sorting_Searching

제로 이동(Move Zeros)
k번째 제일큰 원소 (Kth Largest Element In An Array)
원점에 가장 가까운 지점 (K Closest Points to Origin)
미팅룸(Meeting Room)
미팅룸2(Meeting Room2)
interval 병합(Merge Interval)
로그 파일의 데이터 재정렬

사전지식

1) Arrays.sort()
2) PriortiQueue 사용법

제로 이동(Move Zeros)

설명

정수 배열(nums)이 주어지면 0이 아닌 값은 상대적 순서를 유지하고
nums모든 0은 끝으로 이동하게 만드세요.

Note) 배열의 복사본을 만들지 않고 작업을 수행해야합니다.

입출력

Input: nums = [0,3,2,0,8,5]
Output: [3,2,8,5,0,0]

Input: nums = [0]
Output: [0]

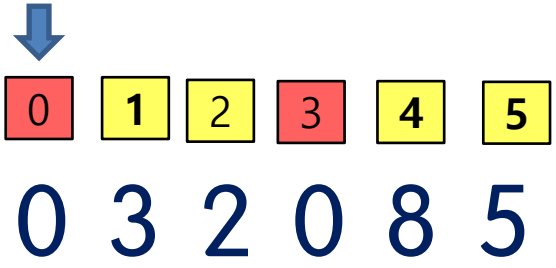
문제 Format

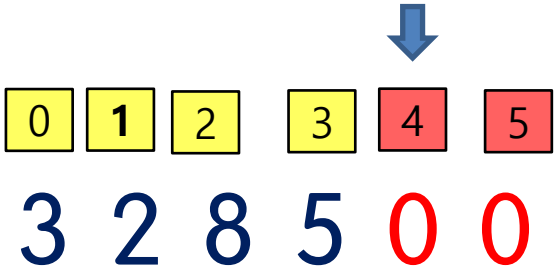
```
class Solution {
 public void solve(int[] nums) { }
}
```

제한사항

$1 <= nums.length <= 10^4$
$-2^{31} <= nums[i] <= 2^{31} - 1$

문제분석





1. 값이 0이 아닌값을먼저 array에 담는다
2. Index를 기억한다
3. 해당 index에 0인 값을 넣는다

시간복잡도/공간복잡도 계산

시간복잡도

1. 대상(Source) : 문제에서 입력받은 파라미터(array 등) (속도)
   Time Complexity : O(N)
   대상 : int[] nums
   이유 : for문 한번 실행

공간복잡도

2. 대상(Source) : 실제 사용되는 저장 공간을 계산(메모리 사용량)
   예) 프로그램을 실행 및 완료하는데 필요한 저장공간
   Space Complexity : O(1)
   대상 :
   이유 : 추가적인 공간을 사용안함.

참고

O(1) : 스택, 큐,Map
O(n) : for문 => 데이터를 한번씩 다 호출하니까 (제일 많음)
O(log N) : sort, prirotiyQueue, binary Search Tree, Tree
O(Klog N) : k번만큼 소팅하는 경우
O(n^2) : 이중for문
O(m*n) : 이중for문인데, n이 다른경우bfs,dfs 류 (예 n=100 인데 m=5인경우)

k번째 제일큰 원소 (Kth Largest Element In An Array)

설명

정수 배열 nums와 정수 k가 주어지면 배열에서 k번째로 큰 요소를 반환합니다 .

Note)
k번째로 큰 요소는 정렬후 값에대한 가장 큰 요소가 아닌 유일한(distinctive) 순서 요소입니다.

입출력

Input: nums = [2,3,1,5,6,4], k = 2
Output: 5

Input: nums = [3,2,3,1,2,4,5,5,6], k = 4
Output: 4

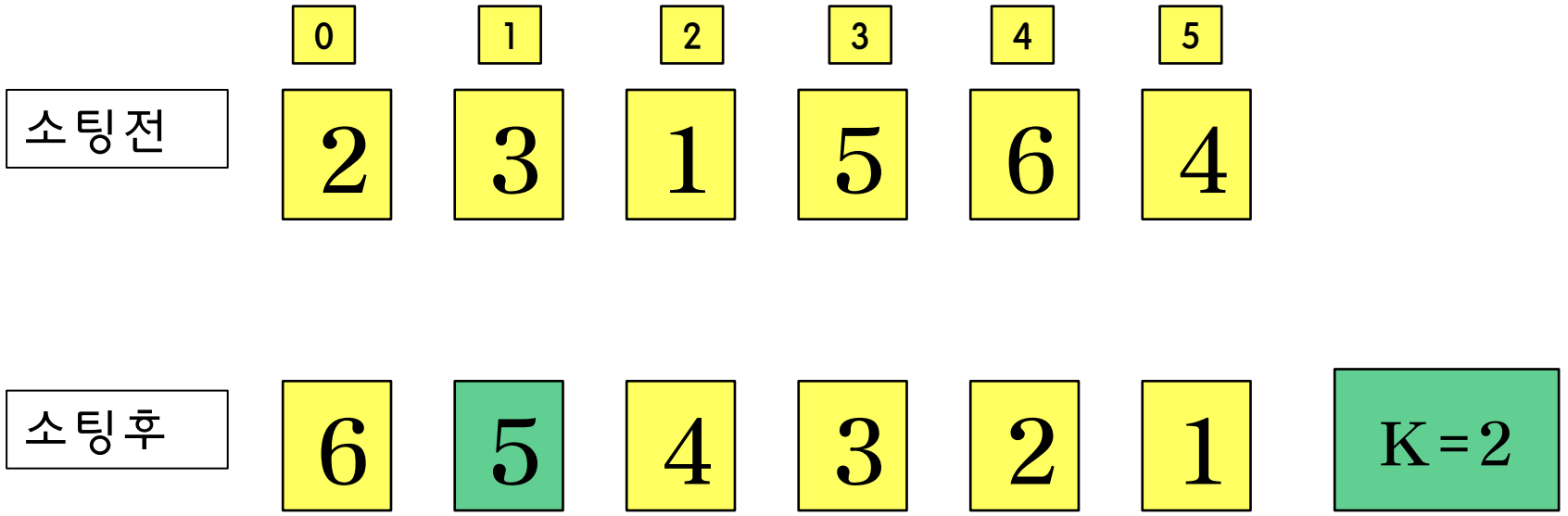
문제 Format

```
class Solution {
 public int solve(int[] nums, int k) { }
}
```

제한사항

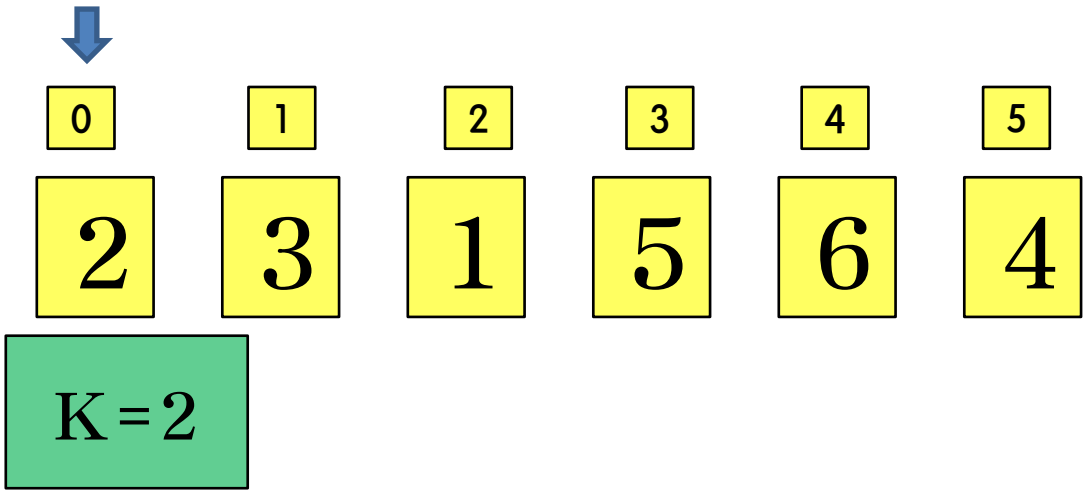
$1 <= k <= nums.length <= 10^4$
$-10^4 <= nums[i] <= 10^4$

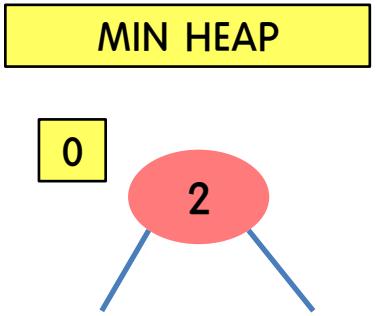
문제분석



1. 값을 소팅한다
2. K=2번째로 큰값을 찾는다

문제분석



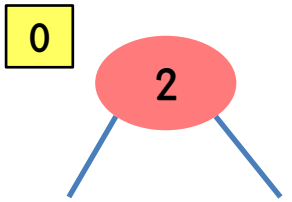


1. pq 를 이용해서 사이즈 2를 유지하면
2. 결국에 5,6만 남게 된다.

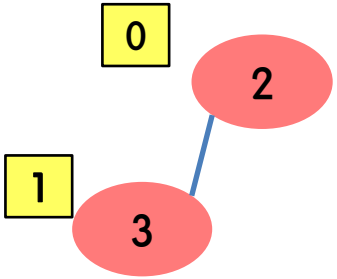
문제분석

0	1	2	3	4	5
2	3	1	5	6	4

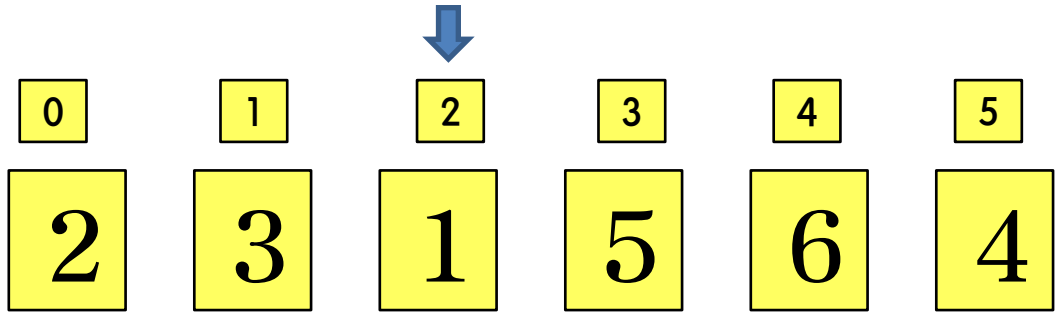
MIN HEAP



K = 2



문제분석



MIN HEAP

K=2

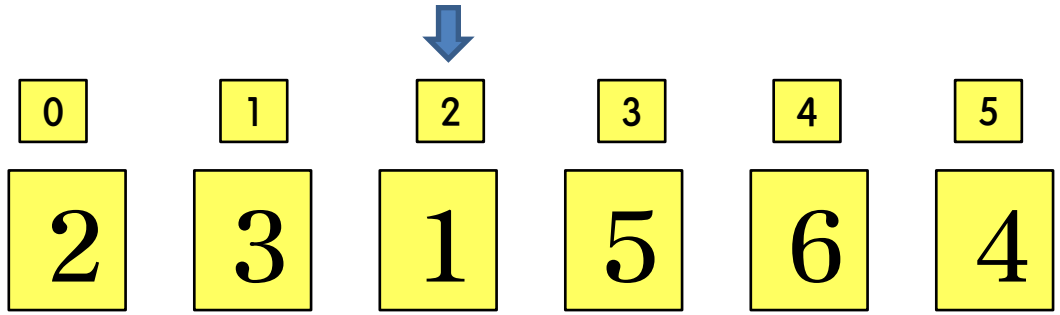
1

2

3

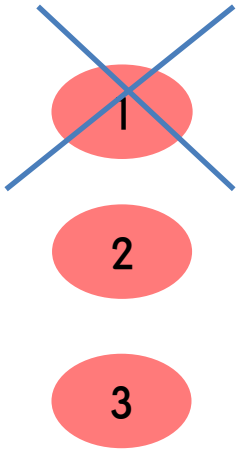
1. 1이 들어오는 순간 pq 구조, pq 사이즈==3
2. 최상단을 제거하면 두개 존재

문제분석



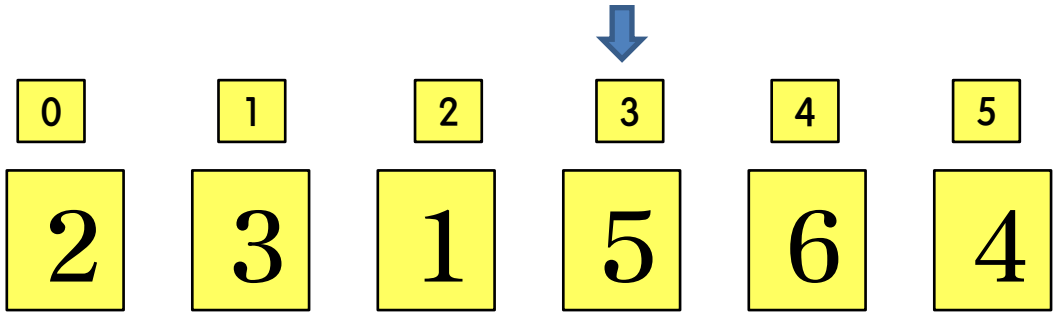
MIN HEAP

K=2



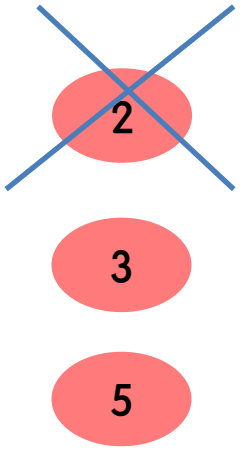
1. 1이 들어오는 순간 pq 구조, pq 사이즈==3
2. 최상단을 제거하면 두개 존재

문제분석



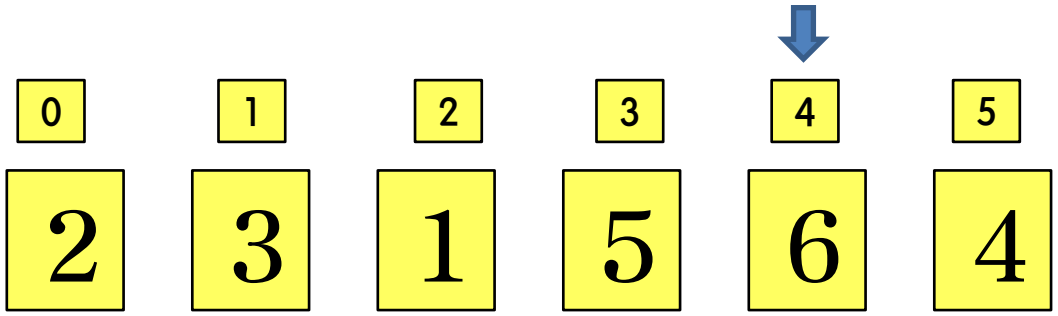
MIN HEAP

K=2



1. 들어오는 순간 pq 구조, pq 사이즈==3
2. 최상단을 제거하면 두개 존재

문제분석



MIN HEAP

K=2

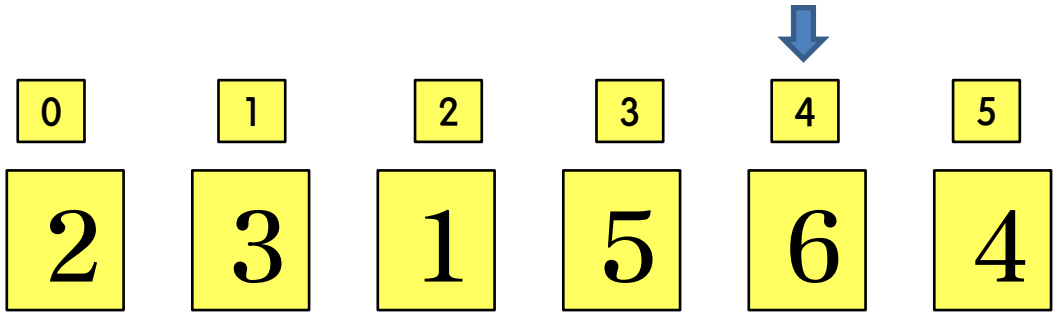
3

5

6

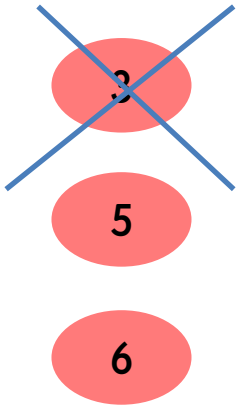
1. 들어오는 순간 pq 구조, pq 사이즈==3
2. 최상단을 제거하면 두개 존재

문제분석



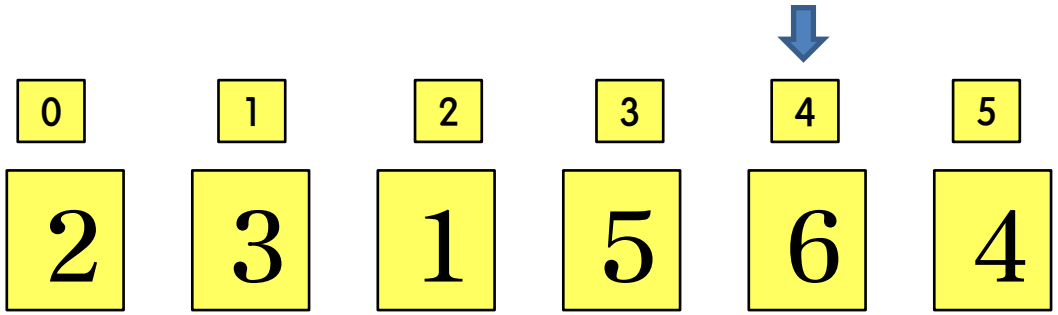
MIN HEAP

K=2



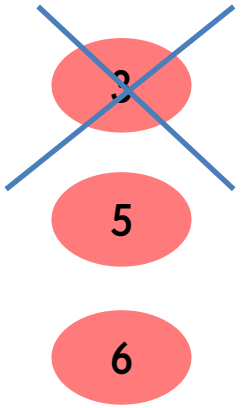
1. 들어오는 순간 pq 구조, pq 사이즈==3
2. 최상단을 제거하면 두개 존재

문제분석



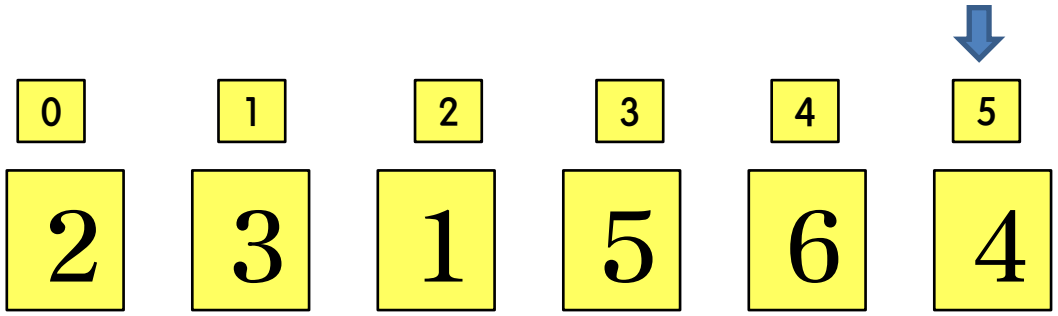
MIN HEAP

K=2



1. 들어오는 순간 pq 구조, pq 사이즈==3
2. 최상단을 제거하면 두개 존재

문제분석



MIN HEAP

K=2

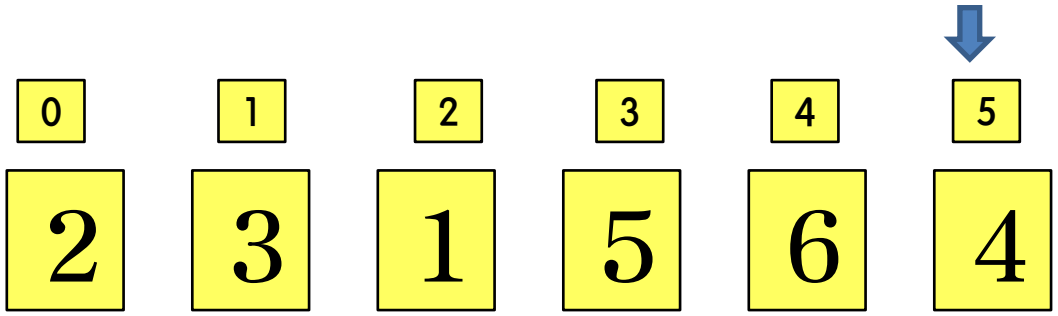
4

5

6

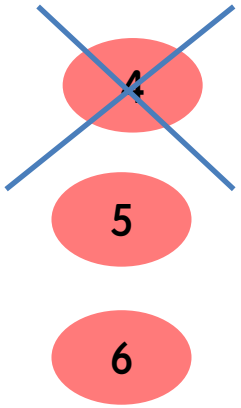
1. 들어오는 순간 pq 구조, pq 사이즈==3
2. 최상단을 제거하면 두개 존재

문제분석



MIN HEAP

K=2



1. 들어오는 순간 pq 구조, pq 사이즈==3
2. 최상단을 제거하면 두개 존재

시간복잡도/공간복잡도 계산

시간복잡도

1. 대상(Source) : 문제에서 입력받은 파라미터(array 등) (속도)
   Time Complexity : O(Nlogk)
   대상 : int[] nums, int k
   이유 : 배열의 개수만큼 n개 * 우선순위큐 사용 logk
   PriorityQueue(우선순위큐) k개 만큼 실행

공간복잡도

2. 대상(Source) : 실제 사용되는 저장 공간을 계산(메모리 사용량)
   예) 프로그램을 실행 및 완료하는데 필요한 저장공간
   Space Complexity : O(k)
   대상 :
   이유 : PriorityQueue(우선순위큐) minHeap 최상단 k개 만큼만 교체
   offer(), poll()반복작업으로 항상k개만큼 저장

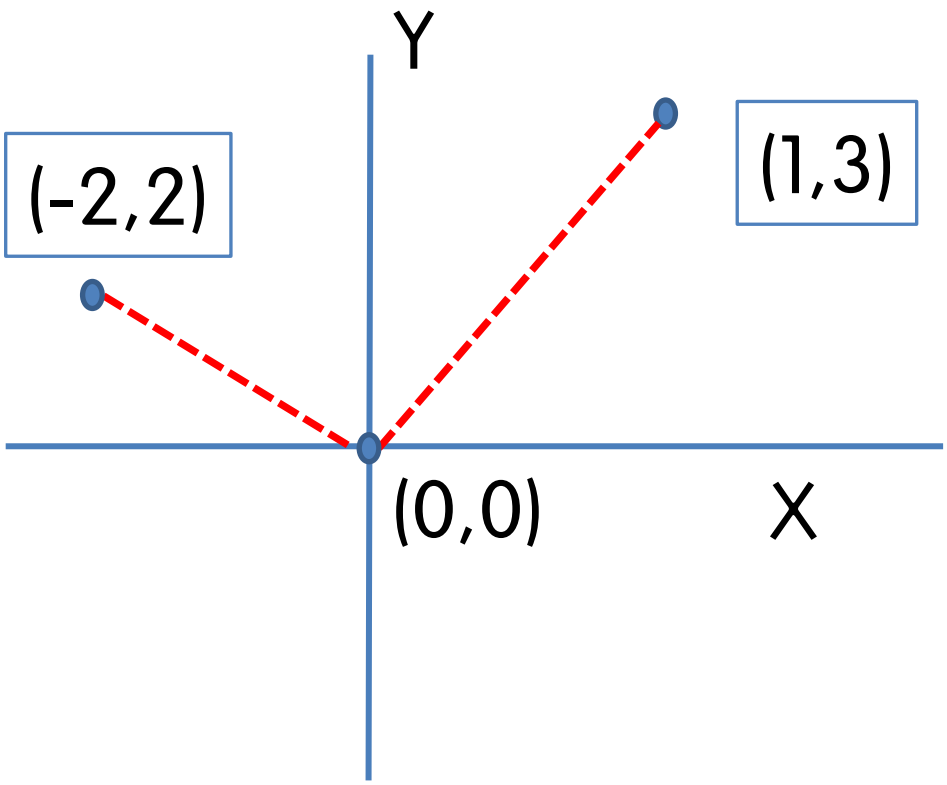
참고

O(1) : 스택, 큐,Map
O(n) : for문 => 데이터를 한번씩 다 호출하니까 (제일 많음)
O(log N) : sort, prirotiyQueue, binary Search Tree, Tree
O(Klog N) : k번만큼 소팅하는 경우
O(n^2) : 이중for문
O(m*n) : 이중for문인데, n이 다른경우bfs,dfs 류 (예 n=100 인데 m=5인경우)

원점에 가장 가까운 지점 (K Closest Points to Origin)

설명

XY 평면의 한 점 과 정수를 나타내는 배열이 주어지면 원점에 가장 가까운 점을 반환합니다.
XY 평면 에서 두 점 사이의 거리 구하는 공식을 이용하세요.$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
원점에서 제일 가까운 좌표를 K개의 갯수만큼 리턴하세요



입출력

Input: points = [[1,3],[-2,2]], k = 1
Output: [[-2,2]]
Explanation: 원점에서 좌표(1, 3)은 두 점사이 거리구하는 공식을 이용하면 10.

원점에서 좌표(-2, 2)은 두점사이 거리구하는 공식을 이용하면 8.

k = 1 일때는 제일 가까운 좌표 1개만 리턴. [[-2,2]] 리턴

원점에 가장 가까운 지점 (K Closest Points to Origin)

Input: points = [[3,3],[5,-1],[-2,4]], k = 2
Output: [[3,3],[-2,4]]

문제 Format

```
class Solution {
  public int[][] solve(int[][] points, int k) { }
}
```

제한사항

$1 <= k <= points.length <= 10^4$
$-10^4 < x_i, y_i < 10^4$

Solution

1. 문제분석

- 문제를 정확히 이해
- 분석 내용 정리(Devide & Conquer)

2. 규칙찾기

- 분석 내용으로 규칙을 찾는다

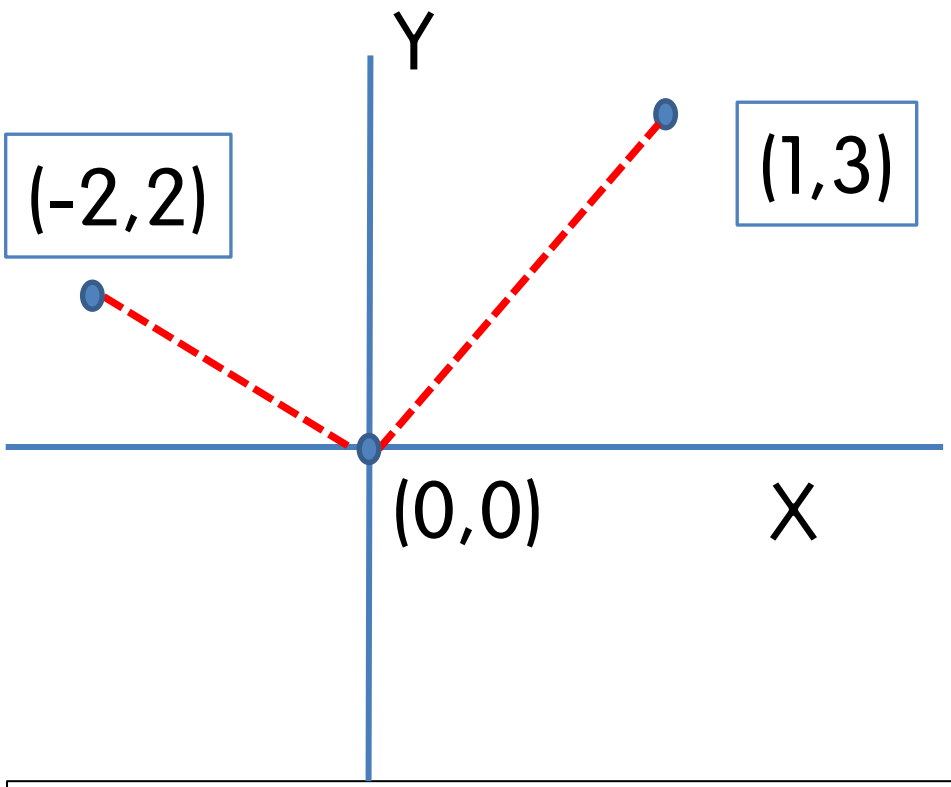
3. 코딩화

- 분석 내용으로 알맞은 구현방법 찾기

4. 알고리즘 적용

- 알고리즘 정하고 담을 그릇 정한다
(사전지식 필요)

문제분석



1. 원점으로 부터의 거리를 구한다.

$$(X2-X1)^2+(Y2-Y1)^2$$

2. 원점에서 제일 작은 거리에 있는 값을 구한다
4. 제일 작은값부터 저장(PriorityQueue 이용한다)

코딩화

1. PriorityQueue에 원점 거리 작은거 부터 저장(MinHeap)

```
Queue< int[] > pq = new PriorityQueue<>((a,b)->
                    (a[0]*a[0]+a[1]*a[1])-(b[0]*b[0]+b[1]*b[1]));
```

points = [[1,3],[-2,2]],

$$(X2-X1)^2+(Y2-Y1)^2$$

2. K개만큼 큐에서 poll해서
   int[][] 타입으로 전달

시간복잡도/공간복잡도 계산

시간복잡도

1. 대상(Source) : 문제에서 입력받은 파라미터(array 등) (속도)
   Time Complexity : O(NlogN)
   대상 : int[][] points
   이유 : 배열의 개수만큼 n개 * 우선순위큐 사용 logN

공간복잡도

2. 대상(Source) : 실제 사용되는 저장 공간을 계산(메모리 사용량)
   예) 프로그램을 실행 및 완료하는데 필요한 저장공간
   Space Complexity : O(k)
   대상 :
   이유 : PriorityQueue(우선순위큐) minHeap 최상단 k만큼 교체

참고

O(1) : 스택, 큐,Map
O(n) : for문 => 데이터를 한번씩 다 호출하니까 (제일 많음)
O(log N) : sort, prirotiyQueue, binary Search Tree, Tree
O(Klog N) : k번만큼 소팅하는 경우
O(n^2) : 이중for문
O(m*n) : 이중for문인데, n이 다른경우bfs,dfs 류 (예 n=100 인데 m=5인경우)

Solution

1. 문제분석

- 문제를 정확히 이해
- 분석 내용 정리(Devide & Conquer)

2. 규칙찾기

- 분석 내용으로 규칙을 찾는다

3. 코딩화

- 분석 내용으로 알맞은 구현방법 찾기

4. 알고리즘 적용

- 알고리즘 정하고 담을 그릇 정한다
(사전지식 필요)

미팅룸(Meeting Room)

설명

미팅 시간 배열이 주어집니다. intervals[i] = [$start_i$, end_i]
Intervals배열을 이용하여 사람들이 모든 회의에 참석할 수 있는지 boolean으로 리턴하세요

입출력

Input: intervals =
[[5,10],[16,20], [0,30]]
Output: false

Input: intervals = [[6,10],[1,3]]
Output: true

문제 Format

```
class Solution {
 public boolean solve(int[][] intervals) { }
}
```

제한사항

0 <= intervals.length <= 10^4
intervals[i].length == 2
0 <= $start_i$ < end_i <= 10^6

Solution

1. 문제분석

- 문제를 정확히 이해
- 분석 내용 정리(Devide & Conquer)

2. 규칙찾기

- 분석 내용으로 규칙을 찾는다

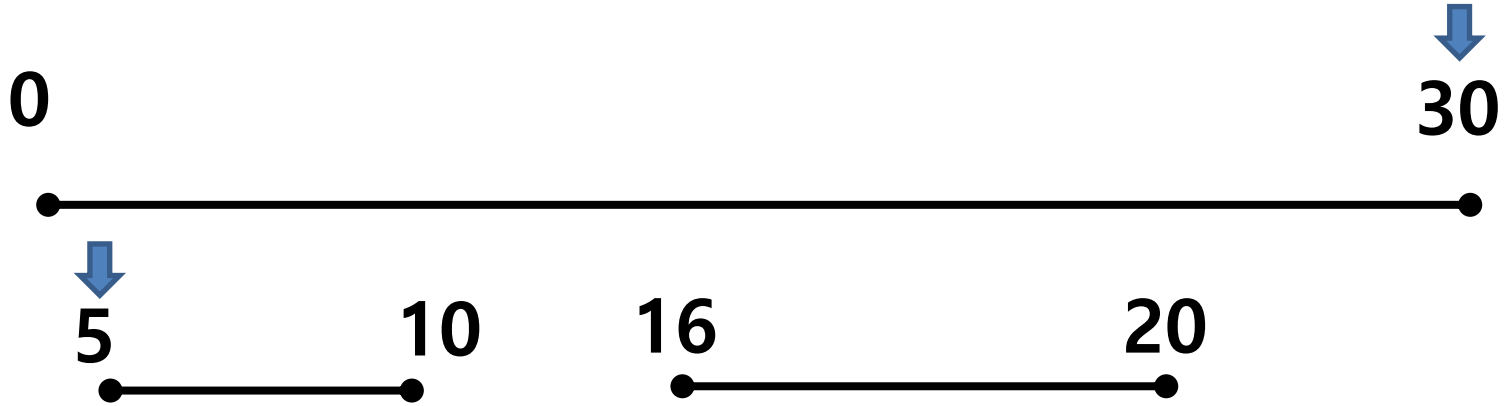
3. 코딩화

- 분석 내용으로 알맞은 구현방법 찾기

4. 알고리즘 적용

- 알고리즘 정하고 담을 그릇 정한다
(사전지식 필요)

문제분석



1. Start 시간으로 소팅
2. 전미팅.end > 현재미팅.start 인 경우 회의실 필요

코딩화

1. 자바 Arrays 소팅
   Arrays.sort(intervals, (a, b) -> a[0] - b[0]);

2. 전미팅.end > 현재미팅.start
   이차원배열 사용.
   첫번째 배열을 빼서, 두번째부터 비교
   end = intervals[0][1]

시간복잡도/공간복잡도 계산

시간복잡도

1. 대상(Source) : 문제에서 입력받은 파라미터(array 등) (속도)
   Time Complexity : O(NlogN)
   대상 : int[][] intervals
   이유 : 소팅 사용 logN , for문 한번 실행!

공간복잡도

2. 대상(Source) : 실제 사용되는 저장 공간을 계산(메모리 사용량)
   예) 프로그램을 실행 및 완료하는데 필요한 저장공간
   Space Complexity : O(1)
   대상 :
   이유 : 추가적인 공간을 사용안함.

참고

O(1) : 스택, 큐,Map
O(n) : for문 => 데이터를 한번씩 다 호출하니까 (제일 많음)
O(log N) : sort, prirotiyQueue, binary Search Tree, Tree
O(Klog N) : k번만큼 소팅하는 경우
O(n^2) : 이중for문
O(m*n) : 이중for문인데, n이 다른경우bfs,dfs 류 (예 n=100 인데 m=5인경우)

미팅룸2(Meeting Room2)

설명

미팅 시간 배열이 주어집니다. intervals[i] = [$start_i$, end_i]
Intervals배열을 이용하여 사람들이 회의에 참석할려면 몇 개의 회의실이 필요한지 리턴하세요

입출력

Input: intervals =
[[5,10],[16,20], [0,30]]
Output: 2

Input: intervals = [[6,10],[1,3]]
Output: 1

문제 Format

```
class Solution {
 public int solve(int[][] intervals) { }
}
```

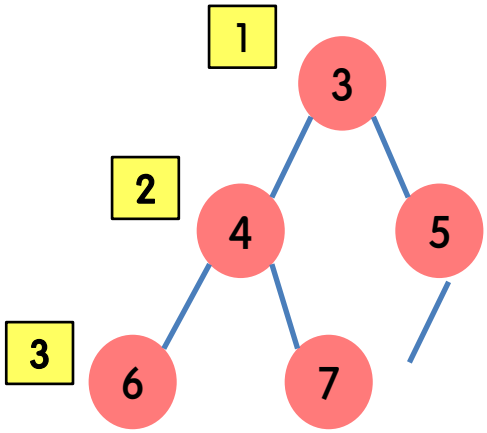
제한사항

0 <= intervals.length <= 10^4
intervals[i].length == 2
0 <= $start_i$ < end_i <= 10^6

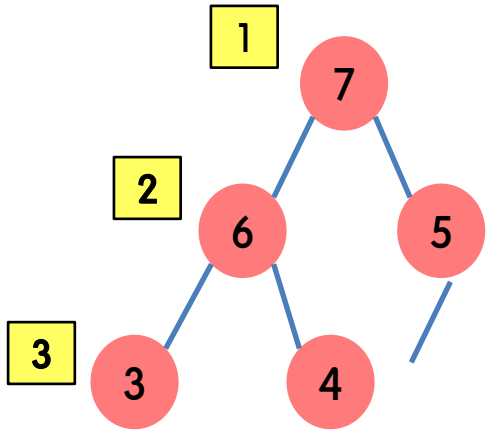
Sort 개념 설명(시험에 나오는거 위주로)

PriorityQueue 시간 복잡도(O(logN)) = 8번 이동해야되는데 3번으로 단축

완전 이진 트리(complete binary tree)

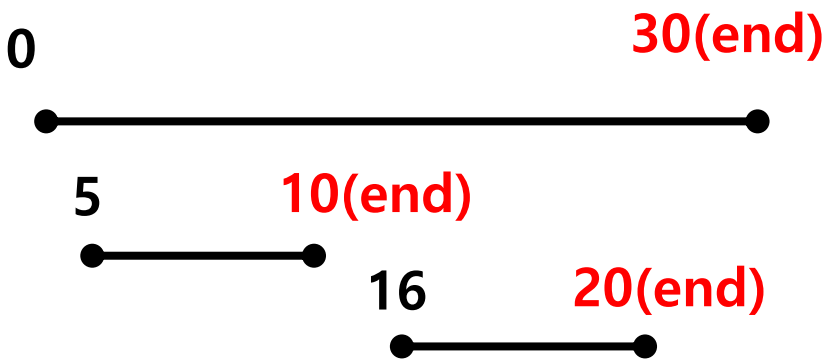


MIN HEAP



MAX HEAP

문제분석

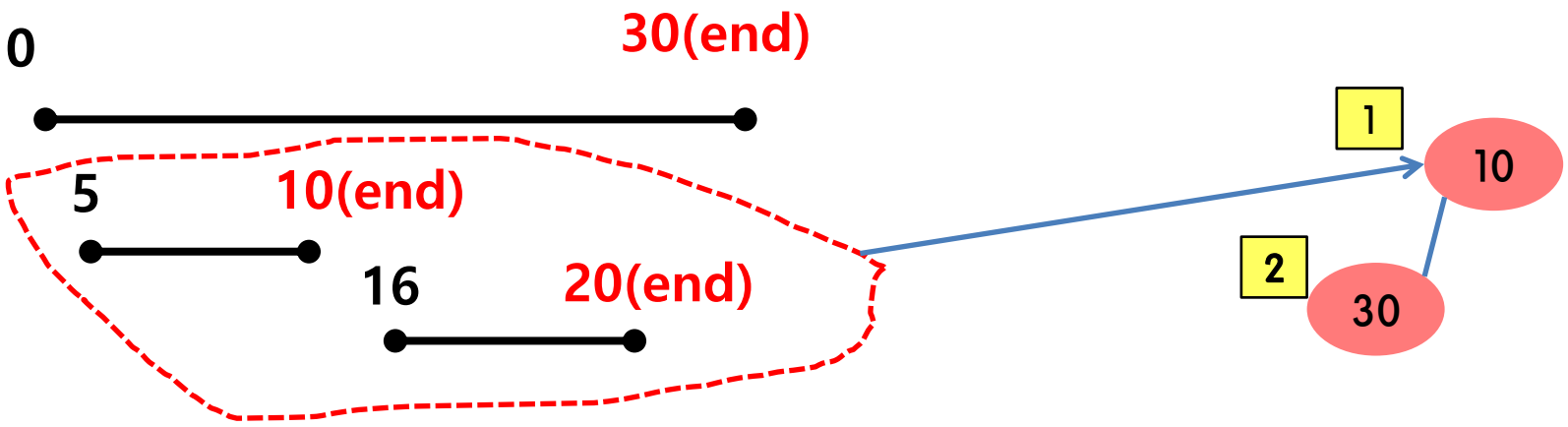


1. 아이디어
회의끝시간이 제일 긴것을 관리한다.
앞.end 뒷.start 시간을 비교

회의실 추가가 필요 없는 것은
하나로 합치고 , 추가가 필요한것은 큐에 넣는다

문제분석

MIN HEAP



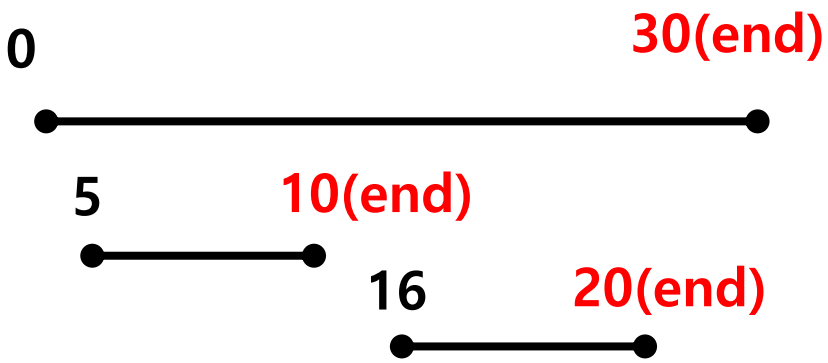
1. 아이디어

우선순위 큐를 이용해서, 회의끝시간이 제일 긴것을 관리한다.
앞.end 뒷.start 시간을 비교해서 회의실 추가가 필요 없는 것은
하나로 합치고 , 추가가 필요한것은 큐에 넣는다

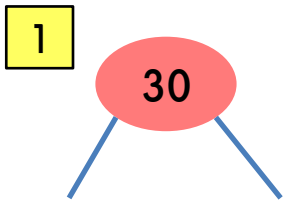
2. minHeap을 만든다

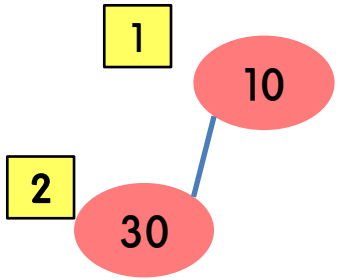
```
Queue<int[]> pq=new PriorityQueue<>((a,b)->a[1]-b[1]);
```

문제분석



MIN HEAP

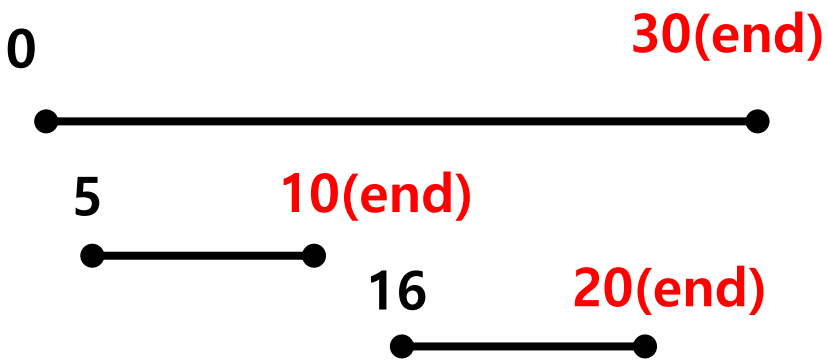




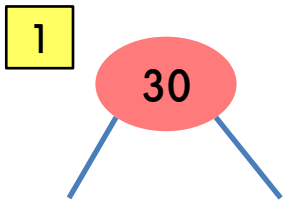
1. 30 >5 앞.end > 뒷.start => 회의실 필요
   : 큐에 추가

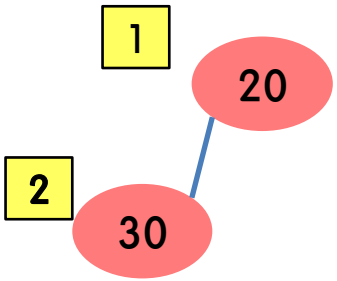
2. 10 <=16 앞.end <= 뒷.start => 회의실 필요 없음
   큐에 있는걸 빼서(poll) 합친 후, 큐에 추가

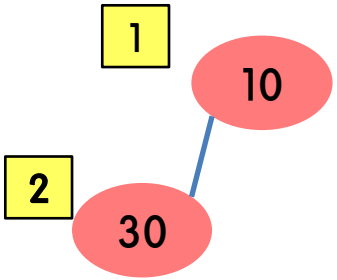
문제분석



MIN HEAP



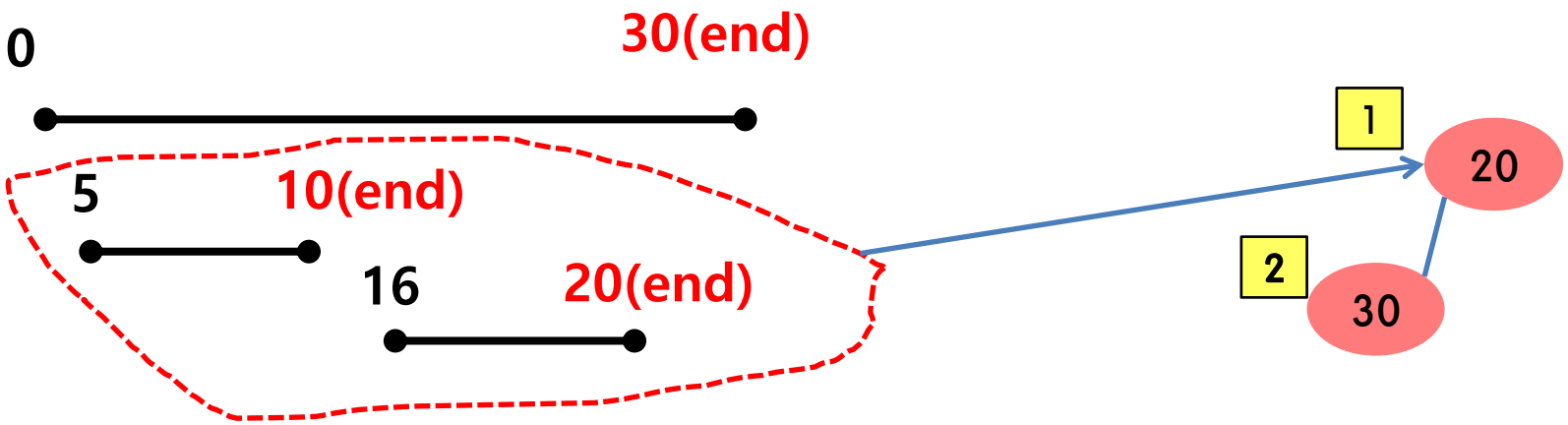




1. 30 >5 앞.end > 뒷.start => 회의실 필요
2. 10 <=16 앞.end <= 뒷.start => 회의실 필요 없음

문제분석

MIN HEAP



1. 결론 : [5,10], [16,20] 은 하나로 관리

시간복잡도/공간복잡도 계산

시간복잡도

1. 대상(Source) : 문제에서 입력받은 파라미터(array 등) (속도)
   Time Complexity : O(NlogN)
   대상 : int[][] intervals
   이유 : 소팅 사용 logN , N개의 요소들이 for문 한번 실행

공간복잡도

2. 대상(Source) : 실제 사용되는 저장 공간을 계산(메모리 사용량)
   예) 프로그램을 실행 및 완료하는데 필요한 저장공간
   Space Complexity : O(N)
   대상 : Queue<int[]> pq=new PriorityQueue<>((a,b)->a[1]-b[1]);
   이유 : 각 배열의 개수만큼pq에 추가 삭제가 일어나므로

참고

O(1) : 스택, 큐,Map
O(n) : for문 => 데이터를 한번씩 다 호출하니까 (제일 많음)
O(log N) : sort, prirotiyQueue, binary Search Tree, Tree
O(Klog N) : k번만큼 소팅하는 경우
O(n^2) : 이중for문
O(m*n) : 이중for문인데, n이 다른경우bfs,dfs 류 (예 n=100 인데 m=5인경우)

interval 병합(Merge Interval)

설명

intervals 배열이 주어지면 겹치는 구간을 병합하여 배열을 반환 합니다 .

입출력

Input: intervals = [[1,4],[2,6],[8,10],[15,18]]
Output: [[1,6],[8,10],[15,18]]
Explanation: [1,4] [2,6] 오버랩되므로, [1,6]으로 병합시킵니다.

Input: intervals = [[1,5],[5,6]]
Output: [[1,6]]

문제 Format

```
class Solution {
 public int[][] solve(int[][] intervals) {}
}
```

제한사항

$1 <= intervals.length <= 10^4$
$intervals[i].length == 2$
$0 <= start_i <= end_i <= 10^4$

Solution

1. 문제분석

- 문제를 정확히 이해
- 분석 내용 정리(Devide & Conquer)

2. 규칙찾기

- 분석 내용으로 규칙을 찾는다

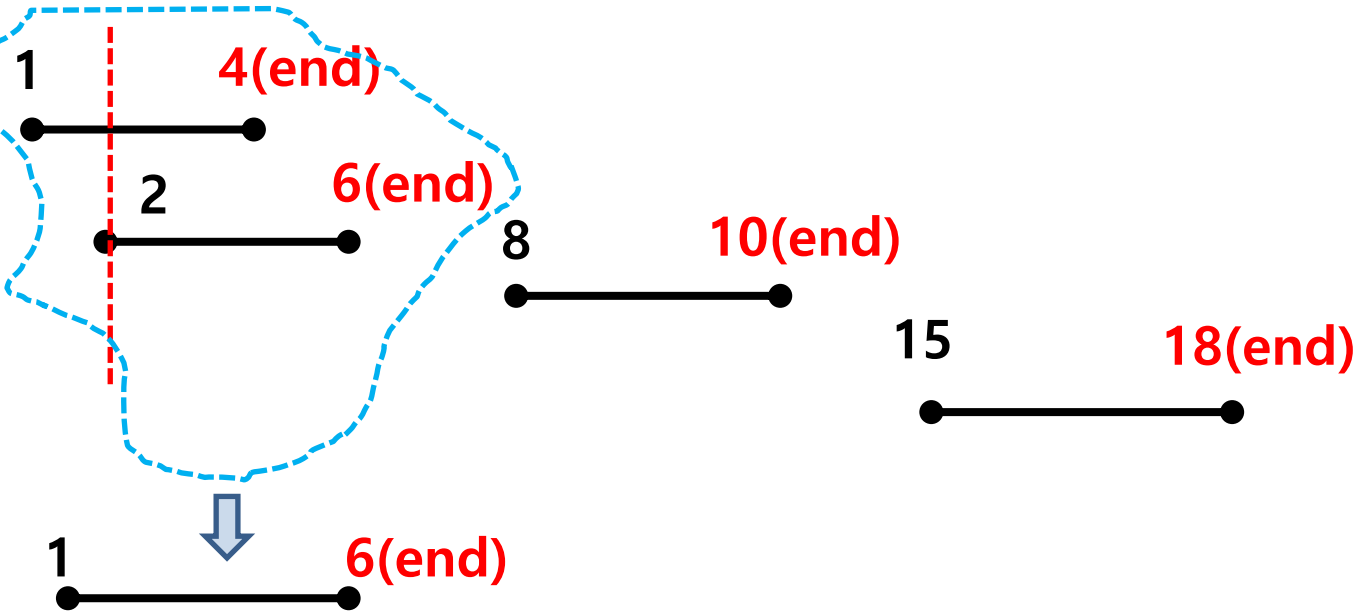
3. 코딩화

- 분석 내용으로 알맞은 구현방법 찾기

4. 알고리즘 적용

- 알고리즘 정하고 담을 그릇 정한다
(사전지식 필요)

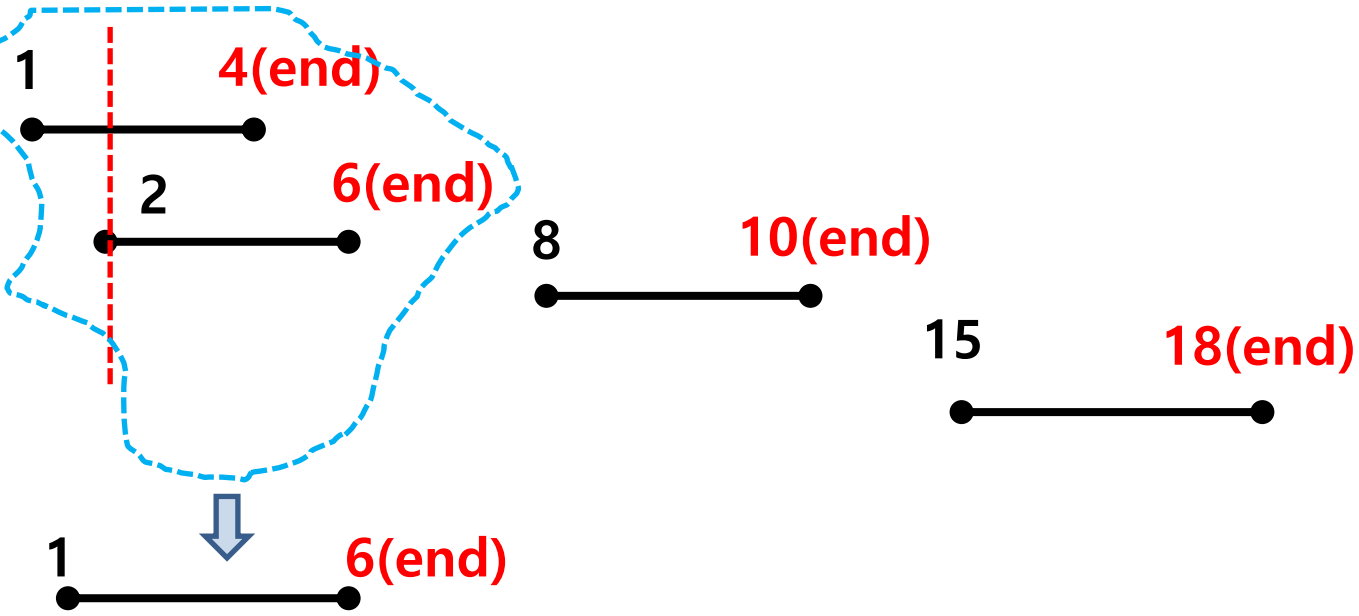
문제분석



1. 아이디어
겹치는 부분을 합친다.
(1,4)에서 4와 (2,6)에서 2를 비교해서 합친다.
(1,6)을 결과리스트에 저장한다.

(1,6) (8,10)을 비교 (8,10)을 결과리스트에 저장한다.
(전.end 6과 < 현.start 8을 비교 도출)

문제분석



00	1	01	4
10	2	11	6
20	8	21	10
30	15	31	18

1. 4 >=2 이면, 1-6으로 만든다
   전.end > 현.start
   end = Math.max(전.end , 현.end)
   새로운 1,6을 만든다
   start=1, end=6을 넣어서 다음 (8,10) 비교

2. 다시 전.end > 현.start 비교해서, 6 >= 8이 아니므로
   (8,10)을 결과값에 저장한다 .
   start=8, end=10을 넣어서 다음 (15,18) 비교

개념) 이차원배열 List로 변환

2DArray

1 `int[ ][ ] grid = new int[3][4]`

	0 열	1 열	2 열	3 열
0 행	0,0	0,1	0,2	0,3
1 행	1,0	1,1	1,2	1,3
2 행	2,0	2,1	2,2	2,3

시간복잡도/공간복잡도 계산

시간복잡도

1. 대상(Source) : 문제에서 입력받은 파라미터(array 등) (속도)
   Time Complexity : O(NlogN)
   대상 : int[][] intervals
   이유 : 소팅 사용 logN , N개의 요소들이 for문 한번 실행

공간복잡도

2. 대상(Source) : 실제 사용되는 저장 공간을 계산(메모리 사용량)
   예) 프로그램을 실행 및 완료하는데 필요한 저장공간
   Space Complexity : O(logN)
   대상 : int[][] intervals
   이유 : 소팅만 함, 다른 스페이스 안씀

참고

O(1) : 스택, 큐,Map
O(n) : for문 => 데이터를 한번씩 다 호출하니까 (제일 많음)
O(log N) : sort, prirotiyQueue, binary Search Tree, Tree
O(Klog N) : k번만큼 소팅하는 경우
O(n^2) : 이중for문
O(m*n) : 이중for문인데, n이 다른경우bfs,dfs 류 (예 n=100 인데 m=5인경우)

로그 파일의 데이터 재정렬

설명

배열로 logs 주어집니다.
각 로그는 공백으로 구분 된 단어 문자열이며 첫 번째 단어는 **식별자** 입니다.

There are two types of logs:
Letter-logs: All words (except the identifier) consist of lowercase English letters.
Digit-logs: All words (except the identifier) consist of digits.

Reorder these logs so that:
The **letter-logs** come before all **digit-logs**.
The **letter-logs** are sorted lexicographically by their contents.
If their contents are the same, then sort them lexicographically by their identifiers.
The **digit-logs** maintain their relative ordering.
Return the final order of the logs.

문제 Format

```
class Solution {
    public String[] solve (String[] logs) {
    }
}
```

3. 로그 파일의 데이터 재정렬

설명

Input: logs = ["dig1 8 2 3 1",
"let1 abc cat",
"dig1 2 5",
"let2 good dog book",
"let3 abc zoo"]
Output: ["let1 abc cat",
"let3 abc zoo",
"let2 good dog book ",
"dig1 8 2 3 1",
"dig1 2 5"]
Explanation:
The letter-log contents are all different, so their ordering is "abc cat", "abc zoo", "good dog book".
The digit-logs have a relative order of "dig1 8 2 3 1", "dig1 2 5".

제한사항

1 <= logs.length <= 100, 3 <= logs[i].length <= 100
All the tokens of logs[i] are separated by a single space.
logs[i] is guaranteed to have an identifier and at least one word after the identifier.

문제분석

1. Idenfier와 나누기

```
"dig1 8 2 3 1",
"let1 abc cat",
"dig1 2 5",
"let2 good dog book",
"let3 abc zoo"
```

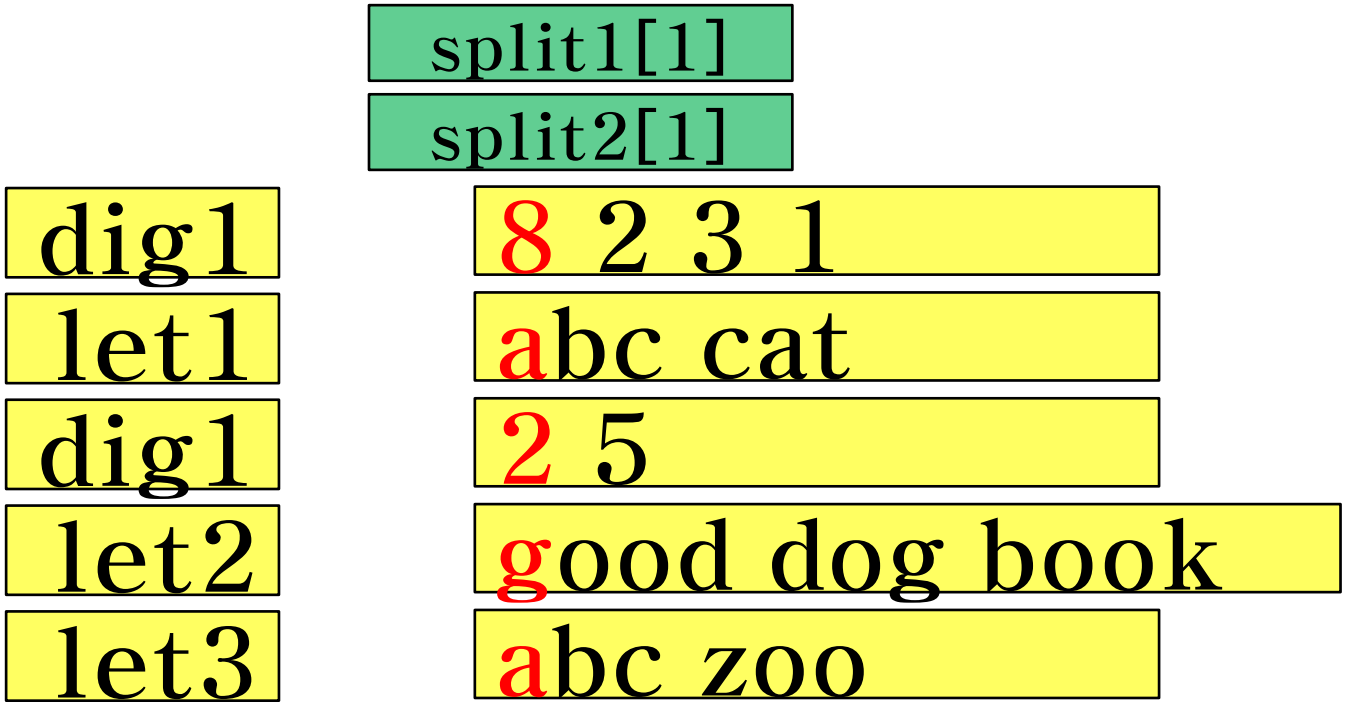
```
1.  String[ ] split1 = s1.split(" ", 2);
2.  String[ ] split2 = s2.split(" ", 2);
```

split1[0]	split1[1]
split2[0]	split2[1]
dig1	8 2 3 1
let1	abc cat
dig1	2 5
let2	good dog book
let3	abc zoo

문제분석

2. 두번째 문자열판단

1. Char 값이 숫자인지 문자인 판단
   isDigit() => 숫자인지 판단
   isLetter() => 문자인지 판단
2. Character.isDigit(split1[1].charAt(0))
   Character.isDigit(split2[1].charAt(0))



문제분석

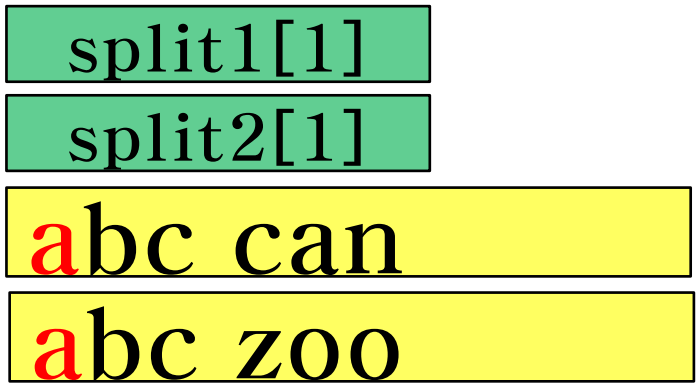
3. Comparator 적용하기

dig1	8 2 3 1
let1	abc cat
dig1	2 5
let2	good dog book
let3	abc zoo

```
if(!isDigit1 && !isDigit2) {
    // 1. 모두 문자
 } else if (isDigit1 && isDigit2) {
    // 2. 모두 숫자.
    return 0;
 } else if (isDigit1 && !isDigit2) {
    // 3. 첫번째는 숫자, 두번째는 문자.
    return 1;
 } else {
    // 4. 첫번째는 문자, 두번째는 숫자.
    return -1;
 }
```

문제분석

4. Comparator 모두 문자인 경우



1. split1[1] 과 split2[1] 오름차순으로 비교한다.
2. 만약같다면 (0인경우) **split1[0].compareTo(split2[0]); 적용한다. 오름차순이다**

```
if(!isDigit1 && !isDigit2) {
    // 모두 문자
    int comp = split1[1].compareTo (split2[1]); // 오름차순 마-1
    if (comp == 0) return split1[0].compareTo(split2[0]);
    else return comp;
}
```

문제분석

5. Comparator 나머지 비교

abc cat

abc zoo

good dog book

8 2 3 1

2 5

```
} else if (isDigit1 && isDigit2) {
    // 2. 모두 숫자
    return 0;
 } else if (isDigit1 && !isDigit2) {
    // 3. 첫번째는 숫자, 두번째는 문자
    return 1;
 } else {
    // 4. 첫번째는 문자, 두번째는 숫자.
    return -1;
 }
```

Sort 개념 설명(시험에 나오는거 위주로)

```
1.  public int compareTo() {
        return A.compareTo(B);
    }
```

1. A와 B가 같으면 0
2. A> B 이면 1
3. A< B 이면 -1 오름차순 (ex 1 2 3 이런식이면 2-3은 -1 음수) 오름마

1. Collections.sort(intervals,(a,b) -> a.start-b.start);

2. Collections.sort(intervals, comp2);

```
Comparator<Interval> comp2 = new Comparator<Interval>() {
@Override
public int compare(Interval o1, Interval o2) {
        if (o1.start > o2.start) {
            return 1;
        } else if (o1.start < o2.start) {
            return -1;
        } else {
            return 0;
        }
        };
```

3. Comparator comp = new Comparator<Interval>() {

```
    public int compare(Interval a, Interval b) {

        return a.start - b.start;

}};
```

2. Comparator

```
Comparator<Interval> Comp2 = new Comparator<Interval>() {
    @Override
        public int compare(Interval o1, Interval o2) {
          // TODO Auto-generated method stub
          return o1.end - o2.end;
          }
};

Comparator<Interval> comp = new Comparator<Interval>() {

    @Override
    public int compare(Interval o1, Interval o2) {
        if(o1.end >o2.end) {
            return 1;
        }else if(o1.end <o2.end) {
              return -1;
        }else {
              return 0;
           }
      }
```

시간복잡도/공간복잡도 계산

시간복잡도

1. 대상(Source) : 문제에서 입력받은 파라미터(array 등) (속도)
   Time Complexity : O(M*N*log N)
   대상 : String[] logs.
   이유 : 소팅하는 부분 O(N* log(N))
   logs 의 length() M개만큼 있음 O(M)
   그래서 O(M*N*log N)

공간복잡도

2. 대상(Source) : 실제 사용되는 저장 공간을 계산(메모리 사용량)
   예) 프로그램을 실행 및 완료하는데 필요한 저장공간
   Space Complexity : O(M*log N)
   logs 길이가 길다. M개만큼 저장공간 필요 O(M)
   소팅하는 부분 O(log(N))

참고

O(1) : 스택, 큐,Map
O(n) : for문 => 데이터를 한번씩 다 호출하니까 (제일 많음)
O(log N) : sort, prirotiyQueue, binary Search Tree, Tree
O(Klog N) : k번만큼 소팅하는 경우
O(n^2) : 이중for문
O(m*n) : 이중for문인데, n이 다른경우bfs,dfs 류 (예 n=100 인데 m=5인경우)