25-1 이니로
알고리즘 멘토링

멘토 - 김수성

공유기 설치 / 2110

백준 2110 / https://www.acmicpc.net/problem/2110

문제

도현이의 집 N개가 수직선 위에 있다. 각각의 집의 좌표는 x_1, ..., x_N이고, 집 여러개가 같은 좌표를 가지는 일은 없다.

도현이는 언제 어디서나 와이파이를 즐기기 위해서 집에 공유기 C개를 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.

C개의 공유기를 N개의 집에 적당히 설치해서, 가장 인접한 두 공유기 사이의 거리를 최대로 하는 프로그램을 작성하시오.

입력

첫째 줄에 집의 개수 N ($2 \le N \le 200,000$)과 공유기의 개수 C ($2 \le C \le N$)이 하나 이상의 빈 칸을 사이에 두고 주어진다. 둘째 줄부터 N개의 줄에는 집의 좌표를 나타내는 x_i ($0 \le x_i \le 1,000,000,000$)가 한 줄에 하나씩 주어진다.

출력

첫째 줄에 가장 인접한 두 공유기 사이의 최대 거리를 출력한다.

공유기 설치 / 2110

출력

첫째 줄에 가장 인접한 두 공유기 사이의 최대 거리를 출력한다.

예제 입력 1 복사

```
5 3
1
2
8
4
9
```

예제 출력 1 복사

```
3
```

힌트

공유기를 1, 4, 8 또는 1, 4, 9에 설치하면 가장 인접한 두 공유기 사이의 거리는 3이고, 이 거리보다 크게 공유기를 3개 설치할 수 없다.

공유기 설치 / 2110

예제 입력	예제 출력
5 3 1 2 8 4 9	3

1 4 9에 공유기를 설치하면 각 거리가 3 5가 됨
거리의 최솟값이 3보다 크게 공유기를 설치할 수 없음

공유기 설치 / 2110

최적화 문제

설치한 공유기 사이의 거리의 최솟값의 최댓값

결정 문제

M개 이상의 공유기를 설치 했을 때

공유기 사이의 거리의 최솟값을 K로 만들 수 있는가

공유기 설치 / 2110

결정 문제

M개 이상의 공유기를 설치 했을 때
공유기 사이의 거리의 최솟값을 K로 만들 수 있는가

최솟값을 크게 만들려면
공유기를 적게 설치해야 함

K 값이 커지면 공유기를 M개 이상 설치 할 수 없음

공유기 설치 / 2110

결정 문제

M개 이상의 공유기를 설치 했을 때
공유기 사이의 거리의 최솟값을 K로 만들 수 있는가

K 값이 커지면 공유기를 M개 이상 설치 할 수 없음
-> decision(K) 1111100000
최댓값을 구하면 됨

공유기 설치 / 2110

결정 문제

M개 이상의 공유기를 설치 했을 때
공유기 사이의 거리의 최솟값을 K로 만들 수 있는가

배열을 정렬하고 전에 공유기를 설치한 장소와
거리가 K이상이 되었을 때 공유기를 설치함
-> 항상 공유기 사이의 거리는 K 이상

이 때 설치한 공유기 개수가 M개 이상이면
Decision(K)는 1이 됨

공유기 설치 / 2110

시간 복잡도

배열 정렬 -> O(N log N)

결정 함수 -> O(N)

매개 변수 탐색 -> O(log 1e9) = 30...

O(N log N + N log 1e9)

공유기 설치 / 2110

C++

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

    cin >> n >> m;
    for(int i = 1;i <= n;i++) cin >> a[i];
    sort(a + 1, a + n + 1);

    cout << maximization();

    return 0;
}
```

```cpp
bool decision(ll cur){
    // cnt -> 설치한 공유기 개수
    // last -> 마지막으로 설치한 공유기 위치
    ll cnt = 0, last = -1e12; // last는 음의 무한대로 초기화
    for(int i = 1;i <= n;i++){
        // 마지막으로 설치한 공유기의 위치와
        // 현재 위치의 차이가 K보다 작다면 건너 뜀
        if(a[i] - last < cur) continue;
        last = a[i]; cnt++;
    }

    // 설치한 공유기 개수가 M 이상이면
    // 공유기 거리의 차이의 최솟값을 K로 만들 수 있음
    return cnt >= m;
}

ll maximization(){
    // 최솟값은 1, 최댓값은 1e9
    // 실수 오차 있을 수 있으니 넉넉하게 2e9로 잡음
    ll lo = 1, hi = 2e9;
    while(lo < hi){
        ll mid = (lo + hi + 1) / 2; // 올림 값 사용
        if(decision(mid)) lo = mid;
        else hi = mid - 1;
    }

    return lo;
}
```

공유기 설치 / 2110

Python

```python
n, m = list(map(int, input().rstrip().split()))
a = []
for _ in range(n):
    x = int(input().rstrip())
    a.append(x)
a.sort()

def decision(cur):
    # cnt -> 설치한 공유기 개수
    # last -> 마지막으로 설치한 공유기 위치
    cnt = 0
    last = -1e12 # last는 음의 무한대로 초기화

    for i in a:
        # 마지막으로 설치한 공유기의 위치와
        # 현재 위치의 차이가 K보다 작다면 건너 뜀
        if i - last < cur:
            continue
        last = i
        cnt += 1

    # 설치한 공유기 개수가 M 이상이면
    # 공유기 거리의 차이의 최솟값을 K로 만들 수 있음
    return cnt >= m
```

```python
def maximization():
    # 최솟값은 1, 최댓값은 1e9
    # 실수 오차 있을 수 있으니 넉넉하게 2e9로 잡음
    lo = 1
    hi = int(2e9)
    while lo < hi:
        mid = (lo + hi + 1) // 2
        if decision(mid):
            lo = mid
        else:
            hi = mid - 1
    return lo

print(maximization())
```

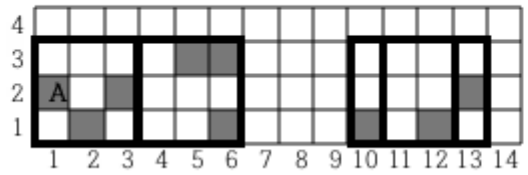
질문?

모자이크 / 2539

백준 2539 / https://www.acmicpc.net/problem/2539

1. 사용되는 색종이는 모두 크기가 같고 정사각형 모양이다.
2. 색종이 크기는 한 변의 길이로 나타내며, 원하는 크기의 색종이는 모두 구할 수 있다.
3. 모든 색종이는 반드시 도화지의 밑변에 맞추어 붙인다. 이때 색종이를 겹쳐서 붙일 수 있다.

도화지 위의 행은 다음 그림과 같이 맨 아래에서 위쪽으로 1번부터 순서대로 번호가 매겨져 있고, 열은 왼쪽에서 오른쪽으로 1번부터 번호가 매겨져 있다. 이 그림은 도화지에 가로선과 세로선을 그어서 4개의 행과 14개의 열, 그리고 56개의 칸으로 나눈 모양을 보여준다. 잘못 칠해진 칸은 회색으로 표시되어 있다.



모자이크 / 2539

입력

첫째 줄에는 도화지 위의 행의 개수와 열의 개수를 나타내는 자연수가 빈칸을 사이에 두고 주어진다. 행의 개수와 열의 개수는 모두 1000000 이하이다. 둘째 줄에는 사용할 색종이의 장수를 나타내는 자연수가 주어진다. 사용할 색종이는 100장 이하이다. 셋째 줄에는 도화지에 잘못 칠해진 칸의 개수를 나타내는 자연수가 주어진다. 잘못 칠해진 칸은 1000개 이하이다. 넷째 줄부터 마지막 줄까지 잘못 칠해진 칸의 위치가 한 줄에 하나씩 주어진다. 잘못 칠해진 칸의 위치는 빈칸을 사이에 두고 행 번호가 주어진 다음 열 번호가 주어진다.

출력

첫째 줄에 주어진 장수의 색종이를 사용하여 잘못 칠해진 칸을 모두 가릴 수 있는 가장 작은 색종이의 크기가 몇 cm인지를 나타내는 자연수를 출력한다.

예제 입력 1 복사

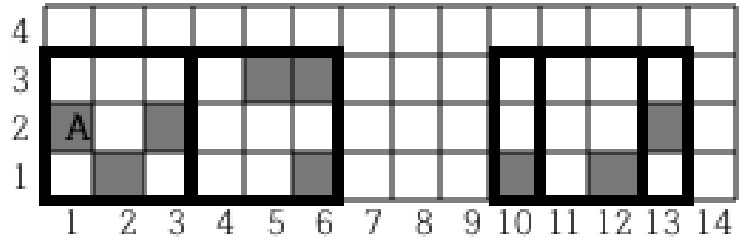
```
4 14
4
9
1 2
2 1
2 3
1 6
3 5
1 10
3 6
1 12
2 13
```

예제 출력 1 복사

```
3
```

모자이크 / 2539

정사각형을 밑변에 맞춰서 L개를 배치 할 때
모든 검은 칸을 가릴 수 있는 정사각형의 최소 크기



다음과 같이 정사각형을 배치하면
크기가 3인 정사각형으로 모든 검은 칸을 가릴 수 있음

모자이크 / 2539

최적화 문제
정사각형을 밑변에 맞춰서 L개를 배치 할 때
모든 검은 칸을 가릴 수 있는 정사각형의 길이의 최솟값

결정 문제
정사각형의 길이가 K일 때 모든 검은 칸을
L개 이하의 정사각형으로 가릴 수 있는가

모자이크 / 2539

결정 문제
정사각형의 길이가 K일 때 모든 검은 칸을
L개 이하의 정사각형으로 가릴 수 있는가

당연히 정사각형의 길이가 커지면 더 적은 개수로
검은 칸을 가릴 수 있음

decision(K) -> 00000**1**1111

최솟값을 찾으면 됨

모자이크 / 2539

결정 문제
정사각형의 길이가 K일 때 모든 검은 칸을
L개 이하의 정사각형으로 가릴 수 있는가

공유기 설치 문제처럼 x 좌표 순으로 정렬 후에
현재 검은 칸이 가려져 있으면 넘어가고
그렇지 않으면 정사각형을 배치하면 됨

y 좌표가 정사각형 보다 크면 검은 칸을 가릴 수 없음
예외 처리로 결정 문제는 O

모자이크 / 2539

시간 복잡도

N = 검은 칸의 개수

배열 정렬 -> O(N log N)

결정 함수 -> O(N)

매개 변수 탐색 -> O(log 1e6) = 20...

O(N log N + N log 1e6)

모자이크 / 2539

C++

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

const ll MAX = 201010;
ll n, m, l, black;
pair <ll, ll> a[MAX];

bool decision(ll cur){
    // cnt -> 사용한 정사각형의 개수
    // last -> 마지막으로 정사각형을 배치한 x 좌표
    ll cnt = 0, last = -1e12;
    for(int i = 1;i <= black;i++){
        // y좌표가 정사각형보다 크면
        // 검은 칸을 가릴 수 없음
        if(a[i].second > cur) return 0;

        // 현재 x좌표가 이미 배치한 정사각형
        // 안에 들어오면 이미 막힌 검은 칸
        if(a[i].first - last + 1 <= cur) continue;
        last = a[i].first; cnt++;
    }

    // 정사각형을 L개 이하로 사용 했으면
    // 검은 칸을 다 막을 수 있음
    return cnt <= l;
}
```

```cpp
ll minimization(){
    // 범위의 최댓값은 1e6
    ll lo = 1, hi = 2e6;
    while(lo < hi){
        ll mid = (lo + hi) / 2;
        if(decision(mid)) hi = mid;
        else lo = mid + 1;
    }

    return lo;
}

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

    cin >> n >> m >> l >> black;
    for(int i = 1;i <= black;i++) cin >> a[i].second >> a[i].first;
    sort(a + 1, a + black + 1);

    cout << minimization();

    return 0;
}
```

모자이크 / 2539

Python

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

n, m = list(map(int, input().rstrip().split()))
l = int(input().rstrip())
black = int(input().rstrip())

a = []
for _ in range(black):
    y, x = list(map(int, input().rstrip().split()))
    a.append((x, y))
a.sort()

def decision(cur):
    cnt = 0 # 사용한 정사각형의 개수
    last = int(-1e12) # 마지막으로 정사각형을 배치한 x 좌표
    for x, y in a:
        # y좌표가 정사각형보다 크면
        # 검은 칸을 가릴 수 없음
        if y > cur:
            return 0

        # 현재 x좌표가 이미 배치한 정사각형
        # 안에 들어오면 이미 막힌 검은 칸
        if x - last + 1 <= cur:
            continue
        last = x
        cnt += 1

    # 정사각형을 L개 이하로 사용 했으면
    # 검은 칸을 다 막을 수 있음
    return cnt <= l
```

```python
def minimization():
    lo = 1
    # 범위의 최댓값은 1e6
    hi = int(2e6)
    while(lo < hi):
        mid = (lo + hi) // 2
        if(decision(mid)):
            hi = mid
        else:
            lo = mid + 1
    return lo

print(minimization())
```

3주차 – 우선순위 큐

우선순위 큐

우선순위 큐

큐 – 먼저 들어온 데이터가 먼저 나감

우선순위 큐 – 우선순위가 높은 데이터가 먼저 나감

원소 삽입

우선순위가 가장 큰 원소 반환

우선순위가 가장 큰 원소 삭제

우선순위 큐

우선순위 큐

루트가 1인 이진 트리로 이루어져 있음

x의 부모 : x / 2

x의 자손 : 2 * x, 2 * x + 1

부모가 자손보다 항상 크게 유지

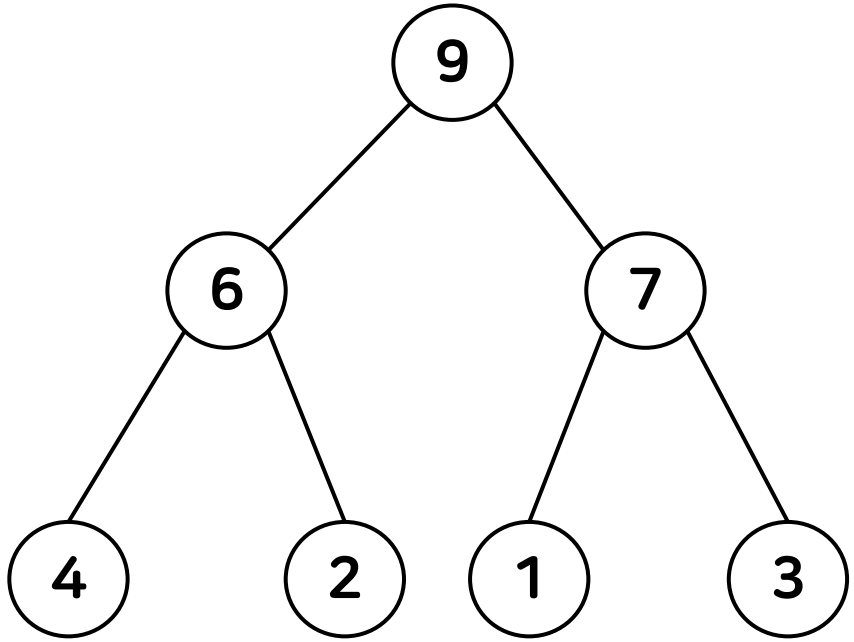
-> 항상 루트가 최댓값을 유지

우선순위 큐

원소 삽입

마지막에 원소 추가

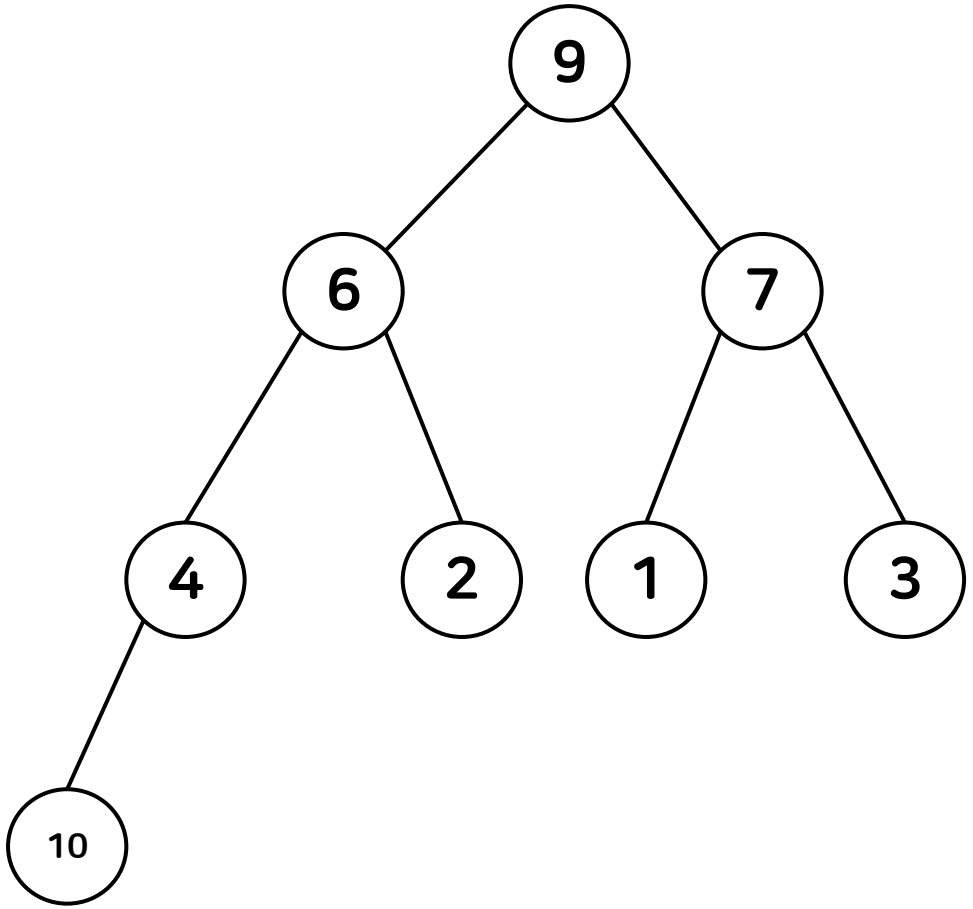
부모가 더 작으면 부모와 위치 교환



우선순위 큐

원소 삽입

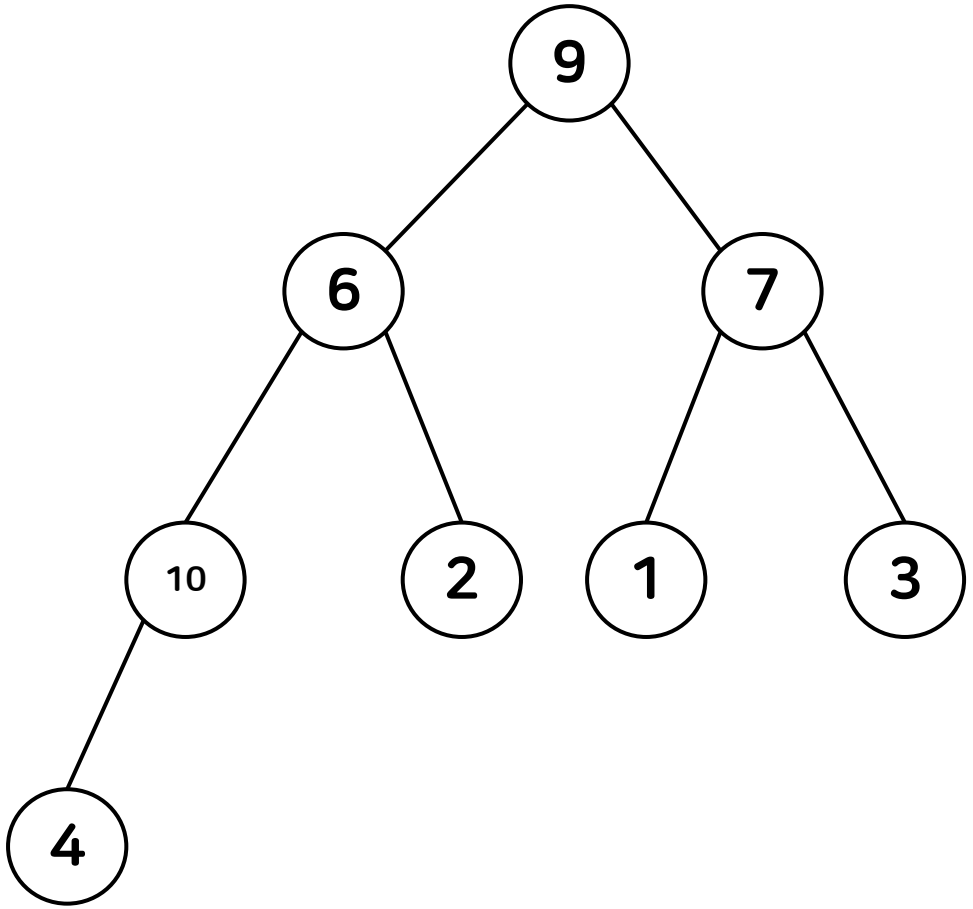
insert 10



우선순위 큐

원소 삽입

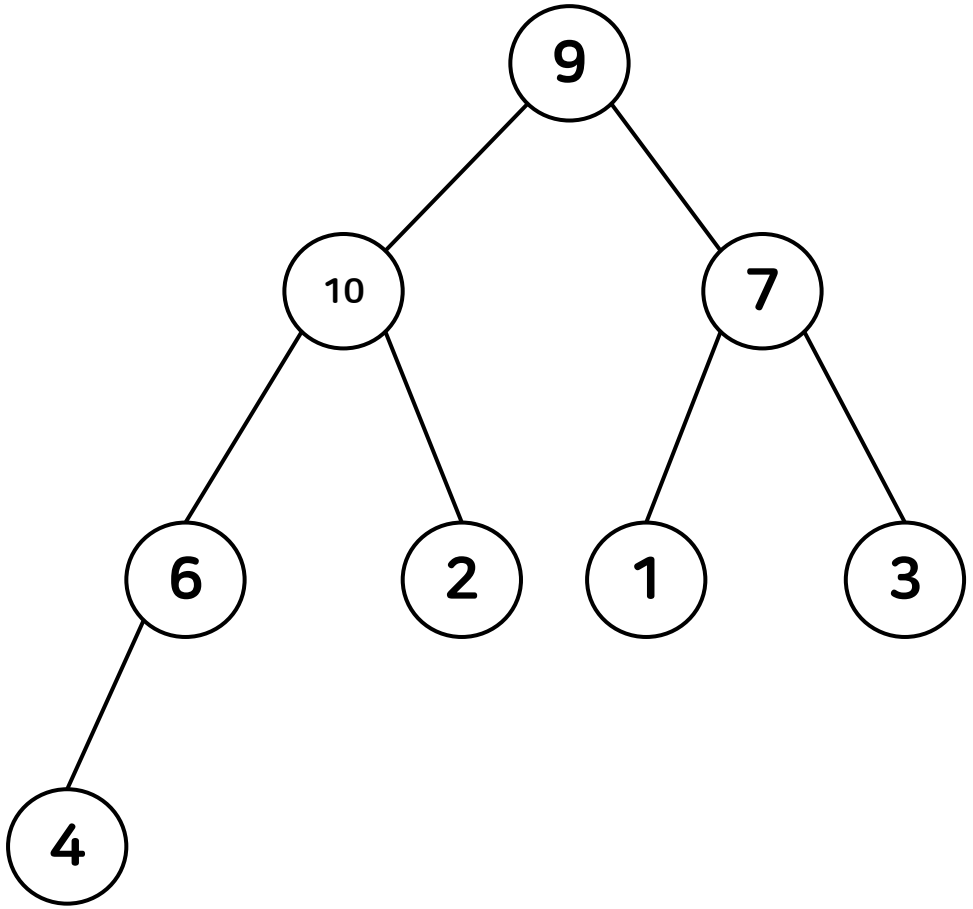
insert 10



우선순위 큐

원소 삽입

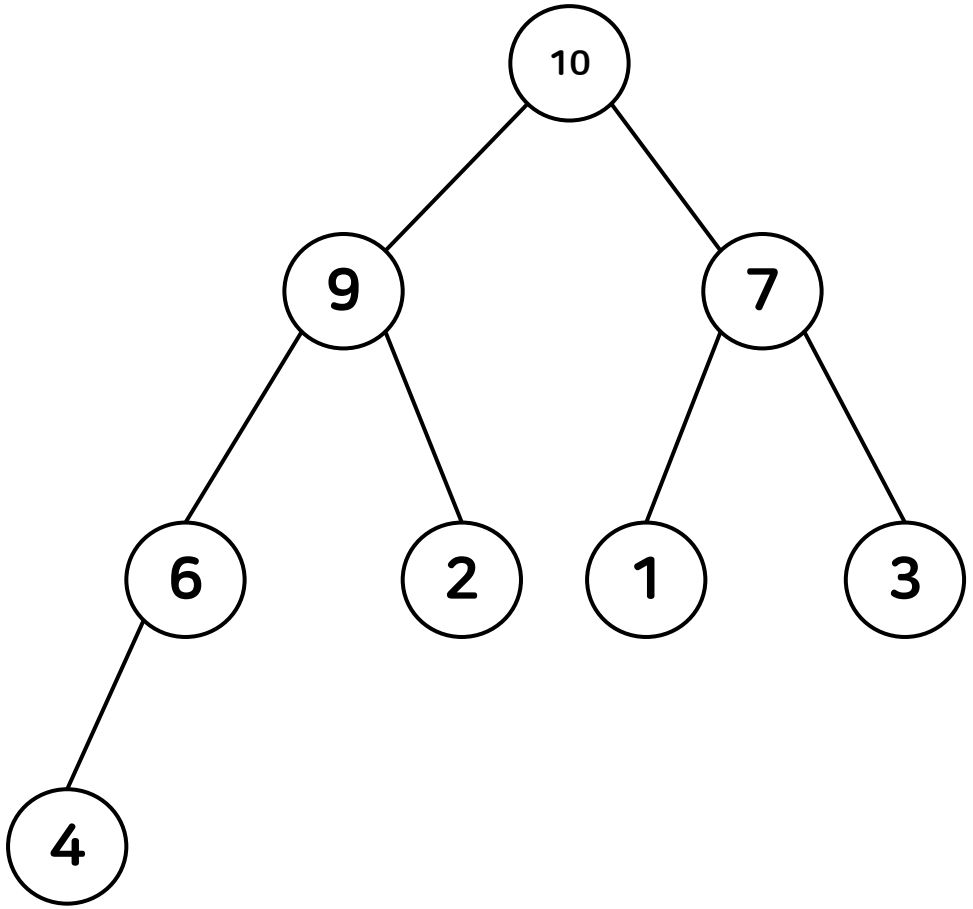
insert 10



우선순위 큐

원소 삽입

insert 10



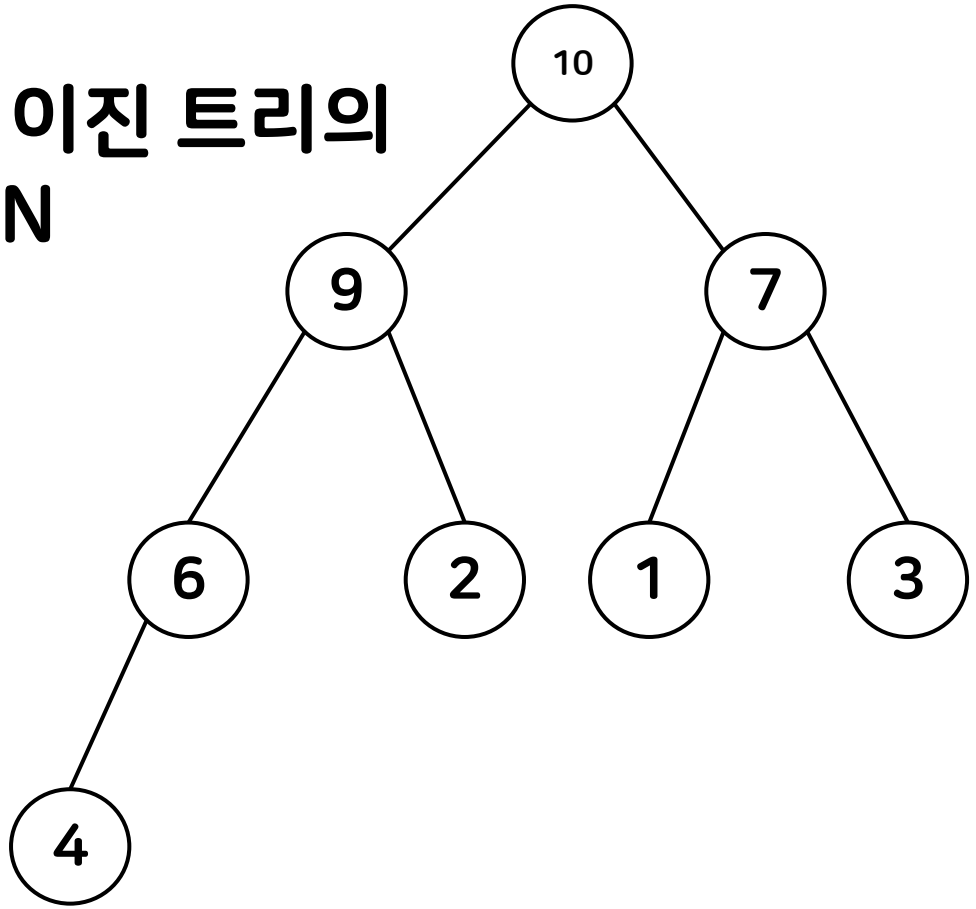
우선순위 큐

시간 복잡도

O(log N)

정점이 N인 이진 트리의

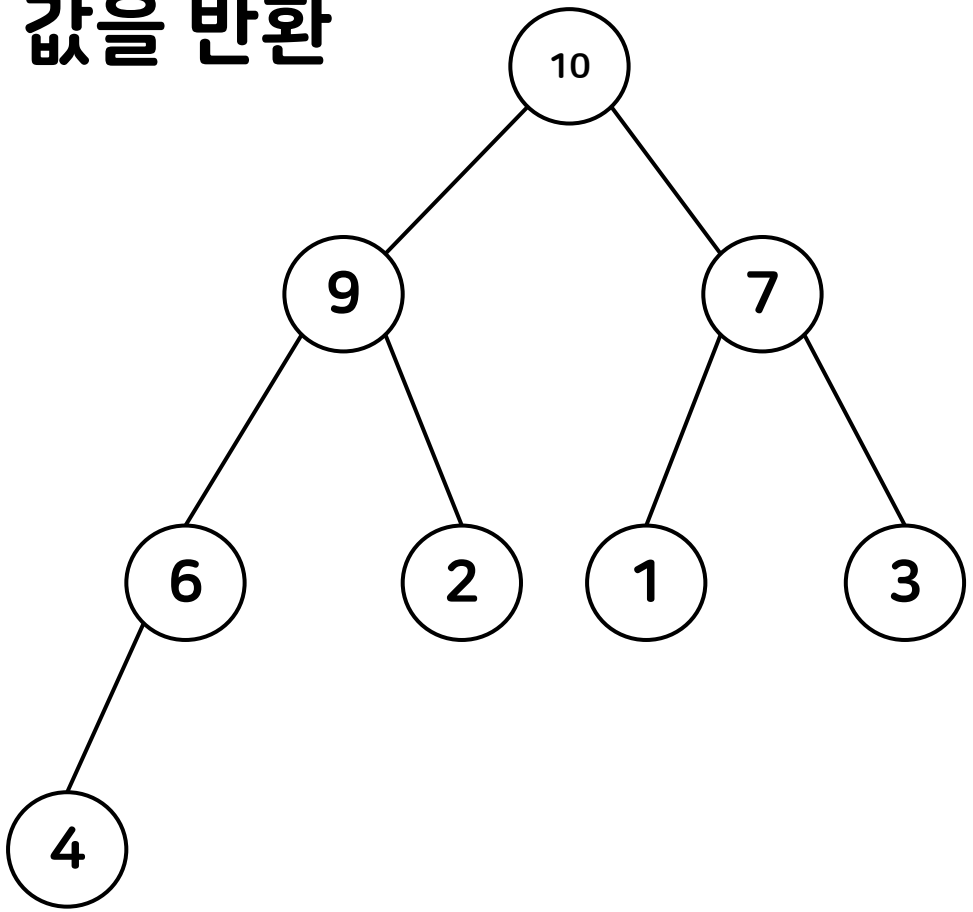
높이는 log N



우선순위 큐

최댓값 반환

루트 노드의 값을 반환

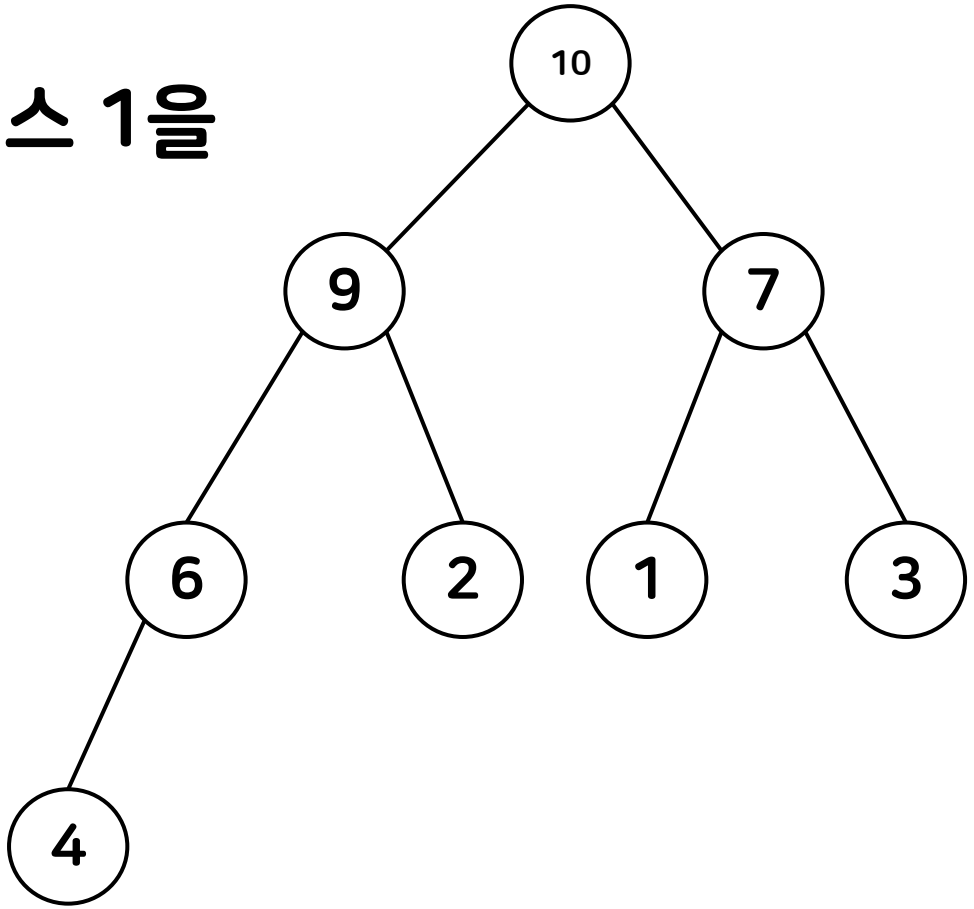


우선순위 큐

시간 복잡도

O(1)

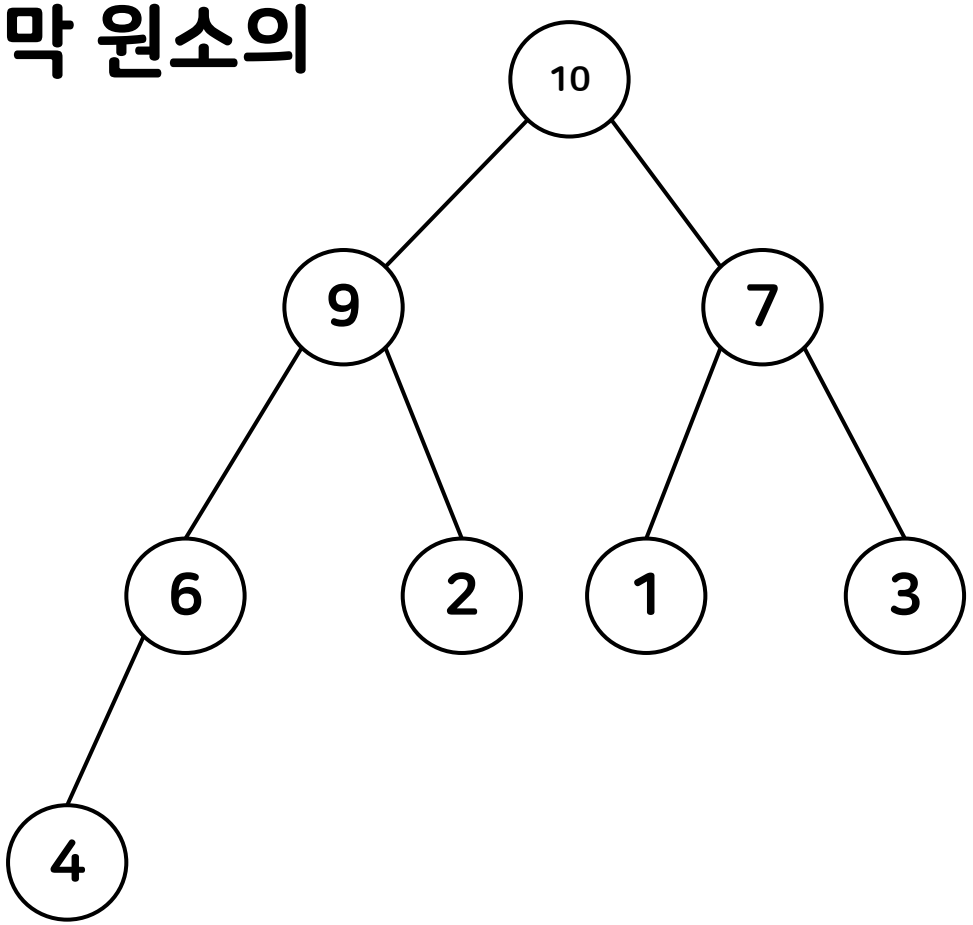
단순히 인덱스 1을 참조하면 됨



우선순위 큐

최댓값 제거

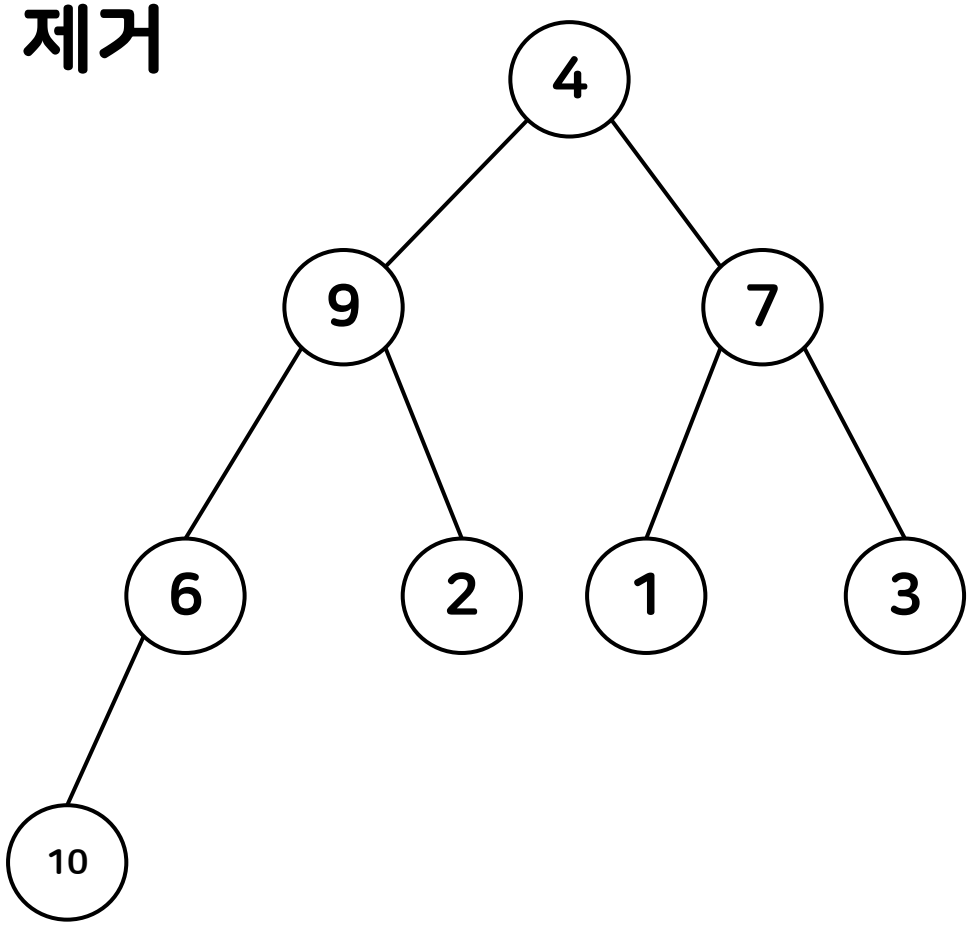
루트와 마지막 원소의 값을 바꿈



우선순위 큐

최댓값 제거

마지막 원소 제거



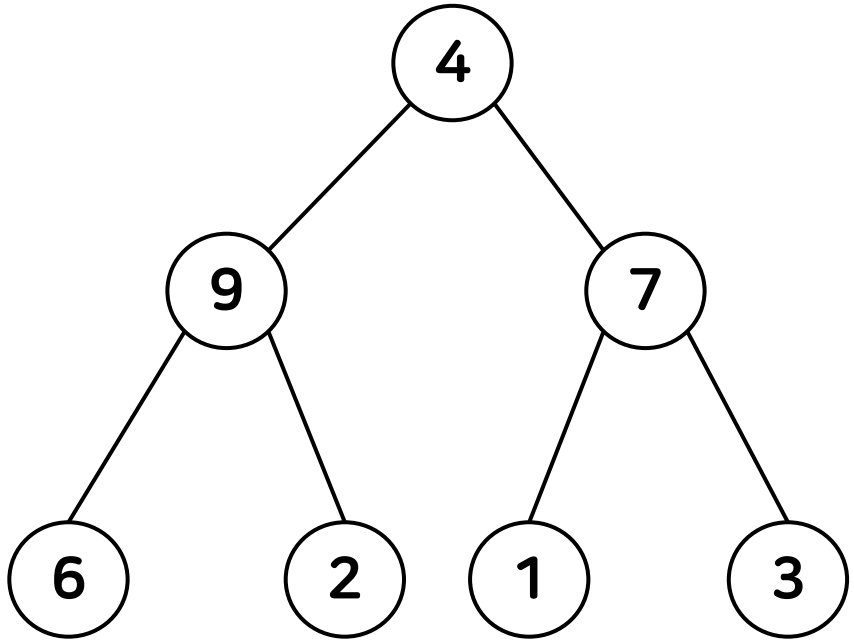
우선순위 큐

최댓값 제거

부모가 항상 자손보다 커야 하므로

자손이 크다면 위치 교환

둘 다 크다면 더 큰 자손이랑 교환



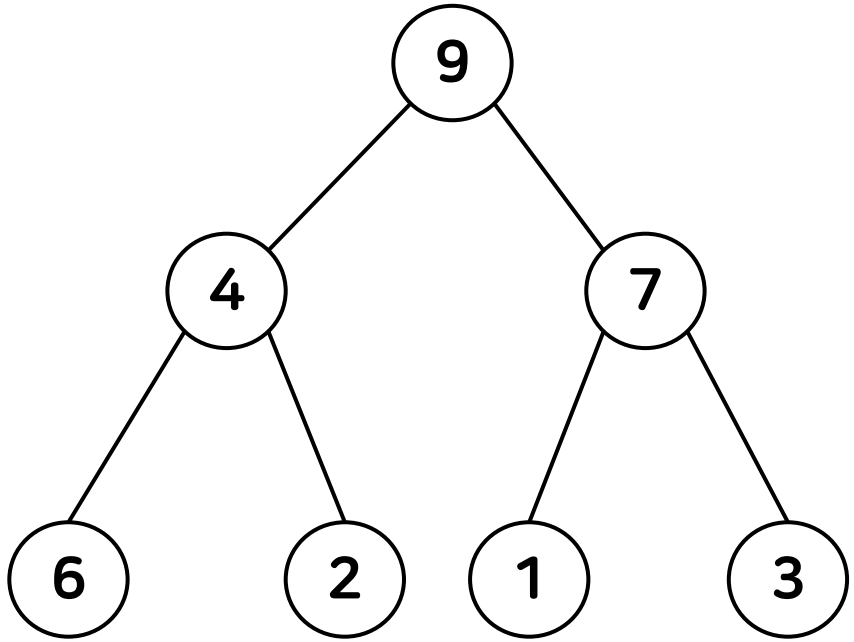
우선순위 큐

최댓값 제거

부모가 항상 자손보다 커야 하므로

자손이 크다면 위치 교환

둘 다 크다면 더 큰 자손이랑 교환



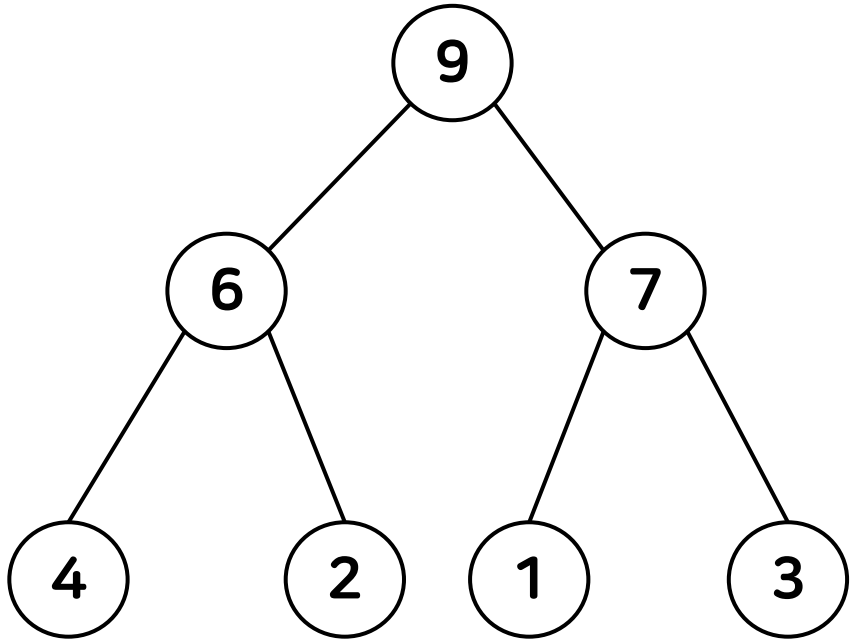
우선순위 큐

최댓값 제거

부모가 항상 자손보다 커야 하므로

자손이 크다면 위치 교환

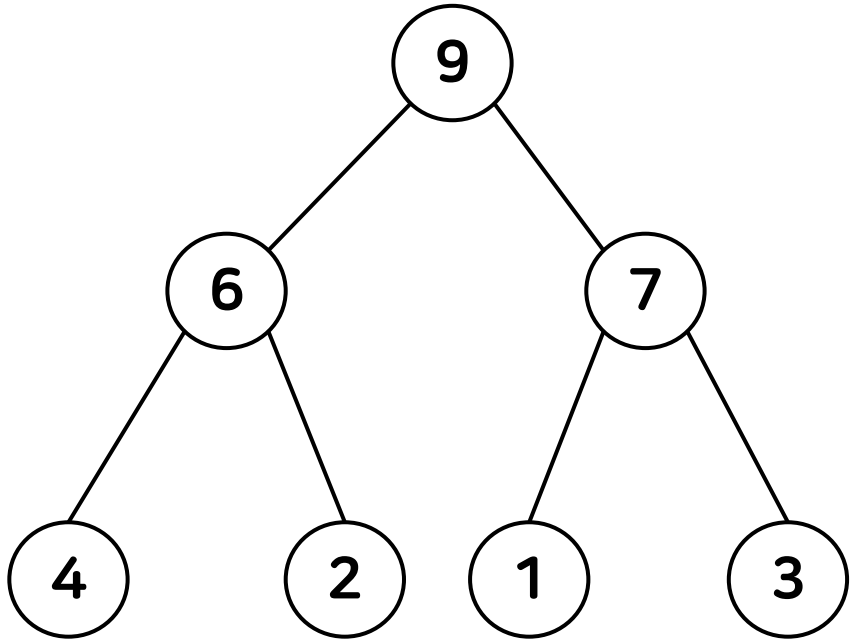
둘 다 크다면 더 큰 자손이랑 교환



우선순위 큐

시간 복잡도

insert와 마찬가지로
트리의 높이가 log N 이므로
O(log N)



우선순위 큐

C++

C++ 표준 라이브러리에 있음

헤더 - <queue>

함수

pop(x) : x를 우선순위 큐에 넣음

top() : 우선순위가 가장 높은 값 반환

pop() : 우선순위가 가장 높은 값 제거

큐, 스택에 있는 empty(), size() 함수도 있음

우선순위 큐

C++

priority_queue <T> pq;

기본적으로 높은 값이
우선순위가 높게 설정 되어 있음

```cpp
#include <iostream>
#include <queue>
using namespace std;
priority_queue <int> pq;

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

    pq.push(1);
    cout << pq.top() << "\n"; // 1
    pq.push(3);
    cout << pq.top() << "\n"; // 3
    pq.push(2);

    pq.pop(); // 3 제거
    cout << pq.top(); // 2

    return 0;
}
```

우선순위 큐

C++

priority_queue
<T, vector <T>, greater<T>>
pq;

작은 값을 높은 우선순위로
사용하고 싶으면 이렇게
우선순위 큐를 정의 해야 함

```cpp
#include <iostream>
#include <queue>
using namespace std;
priority_queue <int, vector<int>, greater<int>> pq;

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

    pq.push(1);
    cout << pq.top() << "\n"; // 1
    pq.push(3);
    cout << pq.top() << "\n"; // 1
    pq.push(2);

    pq.pop(); // 1 제거
    cout << pq.top(); // 2

    return 0;
}
```

우선순위 큐

C++

priority_queue <T, vector <T>, cmp> pq;

따로 cmp 함수를 만들어 우선순위의 값을
지정 해 줄 수 있음

Priority_queue <int, vector<int>, greater<int>> pq

```
struct cmp{
    bool operator()(int a, int b){
        return a > b;
    }
};
priority_queue <int, vector<int>, cmp> pq;
```

우선순위 큐

Python

우선순위 큐와 비슷한 힙 사용

따로 자료구조가 있는 게 아니라 리스트에 사용

pq = []

import heapq

함수

heappush(pq, x) : x를 우선순위 큐에 넣음

heappop(pq) : 우선순위가 가장 높은 값 반환 및 제거

pq[0] : 우선순위가 가장 높은 값 반환

우선순위 큐

Python

빈 리스트로 선언

pq = []

C++과 반대로

기본적으로 낮은 값이

우선순위가 높게 설정 되어 있음

값을 음수로 사용해서 높은 값을

우선순위가 높게 설정

```python
import heapq

pq = []
heapq.heappush(pq, -1) # 1
print(-pq[0])
heapq.heappush(pq, -3) # 3
print(-pq[0])
heapq.heappush(pq, -2)

heapq.heappop(pq) # 3 제거
print(-pq[0]) # 2
```

우선순위 큐

Python

빈 리스트로 선언

pq = []

낮은 값을 우선순위가 높게

사용하고 싶을 때는 그냥 사용

```python
import heapq

pq = []
heapq.heappush(pq, 1) # 1
print(pq[0])
heapq.heappush(pq, 3) # 1
print(pq[0])
heapq.heappush(pq, 2)

heapq.heappop(pq) # 1 제거
print(pq[0]) # 2
```

질문?

최소 힙 / 1927

백준 1927 / https://www.acmicpc.net/problem/1927

문제

널리 잘 알려진 자료구조 중 최소 힙이 있다. 최소 힙을 이용하여 다음과 같은 연산을 지원하는 프로그램을 작성하시오.

1. 배열에 자연수 x를 넣는다.
2. 배열에서 가장 작은 값을 출력하고, 그 값을 배열에서 제거한다.

프로그램은 처음에 비어있는 배열에서 시작하게 된다.

입력

첫째 줄에 연산의 개수 N($1 \le N \le 100,000$)이 주어진다. 다음 N개의 줄에는 연산에 대한 정보를 나타내는 정수 x가 주어진다. 만약 x가 자연수라면 배열에 x라는 값을 넣는(추가하는) 연산이고, x가 0이라면 배열에서 가장 작은 값을 출력하고 그 값을 배열에서 제거하는 경우이다. x는 2^{31}보다 작은 자연수 또는 0이고, 음의 정수는 입력으로 주어지지 않는다.

출력

입력에서 0이 주어진 횟수만큼 답을 출력한다. 만약 배열이 비어 있는 경우인데 가장 작은 값을 출력하라고 한 경우에는 0을 출력하면 된다.

최소 힙 / 1927

단순 최소 우선순위 큐 구현 문제

출력

입력에서 0이 주어진 횟수만큼 답을 출력한다. 만약 배열이 비어 있는 경우인데 가장 작은 값을 출력하라고 한 경우에는 0을 출력하면 된다.

예제 입력 1 복사

```
9
0
12345678
1
2
0
0
0
0
32
```

예제 출력 1 복사

```
0
1
2
12345678
0
```

최소 힙 / 1927

x가 0이고 우선순위 큐가 비어있으면 0
아니면 top() 출력

x가 0이 아니면 우선순위 큐에 x 삽입

최소 힙 / 1927

C++

우선순위 큐가 비어 있을 때 예외 처리

```cpp
#include <iostream>
#include <queue>
using namespace std;
using ll = long long;

const ll MAX = 201010;
ll n;
// 작은 값 먼저 출력
priority_queue <ll, vector<ll>, greater<ll>> pq;

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

    cin >> n;
    for(int i = 1;i <= n;i++){
        ll x; cin >> x;
        if(x) pq.push(x); // x 삽입
        else{
            // 우선순위 큐가 비어있으면 0 출력
            if(pq.empty()) cout << 0 << "\n";

            // 아니면
            else{
                // 최솟값 출력
                cout << pq.top() << "\n";
                pq.pop(); // 값 삭제
            }
        }
    }

    return 0;
}
```

최소 힙 / 1927

Python

우선순위 큐가 비어 있을 때 예외 처리

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

pq = []
n = int(input().rstrip())
for _ in range(n):
    x = int(input().rstrip())

    if x != 0:
        # 우선순위 큐에 x 삽입
        heapq.heappush(pq, x)
    else:
        # 우선순위 큐가 비어있으면
        if len(pq) == 0:
            print(0) # 0 출력

        # 아니면
        else:
            # 최솟값 출력 및 값 삭제
            print(heapq.heappop(pq))
```

최대 힙 / 11279

백준 11279 / https://www.acmicpc.net/problem/11279

문제

널리 잘 알려진 자료구조 중 최대 힙이 있다. 최대 힙을 이용하여 다음과 같은 연산을 지원하는 프로그램을 작성하시오.

1. 배열에 자연수 x를 넣는다.
2. 배열에서 가장 큰 값을 출력하고, 그 값을 배열에서 제거한다.

프로그램은 처음에 비어있는 배열에서 시작하게 된다.

입력

첫째 줄에 연산의 개수 N($1 \le N \le 100,000$)이 주어진다. 다음 N개의 줄에는 연산에 대한 정보를 나타내는 정수 x가 주어진다. 만약 x가 자연수라면 배열에 x라는 값을 넣는(추가하는) 연산이고, x가 0이라면 배열에서 가장 큰 값을 출력하고 그 값을 배열에서 제거하는 경우이다. 입력되는 자연수는 2^{31}보다 작다.

출력

입력에서 0이 주어진 횟수만큼 답을 출력한다. 만약 배열이 비어 있는 경우인데 가장 큰 값을 출력하라고 한 경우에는 0을 출력하면 된다.

최대 힙 / 11279

단순 최대 우선순위 큐 구현 문제

출력

입력에서 0이 주어진 횟수만큼 답을 출력한다. 만약 배열이 비어 있는 경우인데 가장 큰 값을 출력하라고 한 경우에는 0을 출력하면 된다.

예제 입력 1 복사

```
13
0
1
2
0
0
3
2
1
0
0
0
0
0
```

예제 출력 1 복사

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

최대 힙 / 11279

최소 힙 문제에서 우선순위 큐만 최대로 바꾸면 됨

출력

입력에서 0이 주어진 횟수만큼 답을 출력한다. 만약 배열이 비어 있는 경우인데 가장 큰 값을 출력하라고 한 경우에는 0을 출력하면 된다.

예제 입력 1 복사

```
13
0
1
2
0
0
3
2
1
0
0
0
0
0
```

예제 출력 1 복사

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

최대 힙 / 11279

C++

우선순위 큐가 비어 있을 때 예외 처리

```cpp
#include <iostream>
#include <queue>
using namespace std;
using ll = long long;

const ll MAX = 201010;
ll n;
// 큰 값 먼저 출력
priority_queue <ll> pq;

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

    cin >> n;
    for(int i = 1;i <= n;i++){
        ll x; cin >> x;
        if(x) pq.push(x); // x 삽입
        else{
            // 우선순위 큐가 비어있으면 0 출력
            if(pq.empty()) cout << 0 << "\n";

            // 아니면
            else{
                // 최댓값 출력
                cout << pq.top() << "\n";
                pq.pop(); // 값 삭제
            }
        }
    }

    return 0;
}
```

최대 힙 / 11279

Python

우선순위 큐가 비어 있을 때
예외 처리

최대 우선순위 큐이므로
음수로 삽입하고
음수로 반환

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

pq = []
n = int(input().rstrip())
for _ in range(n):
    x = int(input().rstrip())

    if x != 0:
        # 우선순위 큐에 x 삽입
        # 최대 우선순위 큐이므로 음수로 삽입
        heapq.heappush(pq, -x)
    else:
        # 우선순위 큐가 비어있으면
        if len(pq) == 0:
            print(0) # 0 출력

        # 아니면
        else:
            # 최댓값 출력 및 값 삭제
            # 최대 우선순위 큐이므로 음수로 반환
            print(-heapq.heappop(pq))
```

질문?

공정 컨설턴트 호석 / 22254

백준 22254 / https://www.acmicpc.net/problem/22254

문제

거듭된 창업 성공을 이룬 류진국 사장은 이번에는 맞춤형 선물을 제작해주는 공장을 만들기로 했다. 현재 들어온 맞춤형 선물 주문은 총 N개이며, 각 맞춤형 선물마다 제작에 필요한 시간이 정해져있다. 주문의 번호는 1번부터 N번까지 매겨져 있으며, 다음과 같은 규칙에 맞게 공정이 이뤄진다.

1. 공정 라인이 총 K개가 있다면 1번부터 K번까지의 번호가 존재한다.
2. 공정 라인의 사용 시간은 배정된 맞춤형 선물들의 제작 시간의 총합이다.
3. i번 선물은 1번 부터 $i-1$번 선물들이 배정된 이후에 사용 시간이 가장 적은 공정 라인 중 하나에 배정된다.

모든 맞춤형 선물의 제작이 완료될 때까지 최대 X시간이 남아있는 상황인데, 아직 공정 라인의 개수 K가 정해져 있지 않다. 류진국 사장은 이 분야 최고 권위자, 공정 컨설턴트 호석에게 의뢰했다. 공정 컨설턴트 호석은 최소한의 비용을 쓰기 위해서 공정 라인의 개수를 최소화 시키고자 한다. 호석을 도와 필요한 최소 공정 라인 개수를 계산하자.

입력

첫 번째 줄에 맞춤형 선물 주문의 개수 N, 제작 완료까지 남은 시간 X가 공백으로 구분되어 주어진다.

두 번째 줄에는 1번부터 N번 선물이 필요한 공정 시간이 순서대로 주어진다. 단위는 '시간' 이다.

출력

모든 선물을 X시간 이내에 제작하기 위해 필요한 최소 공정 라인 개수를 출력하라.

공정 컨설턴트 호석 / 22254

선물의 개수 N, 시간 제한 M이 주어짐
선물을 1번부터 차례대로 N번 까지 제작 할 때
사용해야 하는 공정 라인 개수의 최솟값을 구해야 함

예제 입력 1 복사

```
6 11
5 2 8 4 3 5
```

예제 출력 1 복사

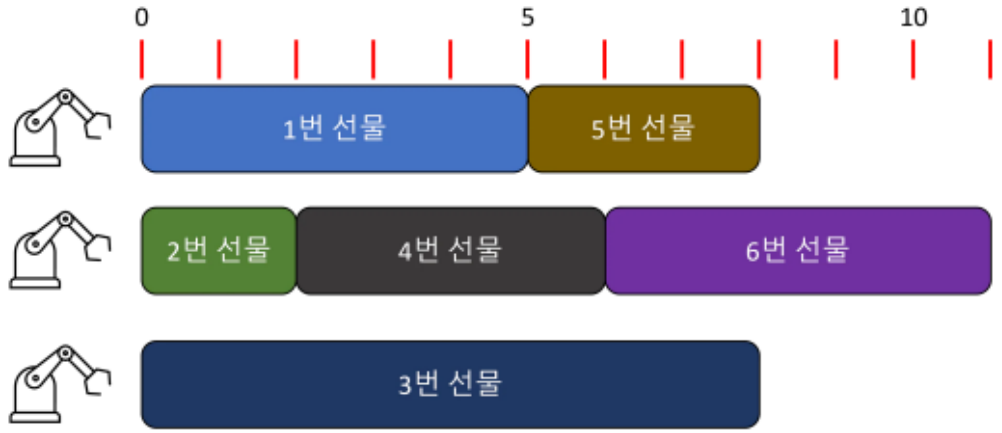
```
3
```

공정 컨설턴트 호석 / 22254

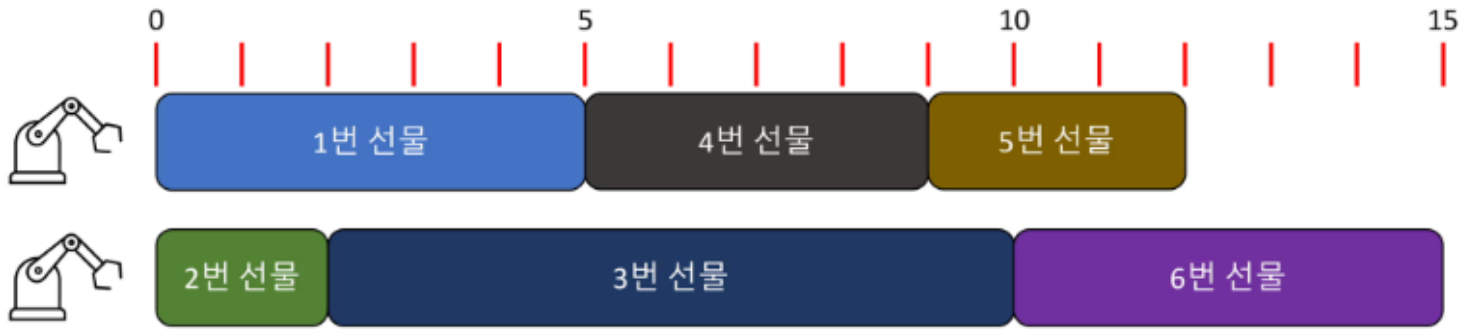
사용해야 하는 공정 라인 개수의 최솟값을 구해야 함

```
6 11
5 2 8 4 3 5
```

3개의 공정 라인을 사용할 경우, 아래 그림 처럼 11 시간이면 완료된다.



2개의 공정 라인을 사용할 경우, 아래 그림 처럼 15 시간이 걸리기 때문에 시간 안에 만들 수 없다.



공정 컨설턴트 호석 / 22254

공정 라인의 개수 K를 정했을 때
모든 선물을 M의 시간 안에 구하는 건 O(N)에 가능함

현재 보고 있는 선물은 최소로 사용한 공정 라인에 배치
-> 우선 순위 큐 사용

우선 순위 큐에 0을 K개 삽입하고, N번 동안
가장 작은 값에 A[i]를 더해서 삽입하면
모든 선물을 제작 했을 때의 시간은 가장 큰 값이 됨

공정 컨설턴트 호석 / 22254

이제 공정 라인의 개수 K를 정했을 때
M시간 안에 만들 수 있는지 없는지 판단 할 수 있음

공정 라인의 개수가 늘어나면 항상
선물을 다 만드는 시간은 줄어 들음
decision(K) -> 00000**1**1111

결정 문제의 최솟값을 매개 변수 탐색으로 찾으면 됨

공정 컨설턴트 호석 / 22254

C++

```cpp
#include <iostream>
#include <queue>
using namespace std;
using ll = long long;

const ll MAX = 101010;
ll n, m, a[MAX];

bool decision(ll cur){
    // 최소 시간 출력
    priority_queue <ll, vector<ll>, greater<ll>> pq;
    // cur개 만큼의 공정 라인 사용 가능
    for(int i = 1;i <= cur;i++) pq.push(0);

    ll mx = 0; // 공정 라인 사용 시간 최댓값
    for(int i = 1;i <= n;i++){
        // 제일 적은 공정 라인 사용 시간
        ll now = pq.top(); pq.pop();
        // 현재 제작 시간을 더함
        pq.push(now + a[i]);
        mx = max(mx, now + a[i]);
    }

    // 공정 라인 사용 시간 최댓값이 M보다 작으면
    // cur개의 라인으로 M시간 내에 제작 가능
    return mx <= m;
}
```

```cpp
ll maximization(){
    ll lo = 1, hi = 2e5;
    while(lo < hi){
        ll mid = (lo + hi) / 2;
        if(decision(mid)) hi = mid;
        else lo = mid + 1;
    }

    return lo;
}

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

    cin >> n >> m;
    for(int i = 1;i <= n;i++) cin >> a[i];
    cout << maximization();

    return 0;
}
```

공정 컨설턴트 호석 / 22254

Python

```python
n, m = list(map(int, input().rstrip().split()))
a = list(map(int, input().rstrip().split()))

def decision(cur):
    # 최소 시간 출력
    # cur개 만큼의 공정 라인 사용 가능
    pq = [0] * cur

    mx = 0 # 공정 라인 사용 시간 최댓값
    for i in a:
        # 제일 적은 공정 라인 사용 시간
        # 현재 제작 시간을 더함
        now = heapq.heappop(pq) + i
        heapq.heappush(pq, now)
        mx = max(mx, now)

    # 공정 라인 사용 시간 최댓값이 M보다 작으면
    # cur개의 라인으로 M시간 내에 제작 가능
    return mx <= m
```

```python
def maximization():
    lo = 1
    hi = int(2e5)
    while lo < hi:
        mid = (lo + hi) // 2
        if(decision(mid)):
            hi = mid
        else:
            lo = mid + 1
    return lo

print(maximization())
```

질문?

연료 채우기 / 1826

백준 1826 / https://www.acmicpc.net/problem/1826

문제

성경이는 트럭을 정글 속에서 운전하다가 트럭의 연료탱크에 갑자기 구멍이 나서 1km를 가는데 1L의 연료가 새 나가게 되었다. 이것을 고치기 위해서는 가장 가까운 마을에 가야 한다. 그런데 그냥 가다가는 중간에 연료가 다 빠질 수가 있다. 다행스럽게도 정글 곳곳에 연료를 채울 수 있는 주유소가 N개 있다. 그런데 정글 속에서 중간에 차를 멈추는 행위는 매우 위험한 행위이므로 주유소에서 멈추는 횟수를 최소화 하려 한다.

그리고 다행이도 성경이의 트럭은 매우 좋아서 연료 탱크에는 연료의 제한이 없이 많이 충분히 많이 넣을 수 있다고 한다. 각각의 주유소의 위치와, 각 주유소에서 얻을 수 있는 연료의 양이 주어져 있을 때, 주유소에서 멈추는 횟수를 구하는 프로그램을 작성하시오.

정글은 일직선이고, 성경이의 트럭과 주유소도 모두 일직선 위에 있다. 주유소는 모두 성경이의 트럭을 기준으로 오른쪽에 있다.

입력

첫째 줄에 주유소의 개수 N($1 \le N \le 10,000$)가 주어지고 두 번째 줄부터 N+1번째 줄 까지 주유소의 정보가 주어진다. 주유소의 정보는 두개의 정수 a,b로 이루어 져 있는데 a($1 \le a \le 1,000,000$)는 성경이의 시작 위치에서 주유소 까지의 거리, 그리고 b($1 \le b \le 100$)는 그 주유소에서 채울 수 있는 연료의 양을 의미한다. 그리고 N+2번째 줄에는 두 정수 L과 P가 주어지는데 L($1 \le L \le 1,000,000$)은 성경이의 위치에서 마을까지의 거리, P($1 \le P \le 1,000,000$)는 트럭에 원래 있던 연료의 양을 의미한다.

모든 주유소와 마을은 서로 다른 좌표에 있고, 마을은 모든 주유소보다 오른쪽에 있다.

출력

첫째 줄에 주유소에서 멈추는 횟수를 출력한다. 만약 마을에 도착하지 못할경우 -1을 출력한다.

연료 채우기 / 1826

N개의 주유소가 있고, M에 도달 해야 함
처음에는 K의 연료를 가지고 있음
거리 1을 움직일 때마다 연료 1을 소모하고,
각 주유소에 도착하면 연료를 충전 할 수 있음

예제 입력 1 복사

```
4
4 4
5 2
11 5
15 10
25 10
```

예제 출력 1 복사

```
3
```

연료 채우기 / 1826

아래에 예제에서는
첫번째, 세번째, 네번째 주유소에서 연료를 충전하면
10 + 4 + 5 + 10 = 29로 종착점인 25에 도달 할 수 있음

예제 입력 1 복사

```
4
4 4
5 2
11 5
15 10
25 10
```

예제 출력 1 복사

```
3
```

연료 채우기 / 1826

연료 1L로 1의 거리를 움직일 수 있음 ->
갖고 있는 연료의 용량 = 움직일 수 있는 거리

최소로 연료를 충전할 때의 숫자를 구해야 함 ->
많은 연료를 충전할 수 있는 주유소부터 충전

주유소에서 충전을 하기 위해서는 그 주유소를 지나야 함
현재 충전 할 수 있는 연료의 양을 우선순위 큐에 삽입

연료 채우기 / 1826

최소로 연료를 주유할 때의 숫자를 구해야 함 ->
많은 연료를 주유할 수 있는 주유소부터 충전

주유소에서 주유를 하기 위해서는 그 주유소를 지나야 함
현재 충전 할 수 있는 연료의 양을 우선순위 큐에 삽입

다음 주유소로 갈 수 없으면 현재 충전 할 수 있는
연료의 양 중에서 큰 것부터 주유

연료 채우기 / 1826

다음 주유소로 갈 수 없으면 현재 주유 할 수 있는
연료의 양 중에서 큰 것부터 충전

만약 다음 주유소로 갈 수 없는데
주유도 할 수 없으면 -1 출력

연료 채우기 / 1826

C++

```cpp
#include <iostream>
#include <algorithm>
#include <queue>
using namespace std;
using ll = long long;

const ll MAX = 10101;
ll n, m, result;
pair <ll, ll> a[MAX];
priority_queue <ll> pq;
```

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

    cin >> n;
    for(int i = 1;i <= n;i++) cin >> a[i].first >> a[i].second;
    ll en, fuel; cin >> en >> fuel;
    a[n + 1].first = en; // 마을도 포함
    sort(a + 1, a + n + 2); // 거리가 가까운 순으로 정렬

    for(int i = 1;i <= n + 1;i++){
        // 현재 주유소의 거리가 갈 수 있는 거리보다 멀면
        while(a[i].first > fuel){
            // 비축해둔 연료가 없으면
            // 마을까지 갈 수 없으므로 -1
            if(pq.empty()){ cout << -1; return 0; }

            // 가장 많은 연료부터 소모
            ll now = pq.top(); pq.pop();
            fuel += now; result++;
        }

        // 연료를 비축해둠
        pq.push(a[i].second);
    }

    cout << result;

    return 0;
}
```

연료 채우기 / 1826

Python

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

n = int(input().rstrip())
a = []
for _ in range(n):
    x, y = list(map(int, input().rstrip().split()))
    a.append((x, y))
en, fuel = list(map(int, input().rstrip().split()))
pq = []

a.append((en, 0)) # 마을도 포함
a.sort() # 거리가 가까운 순으로 정렬
result = 0
```

```python
for x, y in a:
    # 현재 주유소의 거리가 갈 수 있는 거리보다 멀면
    while x > fuel:
        # 비축해둔 연료가 없으면
        # 마을까지 갈 수 없으므로 -1
        if len(pq) == 0:
            print(-1)
            exit()

        result += 1
        # 가장 많은 연료부터 소모
        fuel += -heapq.heappop(pq)

    # 연료를 비축해둠
    heapq.heappush(pq, -y)

print(result)
```

질문?

기본 과제

최소 힙 – https://www.acmicpc.net/problem/1927

최대 힙 - https://www.acmicpc.net/problem/11279

파일 합치기 3 - https://www.acmicpc.net/problem/13975

연료 채우기 - https://www.acmicpc.net/problem/1826

공정 컨설턴트 호석 - https://www.acmicpc.net/problem/22254

가운데를 말해요 - https://www.acmicpc.net/problem/1655

심화 과제

소가 길을 건너간 이유 4 - https://www.acmicpc.net/problem/14464

소수의 곱 - https://www.acmicpc.net/problem/1655

고생하셨습니다

K번째 수 / 1300

백준 1300 / https://www.acmicpc.net/problem/1300

문제

널리 잘 알려진 자료구조 중 최소 힙이 있다. 최소 힙을 이용하여 다음과 같은 연산을 지원하는 프로그램을 작성하시오.

1. 배열에 자연수 x를 넣는다.
2. 배열에서 가장 작은 값을 출력하고, 그 값을 배열에서 제거한다.

프로그램은 처음에 비어있는 배열에서 시작하게 된다.

입력

첫째 줄에 연산의 개수 N($1 \le N \le 100,000$)이 주어진다. 다음 N개의 줄에는 연산에 대한 정보를 나타내는 정수 x가 주어진다. 만약 x가 자연수라면 배열에 x라는 값을 넣는(추가하는) 연산이고, x가 0이라면 배열에서 가장 작은 값을 출력하고 그 값을 배열에서 제거하는 경우이다. x는 2^{31}보다 작은 자연수 또는 0이고, 음의 정수는 입력으로 주어지지 않는다.

출력

입력에서 0이 주어진 횟수만큼 답을 출력한다. 만약 배열이 비어 있는 경우인데 가장 작은 값을 출력하라고 한 경우에는 0을 출력하면 된다.

K번째 수 / 1300

A[i][j] = i * j 일 때 N * N의 배열 A에서 K번째 수 출력

출력

B[k]를 출력한다.

예제 입력 1 복사

```
3
7
```

예제 출력 1 복사

```
6
```

K번째 수 / 1300

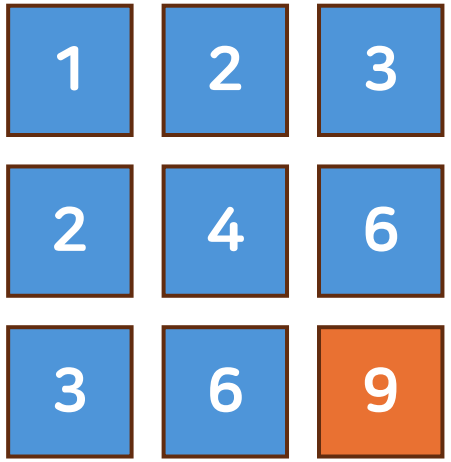
최적화 문제

배열 A에서 작거나 같은 수의 개수가 K개 이상인 수 중 최솟값

예제 입력 예제 출력

3 6

7



6 보다 작거나 같은 수 -> 8개

6 보다 작은 수인 4 보다 작거나 같은 수 - > 6개

4는 최적화 문제를 만족하지 못함

K번째 수 / 1300

최적화 문제

배열 A에서 작거나 같은 수의 개수가 K개 이상인 수 중 최솟값

결정 문제

현재 수가 M일 때

M보다 작거나 같은 수가 K개 이상인가

K번째 수 / 1300

결정 문제

현재 수가 M일 때
M보다 작거나 같은 수가 K개 이상인가

M이 커지면 M보다 작거나 같은 수가 늘어 남
decision(M) = 0000011111

최솟값을 구하면 됨

K번째 수 / 1300

결정 문제

현재 수가 M일 때
M보다 작거나 같은 수가 K개 이상인가

M보다 작거나 같은 수를 나이브하게 구하면
배열은 N*N(N <= 1e6) 이므로 무조건 시간 초과

빠르게 작거나 같은 수를 구하는 법은 없을까?

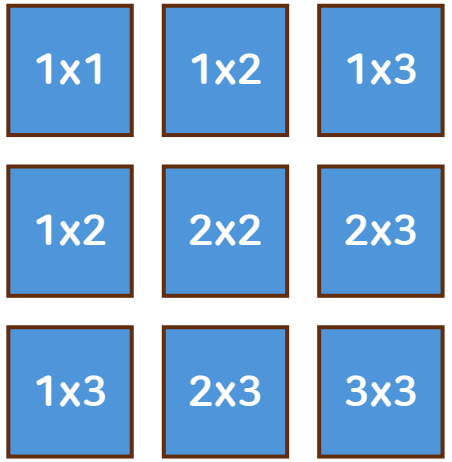
K번째 수 / 1300

결정 문제

현재 수가 M일 때
M보다 작거나 같은 수가 K개 이상인가

배열의 i행은 항상 i의 배수
각 행에 대해서 M를 i로 나누면
작거나 같은 수의 개수가 나옴

단 각 행의 원소 개수는 N이므로
최댓값은 N으로 제한해야 함



K번째 수 / 1300

시간 복잡도

결정 함수 - O(N)

매개 변수 탐색 – O(log 1e10 = 33...)

O(N log 1e10)

K번째 수 / 1300

C++

```cpp
bool decision(ll cur){
    // cur 보다 작은 수의 개수
    ll cnt = 0;
    for(int i = 1;i <= n;i++){
        // i번째 열의 cur 보다 작은 수의 개수를 더함
        cnt += min(cur / i, n);
    }

    return cnt >= m;
}

ll minimization(){
    // 범위의 최댓값은 1e10
    ll lo = 1, hi = 2e10;
    while(lo < hi){
        ll mid = (lo + hi) / 2;
        if(decision(mid)) hi = mid;
        else lo = mid + 1;
    }

    return lo;
}
```

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

    cin >> n >> m;
    cout << minimization();

    return 0;
}
```

K번째 수 / 1300

Python

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

n = int(input().rstrip())
m = int(input().rstrip())

def decision(cur):
    cnt = 0 # cur보다 작은 수의 개수
    for i in range(1, n + 1):
        # i번째 열의 cur보다 작은 수의 개수를 더함
        cnt += min(cur // i, n)
    return cnt >= m

def minimization():
    lo = 1
    # 범위의 최댓값은 1e10
    hi = int(2e10)
    while lo < hi:
        mid = (lo + hi) // 2
        if decision(mid):
            hi = mid
        else:
            lo = mid + 1
    return lo

print(minimization())
```

질문?

세미나 배정 / 28305

백준 28305 / https://www.acmicpc.net/problem/28305

DEVOCEAN에서 세미나 N개를 주최하려고 한다. DEVOCEAN의 행사 기획 담당자 성호는 다음 조건을 지켜 각 세미나의 일정을 정해야 한다.

- 각 세미나는 연속된 T일 동안 진행되어야 한다. 구체적으로, i번째 세미나는 m_i일차부터 $m_i + T - 1$일차까지 매일 진행되어야 한다. 성호는 m_i를 정해야 한다. **m_i는 반드시 양의 정수여야 한다.**
- 각 세미나의 T일 중 하루에는 외부전문가의 특강이 진행되어야 한다. i번째 세미나의 외부전문가가 참석할 수 있는 날이 a_i일차밖에 없기 때문에, a_i일차는 i번째 세미나가 진행되는 T일 중 하루여야 한다. 즉, $m_i \le a_i \le m_i + T - 1$이어야 한다.

성호는 세미나에 사용할 세미나실을 미리 대여하기로 하였다. 하나의 세미나실에서는 같은 날에 최대 하나의 세미나를 진행할 수 있지만, 다른 날에 서로 다른 세미나를 진행하는 것은 가능하다. 한 번이라도 사용할 세미나실은 모두 빌려야 하기 때문에, 성호는 진행되는 세미나가 가장 많은 날의 세미나 수를 최소화하고 싶다. 성호를 도와 이 값을 구해 주자!

입력

첫 번째 줄에 N, T가 공백으로 구분되어 입력된다. ($1 \le N \le 200\,000$; $1 \le T \le 10^9$)

두 번째 줄에 a_1, a_2, $\cdots$, a_N이 공백으로 구분되어 입력된다. ($1 \le a_i \le 10^9$)

출력

진행되는 세미나가 가장 많은 날의 세미나 수의 최솟값을 출력한다.

세미나 배정 / 28305

M일 동안 N개의 세미나를 할 때
i번째 세미나를 A[i]일에 포함 시켜야 할 때
동시에 하는 세미나의 최솟값

출력

진행되는 세미나가 가장 많은 날의 세미나 수의 최솟값을 출력한다.

예제 입력 1 복사

```
5 3
4 6 3 5 7
```

예제 출력 1 복사

```
2
```

세미나 배정 / 28305

출력

진행되는 세미나가 가장 많은 날의 세미나 수의 최솟값을 출력한다.

예제 입력 1 복사

```
5 3
4 6 3 5 7
```

예제 출력 1 복사

```
2
```

첫 번째 예제에서, 다섯 개의 세미나를 다음과 같은 일정으로 진행한다.

- 첫 번째 세미나를 3일차부터 5일차까지 진행한다.
- 두 번째 세미나를 6일차부터 8일차까지 진행한다.
- 세 번째 세미나를 1일차부터 3일차까지 진행한다.
- 네 번째 세미나를 4일차부터 6일차까지 진행한다.
- 다섯 번째 세미나를 7일차부터 9일차까지 진행한다.

이 경우, 1일차부터 9일차까지 각 날에 진행되는 세미나의 수는 차례로 1, 1, 2, 2, 2, 2, 2, 2, 1개이다. 따라서 진행되는 세미나가 가장 많은 날의 세미나 수는 2개이다.

2개 이상의 세미나가 진행되는 날이 없게 일정을 정하는 방법은 없음을 증명할 수 있다.

세미나 배정 / 28305

최적화 문제

M일 동안 N개의 세미나를 할 때
i번째 세미나를 A[i]일에 포함 시켜야 할 때
동시에 하는 세미나의 최솟값

결정 문제

동시에 세미나를 K개 까지 할 수 있을 때
i번째 세미나를 A[i]일에 포함 시킬 수 있는가

세미나 배정 / 28305

결정 문제

동시에 세미나를 K개 까지 할 수 있을 때
i번째 세미나를 A[i]일에 포함 시킬 수 있는가

세미나를 동시에 많이 진행 할 수 있으면
한번에 모든 세미나를 진행 할 수 있음
decision(K) -> 00000**1**1111

최솟값을 구하면 됨

세미나 배정 / 28305

결정 문제

동시에 세미나를 K개 까지 할 수 있을 때
i번째 세미나를 A[i]일에 포함 시킬 수 있는가

예제 입력	예제 출력
5 3 4 6 3 5 7	2

---

1 2 3 4 5 6 7 8 9

세미나 배정 / 28305

결정 문제

동시에 세미나를 K개 까지 할 수 있을 때

i번째 세미나를 A[i]일에 포함 시킬 수 있는가

5 3 2

3 4 5 6 7

최대한 앞에 배치하는 게 이득

세미나 배정 / 28305

결정 문제

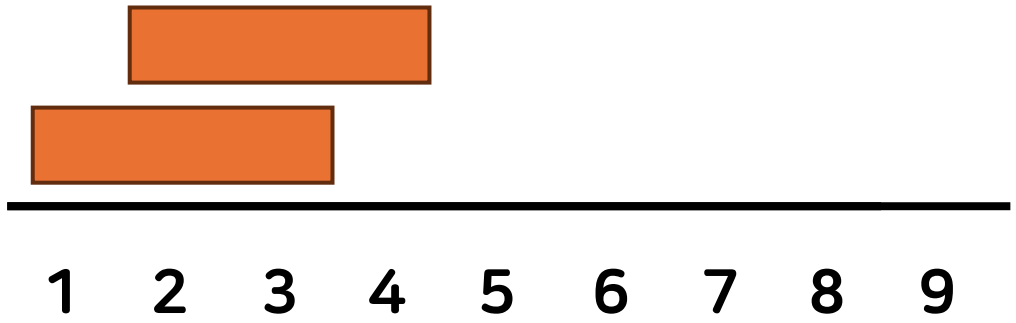
동시에 세미나를 K개 까지 할 수 있을 때

i번째 세미나를 A[i]일에 포함 시킬 수 있는가

5 3 2

3 4 5 6 7

최대한 앞에 배치하는 게 이득



세미나 배정 / 28305

결정 문제

동시에 세미나를 K개 까지 할 수 있을 때

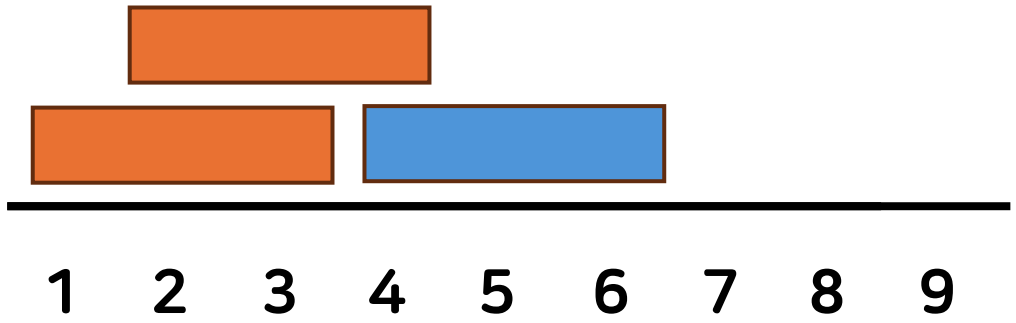
i번째 세미나를 A[i]일에 포함 시킬 수 있는가

5 3 2

3 4 5 6 7

K개를 다 배치 했으므로

K개 이전의 배치 날짜 + M 이후로 배치해야 함



세미나 배정 / 28305

결정 문제

동시에 세미나를 K개 까지 할 수 있을 때

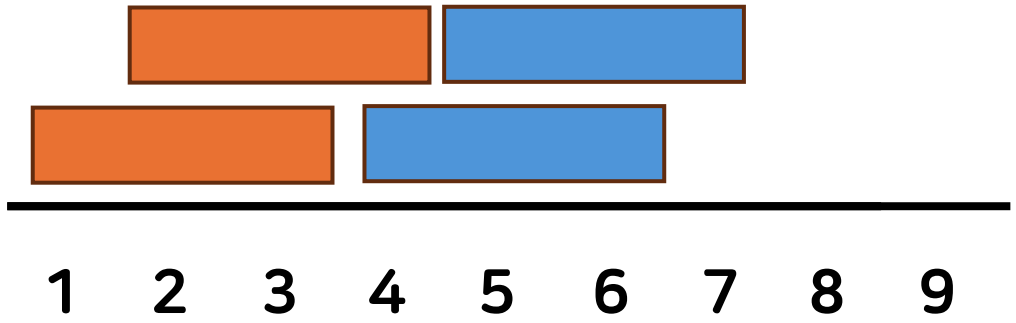
i번째 세미나를 A[i]일에 포함 시킬 수 있는가

5 3 2

3 4 5 6 7

K개를 다 배치 했으므로

K개 이전의 배치 날짜 + M 이후로 배치해야 함



세미나 배정 / 28305

결정 문제

동시에 세미나를 K개 까지 할 수 있을 때

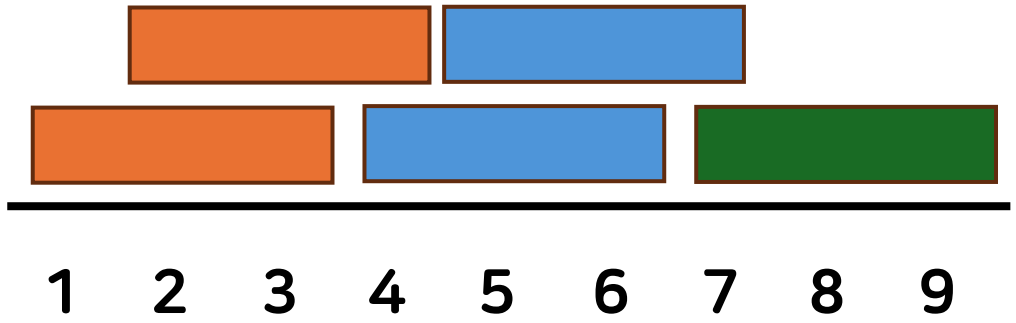
i번째 세미나를 A[i]일에 포함 시킬 수 있는가

5 3 2

3 4 5 6 7

K개를 다 배치 했으므로

K개 이전의 배치 날짜 + M 이후로 배치해야 함



세미나 배정 / 28305

결정 문제

동시에 세미나를 K개 까지 할 수 있을 때

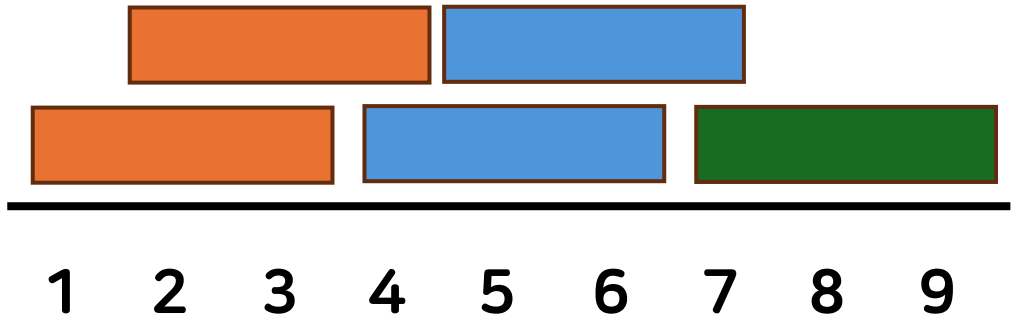
i번째 세미나를 A[i]일에 포함 시킬 수 있는가

5 3 2

3 4 5 6 7

모든 배치가 끝났는데 배치의 시작점이 A[i] 뒤에 있으면

동시에 K개의 세미나로는 배치 할 수 없음



세미나 배정 / 28305

C++

```cpp
ll n, m, a[MAX], num[MAX];

bool decision(ll cur){
    // 현재 사용중인 세미나의 시작날의 최댓값
    ll last = 0;
    for(int i = 1;i <= n;i++){
        // K개의 세미나실을 아직 다 안썼다면
        // 시작날을 max(a[i] - m + 1, 1)로 지정
        if(i <= cur) last = max(a[i] - m + 1, 1ll);

        // K개의 세미나실을 다 썼다면
        // K의 인덱스 전의 세미나의 시작날의 M을 더함
        else last = max({a[i] - m + 1, num[i - cur] + m, 1ll});

        // 세미나의 시작날 저장
        num[i] = last;

        // 세미나의 시작날이 지정한 날보다 늦으면
        // K개의 세미나실로 세미나를 진행 할 수 없음
        if(a[i] - num[i] < 0) return 0;
    }

    // 아니면 K개의 세미나실로 진행 할 수 있음
    return 1;
}
```

```cpp
ll minimization(){
    // 구간의 최댓값은 2e5
    ll lo = 1, hi = 201010;
    while(lo < hi){
        ll mid = (lo + hi) / 2;
        if(decision(mid)) hi = mid;
        else lo = mid + 1;
    }

    return lo;
}

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

    cin >> n >> m;
    for(int i = 1;i <= n;i++) cin >> a[i];
    sort(a + 1, a + n + 1);

    cout << minimization();

    return 0;
}
```

세미나 배정 / 28305

Python

```python
n, m = list(map(int, input().rstrip().split()))
a = list(map(int, input().rstrip().split()))
a.sort()
num = [0] * 201010

def decision(cur):
    # 현재 사용중인 세미나의 시작날의 최댓값
    last = 0
    for i in range(1, n + 1):
        # K개의 세미나실을 아직 다 안썼다면
        # 시작날을 max(a[i - 1] - m + 1, 1)로 지정
        if(i <= cur):
            last = max(a[i - 1] - m + 1, 1)

        # K개의 세미나실을 다 썼다면
        # K개의 인덱스 전의 세미나의 시작날에 M을 더함
        else:
            last = max(a[i - 1] - m + 1, num[i - cur - 1] + m, 1)

        # 세미나의 시작날 저장
        num[i - 1] = last

        # 세미나의 시작날이 지정한 날보다 늦으면
        # K개의 세미나실로 세미나를 진행할 수 없음
        if a[i - 1] - num[i - 1] < 0:
            return 0

    # 아니면 K개의 세미나실로 진행 할 수 있음
    return 1
```

```python
def minimization():
    lo = 1
    # 구간의 최댓값은 2e5
    hi = 201010
    while lo < hi:
        mid = (lo + hi) // 2
        if(decision(mid)):
            hi = mid
        else:
            lo = mid + 1
    return lo

print(minimization())
```

질문?

고생하셨습니다