이진 탐색 알고리즘

이준규

목차

이진탐색이란?

이진탐색이란?

이진 탐색은 리스트의 중간 값과 비교하여 목표값을 찾는 알고리즘입니다.

중간 값을 찾아야 하기 때문에 반드시 정렬된 배열에서만 사용할 수 있습니다.

순차탐색 vs 이진탐색

순차 탐색 (Sequential search)

- 특정한 원소를 찾기 위해 원소를 순차적으로 하나씩 탐색 O(N)의 시간 복잡도를 가짐
- 리스트의 길이가 길수록 비효율적임
- 검색 방법 중 가장 단순하여 구현이 쉽고 정렬되지 않은 리스트에서도 사용할 수 있다는 장점이 있음

이진탐색 (Binary search)

- 오름차순으로 정렬된 리스트에서 특정한 값의 위치를 찾는 알고리즘. 탐색 범위를 절반씩 좁혀가며 데이터를 탐색
- O(logN)의 시간 복잡도를 가짐
- 처음 선택한 중앙값이 만약 찾는 값보다 크면 그 값은 새로운 최댓값, 작으면 그 값은 새로운 최솟값이 됨
- 정렬된 리스트에만 탐색이 가능하다는 단점이 있음
- 검색이 반복될 때마다 목표값을 찾을 확률은 두 배가 되므로 속도가 빠르다는 장점이 있음

이진탐색 동작방식

- 동작 방식

1) 배열의 중간 값을 가져옵니다.

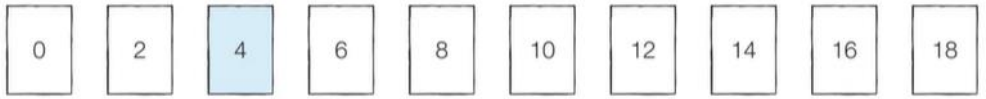
2) 중간 값과 검색 값을 비교합니다.

   중간 값 = 검색 값 : 해당 값 return 후 종료 (mid = key)
   중간 값 > 검색 값 : 중간값 기준 배열의 오른쪽 탐색 (mid < key)
   중간 값 < 검색 값 : 중간값 기준 배열의 왼쪽 탐색 (mid > key)

3) 값을 찾거나 간격이 없어질 때까지 반복합니다.

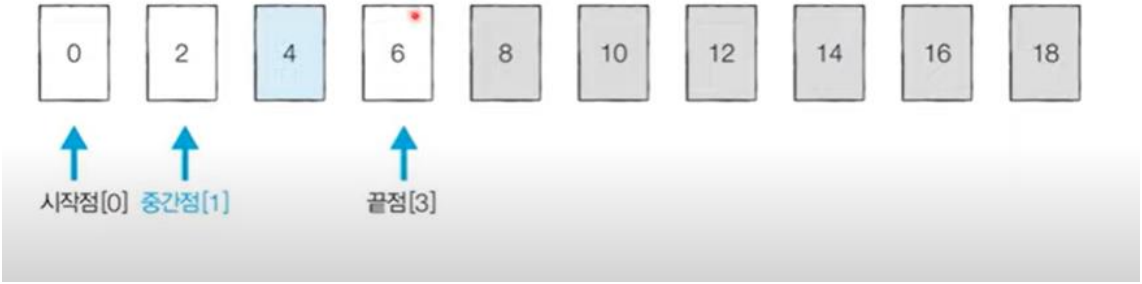
이진탐색 예시

- 이미 정렬된 10개의 데이터 중에서 값이 4인 원소를 찾는 예시를 살펴봅시다.

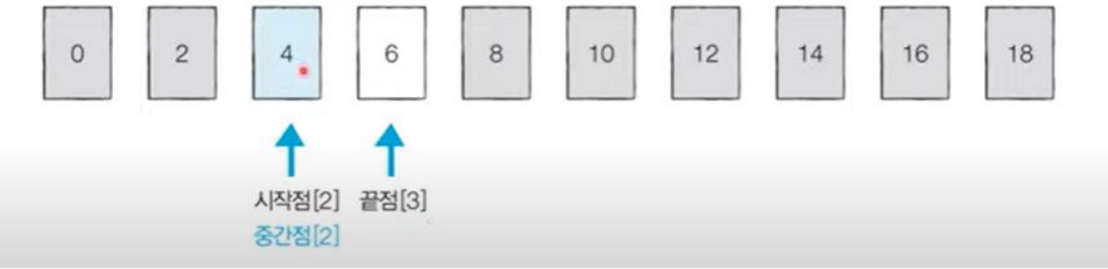


- **[Step 1]** 시작점: 0, 끝점: 9, 중간점: 4 (소수점 이하 제거)

- **[Step 2]** 시작점: 0, 끝점: 3, 중간점: 1 (소수점 이하 제거)



- **[Step 3]** 시작점: 2, 끝점: 3, 중간점: 2 (소수점 이하 제거)



찾고자 했던 값과 일치하는 경우 해당 인덱스를 반환하며 탐색종료.

이진탐색 코드 (재귀함수)

```python
# 이진 탐색 소스코드 구현 (재귀 함수)
def binary_search(array, target, start, end):
    if start > end:
        return None
    mid = (start + end) // 2
    # 찾은 경우 중간점 인덱스 반환
    if array[mid] == target:
        return mid
    # 중간점의 값보다 찾고자 하는 값이 작은 경우 왼쪽 확인
    elif array[mid] > target:
        return binary_search(array, target, start, mid - 1)
    # 중간점의 값보다 찾고자 하는 값이 큰 경우 오른쪽 확인
    else:
        return binary_search(array, target, mid + 1, end)

# n(원소의 개수)과 target(찾고자 하는 값)을 입력 받기
n, target = list(map(int, input().split()))
# 전체 원소 입력 받기
array = list(map(int, input().split()))

# 이진 탐색 수행 결과 출력
result = binary_search(array, target, 0, n - 1)
if result == None:
    print("원소가 존재하지 않습니다.")
else:
    print(result + 1)
```

```
10 7 ↵
1 3 5 7 9 11 13 15 17 19 ↵
4
```

```
10 7 ↵
1 3 5 6 9 11 13 15 17 19 ↵
원소가 존재하지 않습니다.
```

이진탐색 코드 (반복문)

```python
# 이진 탐색 소스코드 구현 (반복문)
def binary_search(array, target, start, end):
    while start <= end:
        mid = (start + end) // 2
        # 찾은 경우 중간점 인덱스 반환
        if array[mid] == target:
            return mid
        # 중간점의 값보다 찾고자 하는 값이 작은 경우 왼쪽 확인
        elif array[mid] > target:
            end = mid - 1
        # 중간점의 값보다 찾고자 하는 값이 큰 경우 오른쪽 확인
        else:
            start = mid + 1
    return None

# n(원소의 개수)과 target(찾고자 하는 값)을 입력 받기
n, target = list(map(int, input().split()))
# 전체 원소 입력 받기
array = list(map(int, input().split()))

# 이진 탐색 수행 결과 출력
result = binary_search(array, target, 0, n - 1)
if result == None:
    print("원소가 존재하지 않습니다.")
else:
    print(result + 1)
```

```
10 7 ↵
1 3 5 7 9 11 13 15 17 19 ↵
4
```

```
10 7 ↵
1 3 5 6 9 11 13 15 17 19 ↵
원소가 존재하지 않습니다.
```

파라메트릭 서치 (Parametric Search)

- **파라메트릭 서치**란 최적화 문제를 결정 문제('예' 혹은 '아니오')로 바꾸어 해결하는 기법입니다.
  - 예시: 특정한 조건을 만족하는 가장 알맞은 값을 빠르게 찾는 최적화 문제
- 일반적으로 코딩 테스트에서 파라메트릭 서치 문제는 **이진 탐색을 이용하여 해결**할 수 있습니다.

문제 1

떡볶이 떡 만들기 문제

- 오늘 동빈이는 여행 가신 부모님을 대신해서 떡집 일을 하기로 했습니다. 오늘은 떡볶이 떡을 만드는 날입니다. 동빈이네 떡볶이 떡은 재밌게도 떡볶이 떡의 길이가 일정하지 않습니다. 대신에 한 봉지 안에 들어가는 떡의 총 길이는 절단기로 잘라서 맞춰줍니다.
- 절단기에 **높이(H)**를 지정하면 줄지어진 떡을 한 번에 절단합니다. 높이가 H보다 긴 떡은 H 위의 부분이 잘릴 것이고, 낮은 떡은 잘리지 않습니다.
- 예를 들어 높이가 19, 14, 10, 17cm인 떡이 나란히 있고 절단기 높이를 15cm로 지정하면 자른 뒤 떡의 높이는 15, 14, 10, 15cm가 될 것입니다. 잘린 떡의 길이는 차례대로 4, 0, 0, 2cm입니다. 손님은 6cm 만큼의 길이를 가져갑니다.
- 손님이 왔을 때 요청한 총 길이가 M일 때 **적어도 M만큼의 떡을 얻기 위해 절단기에 설정할 수 있는 높이의 최댓값을 구하는 프로그램**을 작성하세요.

떡볶이 떡 만들기 문제

난이도 ●●○ | 풀이 시간 40분 | 시간제한 2초 | 메모리 제한 128MB

입력 조건

- 첫째 줄에 떡의 개수 N과 요청한 떡의 길이 M이 주어집니다. (1 ≤ N ≤ 1,000,000, 1 ≤ M ≤ 2,000,000,000)
- 둘째 줄에는 떡의 개별 높이가 주어집니다. 떡 높이의 총합은 항상 M 이상이므로, 손님은 필요한 양만큼 떡을 사갈 수 있습니다. 높이는 10억보다 작거나 같은 양의 정수 또는 0입니다.

출력 조건

- 적어도 M만큼의 떡을 집에 가져가기 위해 절단기에 설정할 수 있는 높이의 최댓값을 출력합니다.

입력 예시

```
4 6
19 15 10 17
```

출력 예시

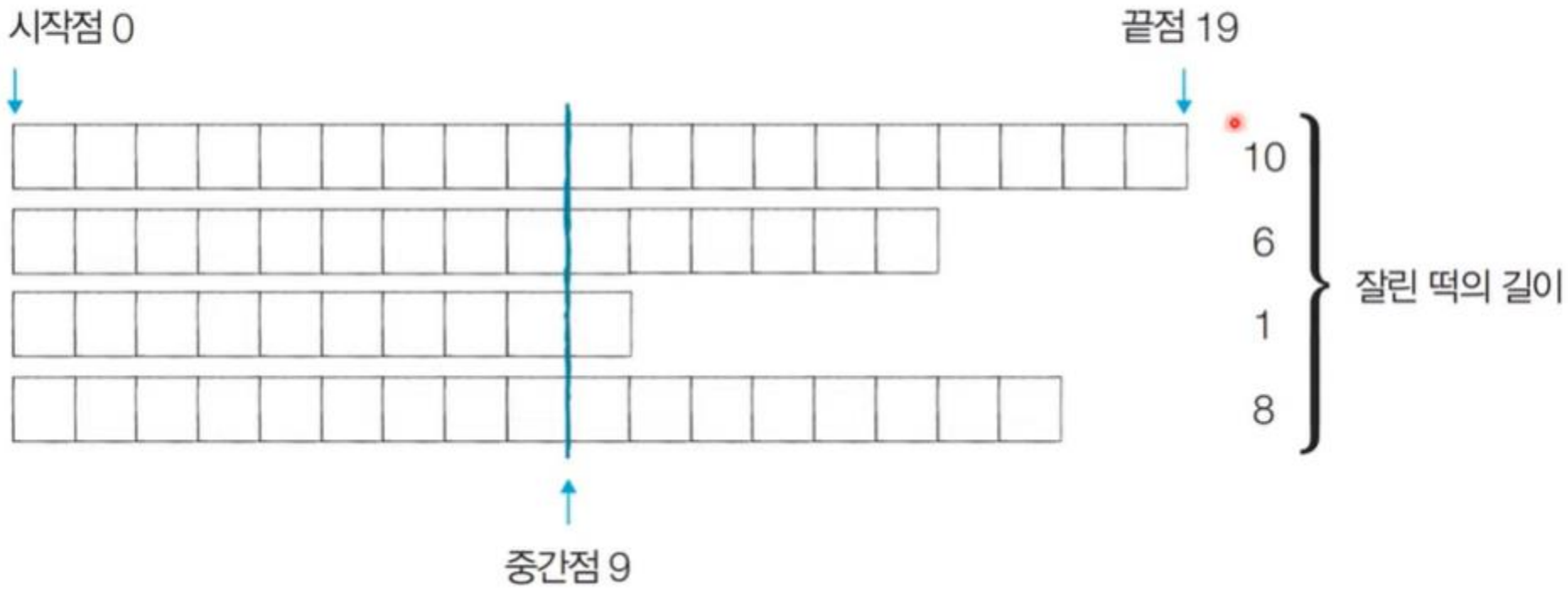
```
15
```

떡볶이 떡 만들기 문제 아이디어

- 적절한 높이를 찾을 때까지 이진 탐색 반복수행, 높이 H를 반복해서 조정한다.
- H로 자르면 조건이 만족하는지 확인 후, 조건 만족 여부에 따라 탐색 범위를 좁혀가면서 해결한다.
- 절단기 높이는 0 ~ 10억 까지 정수 중 하나 -> 범위가 매우 크므로 이진 탐색을 사용해야 한다는 것을 인지한다

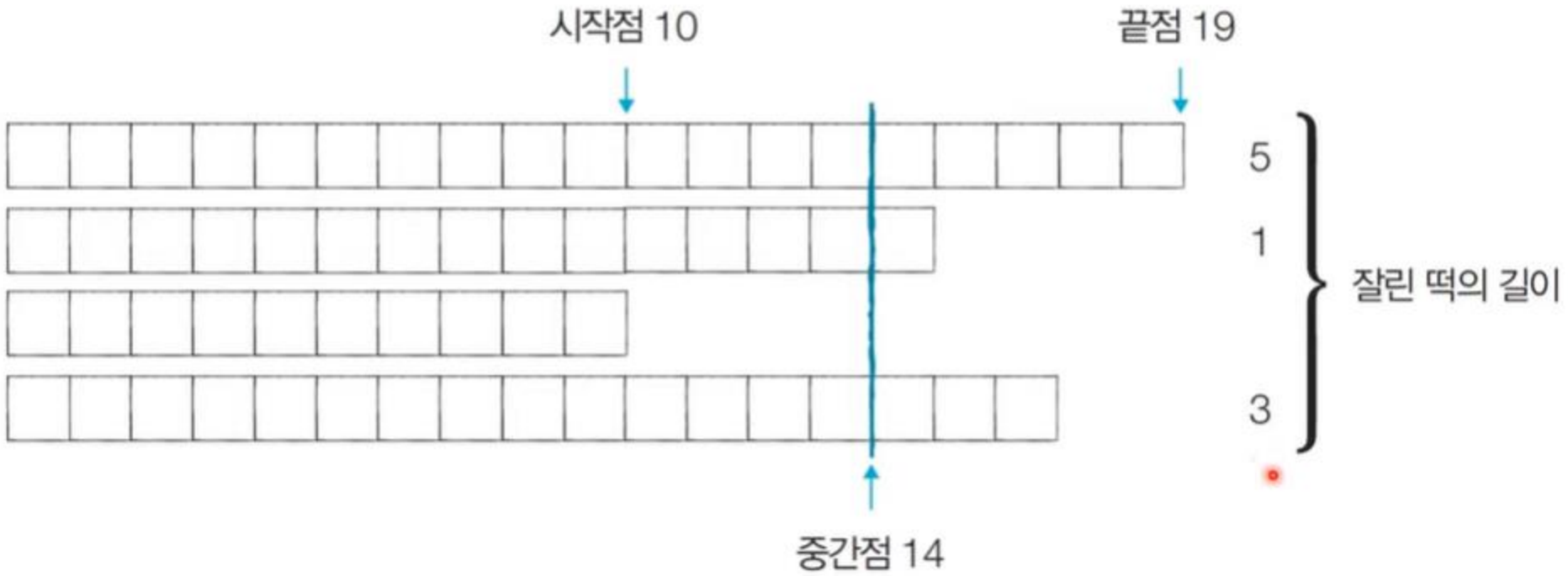
떡볶이 떡 만들기 문제 아이디어

- **[Step 1]** 시작점: 0, 끝점: 19, 중간점: 9 이때 필요한 떡의 크기: M = 6이므로, **결과 저장**



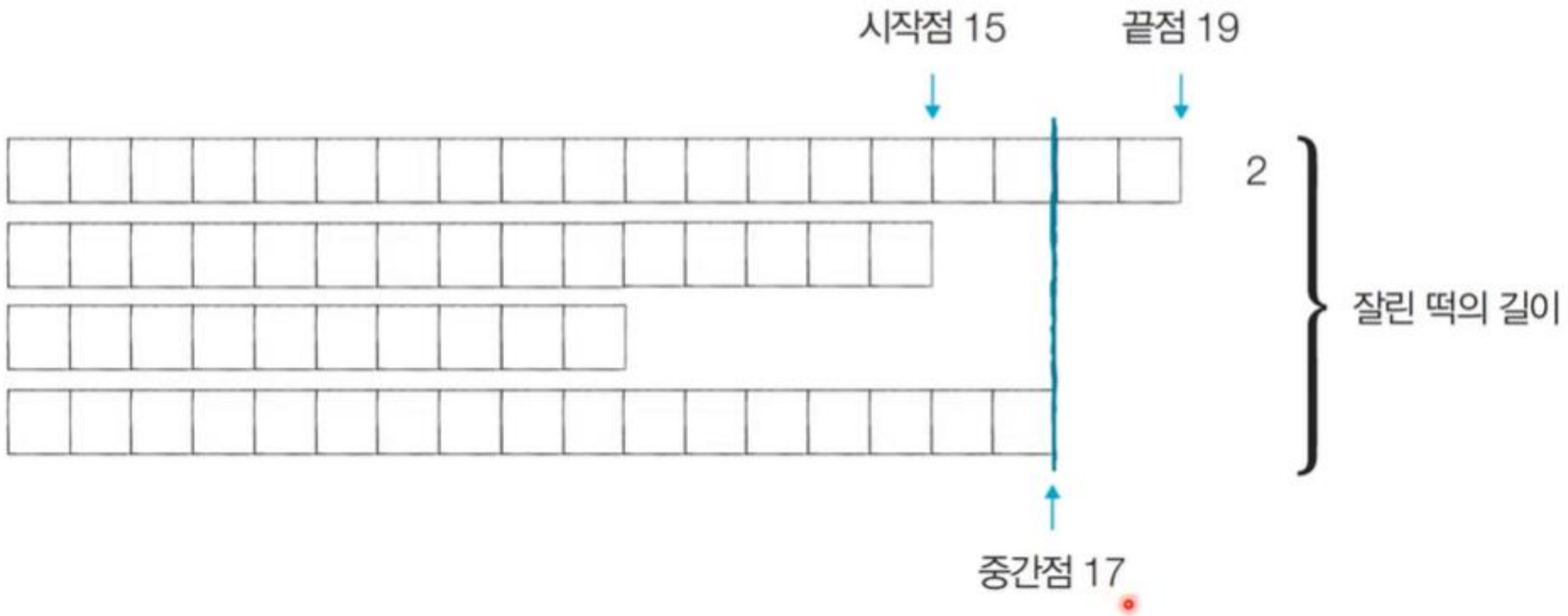
떡볶이 떡 만들기 문제 아이디어

- **[Step 2]** 시작점: 10, 끝점: 19, 중간점: 14 이때 필요한 떡의 크기: M = 6이므로, **결과 저장**



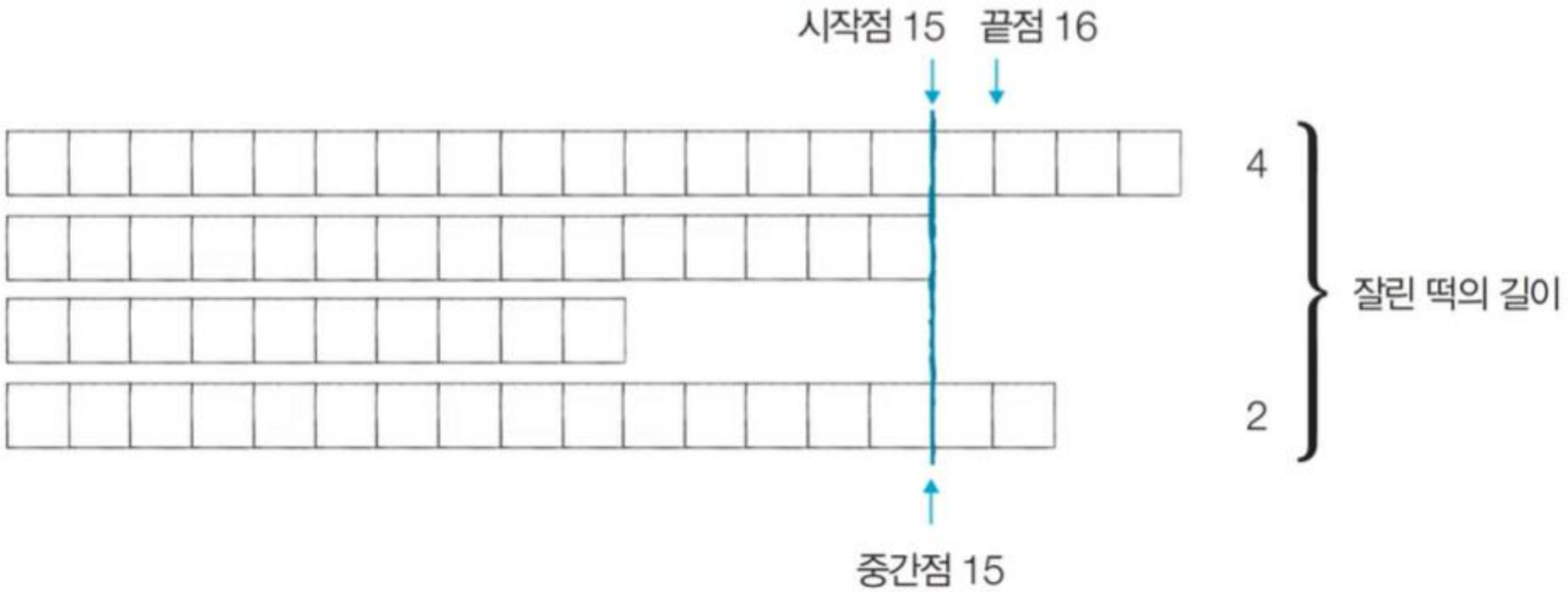
떡볶이 떡 만들기 문제 아이디어

- **[Step 3]** 시작점: 15, 끝점: 19, 중간점: 17 이때 필요한 떡의 크기: M = 6이므로, **결과 저장하지 않음**



떡볶이 떡 만들기 문제 아이디어

- **[Step 4]** 시작점: 15, 끝점: 16, 중간점: 15 이때 필요한 떡의 크기: M = 6이므로, **결과 저장**



- 이러한 이진 탐색 과정을 반복하면 답을 도출할 수 있습니다.
- 중간점의 값은 시간이 지날수록 **'최적화된 값'**이 되기 때문에, 과정을 반복하면서 얻을 수 있는 떡의 길이 합이 필요한 떡의 길이보다 크거나 같을 때마다 **중간점의 값을 기록**하면 됩니다.

떡볶이 떡 만들기 문제 코드

```python
# 떡의 개수(N)와 요청한 떡의 길이(M)을 입력
n, m = list(map(int, input().split(' ')))
# 각 떡의 개별 높이 정보를 입력
array = list(map(int, input().split()))

# 이진 탐색을 위한 시작점과 끝점 설정
start = 0
end = max(array)

# 이진 탐색 수행 (반복적)
result = 0
while(start <= end):
    total = 0
    mid = (start + end) // 2
    for x in array:
        # 잘랐을 때의 떡의 양 계산
        if x > mid:
            total += x - mid
    # 떡의 양이 부족한 경우 더 많이 자르기 (왼쪽 부분 탐색)
    if total < m:
        end = mid - 1
    # 떡의 양이 충분한 경우 덜 자르기 (오른쪽 부분 탐색)
    else:
        result = mid # 최대한 덜 잘랐을 때가 정답이므로, 여기에서 result에 기록
        start = mid + 1

# 정답 출력
print(result)
```

문제 2

정렬된 배열에서 특정 수의 개수 구하는 문제

- N개의 원소를 포함하고 있는 수열이 오름차순으로 정렬되어 있습니다. 이때 이 수열에서 x가 등장하는 횟수를 계산하세요. 예를 들어 수열 {1, 1, 2, 2, 2, 2, 3}이 있을 때 x = 2라면, 현재 수열에서 값이 2인 원소가 4개이므로 4를 출력합니다.
- 단, 이 문제는 시간 복잡도 $O(logN)$으로 알고리즘을 설계하지 않으면 **시간 초과** 판정을 받습니다.

정렬된 배열에서 특정 수의 개수 구하는 문제

난이도 ●●○ | 풀이 시간 30분 | 시간 제한 1초 | 메모리 제한 128MB | 기출 Zoho 인터뷰

입력 조건

- 첫째 줄에 N과 x가 정수 형태로 공백으로 구분되어 입력됩니다.
  $(1 \le N \le 1{,}000{,}000), (-10^9 \le x \le 10^9)$
- 둘째 줄에 N개의 원소가 정수 형태로 공백으로 구분되어 입력됩니다.
  $(-10^9 \le$ 각 원소의 값 $\le 10^9)$

출력 조건

- 수열의 원소 중에서 값이 x인 원소의 개수를 출력합니다. 단, 값이 x인 원소가 하나도 없다면 −1을 출력합니다.

입력 예시 1

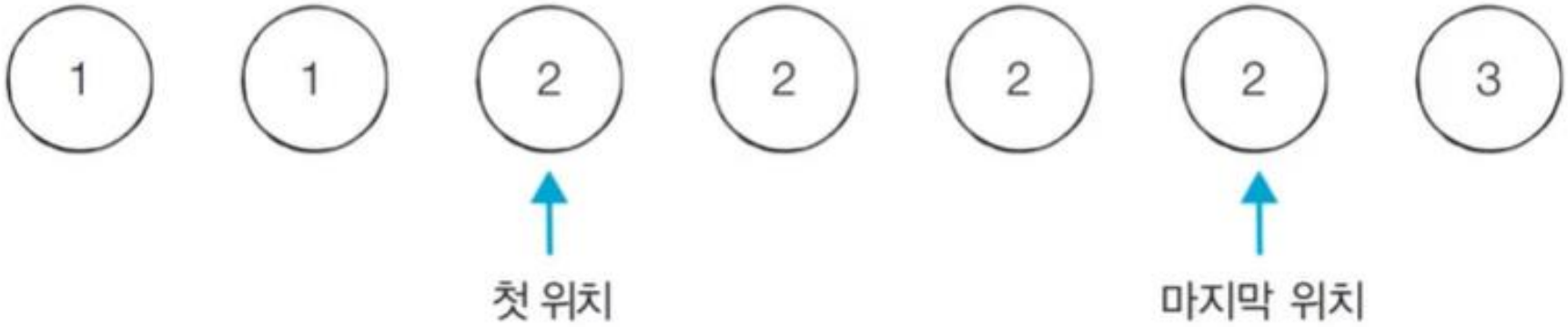
```
7 2
1 1 2 2 2 2 3
```

출력 예시 1

```
4
```

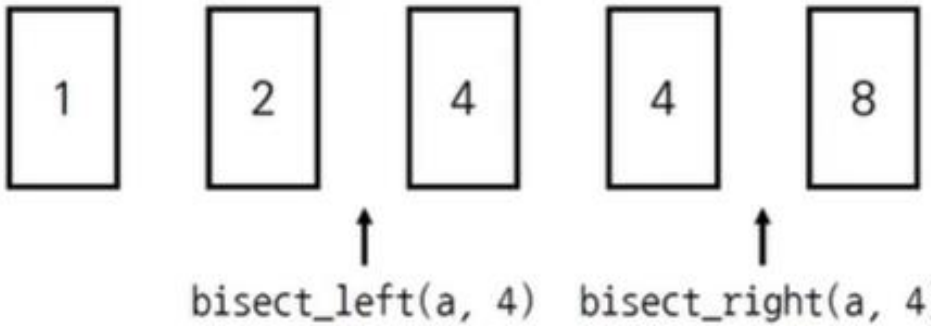
정렬된 배열에서 특정 수의 개수 구하는 문제 아이디어

- 시간 복잡도 $O(logN)$으로 동작하는 알고리즘을 요구하고 있습니다.
  - 일반적인 선형 탐색(Linear Search)로는 시간 초과 판정을 받습니다.
  - 하지만 데이터가 정렬되어 있기 때문에 이진 탐색을 수행할 수 있습니다.
- 특정 값이 등장하는 첫 번째 위치와 마지막 위치를 찾아 위치 차이를 계산해 문제를 해결할 수 있습니다.



정렬된 배열에서 특정 수의 개수 구하는 문제 아이디어

- bisect_left(a, x): 정렬된 순서를 유지하면서 배열 a에 x를 삽입할 가장 왼쪽 인덱스를 반환
- bisect_right(a, x): 정렬된 순서를 유지하면서 배열 a에 x를 삽입할 가장 오른쪽 인덱스를 반환




```
from bisect import bisect_left, bisect_right

a = [1, 2, 4, 4, 8]
x = 4

print(bisect_left(a, x))
print(bisect_right(a, x))
```

실행 결과

```
2
4
```

정렬된 배열에서 특정 수의 개수 구하는 문제 코드

```python
from bisect import bisect_left, bisect_right

# 값이 [left_value, right_value]인 데이터의 개수를 반환하는 함수
def count_by_range(array, left_value, right_value):
    right_index = bisect_right(array, right_value)
    left_index = bisect_left(array, left_value)
    return right_index - left_index

n, x = map(int, input().split()) # 데이터의 개수 N, 찾고자 하는 값 x 입력받기
array = list(map(int, input().split())) # 전체 데이터 입력받기

# 값이 [x, x] 범위에 있는 데이터의 개수 계산
count = count_by_range(array, x, x)

# 값이 x인 원소가 존재하지 않는다면
if count == 0:
    print(-1)
# 값이 x인 원소가 존재한다면
else:
    print(count)
```

Q&A

감사합니다