이진탐색

순차탐색 : 리스트 안에 있는 특정 데이터를 찾기 위해 앞에서부터 데이터를 하나씩 확인하는 방법

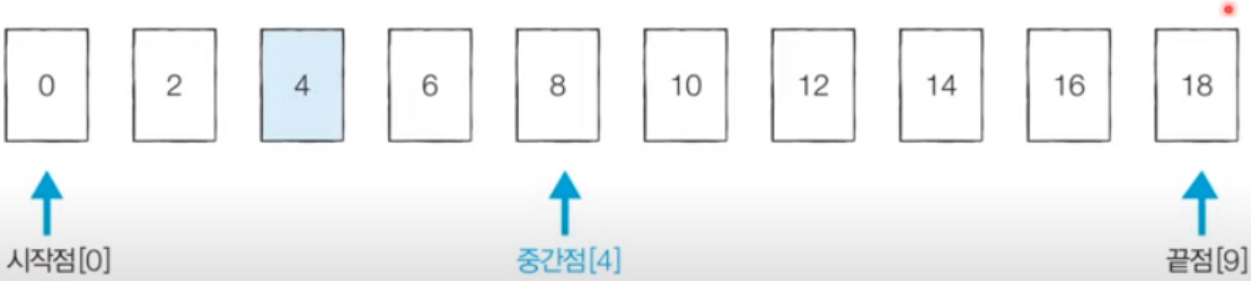
이진탐색 : 정렬된 리스트에서 탐색 범위를 절반씩 좁혀가면서 데이터를 탐색하는 방법

- 이진 탐색은 시작점, 끝점, 중간점을 이용해 탐색 범위를 설정한다.
- 중간점보다 크고 작음을 통해 1 / 2로 범위를 좁혀가며 탐색하기 때문에 시간 복잡도는 O(Log N)을 보장한다.

```
# 배열, 찾을 값, 시작값, 마지막 값
def binary_search(array, target, start, end):
    if start > end: # 만약 시작값이 마지막 값 보다 크다면 원소가 없거나 정렬되지 않은 배열임
        return None

    # 중간값을 지정하고
    mid = (start + end) // 2

    # 만약 중간값이 찾아야 할 값 이라면
    if array[mid] == target:
        return target # 이 값을 return

    # 만약 중간 값이 찾아야 할 값 보다 크다면
    elif array[mid] > target: # 중간값 기준 왼쪽만 찾으면 된다.
        binary_search(array, target, start, mid - 1)

    # 만약 중간 값이 찾아야 할 값 보다 작다면
    else: # 중간값 기준 오른쪽만 찾으면 된다.
        binary_search(array, target, start, mid + 1)

# 라이브러리
# from bisect import bisect_right, bisect_left
# bisect_right(a, x) : 정렬 된 순서를 유지하면서 배열 a에 x를 삽입 할 가장 오른쪽 인덱스를 반환한다.
# bisect_left(a, x) : 정렬 된 순서를 유지하면서 배열 a에 x를 삽입 할 가장 왼쪽 인덱스를 반환한다.
```

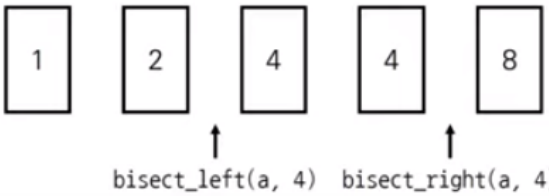
이진 탐색 동작 예시

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



파이썬 이진 탐색 라이브러리

- `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




```python
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
```

Parametric Search, 파라메트릭 서치

최적화 문제를 결정 문제로 바꾸어 해결하는 기법

- 최적화 된 값을 구할 때 예 / 아니오로 바꿔 값을 빠르게 찾는다.
- 일반적으로 이진탐색을 활용해 해결하는 기법

〈문제〉 떡볶이 떡 만들기: 문제 설명

- 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.

〈문제〉 떡볶이 떡 만들기: 문제 조건

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
```

```python
def binary_search():
    n, m = 4, 6 # 최대 갯수와 최소 길이
    arr = [19, 15, 10, 17] # 리스트를 입력받는다.
    # [10, 15, 17, 19]
    arr.sort() # 이진탐색을 위해 리스트를 정렬한다.

    start, end = 0, max(arr) # 시작은 0부터 마지막 값 까지
    result = 0 # 결과를 저장 할 값

    while(start <= end):

        # 떡을 자른 합계를 저장 할 값
        sum = 0
        mid = (start + end) // 2 # 가운데 값을 설정하고

        for i in arr: # 주어진 배열을 돌면서
```

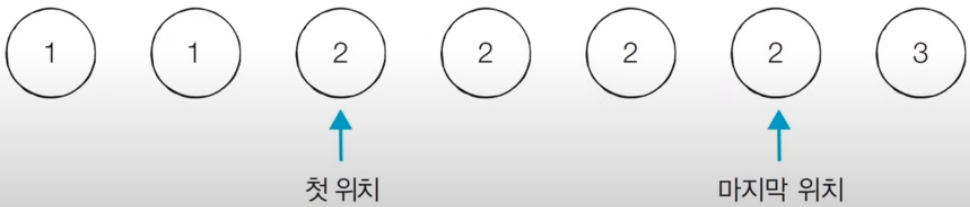
```
            if i > mid: # 만약 현재 값이 중간 값 보다 작아서 떡을 자를 수 있다면
                sum += i - mid # 자른만큼 값을 더한다.

        if sum < m: # 반복을 다 돌고 만약 자른 값이 최소 길이보다 작으면
            end = mid - 1 # 중간 값 이후 오른쪽 값은 모두 날려 범위를 반으로 즐인다.
        else: # 최소 길이보다 합계가 크면
            result = mid # 최대한 덜 잘라야 하므로 결과인 result에 값을 기록하고
            start = mid + 1 # 중간 값 이전 왼쪽 값은 모두 날려 범위를 반으로 줄인다.

    print(result)
```

〈문제〉 정렬된 배열에서 특정 수의 개수 구하기: 문제 설명

- 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.

〈문제〉 정렬된 배열에서 특정 수의 개수 구하기: 문제 해결 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.




```
from bisect import bisect_left, bisect_right

def bisect_search():
    arr = [1, 1, 2, 2, 2, 2, 3]

    def count(arr, a, b):
        end = bisect_right(arr, a)
        start = bisect_left(arr, b)
        return end - start

    result = count(arr, 2, 2)
```

```
    if result == 0:
        print(-1)
    else:
        print(result)
```

이진 탐색 템플릿

- 이진 탐색과 파라메트릭 서치를 각각 문제에 대입한다.
    - 이진 탐색은 범위를 좁혀나가며 탐색을 해야 할 때,
    - 파라메트릭은 특정 범위의 갯수 혹은 특정 범위의 배열이 필요할 때 사용한다.

```
# 이진 탐색
def binary_search():
    target = 6 # 문제에서 주어지는 target을 입력받거나 정의
    arr = [19, 15, 10, 17] # 입력 받은 리스트.
    arr.sort() # 이진탐색을 위해 리스트를 정렬한다.

    start, end = 0, max(arr) # 시작은 0부터 마지막 값 까지, 혹은 문제에 따라 최솟값 부터 최대값

    # 항상 이진 탐색은 절반으로 줄어드므로, 시작과 끝이 중간이 될 때 까지 범위를 줄여나간다.
    while(start <= end):

        sum = 0 # 지문에 따른 결과 값 등
        mid = (start + end) // 2 # 가운데 값을 설정

        for i in arr: # 주어진 배열을 돌면서
            return None # 지문에 따라 무언가의 로직을 수행

        if sum < target: # 로직 수행 후 주어지는 target이 중간 값(혹은 로직의 결과) 보다 크면
            end = mid - 1 # 중간 값 이후 오른쪽 값은 모두 날려 범위를 반으로 즐인다.
        else: # 중간 값 보다 작으면
            start = mid + 1 # 중간 값 이전 왼쪽 값은 모두 날려 범위를 반으로 줄인다.
```

```
# 파라메트릭 서치
# 파라메트릭 서치는 항상 bisect를 import 하고 시작
from bisect import bisect_left, bisect_right

def bisect_search():
    # 지문의 배열을 입력받는다.
    arr = [1, 1, 2, 2, 2, 2, 3]

    # 여기서는 숫자를 세는 예이지만, 지문에 따라 달리 구성한다.
    def count(arr, a, b):

        # 끝 값은 항상 target의 오른쪽
```

```
    end = bisect_right(arr, a)

    # 시작 값은 항상 target의 왼쪽이다.
    start = bisect_left(arr, b)

    # 로직 실행 후 무언가의 결과를 도출
    return end - start
```