

2023 여름학기 동국대학교 SW역량강화캠프

11일차. 이분탐색

시작하기에 앞서

- ▶ 1부터 100까지 중에 생각한 수를 맞추기 위해서는?
- ▶ 이 탐색 횟수를 줄이기 위해서 줄 수 있는 힌트가 있다면?

- ▶ 이 수가 맞는가에 대한 질문의 대답으로 YES/NO 뿐만이 아니라, Up/Down으로 힌트를 준다면
- ▶ 어느 수를 질문하는 것이 최선일까?

● 이분탐색(이진탐색)

- ▶ 탐색 범위를 절반씩 줄여나가며 진행하는 탐색
- ▶ 답이 “무조건” 존재하는 범위 $l \sim r$ 을 구간의 길이가 1이 될 때까지 줄인다.
- ▶ 탐색의 시간복잡도 $O(\lg N)$

● lower bound (2907)

| 문제

n 개로 이루어진 정수 집합에서 k 이상인 수가 처음으로 등장하는 위치를 찾으세요.

단, 입력되는 정수 집합은 오름차순으로 정렬되어 있습니다.

| 입력

첫째 줄에 정수 집합의 크기 n 이 주어진다. 단, n 은 1,000,000이하의 양의 정수이다.

두번째 줄에는 정수 집합에 속한 int범위 이내의 양의 정수가 n 개 주어진다.

셋째 줄에는 찾고자 하는 값 k 가 주어진다.

| 출력

찾고자 하는 원소의 위치를 출력한다.

만약 집합 내 모든 원소가 k 보다 작다면 $n+1$ 을 출력한다.

기본 코드

```
public class main{
    public static void main(String[] args) throws IOException{
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        StringTokenizer st;

        int N = Integer.parseInt(br.readLine());

        int[] arr = new int[N];
        st = new StringTokenizer(br.readLine());
        for(int i=0; i<N; i++) arr[i] = Integer.parseInt(st.nextToken());

        int K = Integer.parseInt(br.readLine());

        System.out.print(lower_bound(N, arr, K));
    }

    static int lower_bound(int N, int[] arr, int K){
        int answer=0;

        return answer;
    }
}
```

```
def lower_bound(N, arr, K):
    answer = 0

    return answer

N = int(input())
arr = list(map(int, input().split()))
K = int(input())

print(lower_bound(N, arr, K))
```

- ▶ 이분탐색을 통한 자료구조 탐색
- ▶ `lower_bound(k)` : k 이상의 수가 처음 등장하는 위치
- ▶ `upper_bound(k)` : k 초과인 수가 처음 등장하는 위치
- ▶ C++과 python에는 두 기능이 구현되어 있지만, java에는..

```
cout << lower_bound(arr, arr+n, k) - arr + 1;
```

```
print(bisect.bisect_left(arr, K) + 1)
```

문제 해설

- ▶ 답이 존재하는 구간 $[L, R]$ 에서 K 이상의 수가 처음 등장하는 위치를 이분탐색 해보자.
- ▶ $mid = (L+R)/2$
- ▶ $arr[mid] < K$ 라면? $[L, R] \rightarrow [mid+1, R]$
- ▶ $arr[mid] > K$ 라면? $[L, R] \rightarrow [L, mid-1]$
- ▶ $arr[mid] == K$ 라면? $[L, R] \rightarrow [L, mid-1]$

$K=8$

L				mid				R




```
static int lower_bound(int N, int[] arr, int K){
    int answer=N, l=0, m, r=N-1;

    while(l <= r){
        m = (l+r)/2;

        if(arr[m] < K) l = m+1;
        else{
            answer = m;
            r = m-1;
        }
    }

    return answer+1;
}
```

```
def lower_bound(N, arr, K):
    answer = N
    l = 0
    r = N-1

    while l <= r:
        m = (l+r)//2
        if arr[m] < K:
            l = m+1
        else:
            answer = m
            r = m-1

    return answer+1
```

● Counting Haybales(1239)

문제

농부 존은 $N(1 \leq N \leq 100,000)$ 개의 건초더미를 일직선 상의 여러 지점에 놓았습니다. 모든 건초더미가 적절하게 놓였는지를 확인하기 위해서 그를 도와 $Q(1 \leq Q \leq 100,000)$ 개의 질문을 처리해주세요. 각 질문은 일직선 상의 특정 구간에 건초더미가 몇개 있는지 물어보는 질문입니다.

입력

첫째 줄에 N 과 Q 가 주어진다.

다음 줄에는 건초더미의 위치를 뜻하는 N 개의 서로 다른 정수가 주어진다. 각각의 정수는 $0 \cdots 1,000,000,000$ 사이 범위이다.

그 다음 Q 개의 줄에는 각각 두개의 정수 A, B 가 주어진다($0 \leq A \leq B \leq 1,000,000,000$). 이는 A 에서 B 구간 사이에 있는 건초더미를 묻는 질문이다.

출력

Q 개의 줄에 각각의 질문에 대하여 해당 구간에 건초더미가 몇개 있는지를 출력한다.

- ▶ 이분탐색을 이용한 풀이
- ▶ 건초더미의 위치를 정렬해놓자.
- ▶ $\text{upper_bound}(k) - \text{lower_bound}(k)$ 를 하면 k 의 개수를 알 수 있다.
- ▶ $\text{upper_bound}(b) - \text{lower_bound}(a)$ 를 하면 a 부터 b 까지의 개수를 알 수 있다.

핵심 코드

```
for(int i=0; i<Q; i++){
    st = new StringTokenizer(br.readLine());
    int A = Integer.parseInt(st.nextToken());
    int B = Integer.parseInt(st.nextToken());

    System.out.println(upper_bound(N, arr, B) - lower_bound(N, arr, A));
}
```

```
for i in range(Q):
    A, B = map(int, input().split())
    print(bisect.bisect_right(arr, B) - bisect.bisect_left(arr, A))
```

● Parametric Search

- ▶ 최적화 문제를 결정문제로 바꾸어 푸는 방법
- ▶ 최적화 문제: \sim 를 만족하는 최적의 답을 구하시오(최소, 최대)
- ▶ 결정 문제: OX 문제, \sim 가 조건을 만족하는가?

- 이분탐색 parametric search를 할 때 고려해야 하는 요소
 - ▶ 이분탐색이 가능한 문제인가? (특정 값을 기준으로 YES/NO가 나뉘는가?)
 - ▶ 답이 범위 내에 “무조건” 존재하는 초기 범위 $L \sim R$ 을 잡았는가?
 - ▶ 구간의 중간점인 mid 가 답에 포함되는가?
 - ▶ 구간의 길이가 언제나 1로 줄어드는가?

● 나무 자르기(11)

문제

상근이는 나무 M미터가 필요하다. 근처에 나무를 구입할 곳이 모두 망해버렸기 때문에, 정부에 벌목 허가를 요청했다. 정부는 상근이네 집 근처의 나무 한 줄에 대한 벌목 허가를 내주었고, 상근이는 새로 구입한 목재절단기를 이용해서 나무를 구할 것이다.

목재절단기는 다음과 같이 동작한다. 먼저, 상근이는 절단기에 높이 H를 지정해야 한다. 높이를 지정하면 톱날이 땅으로부터 H미터 위로 올라간다. 그 다음, 한 줄에 연속해있는 나무를 모두 절단해버린다. 따라서, 높이가 H보다 큰 나무는 H 위의 부분이 잘릴 것이고, 낮은 나무는 잘리지 않을 것이다. 예를 들어, 한 줄에 연속해있는 나무의 높이가 20, 15, 10, 17이라고 하자. 상근이가 높이를 15로 지정했다면, 나무를 자른 뒤의 높이는 15, 15, 10, 15가 될 것이고, 상근이는 길이가 5인 나무와 2인 나무를 들고 집에 갈 것이다. (총 7미터를 집에 들고 간다)

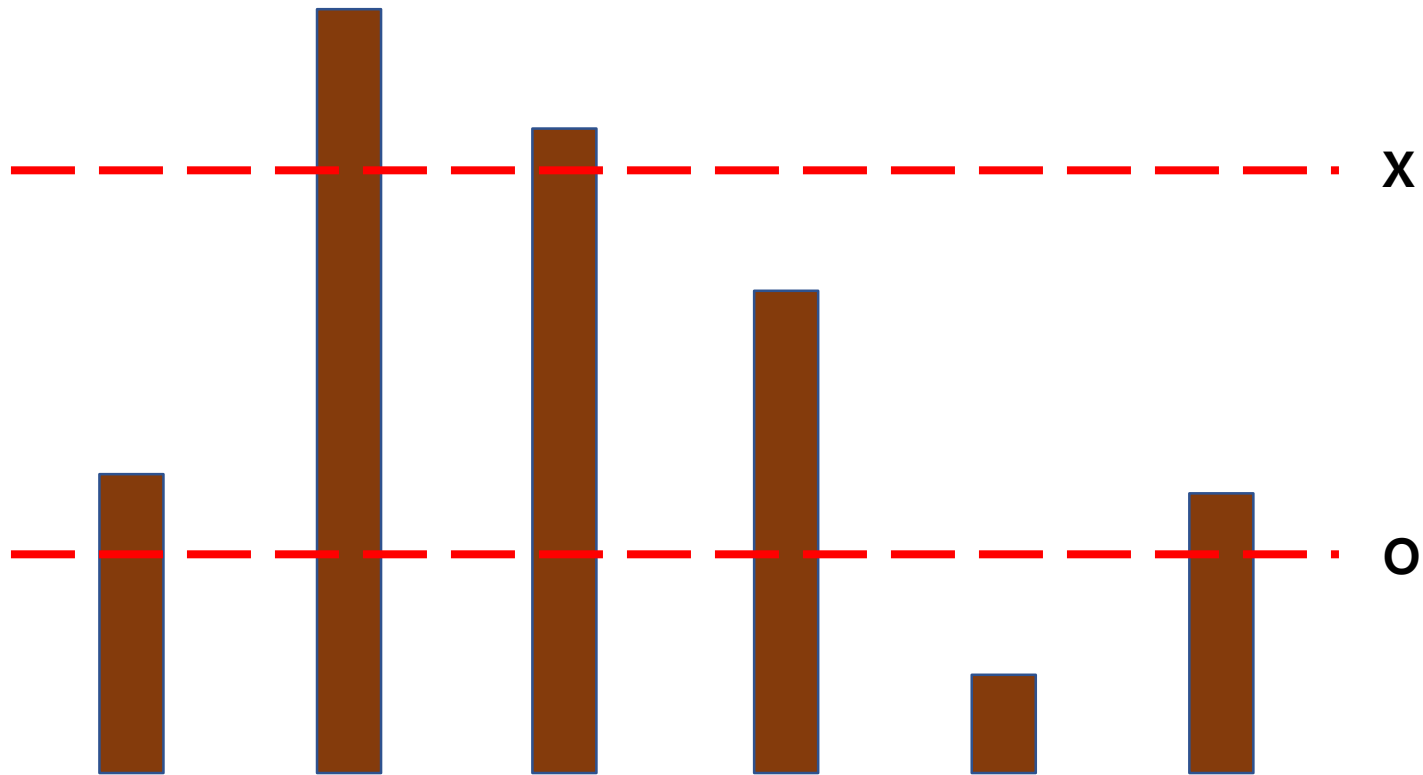
상근이는 환경에 매우 관심이 많기 때문에, 나무를 필요한 만큼만 집으로 가져가려고 한다. 이때, 적어도 M미터의 나무를 집에 가져가기 위해서 절단기에 설정할 수 있는 높이의 최댓값을 구하는 프로그램을 작성하시오.

입력

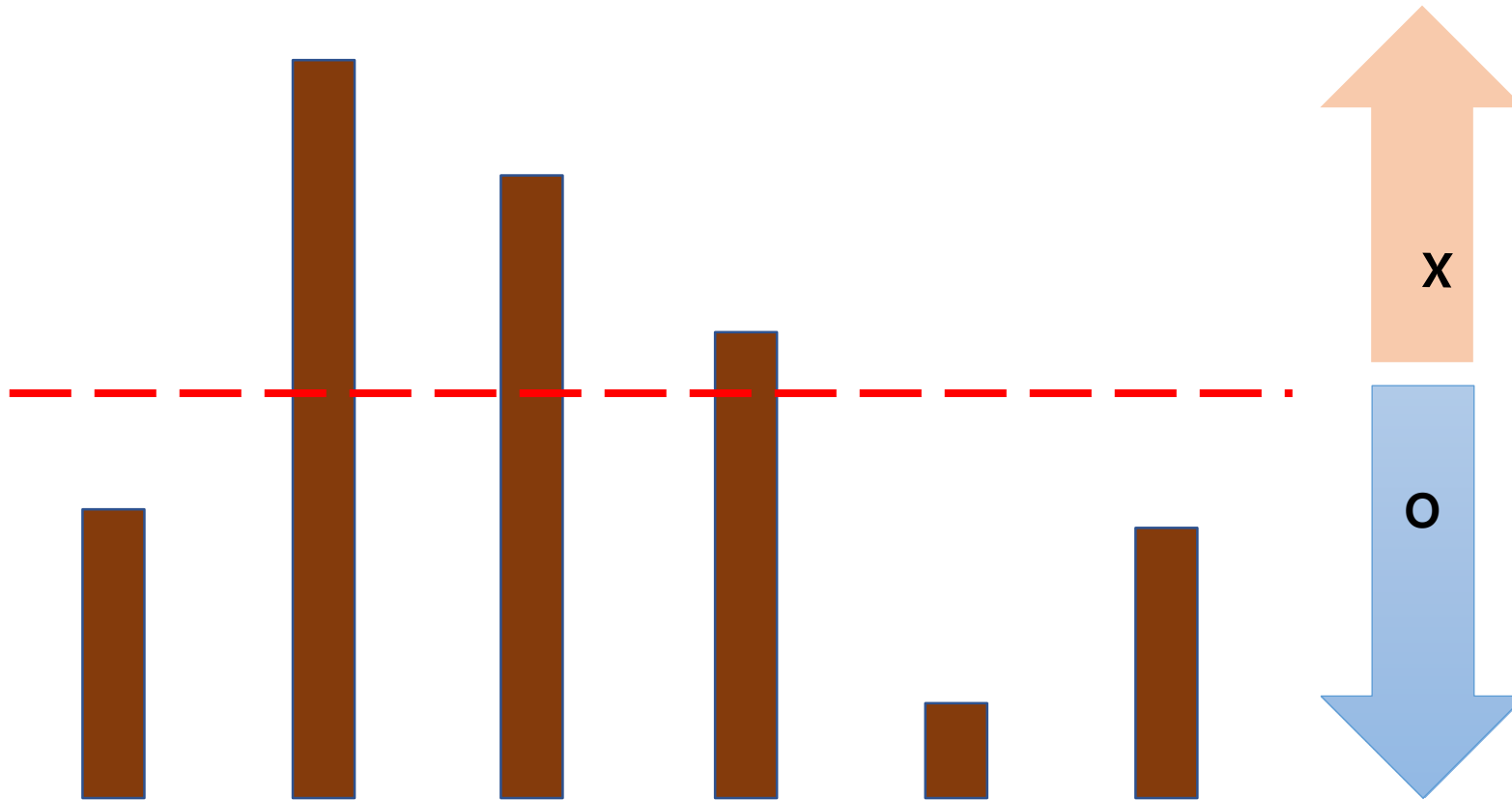
첫째 줄에 나무의 수 N과 상근이가 집으로 가져가려고 하는 나무의 길이 M이 주어진다. ($1 \leq N \leq 1,000,000$, $1 \leq M \leq 2,000,000,000$)

둘째 줄에는 나무의 높이가 주어진다. 나무의 높이의 합은 항상 M을 넘기 때문에, 상근이는 집에 필요한 나무를 항상 가져갈 수 있다. 높이는 1,000,000,000보다 작거나 같은 양의 정수 또는 0이다.

● 나무 자르기(11)



● 나무 자르기(11)



- ▶ Parametric Search를 이용한 풀이
- ▶ 높이의 최댓값을 찾는 최적화 문제
- ▶ “높이 x 에서 자르면 나무의 양이 충분한가?”의 결정 문제로 바꾸자
- ▶ 답은 무조건 $[0, 10억]$ 범위 내에 존재한다.
- ▶ 답이 $[l, r]$ 범위에 존재할 때, 높이 $(l+r)/2$ 에서 자르면 충분한지 검사하여 구간을 절반으로 줄일 수 있다

```
int l=0, r=1000000000;
while(l < r){
    int m = (l+r+1) / 2;

    long sum=0;
    for(int i=0; i<N; i++) sum += h[i] > m ? h[i] - m : 0;

    if(sum >= M) l = m;
    else r = m-1;
}

System.out.print(l);
```

```
N, M = map(int, input().split())
arr = list(map(int, input().split()))

l = 0
r = 1000000000
while l < r:
    m = (l+r+1)//2

    sum = 0
    for h in arr:
        if h > m: sum += h-m

    if sum >= M: l = m
    else: r = m-1

print(l)
```

● 공유기 설치 (4342)

| 문제

집 N 개가 수직선 위에 있다. 각각의 집의 좌표는 x_1, \dots, x_N 이고, 집의 좌표는 모두 다르다.

월리는 집에 공유기 C 개를 설치하려고 한다. 최대한 많은 곳에서 와이파이를 사용하려고 하기 때문에, 한 집에는 공유기를 하나만 설치할 수 있고, 가장 인접한 두 공유기 사이의 거리를 가능한 크게 하여 설치하려고 한다.

C 개의 공유기를 N 개의 집에 설치할 때, 가장 인접한 두 공유기 사이의 거리의 최댓값을 구해보자.

| 입력

첫째 줄에 집의 개수 N ($2 \leq N \leq 200,000$)과 공유기의 개수 C ($2 \leq C \leq N$)이 하나 이상의 빈 칸을 사이에 두고 주어진다.

둘째 줄에 N 개의 집의 좌표를 나타내는 x_i ($0 \leq x_i \leq 1,000,000,000$)가 주어진다.

- ▶ Parametric Search
- ▶ 공유기 C개를 설치할 때, 가장 인접한 두 공유기 사이의 거리의 최댓값
- ▶ 최소 거리 x 로 공유기들을 설치할 때 공유기 C개를 모두 설치할 수 있는가?
 - 최소 거리를 x 로 유지한 채 공유기 설치 시, 공유기 개수가 C개 이상이면 가능
- ▶ 이분탐색 가능

문제 해설

- ▶ 최소 거리 x 로 공유기들을 설치할 때 공유기 C 개를 모두 설치할 수 있는가?
- ▶ 작은 좌표에서부터 보면서 이전 공유기로부터 x 이상 떨어졌으면 설치한다.

```
int l=1, m, r=1000000000;
while(l < r){
    m = (l+r+1) / 2;

    int cnt=1, j=0;
    for(int i=0; i<N; i++){
        if(x.get(i) - x.get(j) >= m){
            cnt++;
            j = i;
        }
    }

    if(cnt < C) r = m-1;
    else l = m;
}
```

```
l = 0
r = 1000000000
while l < r:
    m = (l+r+1) // 2

    cnt=1
    prev_x = x[0]
    for curr_x in x:
        if curr_x - prev_x >= m:
            prev_x = curr_x
            cnt += 1

    if cnt >= C: l = m
    else: r = m-1

print(l)
```

● 동전 금고(4703)

문제

월리는 용돈이 부족해져서 삼촌을 찾아갔다. 월리의 삼촌은 동전의 산으로 가득 찬 큰 금고를 가지고 있는데, 그 중 몇몇 동전은 매우 특별한 가치가 있는 동전이다.

하지만 최근 삼촌은 금고에 있던 코인을 옮기는 중, 우연히 아끼던 특별한 동전이 금고로 옮겨져 극도의 스트레스를 받고 있다. 다행히 그 코인을 찾았지만, 아쉽게도 금고 입구와 완전히 반대이고 금고 안에 동전이 산더미처럼 쌓여 있기 때문에 실제로 동전에 닿는 것은 쉬운 일이 아니다.

삼촌은 월리가 동전의 산에서 특별한 동전을 회수해 올 수 있다면, 용돈을 줄 생각이 있다. 월리는 동전을 회수하기 위한 장비로 사다리를 가져오기로 결정했지만 사다리의 길이를 얼마짜리로 준비해야 할 지 정하지 못했다. 더 긴 사다리는 더 높은 동전 절벽을 오를 수 있다는 것을 의미하지만 더 많은 비용이 든다. 따라서 월리는 특별한 동전에 도달할 수 있는 가장 짧은 사다리를 사고 싶어한다.

금고는 왼쪽 위 모서리에 입구가 있고 특별한 동전은 오른쪽 아래 모서리에 있는 다양한 높이의 동전 더미의 직사각형 그리드로 표시될 수 있다. 월리는 현재 서 있는 동전 더미에서 상하좌우로 인접한 동전 더미로 이동할 수 있다. 월리는 점프하거나 날 수 없기 때문에 성공적으로 높이 x 가 높은 동전더미로 이동하려면 길이 x 의 사다리가 필요하다. 하지만 내려가는 것은 높이 제한 없이 중력에 모든 것을 맡길 수 있다.

월리가 사야 할 사다리의 최단 길이를 출력해보자.

- ▶ Parametric Search
- ▶ 사야 할 사다리의 최단 길이
- ▶ 사다리 길이 x 로 $(1,1)$ 에서 (M,N) 까지 이동 가능한가?
- ▶ 이분탐색 가능

- ▶ BFS
- ▶ 사다리 길이 x 로 $(1,1)$ 에서 (M,N) 을 방문할 수 있는지는 BFS로 탐색 가능

```
int l=0, m, r=1000000000;
while(l < r){
    m = (l+r)/2;

    bfs(m);
    if(chk[M-1][N-1]) r = m;
    else l = m+1;
}

System.out.print(l);
```

```
while(!que.isEmpty()){
    int x = que.remove();
    int y = que.remove();

    for(int i=0; i<4; i++){
        int x_ = x+dx[i];
        int y_ = y+dy[i];

        if(x_ < 0 || x_ >= M || y_ < 0 || y_ >= N) continue;
        if(arr[x_][y_] - arr[x][y] > d) continue;
        if(chk[x_][y_]) continue;

        chk[x_][y_] = true;
        que.add(x_);
        que.add(y_);
    }
}
```