Chapter 01.
우선순위 큐

Clip 01 | [11279] 최대 힙
힙 자료구조와 우선순위 큐 이론

Clip 02 | [1927] 최소 힙
힙의 응용, 라이브러리 사용법

Clip 03 | [11286] 절댓값 힙
비교자를 정의하는 우선순위 큐

Clip 04 | [1655] 가운데를 말해요
두개의 힙을 이용한 중앙 값

Clip 05 | [2075] N번째 큰 수
메모리 최적화된 수 정렬

Clip 06 | [19598] 최소 회의실 개수
힙 적용 아이디어 떠올리기

Ch01. 우선순위 큐

1. [11279] 최대 힙

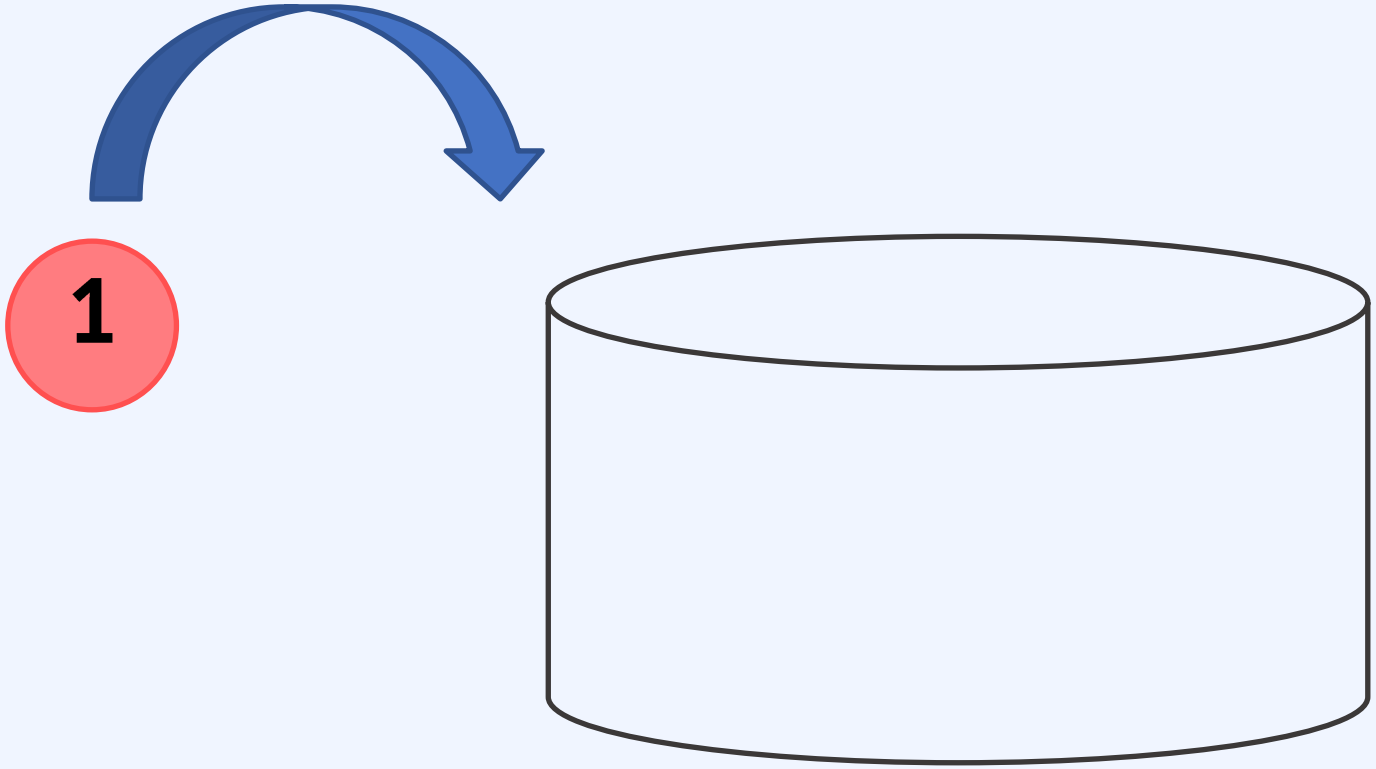
BOJ11279: 최대 힙

우선순위 큐? 힙?

- 데이터를 큐처럼 추가 / 삭제 할 수 있는 자료공간
- 단, 데이터가 뽑히는 순서는 입력할 때 넣은 가중치를 기준으로 “높은 우선순위” 부터 나온다

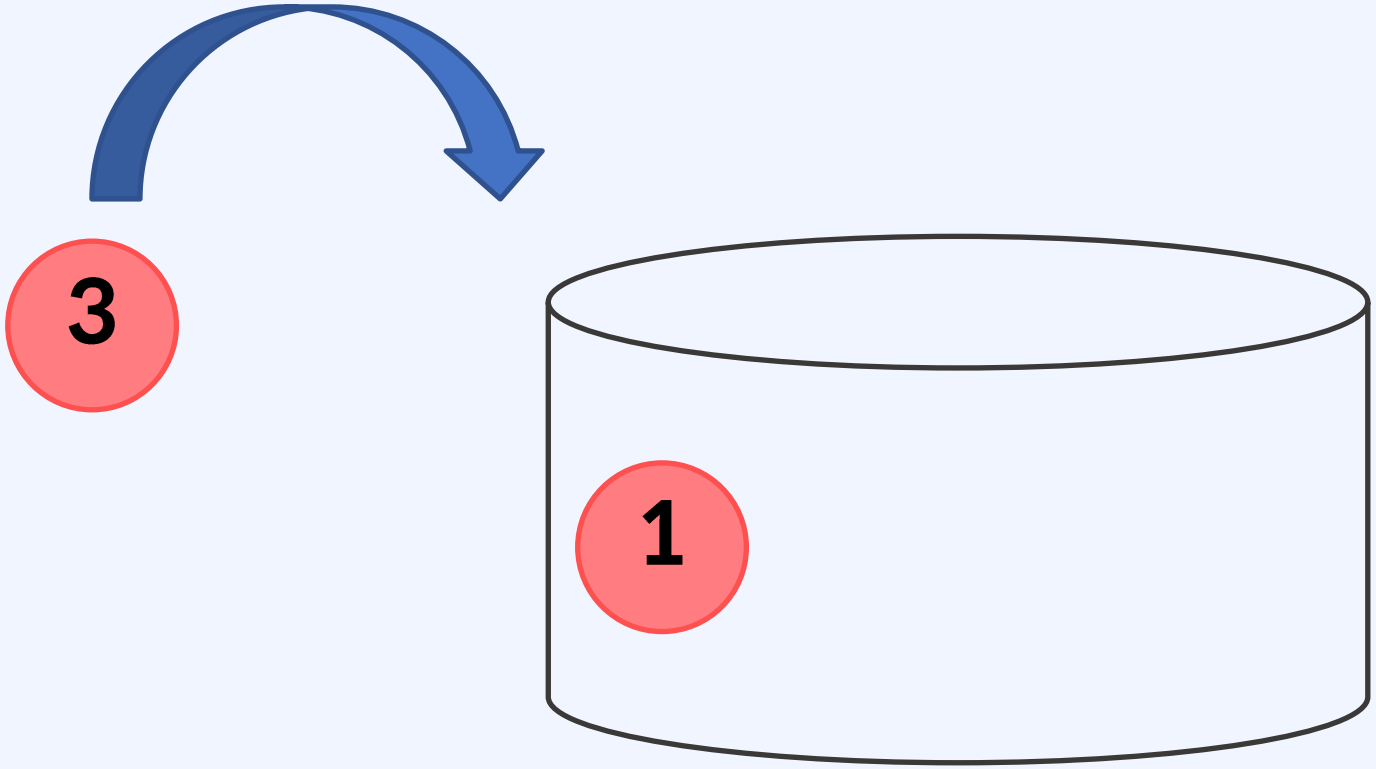
BOJ11279: 최대 힙

우선순위: 큰 수가 높은 우선순위를 가짐



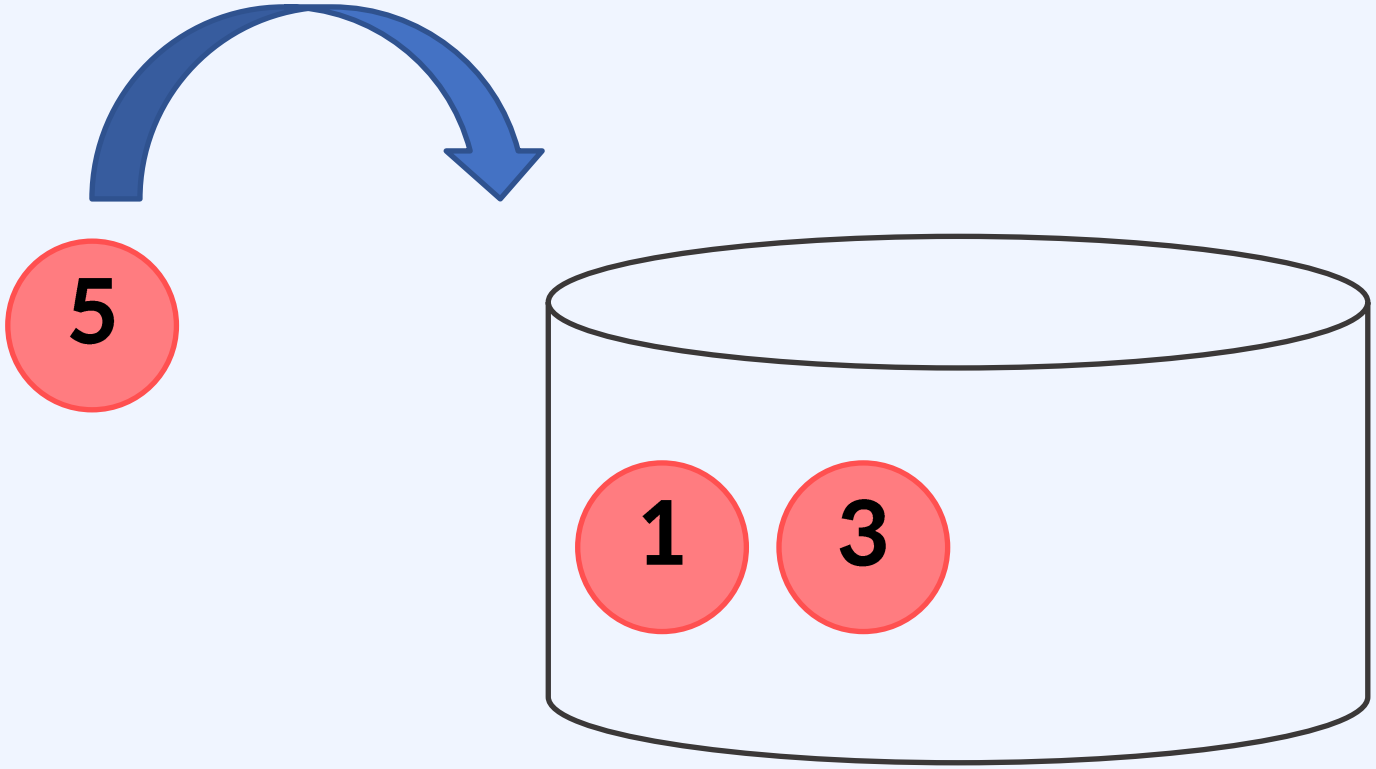
BOJ11279: 최대 힙

우선순위: 큰 수가 높은 우선순위를 가짐



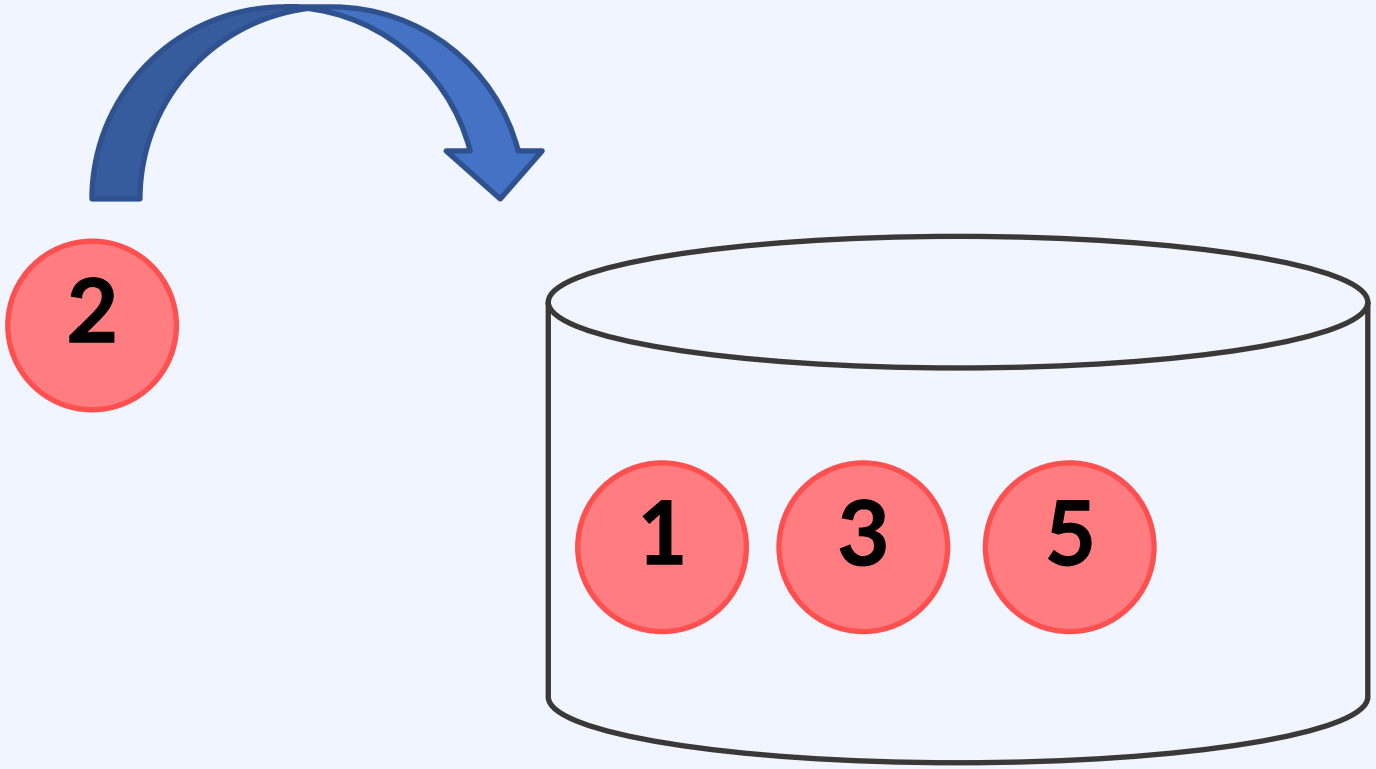
BOJ11279: 최대 힙

우선순위: 큰 수가 높은 우선순위를 가짐



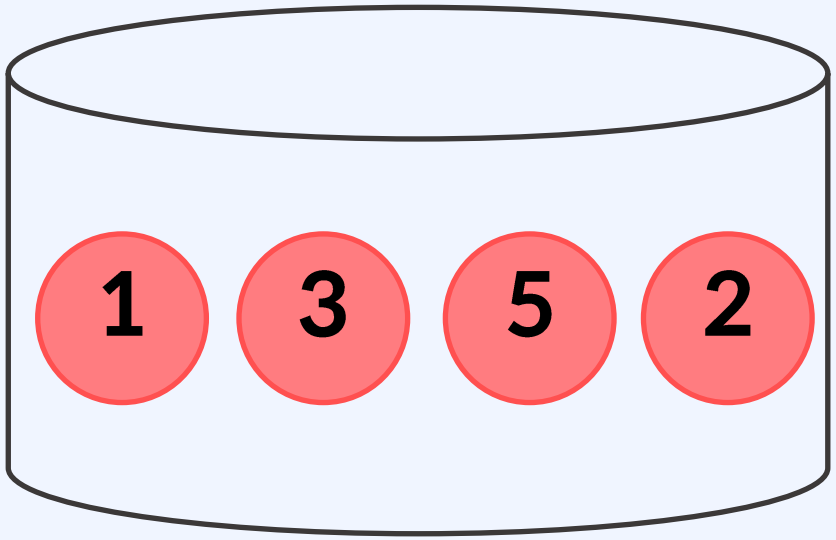
BOJ11279: 최대 힙

우선순위: 큰 수가 높은 우선순위를 가짐



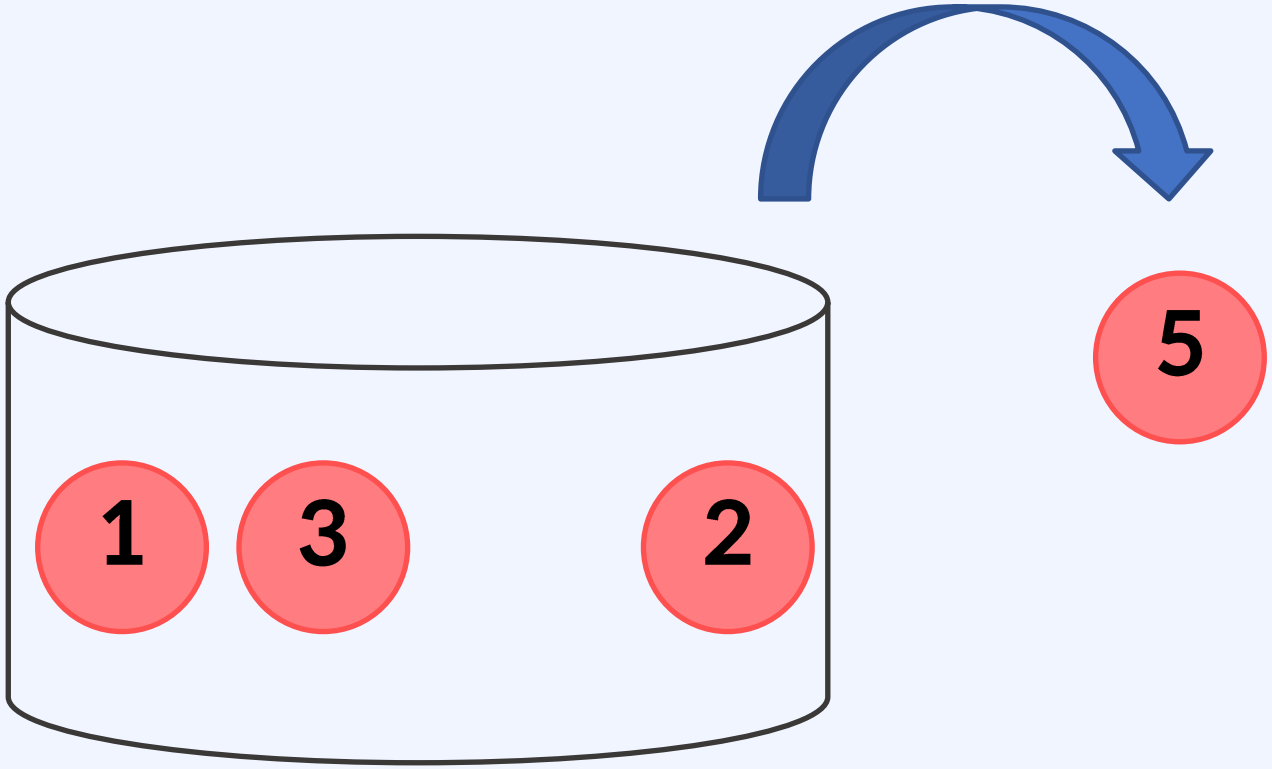
BOJ11279: 최대 힙

우선순위: 큰 수가 높은 우선순위를 가짐



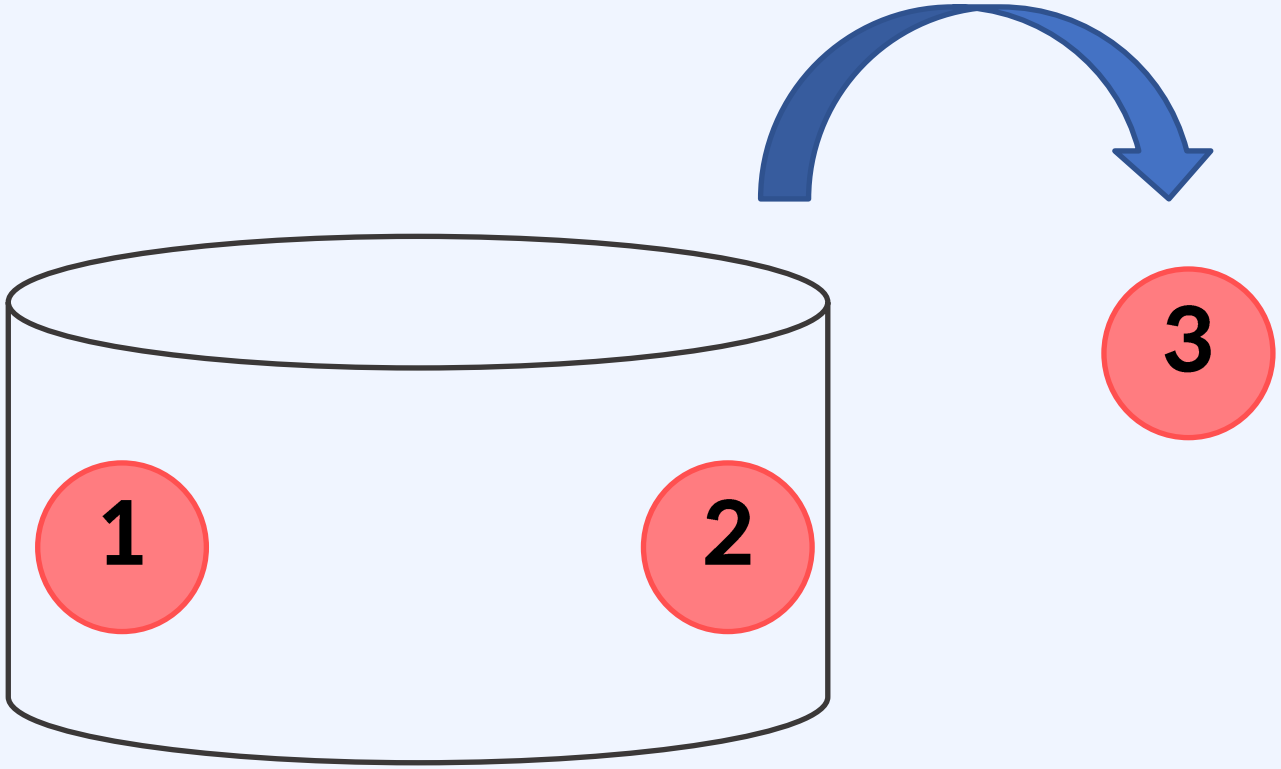
BOJ11279: 최대 힙

우선순위: 큰 수가 높은 우선순위를 가짐



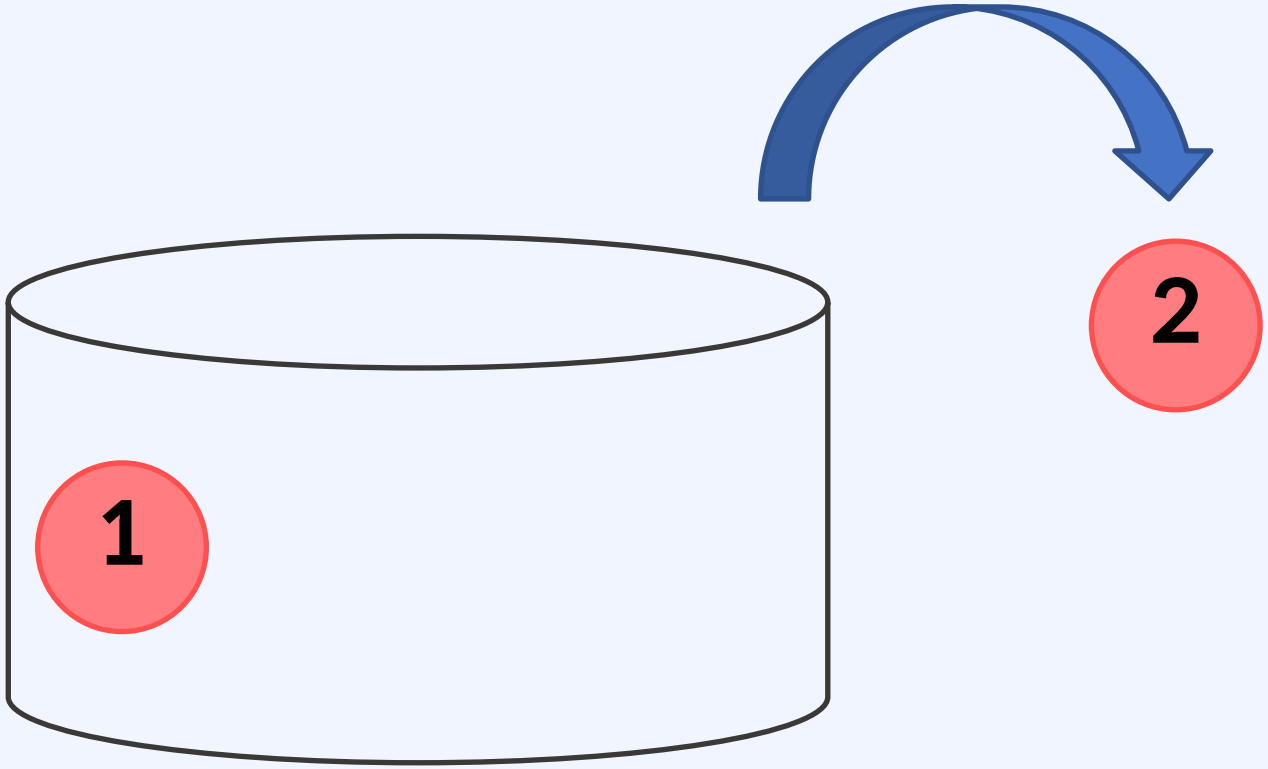
BOJ11279: 최대 힙

우선순위: 큰 수가 높은 우선순위를 가짐



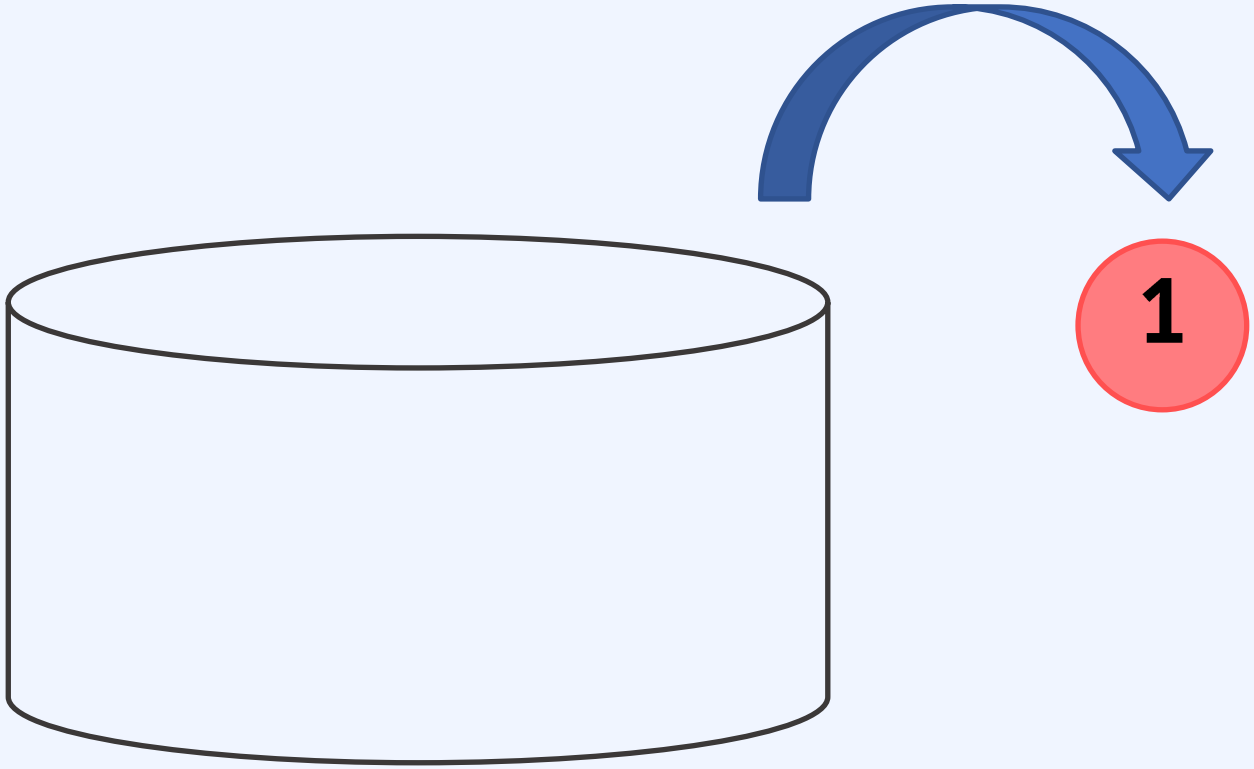
BOJ11279: 최대 힙

우선순위: 큰 수가 높은 우선순위를 가짐



BOJ11279: 최대 힙

우선순위: 큰 수가 높은 우선순위를 가짐



BOJ11279: 최대 힙

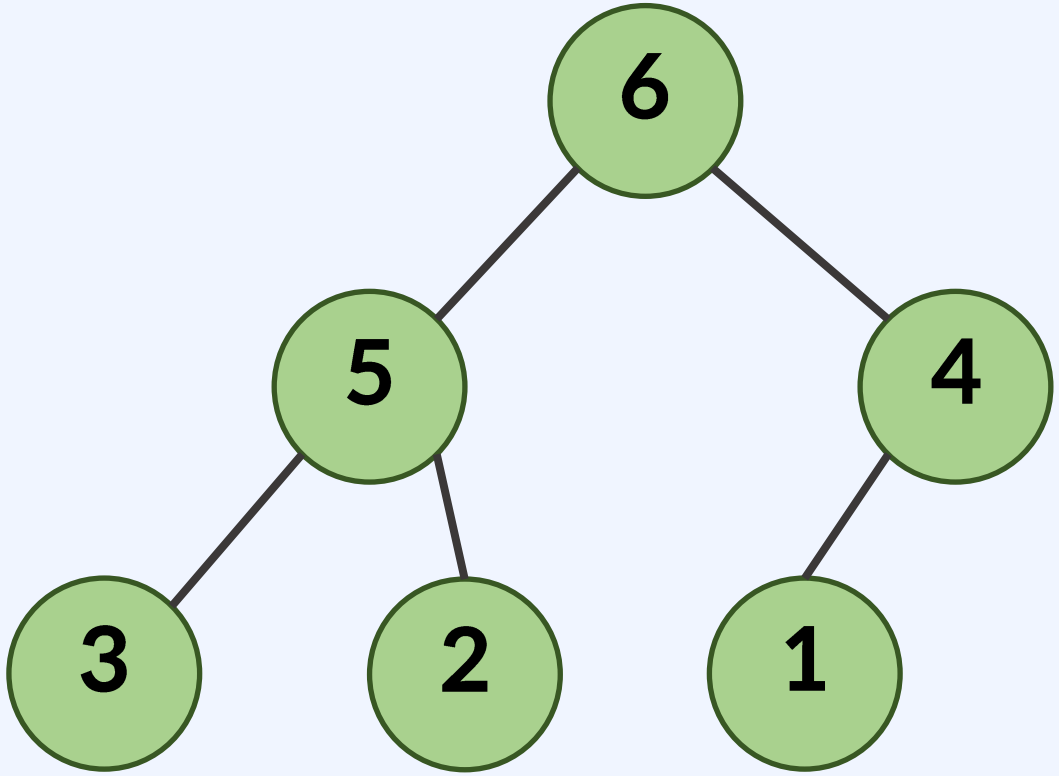
힙 자료구조

- 우선순위는 문제에 따라 다르게 지정될 수 있다
  - 일반적으로 {최대 값}, {최소 값} 을 기준으로 구현된다

- 편의상 이름은 큐라고 지칭하지만, 빠른 처리를 위해서는 트리 형태로 힙 자료구조를 구현해야 한다

BOJ11279: 최대 힙

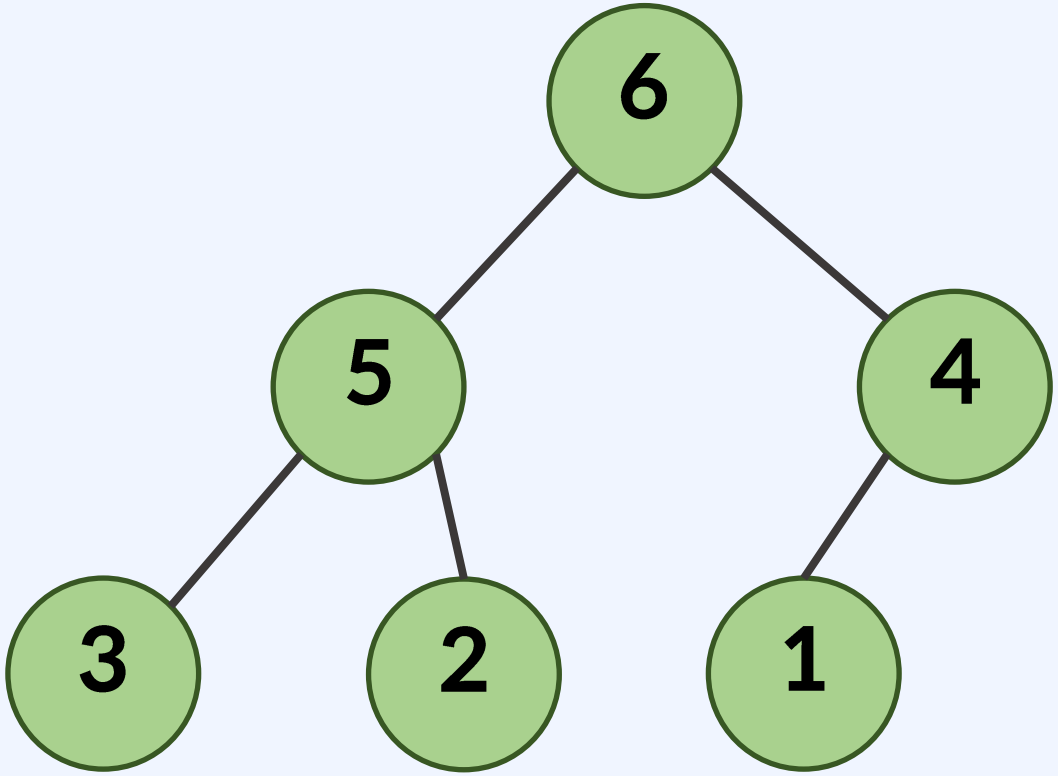
힙 자료구조 (최대 힙)



- 데이터 추가
  - O(n log n)
- 데이터 삭제
  - O(n log n)

BOJ11279: 최대 힙

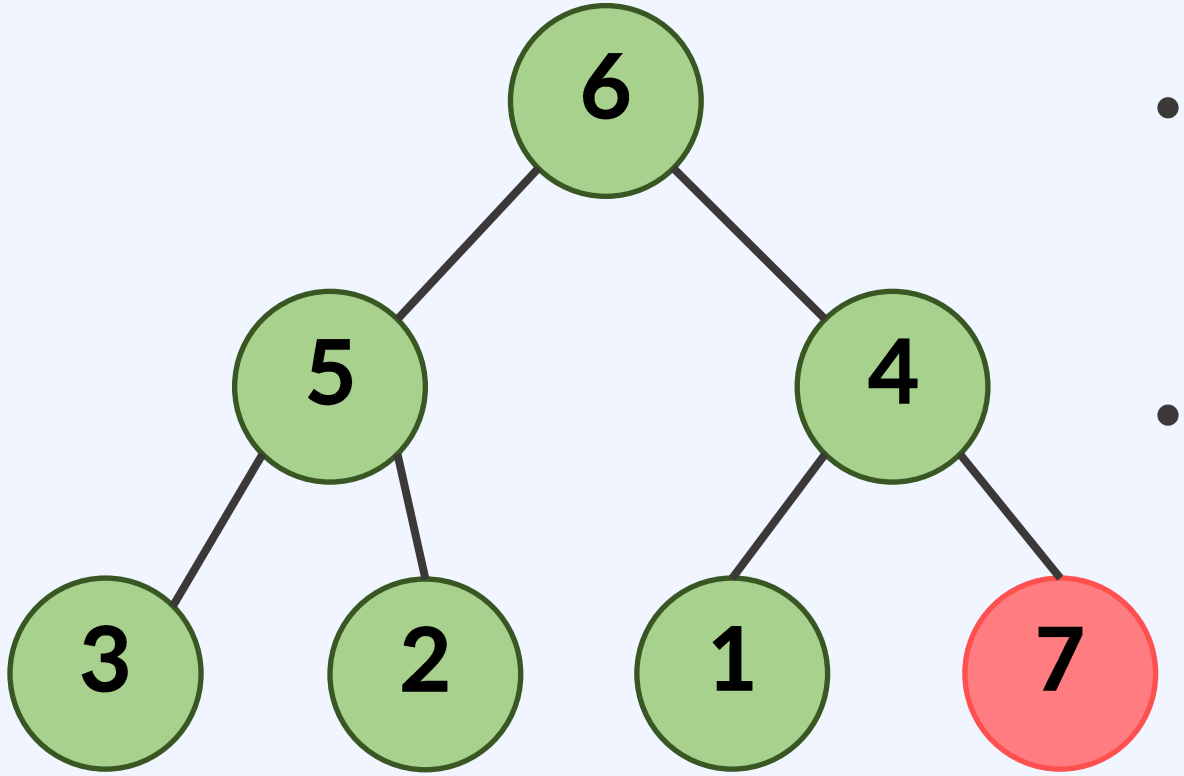
힙 자료구조 (최대 힙)



- 부모 노드는 자식 노드보다 크거나 같은 값을 가짐
- 루트 노드는 가장 큰 값을 가짐
- 힙의 서브트리는 동일하게 힙 구조를 만족함

BOJ11279: 최대 힙

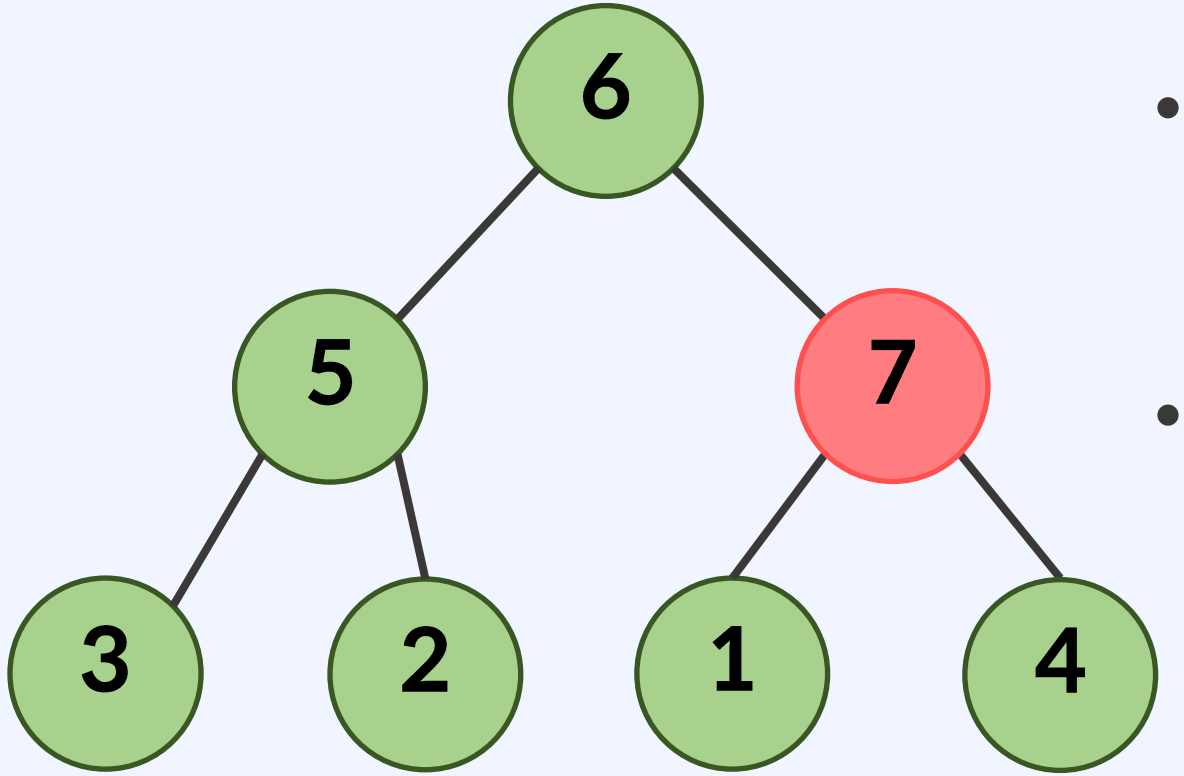
힙 자료구조 (최대 힙) – 데이터 추가



- (완전 이진트리를 만족하면서)
  가장 마지막 위치에 노드를 추가
- 부모가 자신보다 커질 때까지
  {자식노드} <-> {부모노드}
  위치를 변경

BOJ11279: 최대 힙

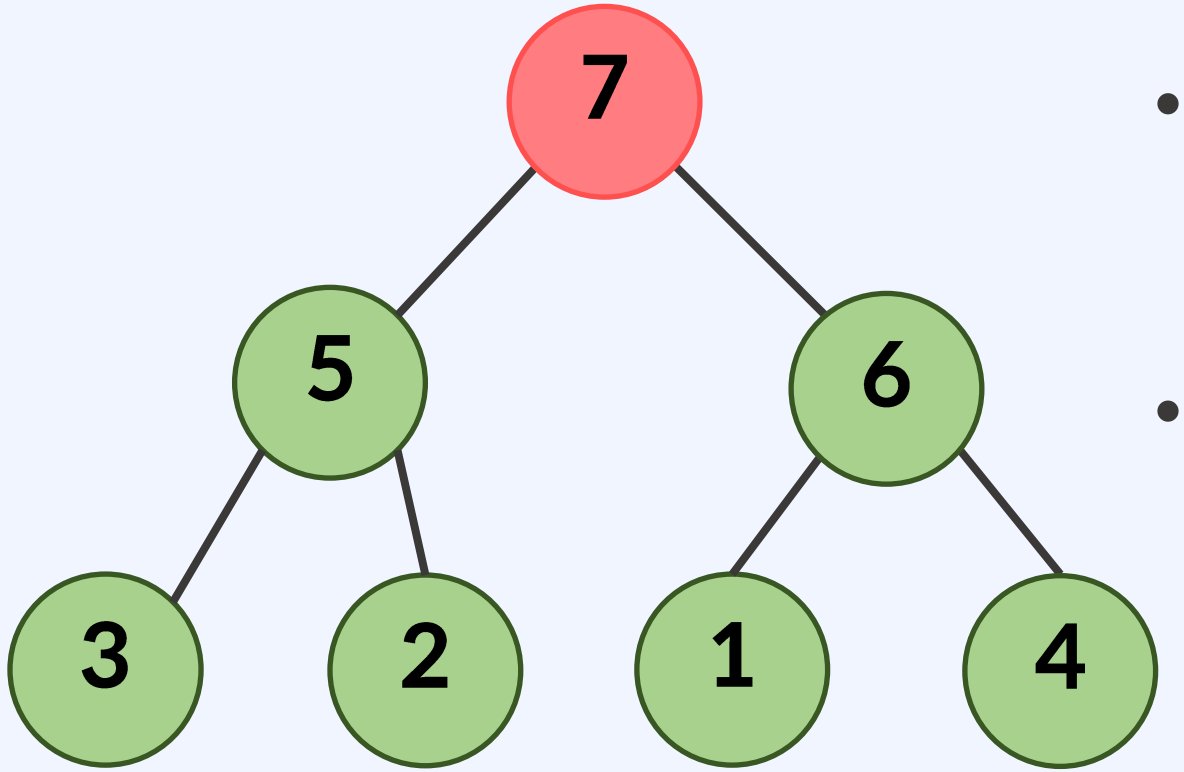
힙 자료구조 (최대 힙) – 데이터 추가



- (완전 이진트리를 만족하면서) 가장 마지막 위치에 노드를 추가
- 부모가 자신보다 커질 때까지 {자식노드} <-> {부모노드} 위치를 변경

BOJ11279: 최대 힙

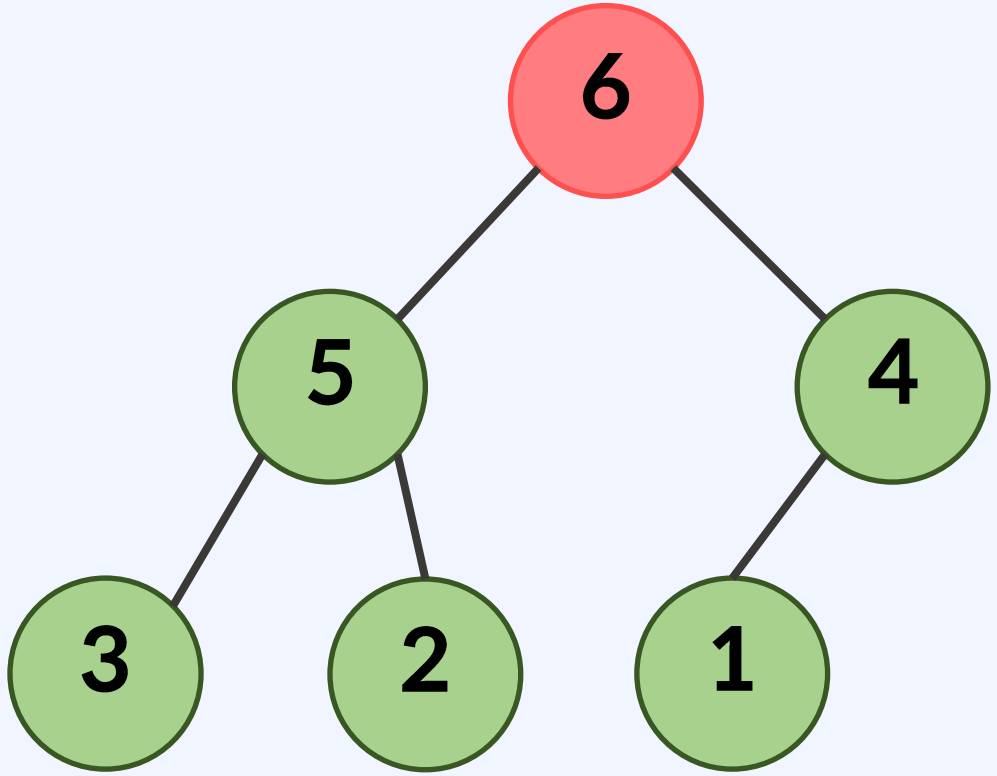
힙 자료구조 (최대 힙) – 데이터 추가



- (완전 이진트리를 만족하면서) 가장 마지막 위치에 노드를 추가
- 부모가 자신보다 커질 때까지 {자식노드} <-> {부모노드} 위치를 변경

BOJ11279: 최대 힙

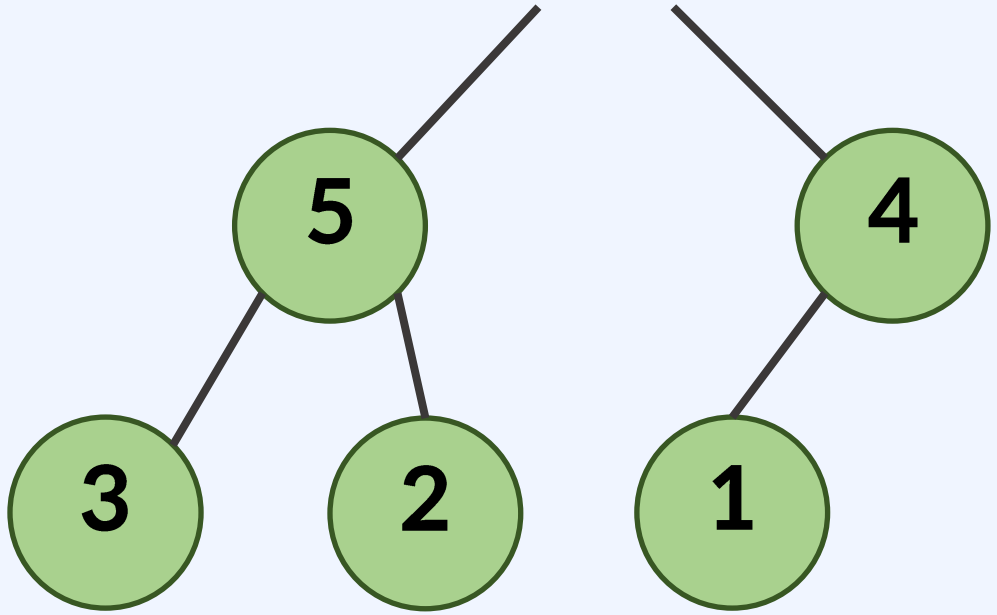
힙 자료구조 (최대 힙) – 데이터 삭제



- 삭제는 오직 루트노드에서만
- 루트노드를 제거하고
  가장 마지막 노드를 루트에 배치
- 현재 노드가 자식보다 작으면
  {현재 노드} <-> {자식 노드}
  위치를 변경
  (자식중에 더 작은 값 이랑 교환)

BOJ11279: 최대 힙

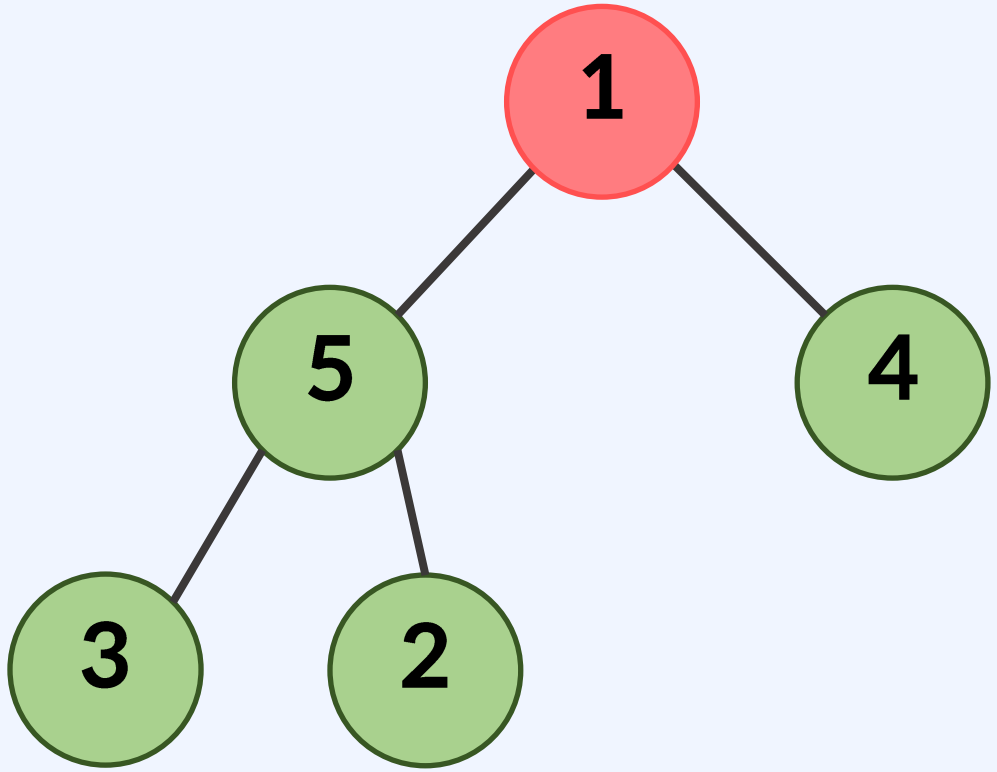
힙 자료구조 (최대 힙) – 데이터 삭제



- 삭제는 오직 루트노드에서만
- 루트노드를 제거하고
  가장 마지막 노드를 루트에 배치
- 현재 노드가 자식보다 작으면
  {현재 노드} <-> {자식 노드}
  위치를 변경
  (자식중에 더 작은 값 이랑 교환)

BOJ11279: 최대 힙

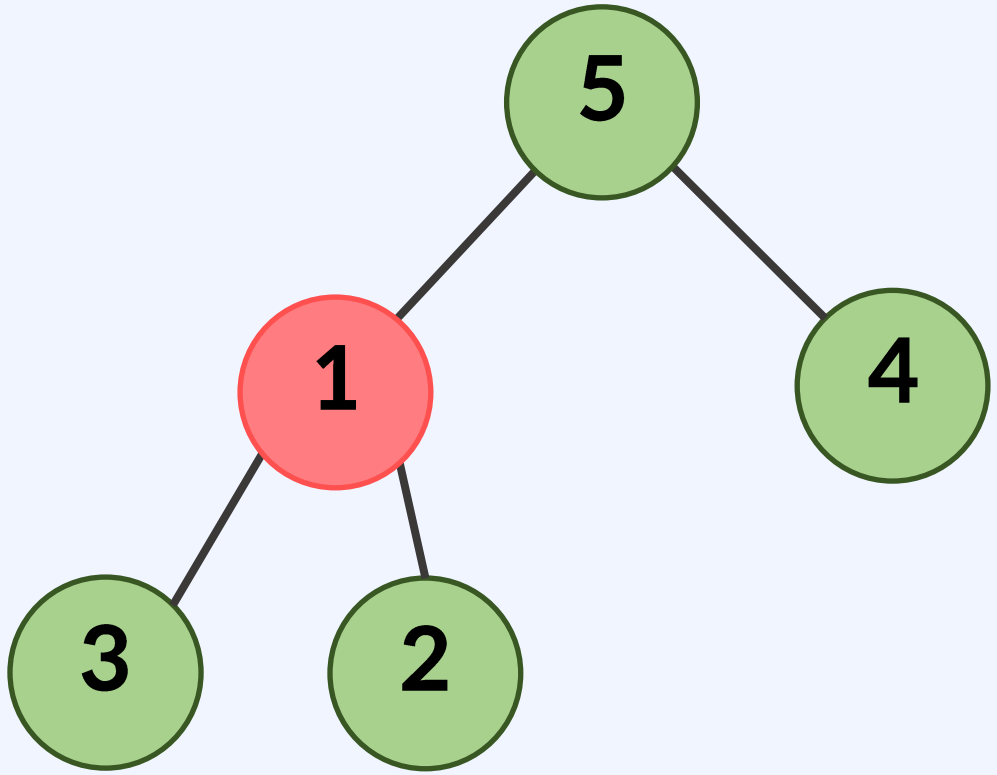
힙 자료구조 (최대 힙) – 데이터 삭제



- 삭제는 오직 루트노드에서만
- 루트노드를 제거하고
  가장 마지막 노드를 루트에 배치
- 현재 노드가 자식보다 작으면
  {현재 노드} <-> {자식 노드}
  위치를 변경
  (자식 중에 더 큰 값 이랑 교환)

BOJ11279: 최대 힙

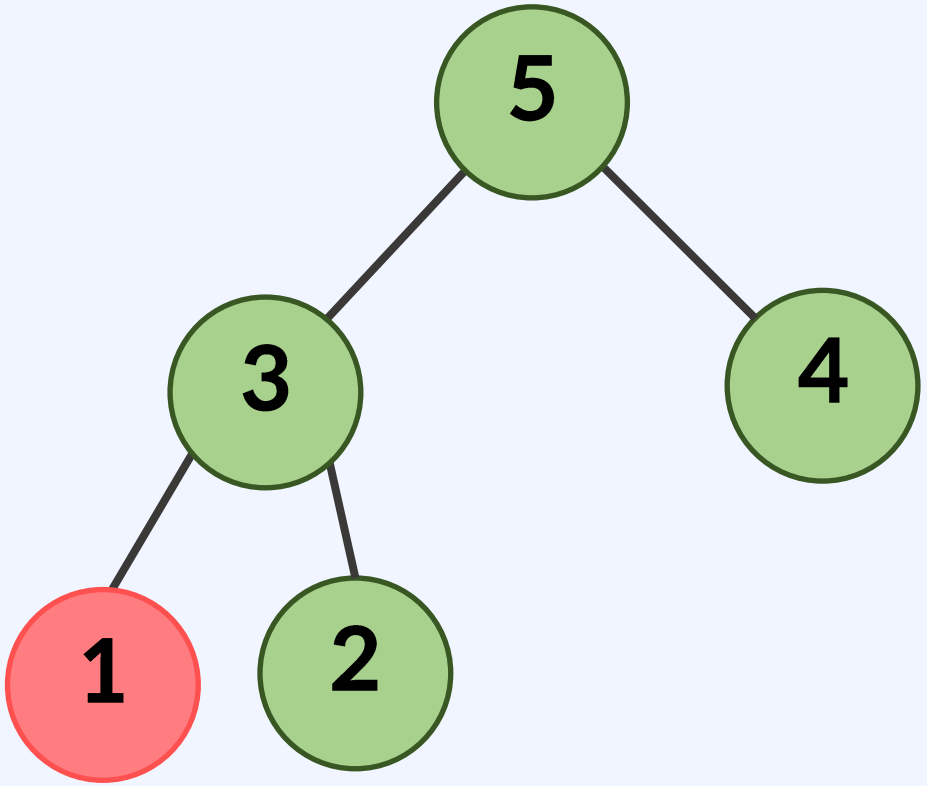
힙 자료구조 (최대 힙) – 데이터 삭제



- 삭제는 오직 루트노드에서만
- 루트노드를 제거하고
  가장 마지막 노드를 루트에 배치
- 현재 노드가 자식보다 작으면
  {현재 노드} <-> {자식 노드}
  위치를 변경
  (자식 중에 더 큰 값 이랑 교환)

BOJ11279: 최대 힙

힙 자료구조 (최대 힙) – 데이터 삭제



- 삭제는 오직 루트노드에서만
- 루트노드를 제거하고
  가장 마지막 노드를 루트에 배치
- 현재 노드가 자식보다 작으면
  {현재 노드} <-> {자식 노드}
  위치를 변경
  (자식 중에 더 큰 값 이랑 교환)

BOJ11279: 최대 힙

힙 자료구조 (최대 힙)

- 데이터 추가
  - O(n log n)
  - 데이터를 맨 뒤에 추가하고 힙의 균형을 맞추므로

- 데이터 삭제
  - O(n log n)
  - 루트를 삭제하고, 맨 뒤의 데이터를 루트로 올려 힙의 균형을 맞추므로

BOJ11279: 최대 힙

힙 자료구조 (최대 힙)

- 힙의 균형을 맞추는 작업은 왜 로그시간 복잡도인가?
  - 트리의 높이에 비례하므로
  - 각 높이별로 원소가 2의 거듭제곱 개수만큼 있다
  - 트리의 높이는 원소의 개수에 log2를 취한 결과와 같다

BOJ11279: 최대 힙

문제 요약

- 최대 힙 자료구조에 자연수 x 를 넣기
- x에 0이 들어오면 pop을 수행하며 출력
- 힙이 비어있다면 0을 출력

BOJ11279: 최대 힙

문제 분석

- 최대 힙을 구현하고, pop을 수행할때마다 데이터를 출력

BOJ11279: 최대 힙

구현

- 완전 이진 트리이므로, 트리의 구현은 간단하게 배열로 구현
- 현재 노드가 {i} 일때
  - 왼쪽자식: {i} * 2
  - 오른쪽 자식: {i} * 2 + 1

```
class MaxPriorityQueue {
    private int[] heap;
    private int size;

    public MaxPriorityQueue() {
        heap = new int[100001];
        size = 0;
    }
```

BOJ11279: 최대 힙

구현 - push

```
public void push(int x) {
    heap[++size] = x;
    int idx = size;
    while(idx > 1) {
        int current = idx;
        int parent = idx / 2;
        if(heap[parent] >= heap[current]) break;
        swap(parent, current);
        idx = parent;
    }
}
```

가장 마지막에 노드 추가

현재 위치의 노드와
부모 노드의 index를 획득

부모가 더 크거나 같으면
힙을 만족하므로 중단

그렇지 않다면
부모와 자식의 위치를 변경

BOJ11279: 최대 힙

구현 - pop

```
public int pop() {
    if(size == 0) return 0;
    int ret = heap[1];
    heap[1] = heap[size--];
    int idx = 1;
    while(idx * 2 <= size) {
        int left = idx * 2;
        int right = idx * 2 + 1;
// ...
```

루트노드를 리턴을위해 변수로 이동

마지막 노드를 루트로 배치

루트 노드부터 시작해서

왼쪽 자식과 오른쪽 자식 획득

BOJ11279: 최대 힙

구현 - pop

```
int child = left;
if(right <= size && heap[right] > heap[left]) {
    child = right;
}
if(heap[idx] >= heap[child]) break;
swap(idx, child);
current = child;
```

왼쪽 자식을 후보로 선택

만약 오른쪽 자식이 더 크다면,
오른쪽 방향으로 스왑 (단, 범위내)

자식이 더 작거나 같으면 중단
그렇지 않으면 위치를 바꾸고 반복

BOJ11279: 최대 힙

구현 - main

```
MaxPriorityQueue pq = new MaxPriorityQueue();
int n = sc.nextInt();
StringBuilder ans = new StringBuilder();
while(n-- > 0) {
    int x = sc.nextInt();
    if(x == 0) {
        ans.append(pq.pop()).append('\n');
    } else {
        pq.push(x);
    }
}
```

x에 0이 들어오면, pop()을 수행하며 데이터를 출력

Ch01. 우선순위 큐

2. [1927] 최소 힙

BOJ1927 : 최소 힙

문제 요약

- 최소 힙 자료구조에 자연수 x 를 넣기
- x에 0이 들어오면 pop을 수행하며 출력
- 힙이 비어있다면 0을 출력

BOJ1927 : 최소 힙

문제 분석

- 힙의 우선순위 조건이 {최대} → {최소}로 변경되었다
- 부등호의 방향만 반대로 변경해도
  정답을 받을 수 있다
- 이번 문제에서는 응용과 STL 사용법을 활용해본다

BOJ1927 : 최소 힙

구현 – [최대힙의 부등호 방향 변경]

push()

```
if (heap[parent] <= heap[current]) break;
```

pop()

```
if (right <= size && heap[right] < heap[left]) {
    child = right;
}
if (heap[current] <= heap[child]) break;
```

BOJ1927 : 최소 힙

구현 – [최대힙의 부등호 방향 변경]

문제	결과	메모리	시간
1927	맞았습니다!!	111968 KB	980 ms

부등호의 방향만 바꿔도 정답이 나온다

하지만 매번 힙 문제마다 구현하기에는 번거로우므로
라이브러리 사용법을 익혀보자

```
if (heap[current] <= heap[child]) break;
```

BOJ1927 : 최소 힙

구현 – [번외: 데이터의 부호 변경]

```
while(n-- > 0) {
    int x = sc.nextInt();
    if(x == 0) {
        ans.append(-pq.pop()).append('\n');
    } else {
        pq.push(-x);
    }
}
```

큐에 데이터를 넣고 뺄 때 부호를 반전시키면 최대/최소가 반전된다

BOJ1927 : 최소 힙

구현 – [STL 사용]

```
PriorityQueue<Integer> pq = new PriorityQueue<>();
while (n-- > 0) {
    int x = sc.nextInt();
    if (x == 0) {
        if(pq.isEmpty())
            ans.append(0).append('\n');
        else
            ans.append(pq.poll()).append('\n');
    }
    else pq.offer(x);
}
```

힙이 비어있는지 검사: isEmpty()

데이터 꺼내기:
- poll()
- remove()

데이터 추가:
- offer()
- add()

BOJ1927 : 최소 힙

구현 – [STL 사용]

- 추가
  - offer()
  - add()
- 삭제
  - poll()
  - remove()

- 루트 데이터 확인
  - peek()
  - element()

- 데이터에 문제가 있다면?
  - false / null 반환
  - Exception 반환

Ch01. 우선순위 큐

3. [11286] 절댓값 힙

BOJ11286 : 절댓값 힙

문제 요약

- 우선순위가 절댓값이 작을수록 높은 힙
- 단, 같은 절댓값을 가지고 있다면?
  - 음수가 우선순위가 더 높다
- 입력으로 0이 들어오면 힙에서 원소를 꺼내서 출력

BOJ11286 : 절댓값 힙

구현 – [STL 사용]

우선순위 조건 변경이 필요한 경우

```
PriorityQueue<Integer> pq = new PriorityQueue<>((o1, o2) -> {
    if(Math.abs(o1) == Math.abs(o2))
        return o1 < o2 ? -1 : 1;
    return Math.abs(o1) < Math.abs(o2) ? -1 : 1;
});
```

- Comparator를 람다함수로 정의하면 된다
- 이를 이용해서 임의의 클래스에서도 사용이 가능하다 (ex. Point)

Ch01. 우선순위 큐

4. [2075] N번째 큰 수

BOJ2075 : N번째 큰 수

문제 요약

- N * N 표 안에 있는 수중에 N번째로 큰 수를 찾는 문제

- 단, 메모리를 12MB만 사용해야 한다
  - (1,500 * 1,500) 데이터를 모두 입력 받으면 메모리 초과가 발생한다

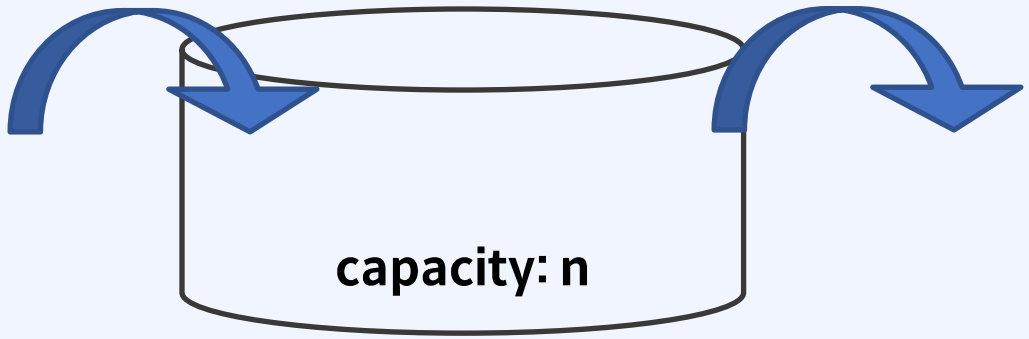
- [r+1][c]의 수는 [r][c]보다 크다는 것이 보장된다
  - (힙을 이용해 구현하면, 무시해도 되는 조건)

BOJ2075 : N번째 큰 수

문제 분석

1	2	3	4	5	6	7	8	9

- (n == 3) 으로
  9개의 숫자 중 3번째로 큰 수를 찾는다고 생각해보자

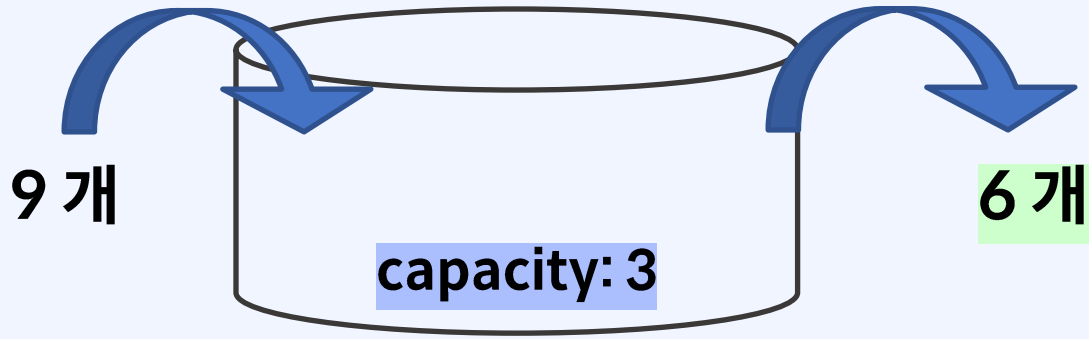


BOJ2075 : N번째 큰 수

문제 분석

1	2	3	4	5	6	7	8	9

- 모든 데이터를 넣었다 뺐을 때
  힙 안에 오름차순으로 n개의 데이터만 남기려면?



BOJ2075 : N번째 큰 수

문제 분석

- 최소 힙으로 구현하면?
  - 작은 수부터 힙에서 제거되므로
    힙 내부에는 내림차순 n개의 값이 남아있다
    → 힙의 ROOT에 해당하는 수가 N번째 큰 수다

BOJ2075 : N번째 큰 수

구현

최소힙 선언

```
PriorityQueue<Integer> pq = new PriorityQueue<>() ;
```

N개 데이터 추가

```
for(int i = 0; i < n; i++) pq.offer(sc.nextInt());
```

BOJ2075 : N번째 큰 수

구현

```
for(int i = 0; i < n; i++) {
    for(int j = 0; j < n; j++) {
        pq.offer(sc.nextInt());
        pq.poll();
    }
}
System.out.println(pq.poll());
```

N개 데이터를 추가

N개 데이터를 제거

작은 수부터 제거되어, 큰 N개의 수가 힙에 남아있다
최소 힙이므로 루트 노드가 N번째로 큰 값

Ch01. 우선순위 큐

5. [1655] 가운데를 말해요

BOJ1655 : 가운데를 말해요

문제 요약

- 정수가 들어올 때 마다,
  모든 수들의 중앙값을 출력하는 문제

- 정수의 개수가 짝수개라면 작은 값을 출력

- 정수의 개수: N (1 <= N <= 100,000)
  - 정수의 범위: (-10,000 <= {정수} <= 10,000)

BOJ1655 : 가운데를 말해요

문제 분석

- 최대힙 / 최소힙 은 각각 모든 수 중에서
  최소 / 최대 값을 빠르게 획득하는 자료 구조이다

- 중앙 값을 빠르게 획득하려면?
  - 최소 최대힙 2개를 활용하는 방법이 있다

BOJ1655 : 가운데를 말해요

문제 분석

중앙 값 보다 작거나
같은 값을 담는 힙 (최대 힙)

중앙 값보다
큰 값을 담는 힙 (최소 힙)

1. 두개의 힙에 수를 번갈아가며 넣는다
   (사이즈가 같다면 최대 힙 먼저)
2. 각 힙의 루트에서 수를 뽑은 뒤, 대소관계를 비교한다
3. 오른쪽힙에 더 작은값이 있다면 위치를 서로 바꾼다

BOJ1655 : 가운데를 말해요

1. 두개의 힙에 수를 번갈아가며 넣는다
   (사이즈가 같다면 최대 힙 먼저)
2. 각 힙의 루트에서 수를 뽑은 뒤, 대소관계를 비교한다
3. 오른쪽힙에 더 작은 값이 있다면 위치를 서로 바꾼다
4. 중앙 값은 최대힙의 ROOT에 위치한다

문제 분석

중앙 값 보다 작거나
같은 값을 담는 힙 (최대 힙)

중앙 값보다
큰 값을 담는 힙 (최소 힙)

BOJ1655 : 가운데를 말해요

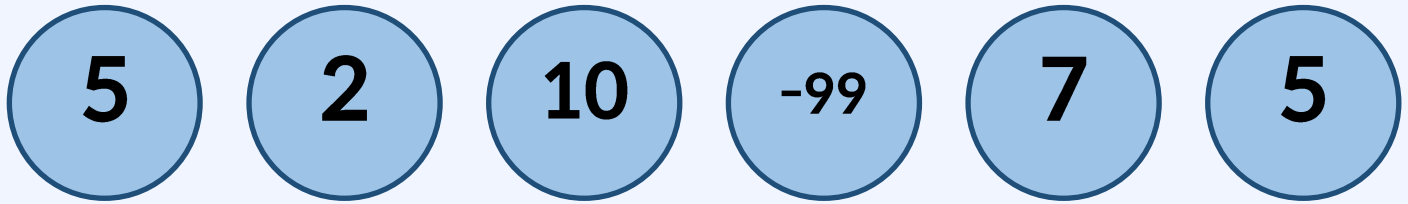
1. 두개의 힙에 수를 번갈아가며 넣는다
   (사이즈가 같다면 최대 힙 먼저)
2. 각 힙의 루트에서 수를 뽑은 뒤, 대소관계를 비교한다
3. 오른쪽힙에 더 작은 값이 있다면 위치를 서로 바꾼다
4. 중앙 값은 최대힙의 ROOT에 위치한다

말해요

문제 분석

중앙 값 보다 작거나
같은 값을 담는 힙 (최대 힙)

중앙 값보다
큰 값을 담는 힙 (최소 힙)



BOJ1655 : 가운데를 말해요

1. 두개의 힙에 수를 번갈아가며 넣는다
   (사이즈가 같다면 최대 힙 먼저)
2. 각 힙의 루트에서 수를 뽑은 뒤, 대소관계를 비교한다
3. 오른쪽힙에 더 작은 값이 있다면 위치를 서로 바꾼다
4. 중앙 값은 최대힙의 ROOT에 위치한다

문제 분석

중앙 값 보다 작거나
같은 값을 담는 힙 (최대 힙)

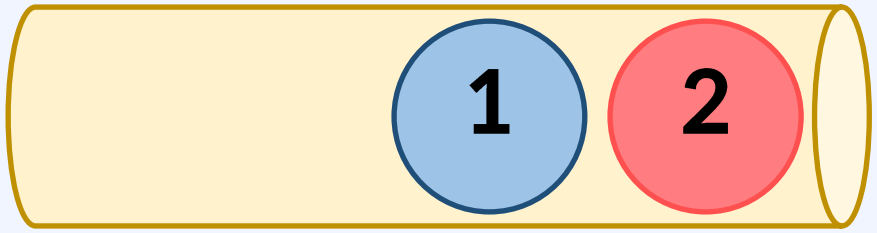
중앙 값보다
큰 값을 담는 힙 (최소 힙)

BOJ1655 : 가운데를 말해요

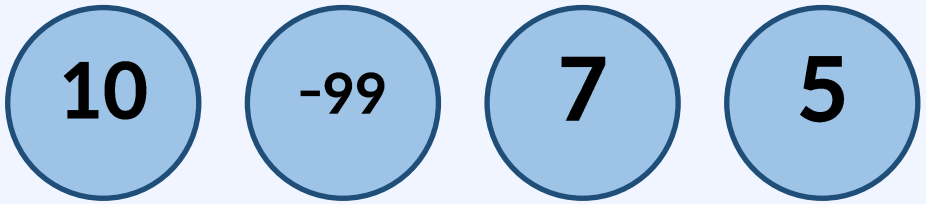
1. 두개의 힙에 수를 번갈아가며 넣는다
   (사이즈가 같다면 최대 힙 먼저)
2. 각 힙의 루트에서 수를 뽑은 뒤, 대소관계를 비교한다
3. 오른쪽힙에 더 작은 값이 있다면 위치를 서로 바꾼다
4. 중앙 값은 최대힙의 ROOT에 위치한다

문제 분석

중앙 값 보다 작거나
같은 값을 담는 힙 (최대 힙)



중앙 값보다
큰 값을 담는 힙 (최소 힙)

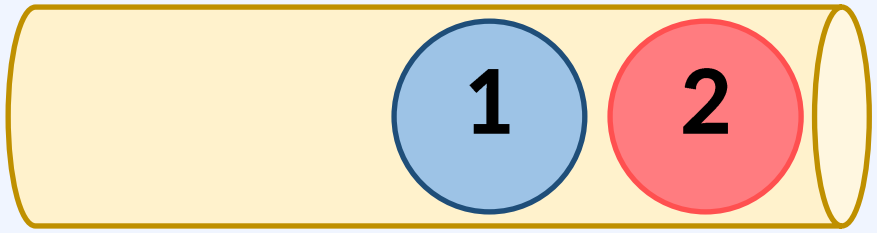


BOJ1655 : 가운데를 말해요

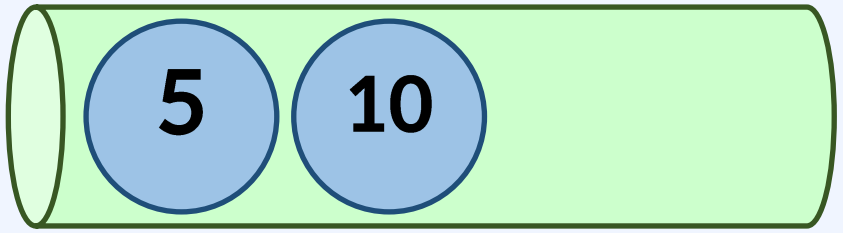
1. 두개의 힙에 수를 번갈아가며 넣는다
   (사이즈가 같다면 최대 힙 먼저)
2. 각 힙의 루트에서 수를 뽑은 뒤, 대소관계를 비교한다
3. 오른쪽힙에 더 작은 값이 있다면 위치를 서로 바꾼다
4. 중앙 값은 최대힙의 ROOT에 위치한다

문제 분석

중앙 값 보다 작거나
같은 값을 담는 힙 (최대 힙)



중앙 값보다
큰 값을 담는 힙 (최소 힙)

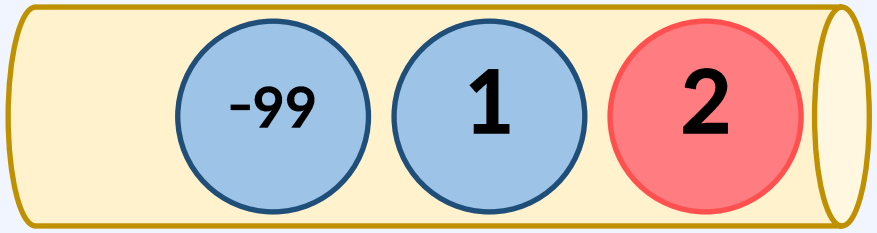


BOJ1655 : 가운데를 말해요

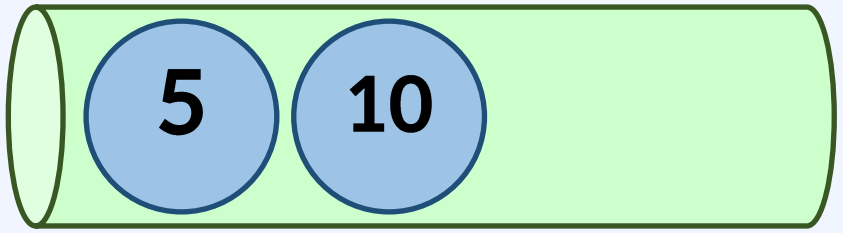
1. 두개의 힙에 수를 번갈아가며 넣는다
   (사이즈가 같다면 최대 힙 먼저)
2. 각 힙의 루트에서 수를 뽑은 뒤, 대소관계를 비교한다
3. 오른쪽힙에 더 작은 값이 있다면 위치를 서로 바꾼다
4. 중앙 값은 최대힙의 ROOT에 위치한다

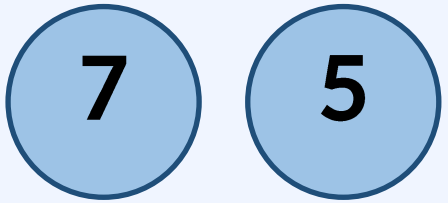
문제 분석

중앙 값 보다 작거나
같은 값을 담는 힙 (최대 힙)



중앙 값보다
큰 값을 담는 힙 (최소 힙)





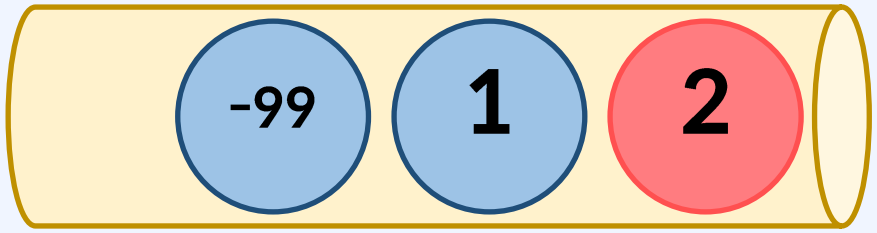
BOJ1655 : 가운데를 말해요

1. 두개의 힙에 수를 번갈아가며 넣는다
   (사이즈가 같다면 최대 힙 먼저)
2. 각 힙의 루트에서 수를 뽑은 뒤, 대소관계를 비교한다
3. 오른쪽힙에 더 작은 값이 있다면 위치를 서로 바꾼다
4. 중앙 값은 최대힙의 ROOT에 위치한다

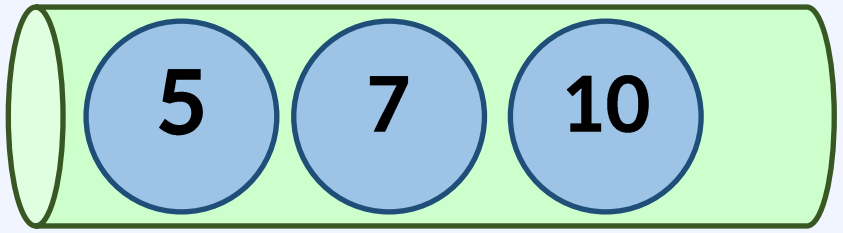
말해요

문제 분석

중앙 값 보다 작거나
같은 값을 담는 힙 (최대 힙)



중앙 값보다
큰 값을 담는 힙 (최소 힙)



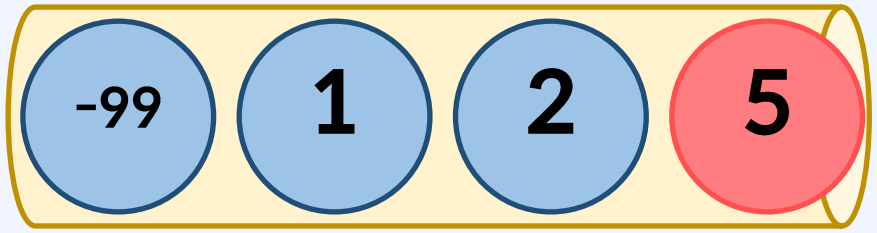
BOJ1655 : 가운데를 말해요

1. 두개의 힙에 수를 번갈아가며 넣는다
   (사이즈가 같다면 최대 힙 먼저)
2. 각 힙의 루트에서 수를 뽑은 뒤, 대소관계를 비교한다
3. 오른쪽힙에 더 작은 값이 있다면 위치를 서로 바꾼다
4. 중앙 값은 최대힙의 ROOT에 위치한다

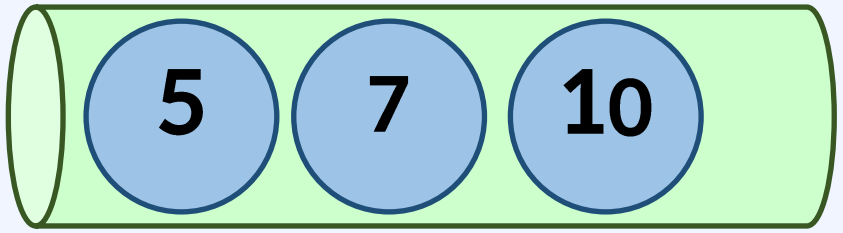
말해요

문제 분석

중앙 값 보다 작거나
같은 값을 담는 힙 (최대 힙)



중앙 값보다
큰 값을 담는 힙 (최소 힙)

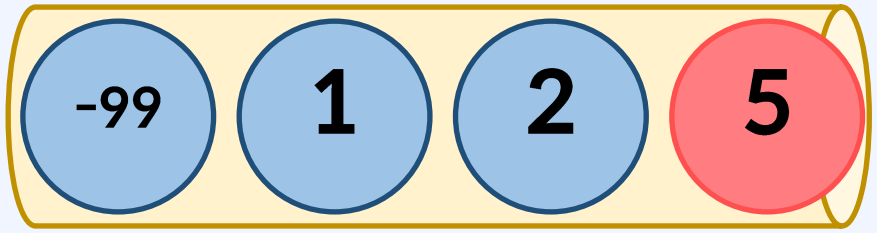


BOJ1655 : 가운데를 말해요

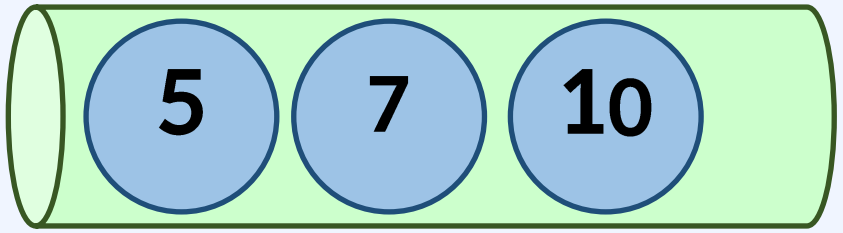
1. 두개의 힙에 수를 번갈아가며 넣는다
   (사이즈가 같다면 최대 힙 먼저)
2. 각 힙의 루트에서 수를 뽑은 뒤, 대소관계를 비교한다
3. 오른쪽힙에 더 작은 값이 있다면 위치를 서로 바꾼다
4. 중앙 값은 최대힙의 ROOT에 위치한다

문제 분석

중앙 값 보다 작거나
같은 값을 담는 힙 (최대 힙)



중앙 값보다
큰 값을 담는 힙 (최소 힙)



{2} 가 추가된다면?

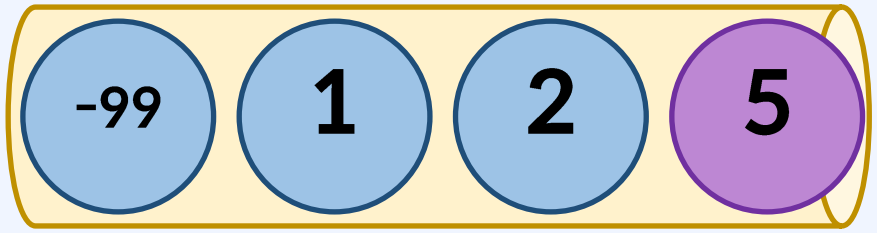
BOJ1655 : 가운데를 말해요

1. 두개의 힙에 수를 번갈아가며 넣는다
   (사이즈가 같다면 최대 힙 먼저)
2. 각 힙의 루트에서 수를 뽑은 뒤, 대소관계를 비교한다
3. 오른쪽힙에 더 작은 값이 있다면 위치를 서로 바꾼다
4. 중앙 값은 최대힙의 ROOT에 위치한다

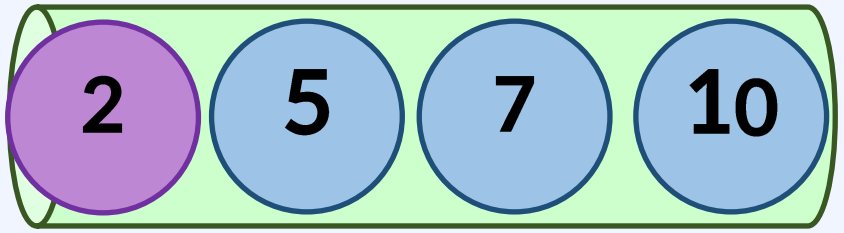
말해요

문제 분석

중앙 값 보다 작거나
같은 값을 담는 힙 (최대 힙)



중앙 값보다
큰 값을 담는 힙 (최소 힙)



오른쪽 힙에 더 작은 값이 ROOT에 있다

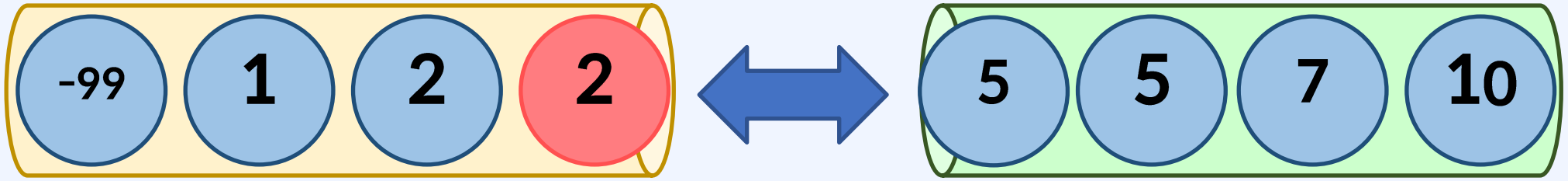
BOJ1655 : 가운데를 말해요

1. 두개의 힙에 수를 번갈아가며 넣는다
   (사이즈가 같다면 최대 힙 먼저)
2. 각 힙의 루트에서 수를 뽑은 뒤, 대소관계를 비교한다
3. 오른쪽힙에 더 작은 값이 있다면 위치를 서로 바꾼다
4. 중앙 값은 최대힙의 ROOT에 위치한다

문제 분석

중앙 값 보다 작거나
같은 값을 담는 힙 (최대 힙)

중앙 값보다
큰 값을 담는 힙 (최소 힙)



두 수를 힙에서 뽑아 위치를 바꾼다

BOJ1655 : 가운데를 말해요

구현 – 큐 선언

```
PriorityQueue<Integer> small = new PriorityQueue<>((o1, o2) -> {
    return o1 < o2 ? 1 : -1;
});

PriorityQueue<Integer> big = new PriorityQueue<>();
```

BOJ1655 : 가운데를 말해요

```
for (int i = 0; i < n; i++) {
    int num = sc.nextInt();
    if (small.size() == big.size()) small.offer(num);
    else big.offer(num);
    if (!small.isEmpty() && !big.isEmpty()) {
        int s = small.peek(); int b = big.peek();
        if (s > b) {
            small.poll();
            big.poll();
            small.offer(b);
            big.offer(s);
        }
    }
```

힙에 번갈아가며 수 추가

오른쪽 힙 루트에 작은값이 있다면?

둘의 위치를 교환

Ch01. 우선순위 큐

6. [19598] 최소 회의실 개수

BOJ19598 : 최소 회의실 개수

문제 요약

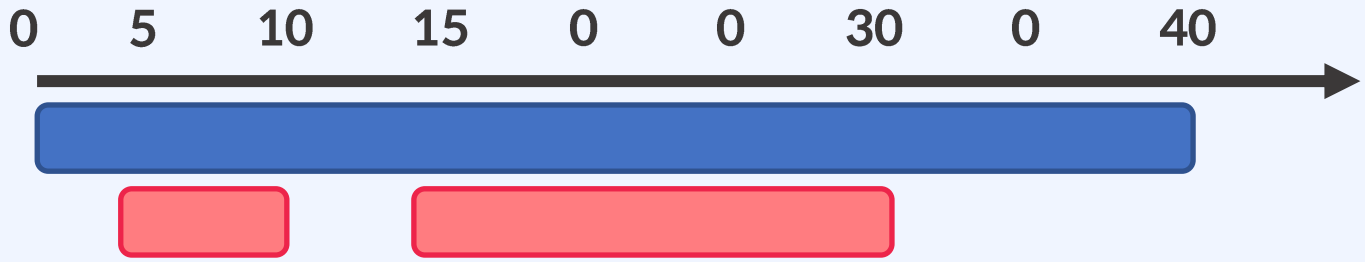
- N개의 회의의 {시작, 종료} 시각이 주어진다
- 한 회의실에서 동시에 두개의 회의가 진행될 수 없다
- 최소 몇 개의 회의실이 필요한지 카운트
- 단, 회의가 끝나는 것과 동시에 다음 회의가 시작될 수 있다

BOJ19598 : 최소 회의실 개수

문제 요약

입력 데이터
3 0 40 15 30 5 10

출력 데이터
2



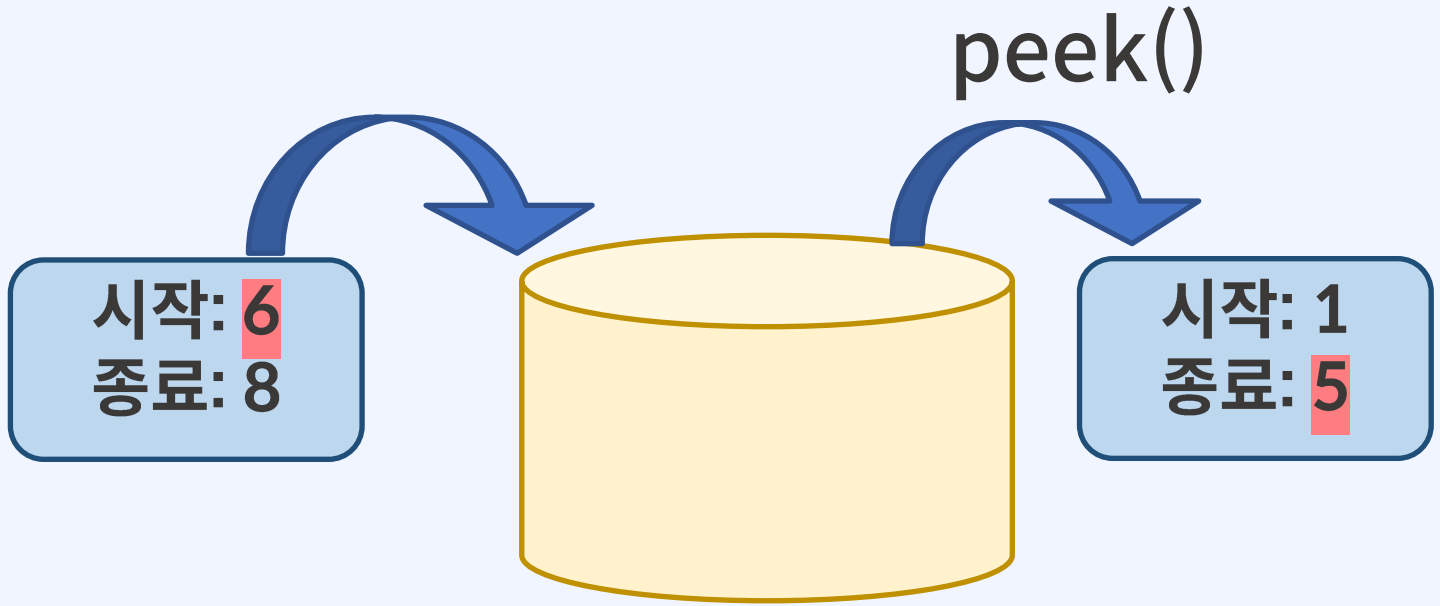
BOJ19598 : 최소 회의실 개수

문제 분석

- 아직 회의가 끝나지 않았다면, 새로운 회의실이 필요하다
- 회의가 끝나는 시각을 기준으로, 가장 먼저 끝나는 회의를 찾을 수 있는 최소 힙을 만들어 보자
- 새로운 회의의 시작 시각이, 가장 먼저 끝날 회의보다 뒤라면, 회의실을 재사용할 수 있다

BOJ19598 : 최소 회의실 개수

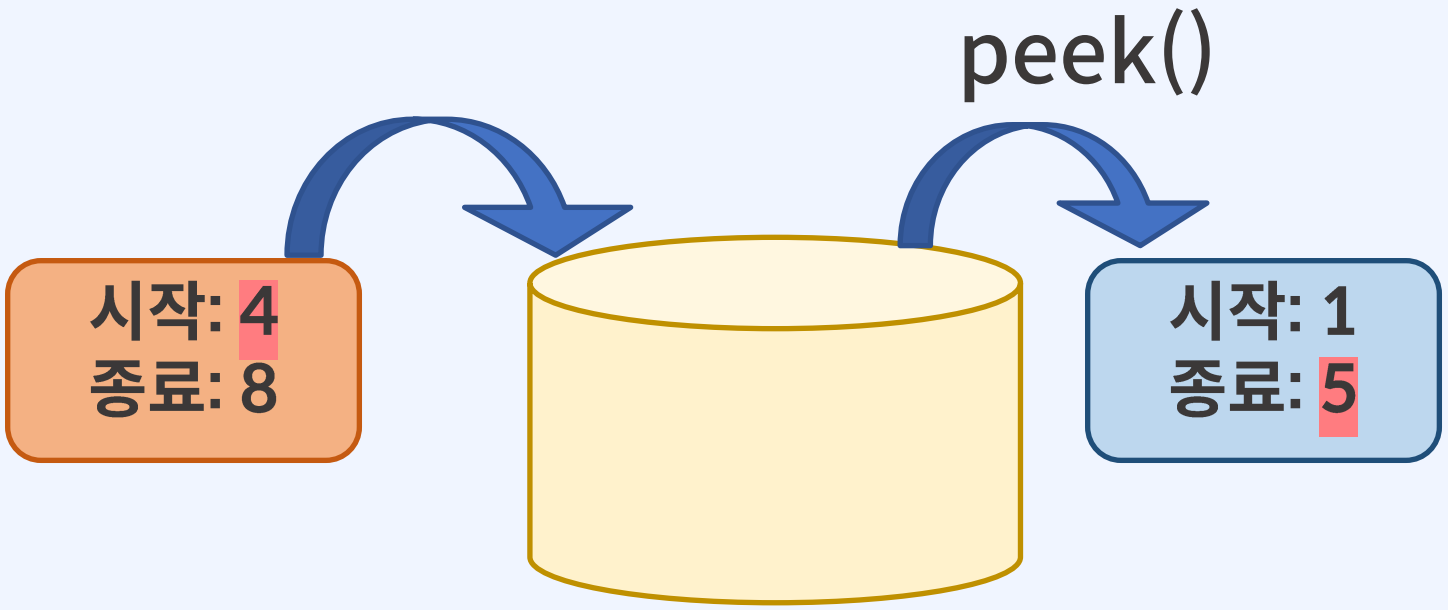
문제 분석



같은 회의실 사용 가능

BOJ19598 : 최소 회의실 개수

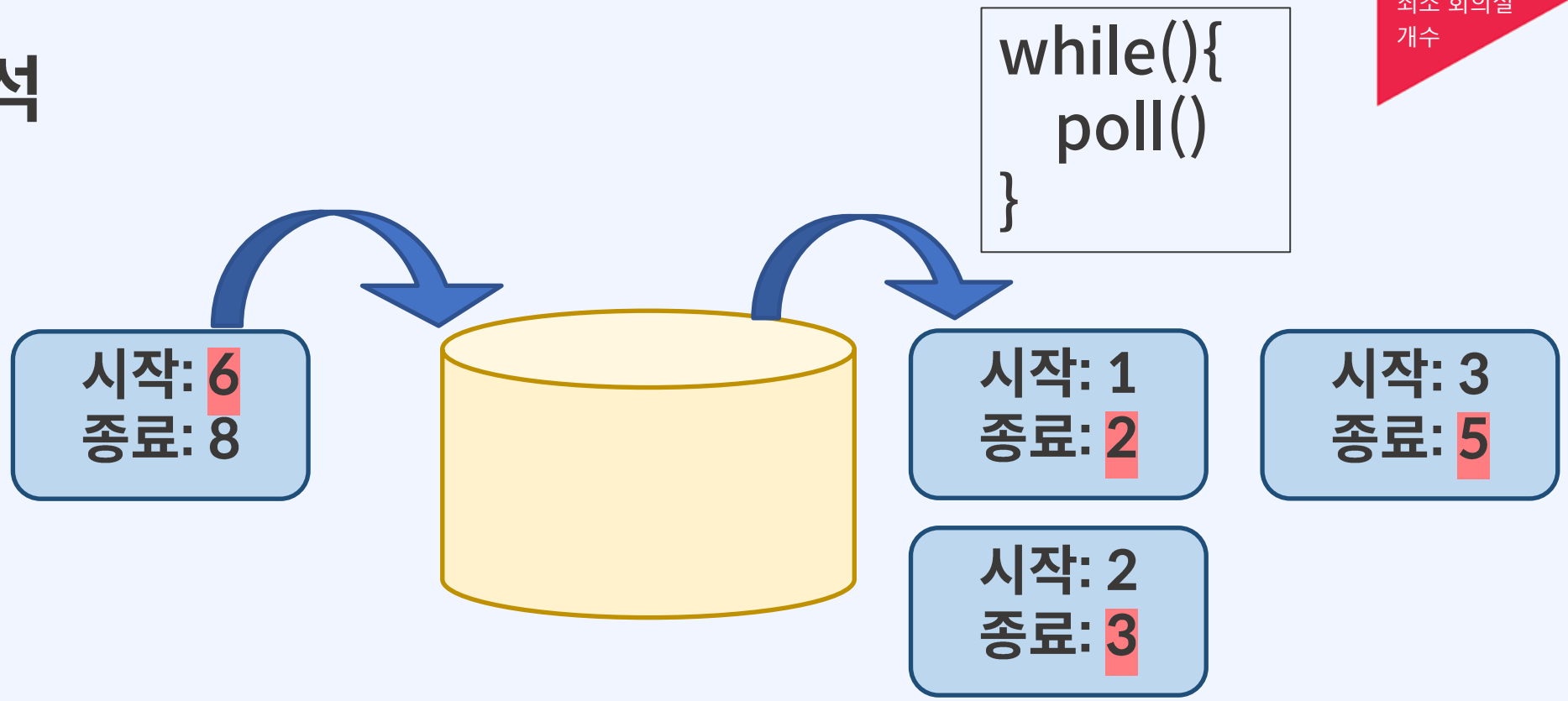
문제 분석



같은 회의실 사용 불가능

BOJ19598 : 최소 회의실 개수

문제 분석



[주의할 점] 새로운 회의보다 종료가 먼저인 회의가 여러 개일 수 있다
반복문으로 더 이상 해당 케이스가 없을 때 까지 뽑는다

BOJ19598 : 최소 회의실 개수

구현

```
class Meeting {
    int start;
    int end;

    public Meeting(int start, int end) {
        this.start = start;
        this.end = end;
    }
}
```

회의의 시작 / 끝나는 시각을 기록할 클래스

BOJ19598 : 최소 회의실 개수

구현

```
meeting.sort((o1, o2) -> {
    if (o1.start == o2.start) return o1.end - o2.end;
    return o1.start - o2.start;
});

PriorityQueue<Meeting> pq = new PriorityQueue<>((o1, o2) -> {
    return o1.end - o2.end;
});
```

회의는 시작하는 순서대로 정렬, 우선순위 큐는 끝나는 시각을 기준으로
둘 다 오름차순으로 관리

BOJ19598 : 최소 회의실 개수

구현

```
for (Meeting m : meeting) {
    if (pq.isEmpty()) {
        pq.offer(m);
        continue;
    }
    while (!pq.isEmpty() && pq.peek().end <= m.start)
        pq.poll();
    pq.offer(m);
    ans = Math.max(ans, pq.size());
}
```

회의실이 모두 비어 있다면 힙에 넣고 스킵

끝나는 시각이 새 회의보다 더 빠른 회의 모두 제거

회의 추가

최대 회의실은 힙의 최대 사이즈