이진탐색

기본 개념
문제풀이

23.11.13 송이

이진탐색이란?

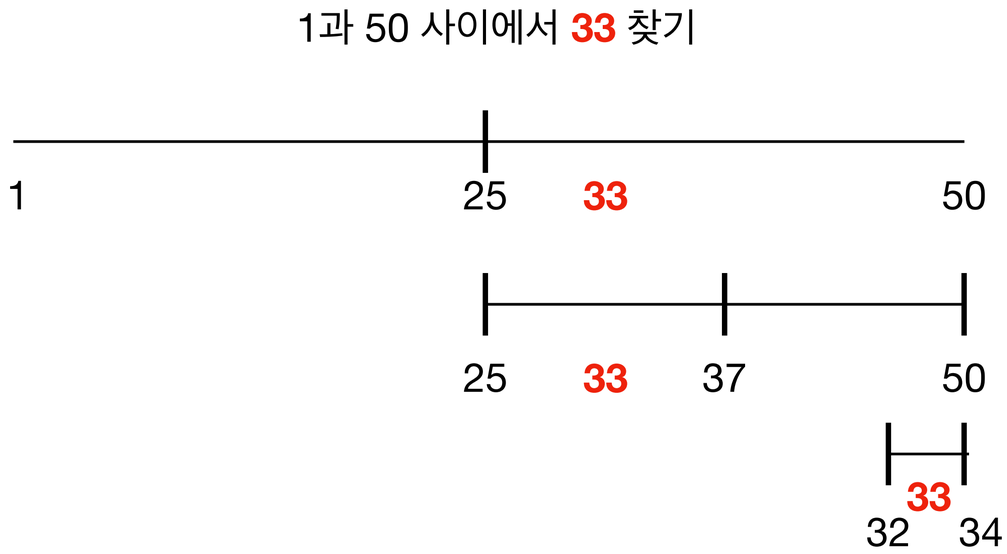
- 여러분에게 몇억개의 데이터가 주어졌다고 가정해 봅시다.
- 이런 상황에서 특정 데이터를 찾으면 어떻게 탐색해야 할까요?
- 엄청나게 많은 데이터 중에서 원하는 데이터를 마법처럼 찾는 방법은 존재하지 않습니다.
- 하지만 데이터가 정렬 되었다면 이야기가 달라집니다.
- 데이터가 정렬되어 있으면 이진 탐색을 사용할 수 있습니다.

8.1.1 이진 탐색이 가지는 이점.

- 이진탐색이란 전체를 절반으로 구분해 현재 자기 위치가 찾는 위치보다 값이 큰지 작은지에 따라 다음 탐색 방향을 결정한다.
- -> 필요없는 부분을 탐색하지 않는다.
- 따라서 매 탐색마다 탐색범위가 1/2씩 줄어든다.
- 이진탐색의 가장 큰 장점은 주어진 데이터가 정렬되어 있다면
- **O(logn)**으로 탐색할 수 있다.

따라서 문제에서 주어진 입력 크기가 과할 정도로 크다면
이진 탐색을 사용하자.

(이보다 빠른 탐색 방법은 해시 탐색밖에 없다(O(1))
하지만 여기서는 공간 복잡도까지 고려해야 하기 때문에
일반적인 상황에서 이진 탐색 이상의 효율을 가진 탐색 알고리즘은 찾기 어렵다.



8.1.2 구현방법

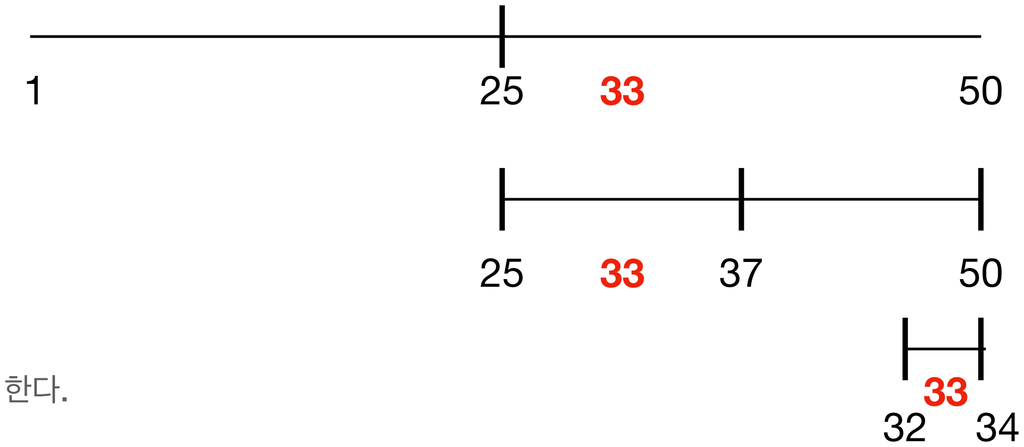
① 직접 구현하기

- 이진탐색 문제는 처음부터 데이터가 주어지고 여기서 이진 탐색만 사용하는 상황보다는, **탐색할 데이터를 직접 정의**하고 그 기반 아래에서 정렬한 뒤 이진 탐색을 진행하도록 유도한다.

- -> 탐색도 중요하지만 탐색원리 자체를 사용해서 풀라는 문제가 많이 주어진다.

```
def bisect(a, x, lo=0, hi=None):
    if lo<0:
        raise ValueError('lo must be non-negative')
    if hi is None:
        hi= len(a)
    while lo < hi:
        mid = (lo+hi) //2
        if a[mid] <x:
            lo = mid+1
        else:
            hi = mid
    return lo

mylist = [1,2,3,7,9,11,33]
print(bisect(mylist,3))
```



이 기본방법을 응용해서 문제를 풀어야 하므로 위의 기본 코드는 암기해 두어야 한다.

8.1.2 구현방법

① bisect 라이브러리 사용하기

- 이진탐색이 메인이 아니라 부가 작업 중 하나로 들어갈 경우 굳이 직접 구현하지 않고 라이브러리를 사용해도 된다.

```
from bisect import bisect
mylist = [1,2,3,7,9,11,33]
print(bisect(mylist,3))
```

bisect(<사용할 배열>,<찾을 값>)

- 만약 정렬된 데이터에 같은 값이 여러 개 있을 때 이진 탐색을 수행하면 어떻게 될까?
- bisect 함수는 오른쪽 방향(끝)을 우선으로 탐색하므로 **중복된 값이 처음 등장하는 위치를 찾을 수 없다.**
- 대신 이런 상황을 대비하여 라이브러리에서 **왼쪽(처음) 방향**으로 우선 탐색하는 **bisect_left()** 함수를 지원한다.

```
from bisect import bisect_left, bisect_right
mylist = [1,2,3,3,3,7,9]
x = 3
print(bisect_left(mylist,x)) #2
print(bisect_right(mylist,x)) #5
```

문제31

문제 이해하기

출발지점부터 distance만큼 떨어진 곳에 도착지점이 있습니다. 그리고 그사이에는 바위들이 놓여있습니다. 바위 중 몇 개를 제거하려고 합니다.
예를 들어, 도착지점이 25만큼 떨어져 있고, 바위가 [2, 14, 11, 21, 17] 지점에 놓여있을 때 바위 2개를 제거하면 출발지점, 도착지점, 바위 간의 거리가 아래와 같습니다.

제거한 바위의 위치	각 바위 사이의 거리	거리의 최솟값
[21, 17]	[2, 9, 3, 11]	2
[2, 21]	[11, 3, 3, 8]	3
[2, 11]	[14, 3, 4, 4]	3
[11, 21]	[2, 12, 3, 8]	2
[2, 14]	[11, 6, 4, 4]	4

위에서 구한 거리의 최솟값 중에 가장 큰 값은 4입니다.

출발지점부터 도착지점까지의 거리 distance, 바위들이 있는 위치를 담은 배열 rocks, 제거할 바위의 수 n이 매개변수로 주어질 때, 바위를 n개 제거한 뒤 각 지점 사이의 거리의 최솟값 중에 가장 큰 값을 return 하도록 solution 함수를 작성해주세요.

제한사항

- 도착지점까지의 거리 distance는 1 이상 1,000,000,000 이하입니다.
- 바위는 1개 이상 50,000개 이하가 있습니다.
- n 은 1 이상 `바위의 개수` 이하입니다.

입출력 예

distance	rocks	n	return
25	[2, 14, 11, 21, 17]	2	4

문제31

접근 방법 소개

1. 이진 탐색의 핵심 기억
2. 어떤 점을 탐색해야 하는지 정하기
3. 다음 탐색 범위 어떻게 정할지 결정하기

완전탐색으로 해결해야 할 것처럼 보이는 문제를 어떻게 이진탐색 문제로 바꾸어서 풀 것인가?

전체 징검다리 중 주어진 n개 만큼 제거해야 하니 모든 경우의 수를 뽑기 위해 조합을 사용하는 것을 우선적으로 떠올리게 된다.
하지만 조합은 완전탐색의 한 종류이기 때문에 시간복잡도가 너무 커진다(이 문제에선 입력이 굉장히 크다)

입력이 매우 크니까 이진탐색 방법을 생각해낼 수는 있는데
그렇다면 제거할 수 있는 모든 경우의 수를 어떻게 이진탐색으로 녹여낼 것인가에 대한 고민이 생긴다.

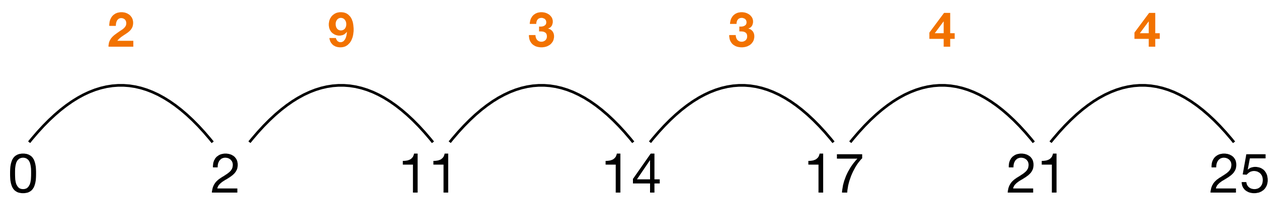
무슨 작업을 했더니 어떤 결과가 나왔다 ⟶ ?

결론 **가설**

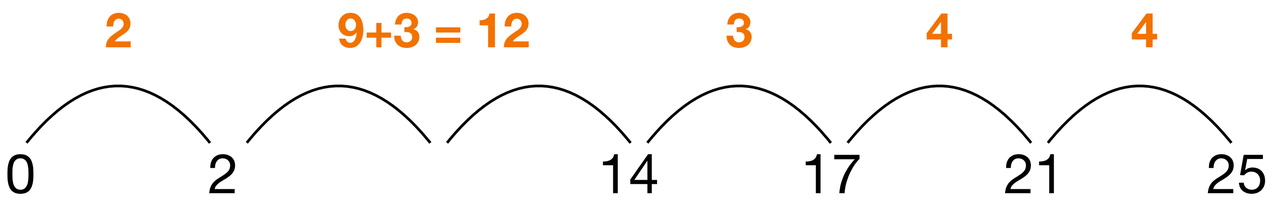
문제31

접근 방법 소개

1. 이진 탐색의 핵심 기억
2. 어떤 점을 탐색해야 하는지 정하기
3. 다음 탐색 범위 어떻게 정할지 결정하기



가장 짧은 거리를 만들기 위해서는 바위를 하나씩
제거해보고 그 사이의 모든 거리를 하나씩 살펴봐야 한다.
-> 비효율적



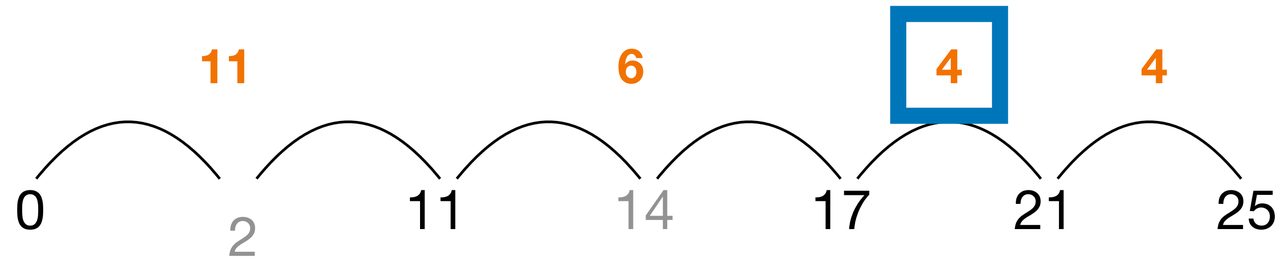
거꾸로
처음부터 **가장 짧게 만들 바위 간 거리**를 하나 정하고
나머지 바위를 제거할 때 무조건 이 값보다 큰 값이 나오도록 해보자.

문제31

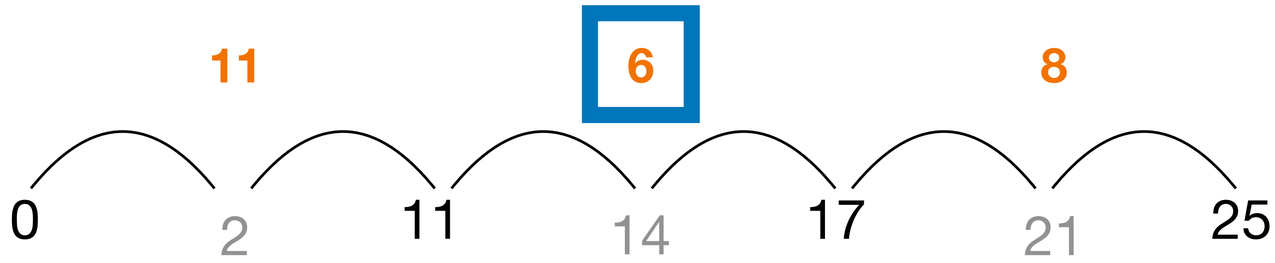
접근 방법 소개

1. 이진 탐색의 핵심 기억
2. 어떤 점을 탐색해야 하는지 정하기
3. 다음 탐색 범위 어떻게 정할지 결정하기

n=2, 최소거리 = 4



n=3, 최소거리 = 6



거리가 늘어날수록 제거해야 할 바위 개수가 늘어나고 거리가 줄어들면 제거해야 할 바위 개수가 줄어든다.

제거해야 할 바위가 목표한 것보다 많다면 거리를 줄이고, 목표한 것보다 적다면 거리를 늘리는 방식으로 이진탐색을 하자

문제31

접근 방법 소개

무슨 작업을 했더니 어떤 결과가 나왔다 ⟶ ?

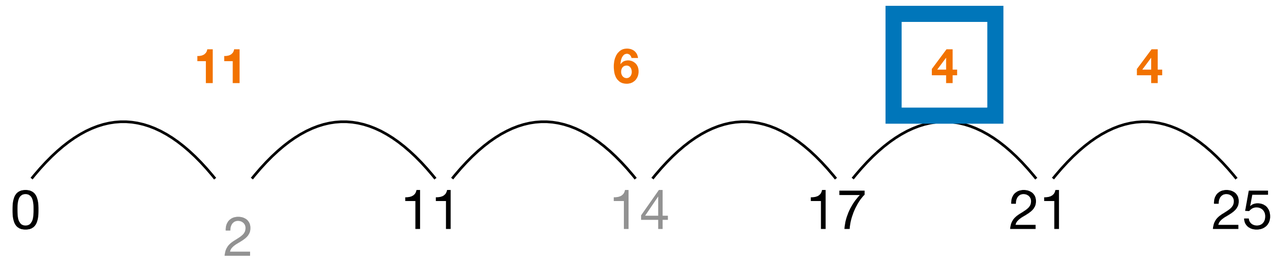
결론 **가설**

1.시작 지점 0과 끝 지점으로 전체 범위를 결정하고 바위 위치를 정렬한다.

2. 범위의 절반이 가장 짧은 거리라고 가정하고, 이 거리를 만들기 위해 바위를 몇 개 제거해야 하는지 센다.

3. 목표하는 바위보다 더 많이 제거했으면 거리를 줄이고 더 적게 제거했으면 거리를 늘린다.

n=2, 최소거리 = 4



문제31

STEP1) 시작 지점 0과 끝 지점으로 전체 범위를 결정하고 바위 위치를 정렬한다.

이진 탐색을 위한 준비단계

```
def solution(distance, rocks, n):
    answer = 0
    start, end = 0, distance
```

→ 시작과 끝 범위를 결정한다.

```
    rocks.sort()
```

→ 탐색을 할 배열을 미리 오름차순으로 정렬한다.

문제31

STEP2) 범위의 절반이 가장 짧은 거리라고 가정하고, 이 거리를 만들기 위해 바위를 몇 개 제거해야 하는지 센다.

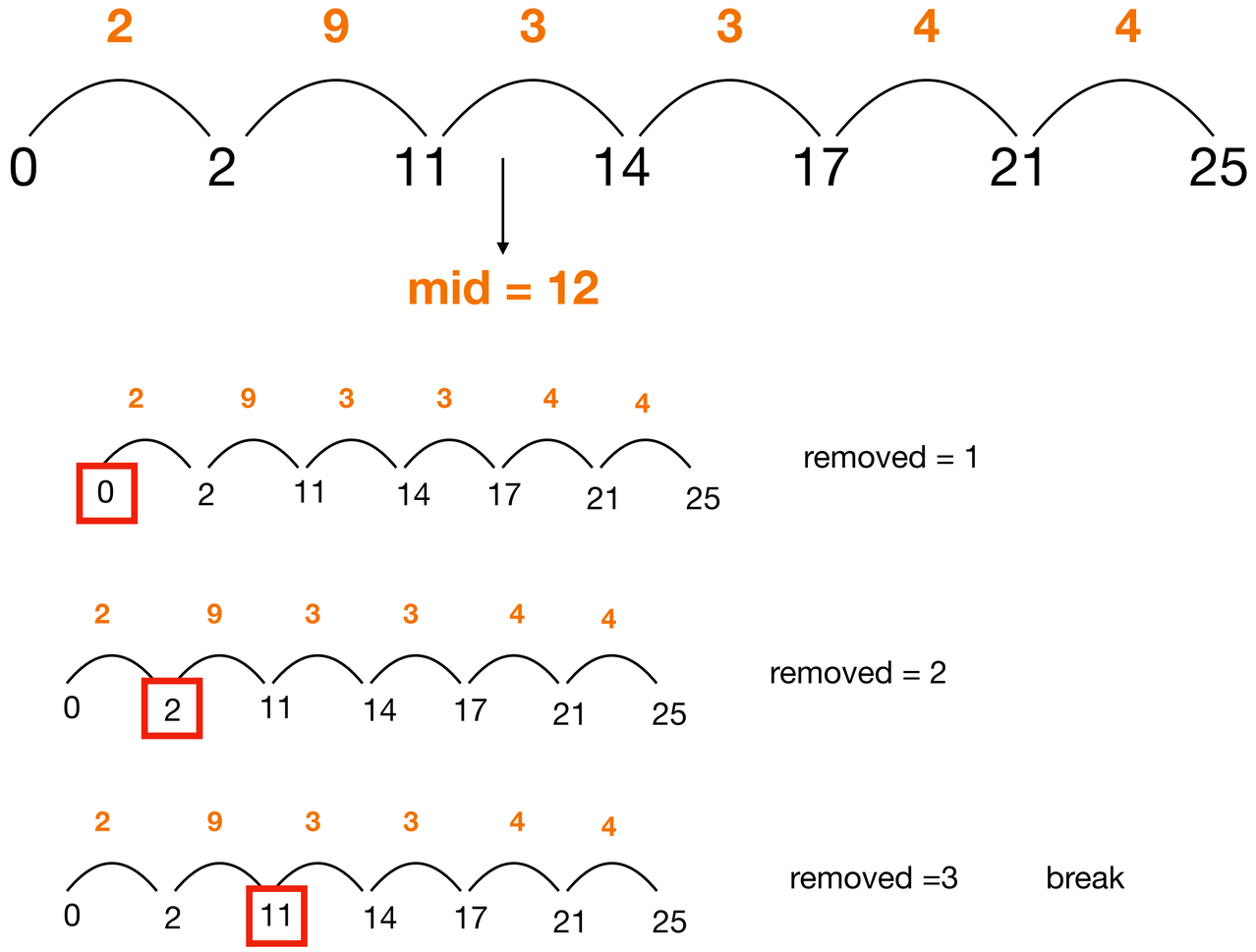
바위를 몇개 제거해야 하는지 판단

```
while start <=end:
    mid = (start+end) //2
    removed = 0
    temp = 0

    for rock in rocks:
        if rock - temp < mid : removed +=1
        else: temp = rock

        if removed > n : break
```

우리가 지워야 하는 개수 n보다는 더 지울 수 없다.



문제31

STEP3) 목표하는 바위보다 더 많이 제거했으면 거리를 줄이고 더 적게 제거했으면 거리를 늘린다.

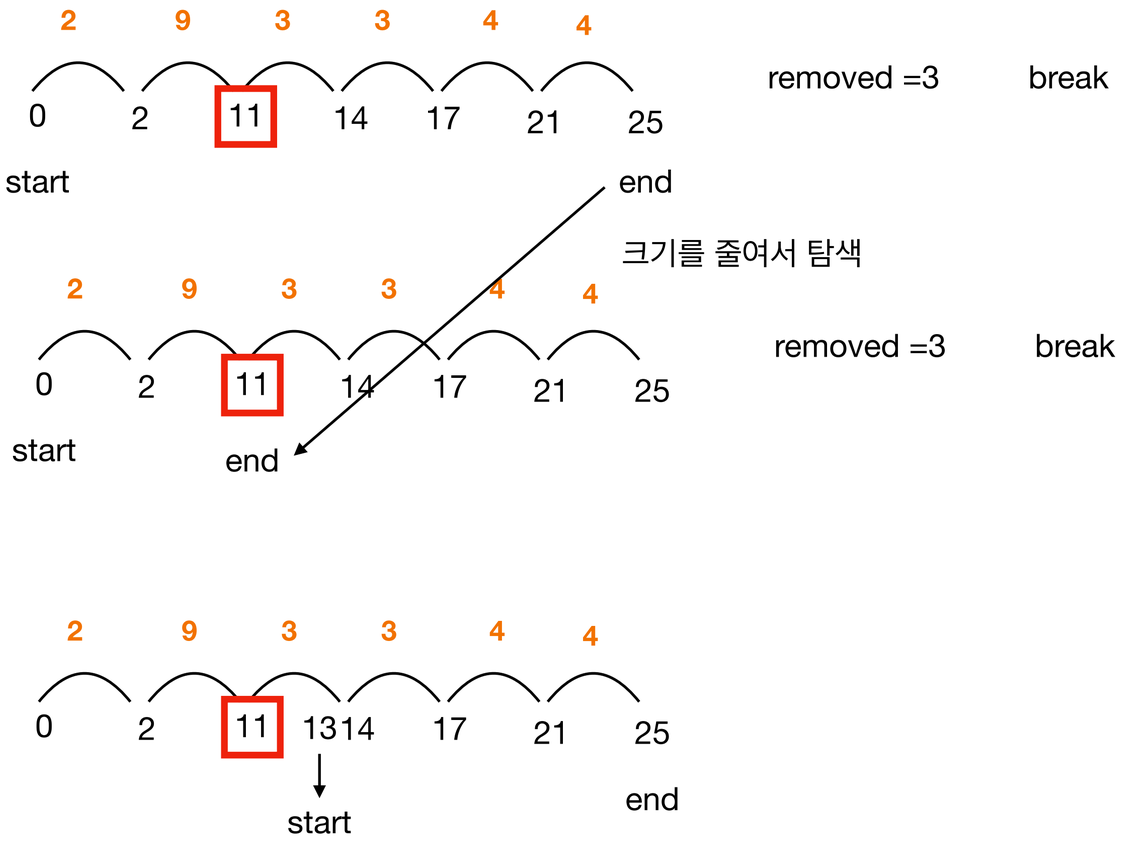
바위를 몇개 제거해야 하는지 판단

목표보다 더 제거

```
if removed >n:
    end = mid-1
```

목표보다 덜 제거

```
else:
    answer = mid
    start = mid + 1
```



문제31

전체 코드

```
while start <= end:
    mid = (start+end) //2
    del_stones = 0
    pre_stones = 0

    for rock in rocks:
        if rock-pre_stones < mid : del_stones +=1
        else: pre_stone = rock

        if del_stones >n: break

    if del_stones > n:
        end = mid -1
    else:
        answer = mid
        start = mid +1
```

문제31

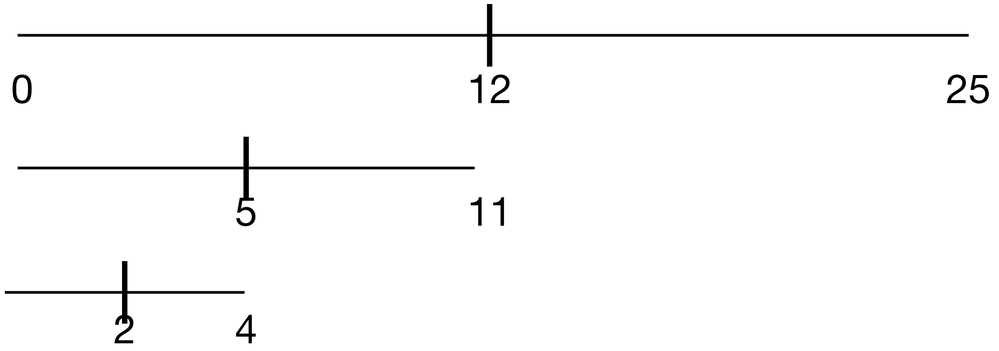
실제로 돌아가는 과정

left	mid	right	제거한 바위수
0	12	25	3
0	5	11	3
0	2	4	0
3	3	4	1
4	4	4	2

왼쪽 탐색

오른쪽 탐색

우리의 목표



answer = 2

answer = 3

answer = 4