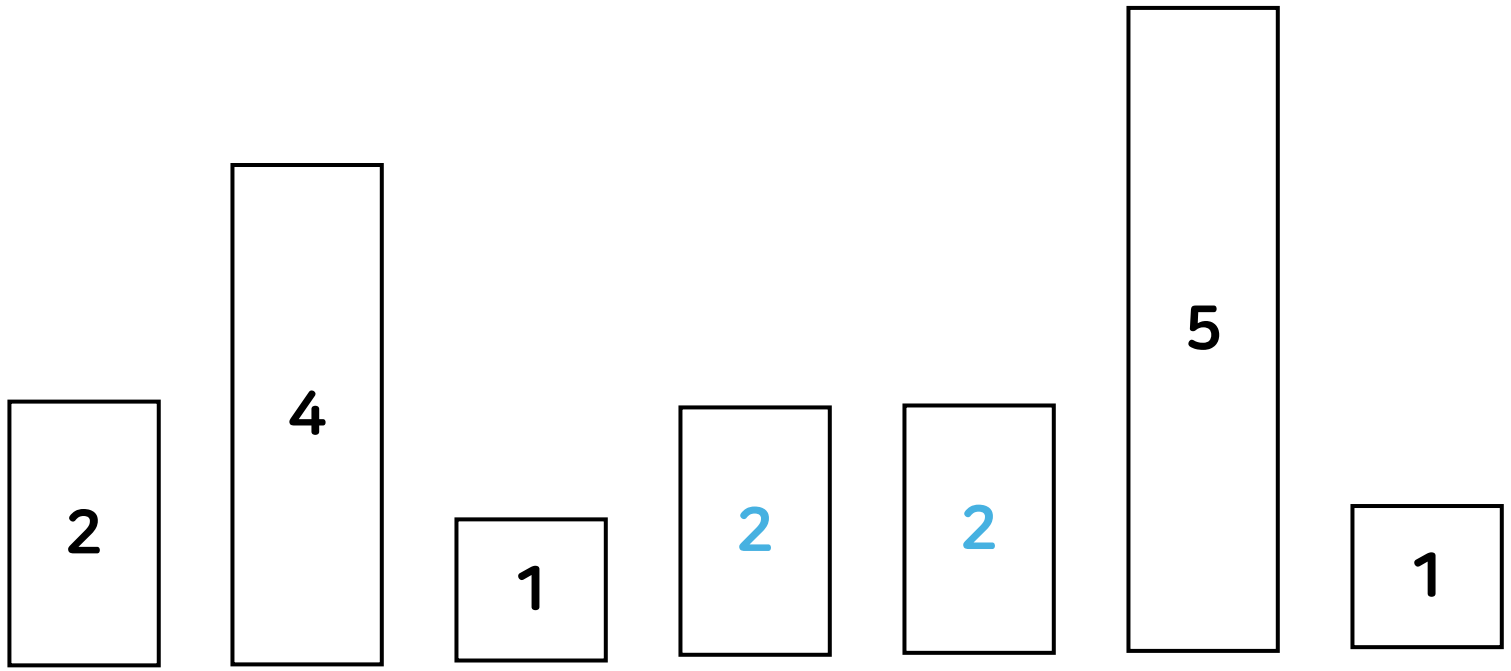
예제 입력 1

7
2 4 1 2 2 5 1

예제 출력 1

10

정답 7



오아시스 재결합 / 3015

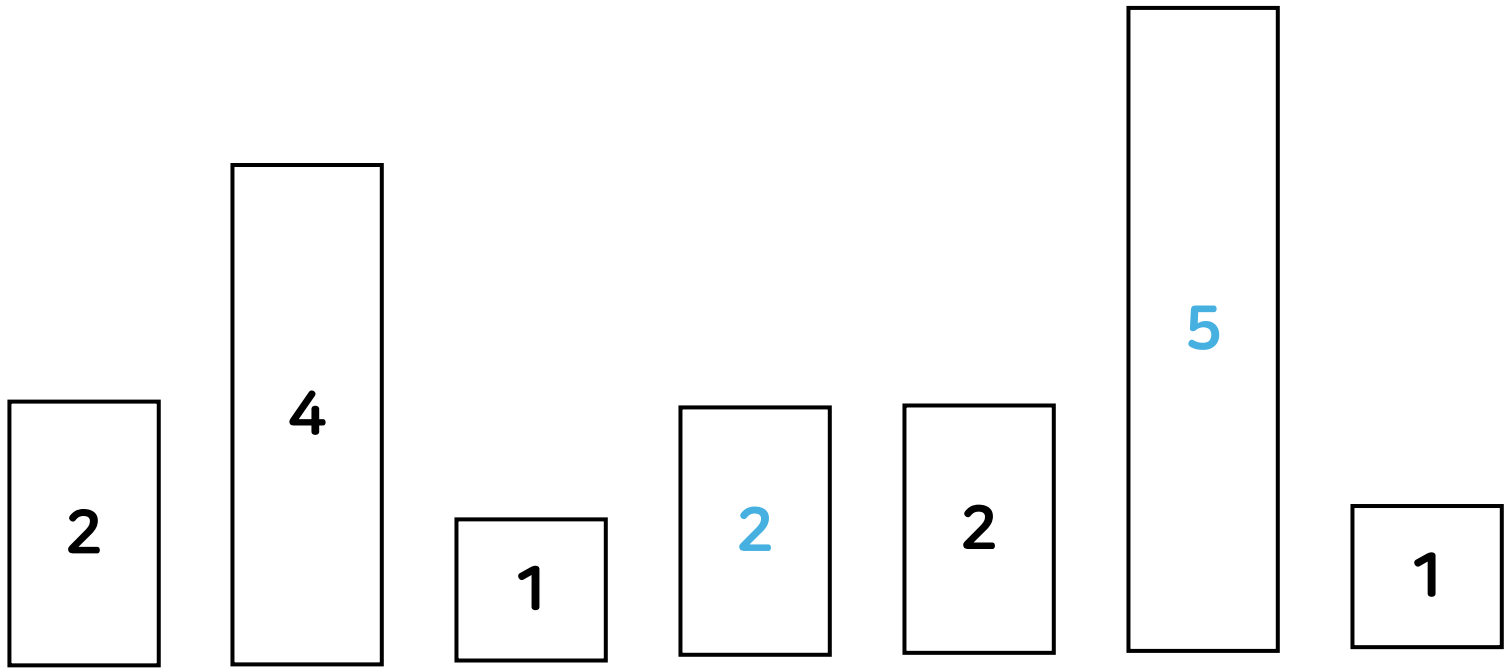
예제 입력 1

7
2 4 1 2 2 5 1

예제 출력 1

10

정답 8



오아시스 재결합 / 3015

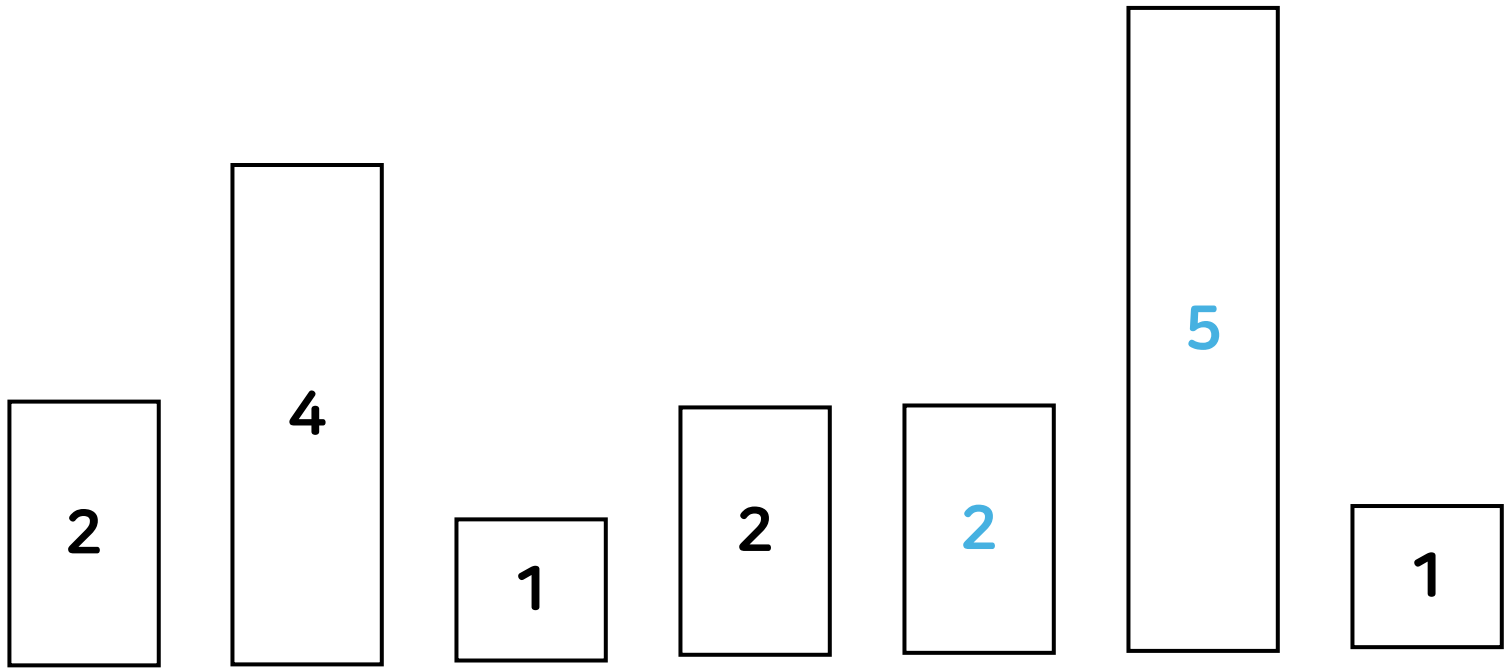
예제 입력 1

7
2 4 1 2 2 5 1

예제 출력 1

10

정답 9



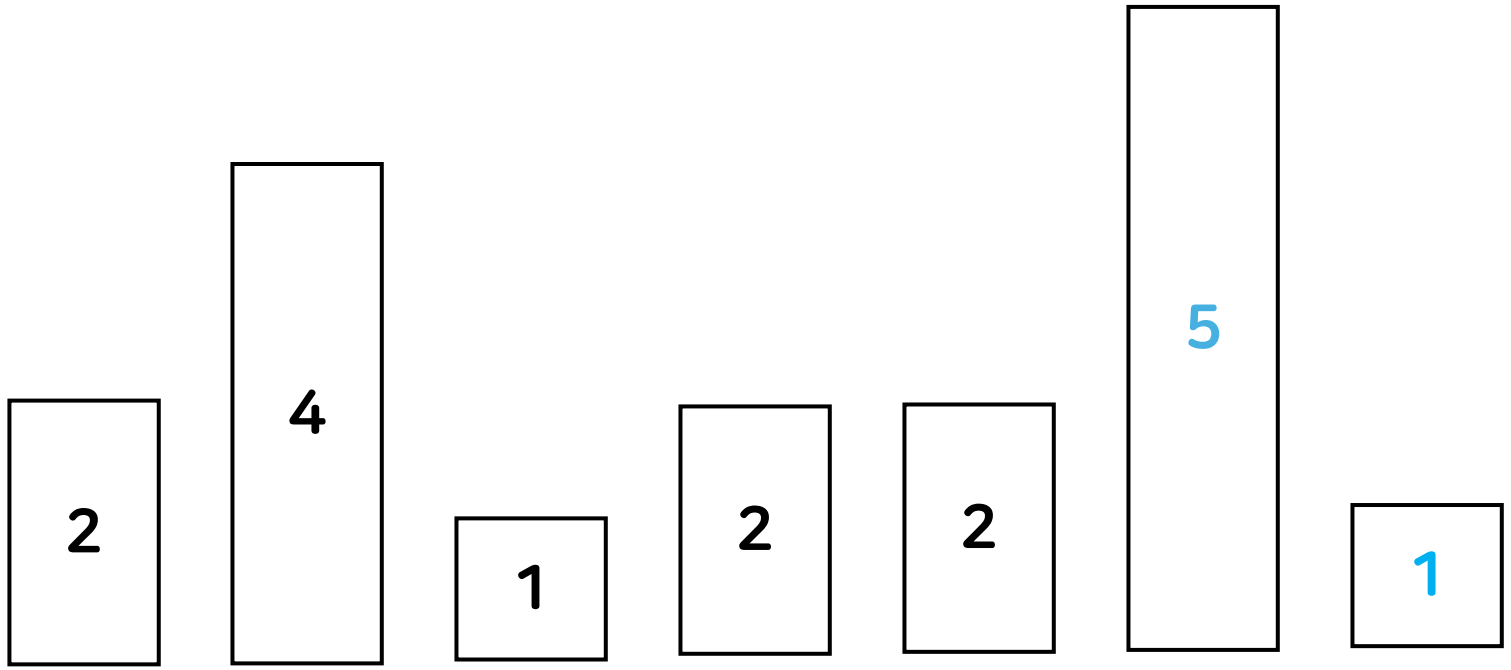
오아시스 재결합 / 3015

예제 입력 1 **예제 출력 1**

7

2 4 1 2 2 5 1 10

정답 10



오아시스 재결합 / 3015

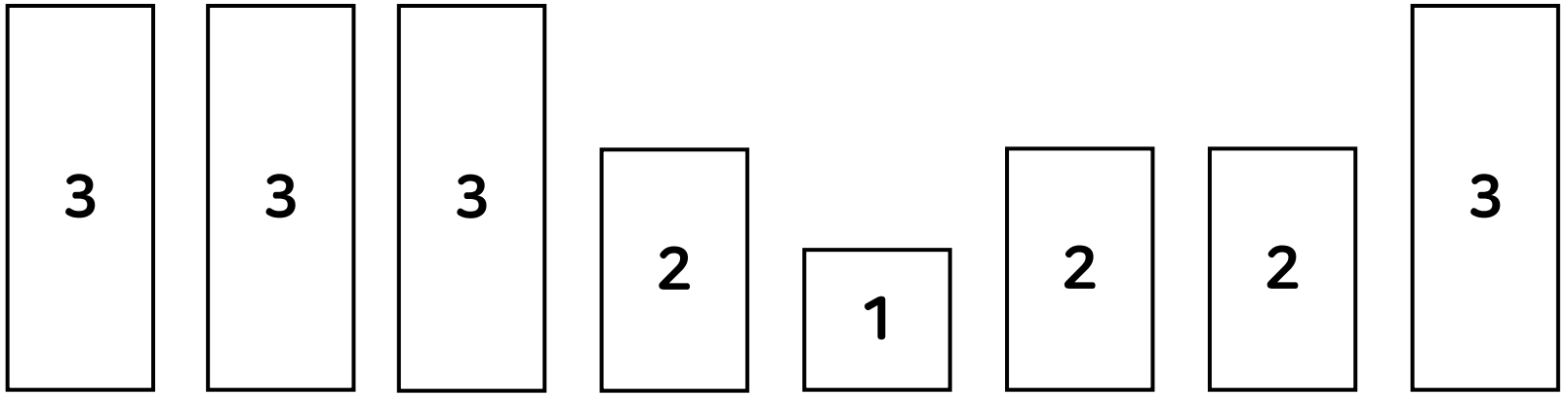
각 인덱스에 대해서 쌍을 구하면 O(N^2)
정답이 될 수 있는 쌍만 볼 수는 없을까?

오큰수를 풀었을 때처럼
스택에 정답의 쌍이 될 수 있는 수들만 관리 해주자

오아시스 재결합 / 3015

8 스택

3 3 3 2 1 2 2 3 정답 0



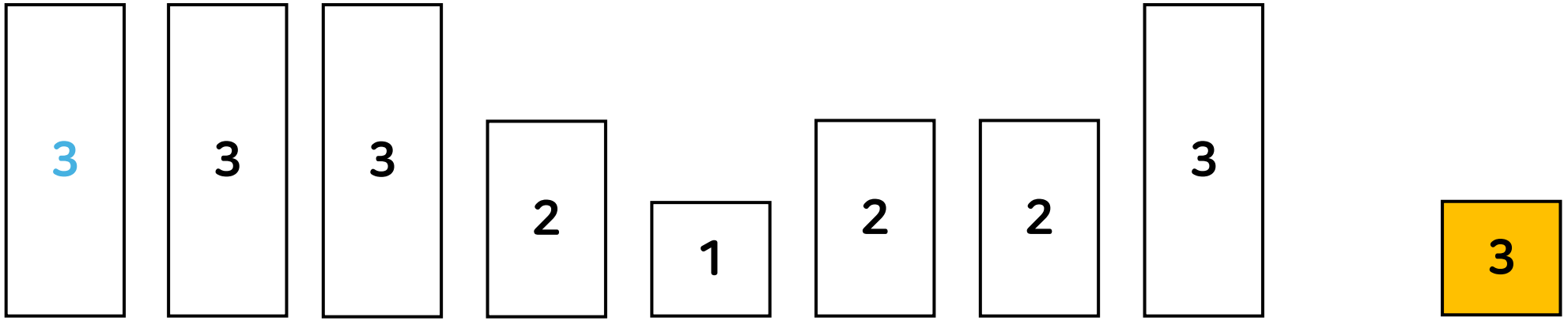
오아시스 재결합 / 3015

8

3 3 3 2 1 2 2 3

정답 0

스택

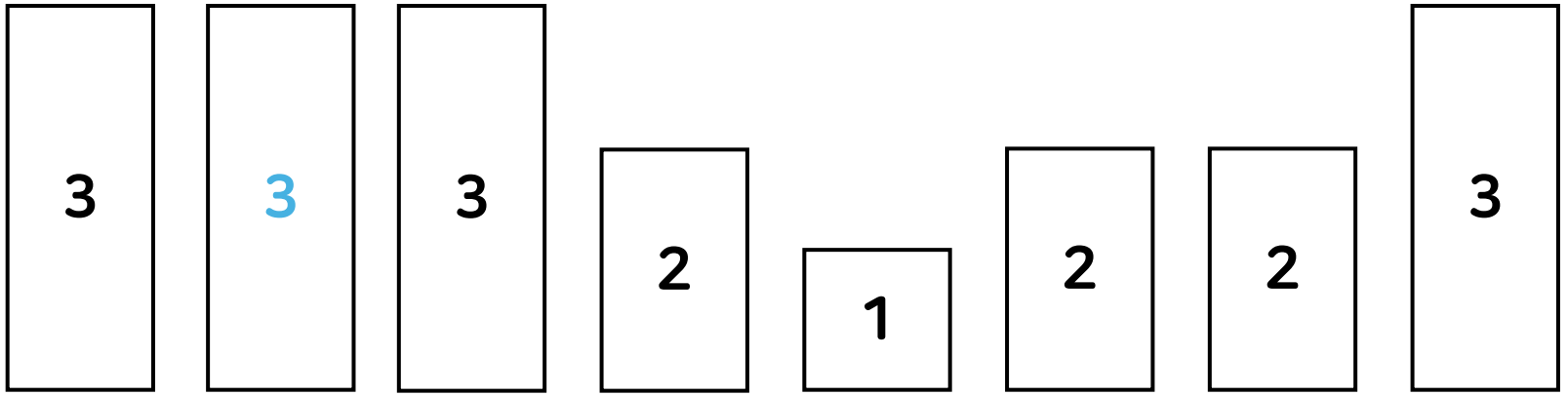


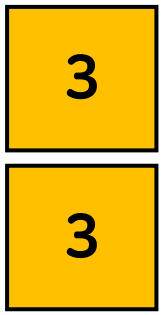
오아시스 재결합 / 3015

8 스택

3 3 3 2 1 2 2 3 정답 1

같은 수들은 만날 수 있으므로 정답을 1 증가 시킨다



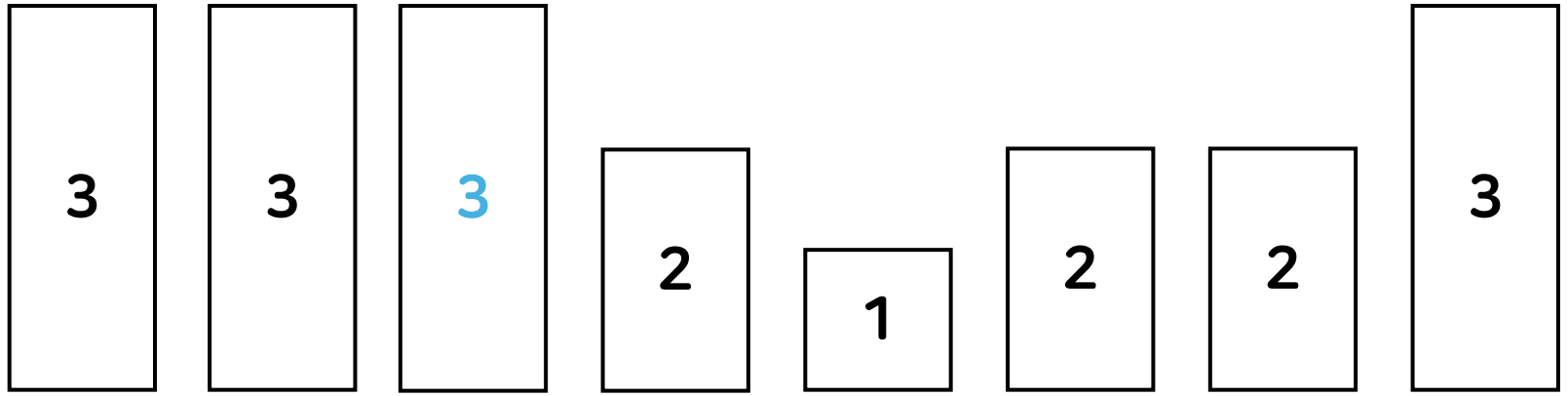


오아시스 재결합 / 3015

8 스택

3 3 3 2 1 2 2 3 정답 3

같은 수가 스택에 쌓였을 때는
그 전 스택의 같은 수의 개수 만큼 더해줘야함

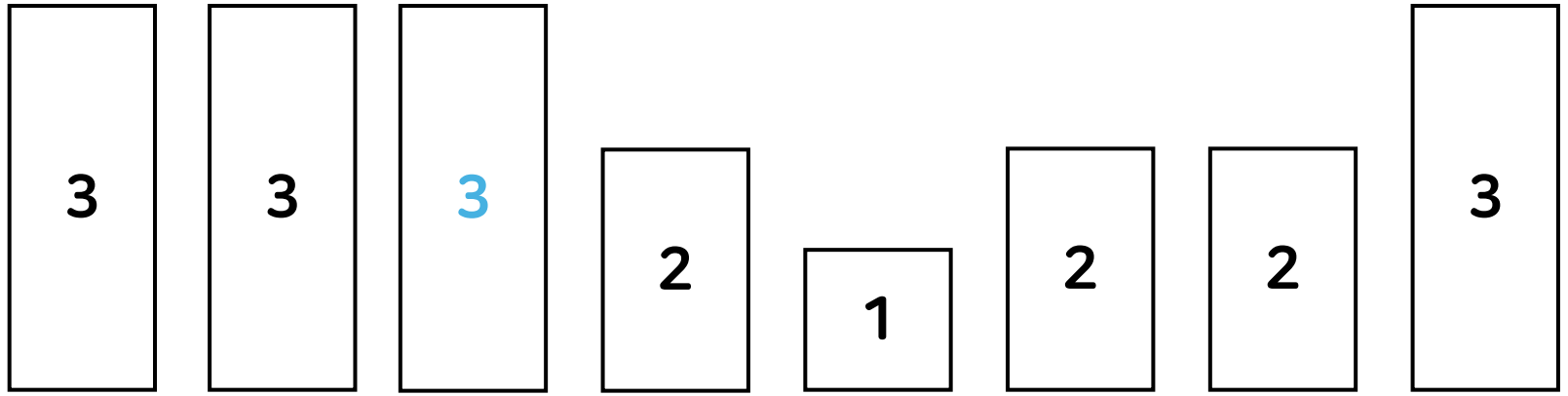


오아시스 재결합 / 3015

8 스택

3 3 3 2 1 2 2 3 정답 3

하지만 스택에는 값만 저장 되어 있고
값의 개수는 저장 되어 있지 않음

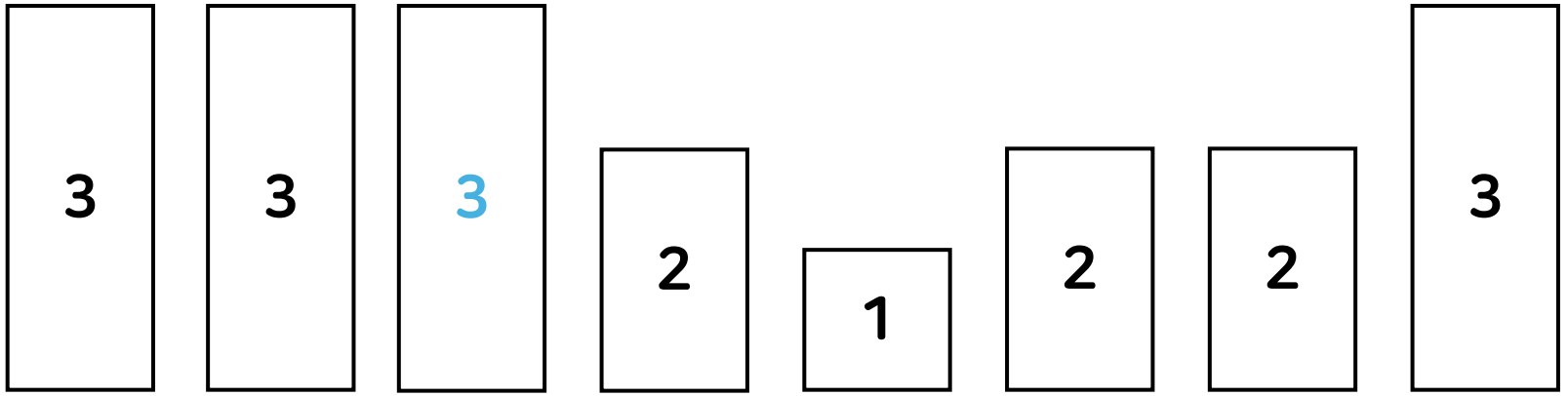


오아시스 재결합 / 3015

8 스택

3 3 3 2 1 2 2 3 정답 3

스택의 값을 크기와 개수의 쌍으로 관리하면 됨

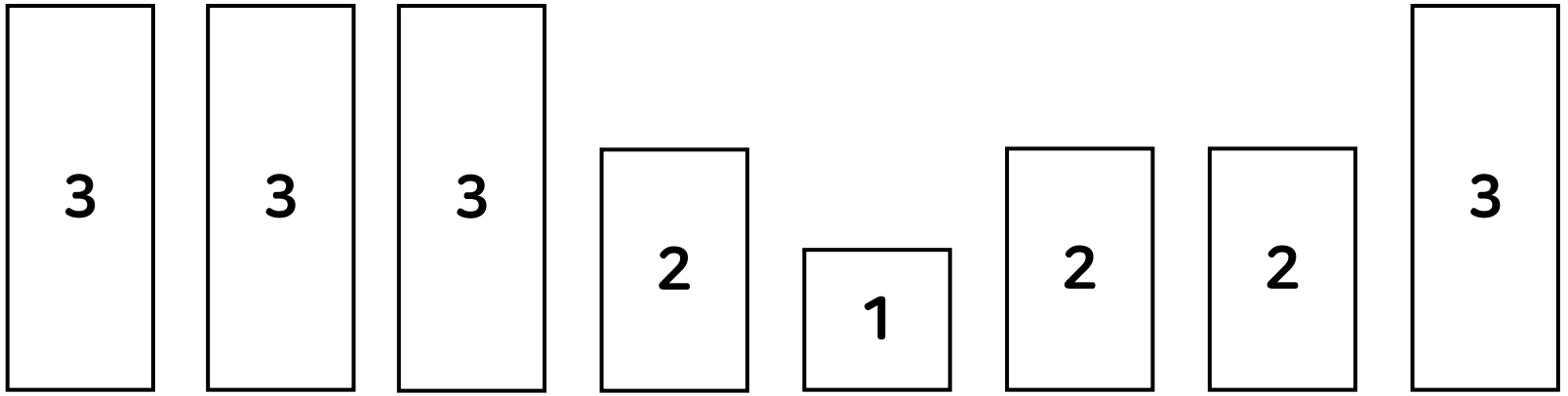


오아시스 재결합 / 3015

8 스택

3 3 3 2 1 2 2 3 정답 0

스택에 크기/개수 쌍으로 넣어보자

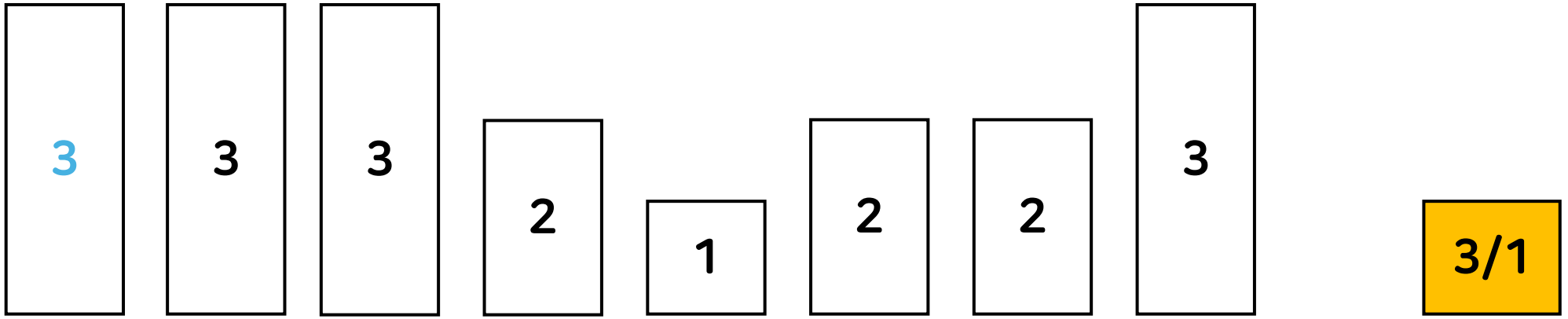


오아시스 재결합 / 3015

8 스택

3 3 3 2 1 2 2 3 정답 0

스택에 크기/개수 쌍으로 넣어보자

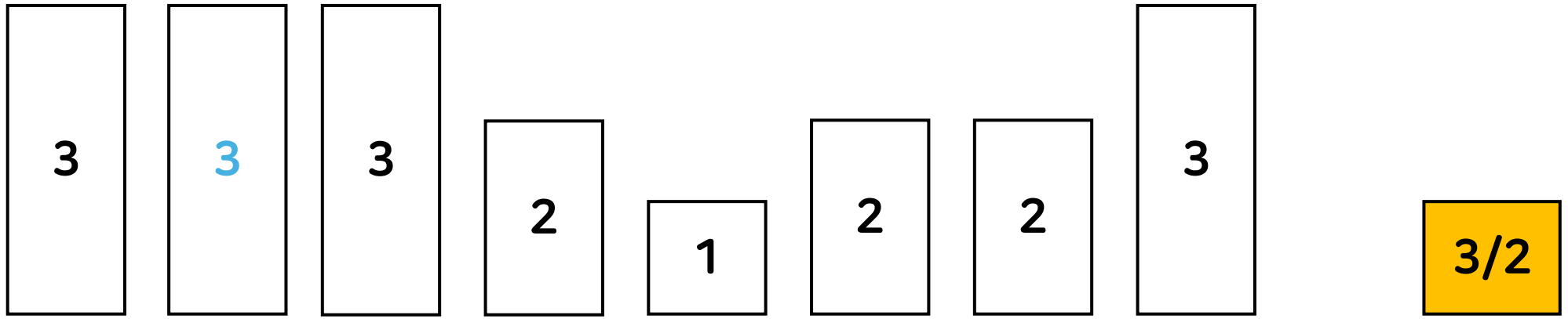


오아시스 재결합 / 3015

8 스택

3 3 3 2 1 2 2 3 정답 1

현재 스택의 크기와 같으므로
정답에 스택의 개수 추가

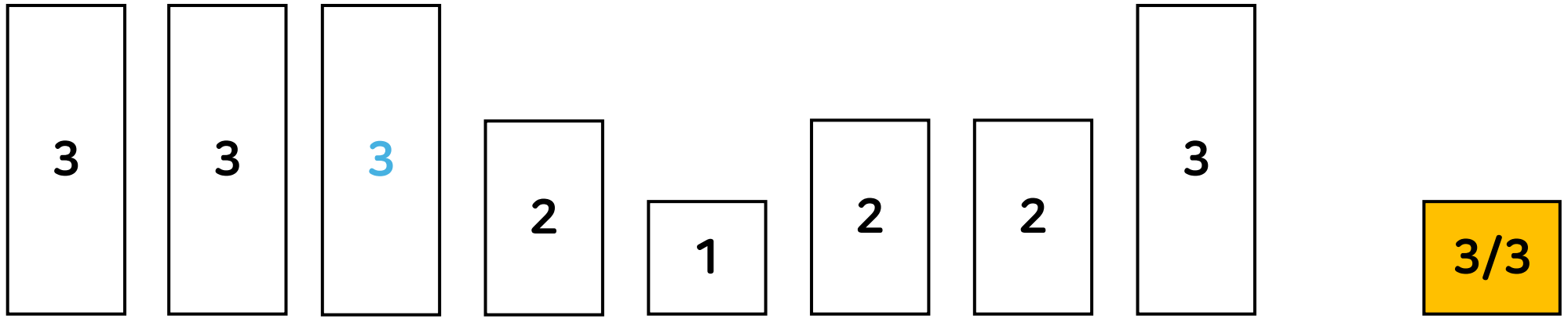


오아시스 재결합 / 3015

8 스택

3 3 3 2 1 2 2 3 정답 3

현재 스택의 크기와 같으므로
정답에 스택의 개수 추가

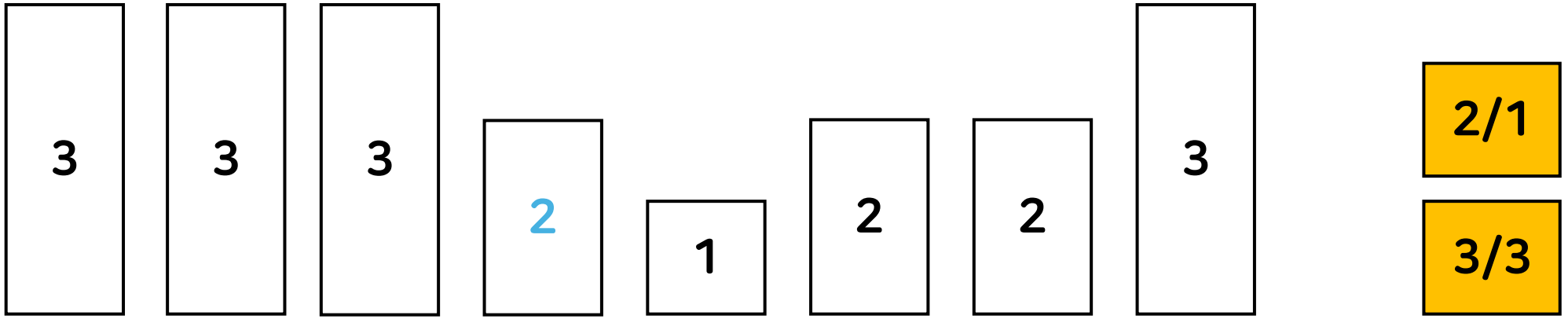


오아시스 재결합 / 3015

8 스택

3 3 3 2 1 2 2 3 정답 4

크기가 작은 수가 들어 왔을 때
스택에 값이 있으면 그 전 값 한 개와 만날 수 있음



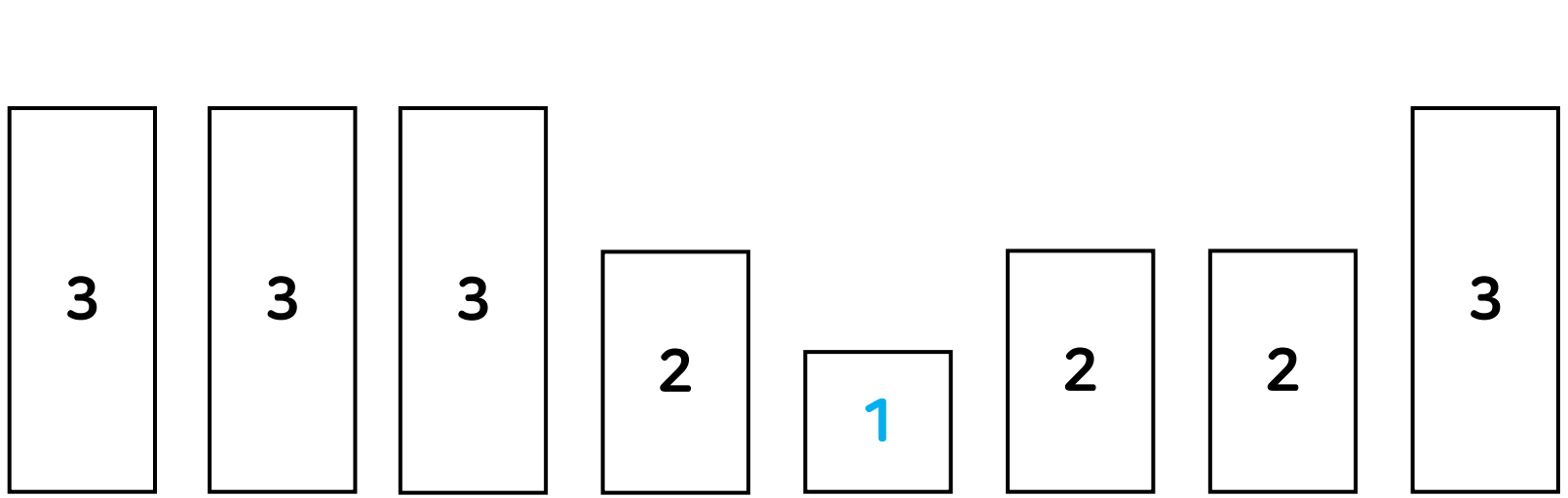
오아시스 재결합 / 3015

8

3 3 3 2 1 2 2 3

정답 5

스택



오아시스 재결합 / 3015

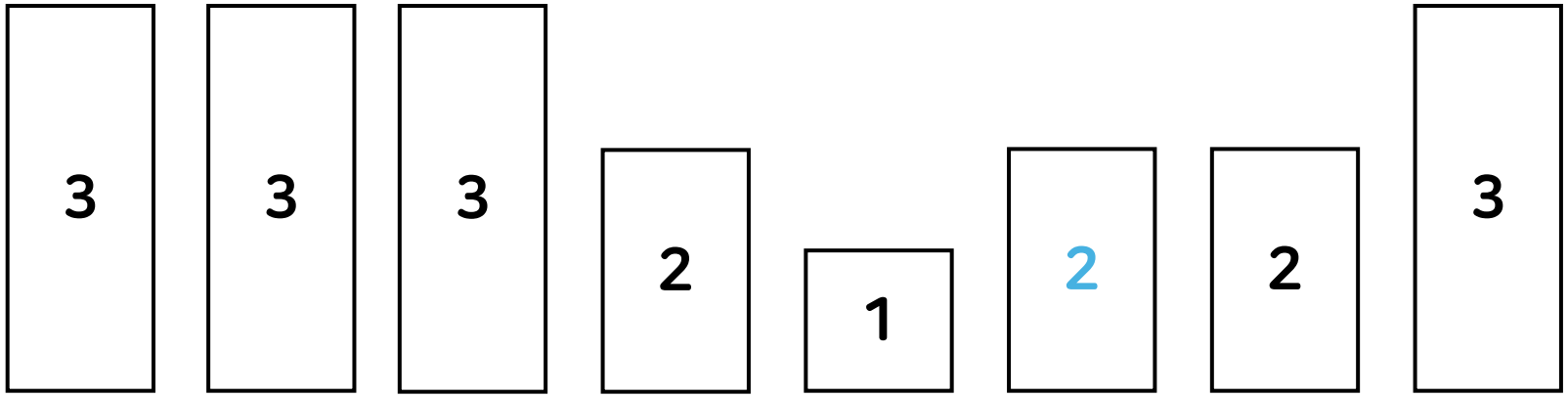
8

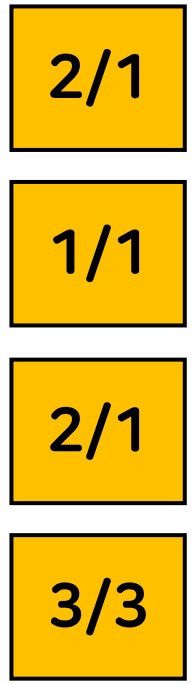
3 3 3 2 1 2 2 3　　정답 6

큰 수가 들어오면 큰 수는 작은 수와 모두 만날 수 있음

작은 수의 개수만큼 정답에 더함

스택





오아시스 재결합 / 3015

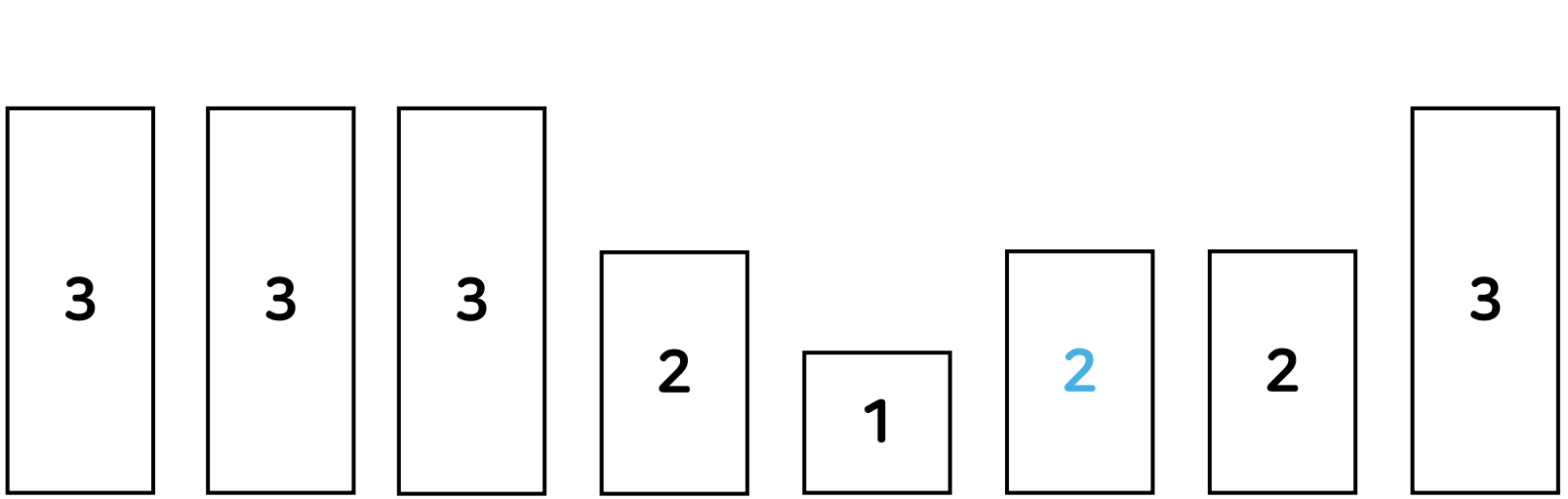
8

3 3 3 2 1 2 2 3

정답 6

또한 그 이후로는 큰 수에 막혀 작은 수는
더 이상 정답이 되는 쌍을 이룰 수 없음
-> 스택에서 삭제

스택



오아시스 재결합 / 3015

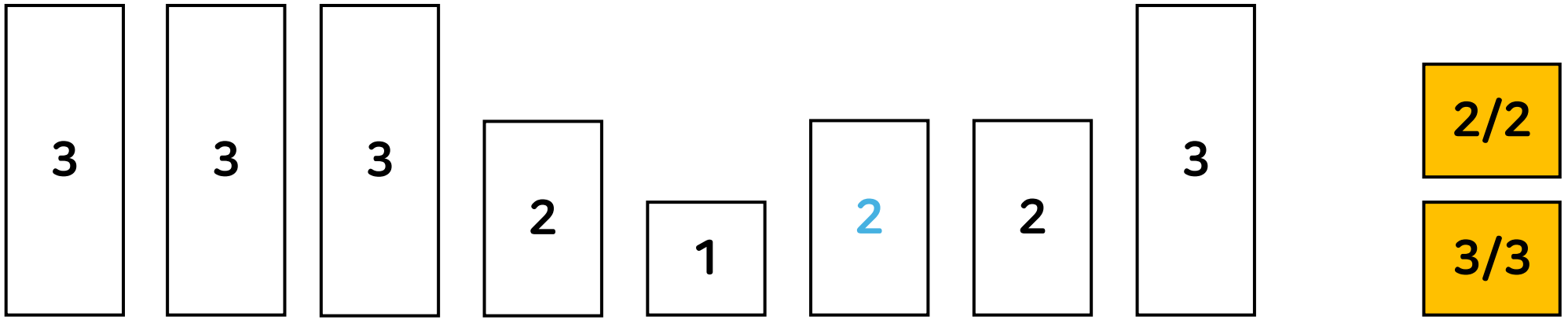
8 스택

3 3 3 2 1 2 2 3 정답 7

같은 수가 스택에 연속으로 존재하면 합쳐줘야 함

또한 같은 수들은 만날 수 있으므로

정답에 개수 추가



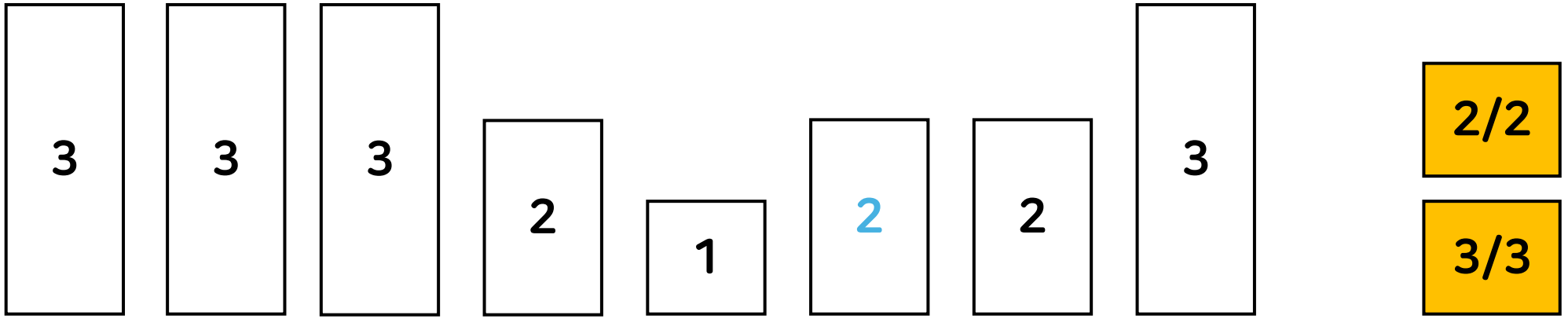
오아시스 재결합 / 3015

8 스택

3 3 3 2 1 2 2 3 정답 8

스택에 값이 두개 이상이면

그 전 값과 만날 수 있으므로 1 추가



오아시스 재결합 / 3015

8

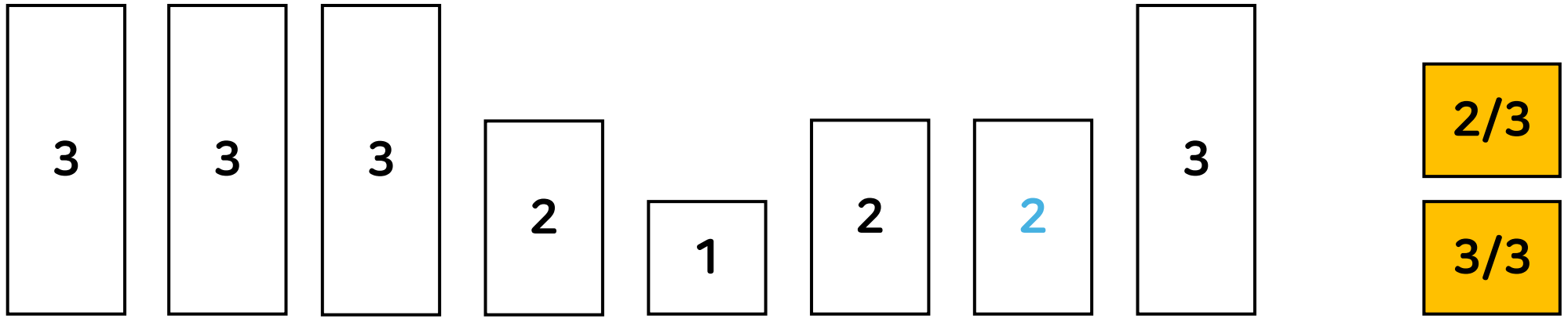
3 3 3 2 1 2 2 3

스택

정답 11

같은 수의 개수만큼 정답에 추가

또한 스택에 값이 2개 이상 있으므로 1 추가



오아시스 재결합 / 3015

8

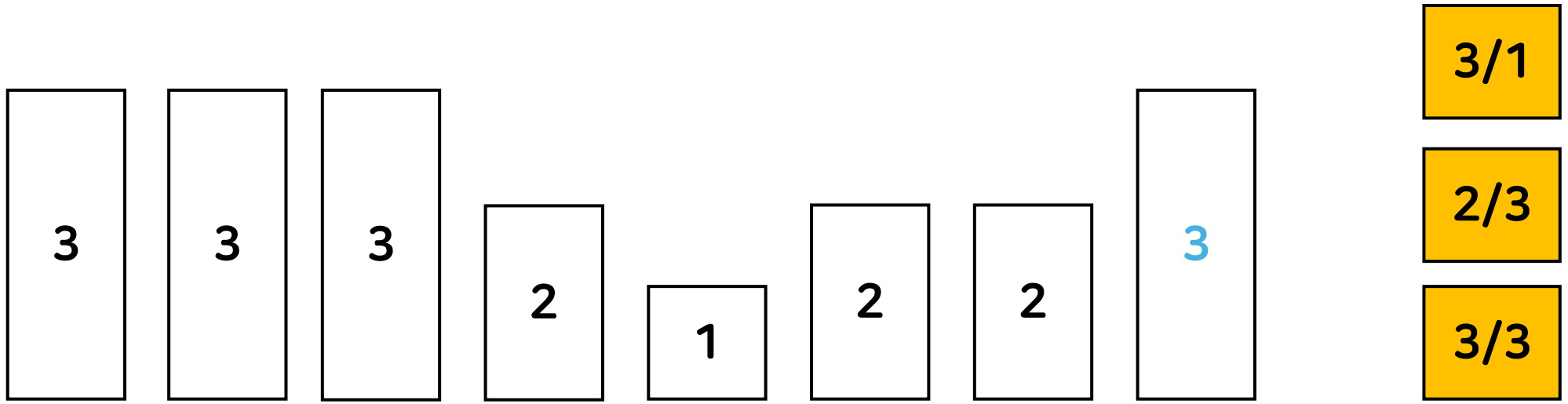
3 3 3 2 1 2 2 3 정답 14

큰 수가 들어왔으므로

작은 수들의 개수를 정답에 추가 후

스택에서 제거

스택



오아시스 재결합 / 3015

8

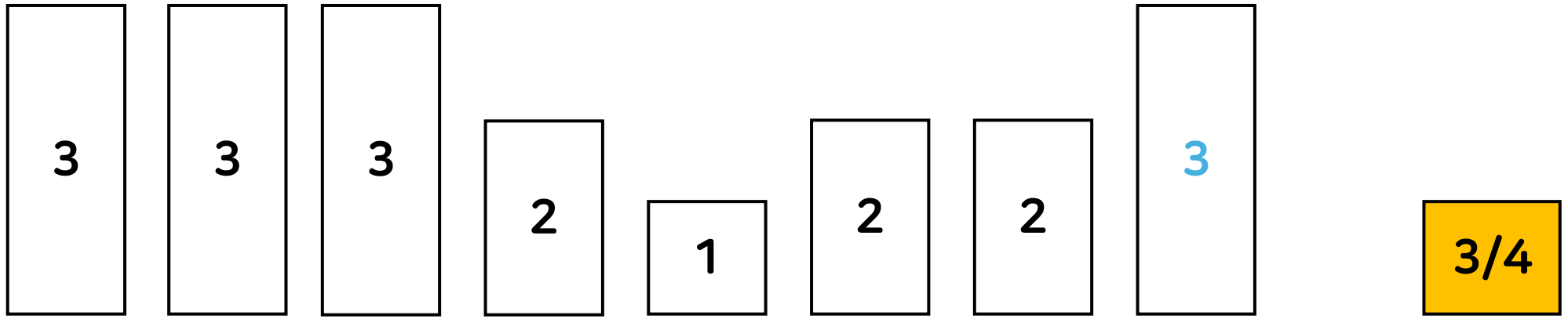
3 3 3 2 1 2 2 3

스택

정답 17

스택에 같은 수가 있으므로

개수 만큼 정답에 더함



오아시스 재결합 / 3015

C++

```cpp
stack <pair <int, int>> st;
void comp(){
    if(st.size() <= 1) return; // 크기가 1이면 종료

    auto top = st.top(); st.pop();
    // 연속된 두개의 값이 다르면
    // 다시 스택에 넣고 종료
    if(st.top().first != top.first){
        st.push(top); return;
    }

    // 연속된 두개의 값이 같으면 개수를 합쳐줌
    int num = st.top().second + top.second;
    st.pop(); st.push({top.first, num});
}
```

```cpp
int a[501010];
int main(){
    fastio;
    int n; cin >> n;
    long long result = 0;
    for(int i = 1;i <= n;i++) cin >> a[i];

    for(int i = 1;i <= n;i++){
        // 이전의 값이 현재 값보다 작을 때
        while(!st.empty() && st.top().first < a[i]){
            // 정답에 이전의 값의 개수만큼 더함
            // 작은 값들은 더 이상 다음 값들을 만날 수 없으므로
            // 스택에서 빼줌
            result += st.top().second; st.pop();
        }

        // 스택이 비었거나 이전의 값이 현재 값이랑 다르면 한 개 추가
        if(st.empty() || st.top().first != a[i]) st.push({a[i], 1});
        else{
            // 이전의 값이 현재 값이랑 같으면
            // 이전 값에 1 추가해서 넣어 줌
            int num = st.top().second + 1;
            st.pop(); st.push({a[i], num});

            // 같은 값들끼리는 서로 볼 수 있으므로 정답에 더함
            result += num - 1;
        }

        // 스택 압축
        comp();

        // 스택의 값이 2개 이상이면 정답에 1 추가
        if(st.size() >= 2) result++;
    }

    cout << result;
    return 0;
}
```

오아시스 재결합 / 3015

Python

```python
def comp():
    if len(st) <= 1: # 크기가 1이면 종료
        return

    top = st.pop()
    # 연속된 두개의 값이 다르면
    # 다시 스택에 넣고 종료
    if st[-1][0] != top[0]:
        st.append(top)
        return

    # 연속된 두개의 값이 같으면 개수를 합침
    num = st[-1][1] + top[1]
    st.pop()
    st.append((top[0], num))
```

```python
import sys
input = sys.stdin.readline
n = int(input().rstrip())
a = [int(input().rstrip()) for _ in range(n)]

st = []
result = 0

for i in a:
    # 이전의 값이 현재 값보다 작을 때
    while len(st) and st[-1][0] < i:
        # 정답에 이전 값의 개수만큼 더함
        # 작은 값들은 더 이상 다음 값들을 만날 수 없으므로
        # 스택에서 빼줌
        result += st[-1][1]
        st.pop()

    # 스택이 비었거나 현재 값이랑 다르면 한 개 추가
    if len(st) == 0 or st[-1][0] != i:
        st.append((i, 1))
    # 이전 값이랑 현재 값이랑 같으면
    # 이전 값에 1을 추가해서 넣어줌
    else:
        num = st[-1][1] + 1
        st.pop()
        st.append((i, num))
        # 같은 값들끼리는 서로 볼 수 있으므로 정답에 더함
        result += num - 1

    #스택 압축
    comp()

    # 스택의 값이 2개 이상이면 정답에 1 추가
    if(len(st) >= 2):
        result += 1

print(result)
```

질문?

고생하셨습니다