Sort 개념 설명(시험에 나오는거 위주로)

1. 개념

1) CompareTo()
2) Comparator
3) PriorityQueue 시간 복잡도(O(logn))
4) Binary Search 시간 복잡도(O(logn))
5) Quick Sort

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)

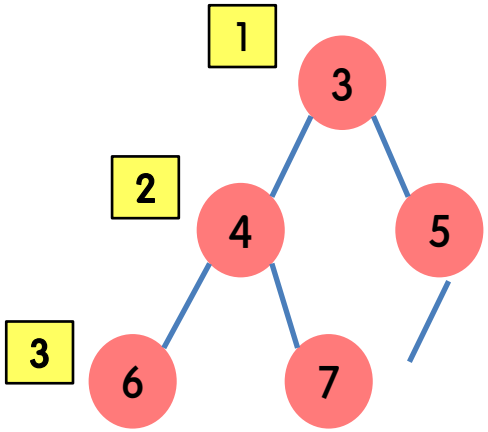
Sort 개념 설명(시험에 나오는거 위주로)

2. Comparator

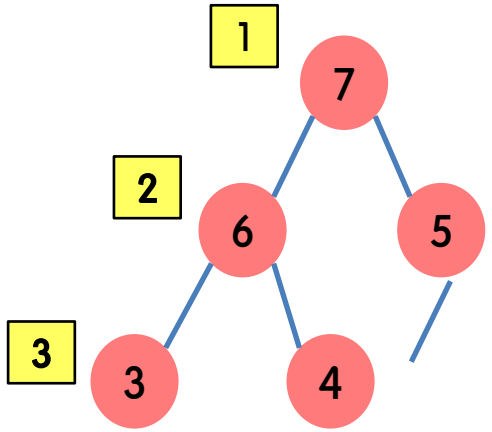
```
Comparator<Interval> Comp2 = new Comparator<Interval>() {
    @Override
            public int compare(Interval o1, Interval o2) {
              // TODO Auto-generated method stub
              return o1.end - o2.end;
              }
};


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;
             }
      }
};
```

Sort 개념 설명(시험에 나오는거 위주로)

3. PriorityQueue 시간 복잡도(O(logn)), 완전 이진 트리(complete binary tree)



MIN HEAP



MAX HEAP

1. Sort (compareTo())

Problem

- 버전 번호를 version1하고 version2, 비교한다.

- 다음을 반환합니다.
- 이면 version1 < version2 반환 -1합니다.
- 이면 version1 > version2 반환 1합니다.
- 그렇지 않으면을 0반환.

Example

Input: String version1 = "8.5.2.4",
version2 = "8.5.3";

Output: -1

Note

Integer a =1, b=3;
// 오름차순 -1, 오른쪽 큰값
a.compareTo(b);//-1

2. Solution

CompareTo()

String version1 = "8.5.2.4",
version2 = "8.5.3";

8	5	2	4
8	5	3	

1	0	1
1	0	0

0	0	-1

0	0	1

생각
1. 앞자리를 비교한다.
2. .을 처리한다=> split
3. 2,3을 비교하면 결정됨

1. Sort (compareTo())

Problem

- 버전 번호를 version1하고 version2, 비교한다.

- 다음을 반환합니다.
- 이면 version1 < version2 반환 -1합니다.
- 이면 version1 > version2 반환 1합니다.
- 그렇지 않으면을 0반환.

Example

Input: String version1 = "8.5.2.4",
version2 = "8.5.3";

Output: -1

Note

Integer a =1, b=3;
// 오름차순 -1, 오른쪽 큰값
a.compareTo(b);//-1

2. Solution

CompareTo()

String version1 = "8.5.2.4",
version2 = "8.5.3";

8	5	2	4
8	5	3	

1	0	1
1	0	0

0	0	-1

0	0	1

생각
1. 앞자리를 비교한다.
2. .을 처리한다=> split
3. 2,3을 비교하면 결정됨

2. Comparator

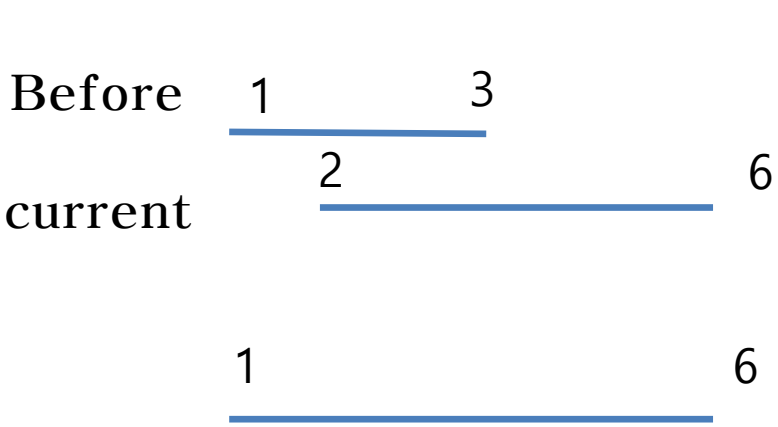
Comparator

Problem

-

Example

Input: [[1,3],[2,6],[8,10],[15,18]]
Output: [[1,6],[8,10],[15,18]]



before.end >= cur.start

before.end = Math.max
(curr.end, before.end);

Solution

Map, Array

1. 문제를 정확히 이해

2. 알고리즘 정하고 담을 그릇 정한다

3. for 문 돌리기

생각->프로그램(한국말로 생각하고->Java)
결과를 해석하여 이미지화시킨다

1. 클래스(Interval)
2. 리스트화(자료)
3. Comparator(버젼)
4. 현재Start< 전 end , 새롭게 1,6짜리 클래스를 만든다.

Solution

Map, Array

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;
```

3. PriorityQueue

PriorityQueue (logn)

Problem

양의 정수 길이의 두 막대기 연결할 수있다.
x와 y의 비용을 지불한다 스틱 x + y =>
이런식으로 연결하여 스틱이 하나만 남을 때까지 모든 스틱을 연결 최소 비용을 반환 합니다 .

Example

입력 : 스틱 = [1,8,3,5]
출력 : 30
설명 :
1. 1 + 3 = 4의 비용, = [4,8,5]가됩니다.
2. 4 + 5 = 9의 비용, = [9,8]이됩니다.
3. 9 + 8 = 17의 비용, = [17]이됩니다.
스틱이 하나만 남아 있으므로 완료
총 비용은 4 + 9 + 17 = 30입니다

Solution

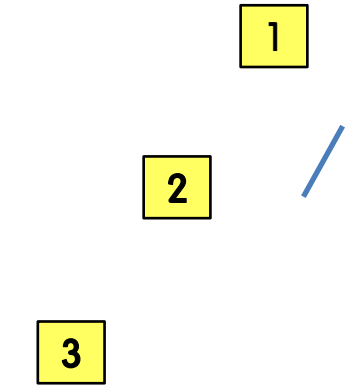
PriorityQueue

1. 문제를 정확히 이해

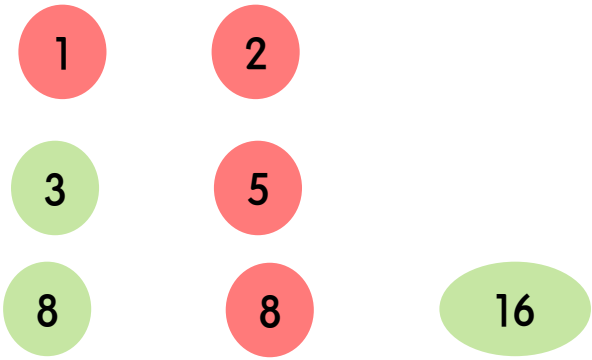
2. 알고리즘 정하고 담을 그릇 정한다

3. for 문 돌리기

생각->프로그램(한국말로 생각하고->Java)
결과를 해석하여 이미지화시킨다

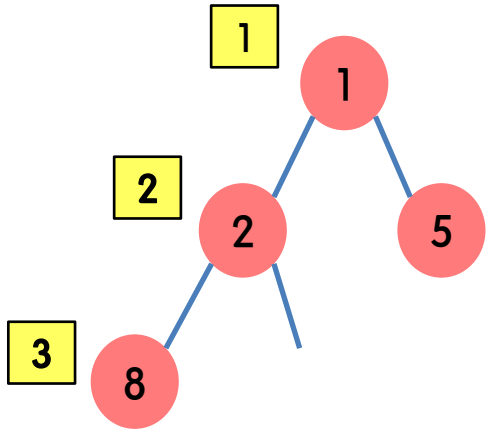


1. PriorityQueue 생성 pq.offer
2. pq.poll



Solution

Comparator



1. PriorityQueue 생성 pq.offer (우선순위가 낮은 순)
2. pq.poll