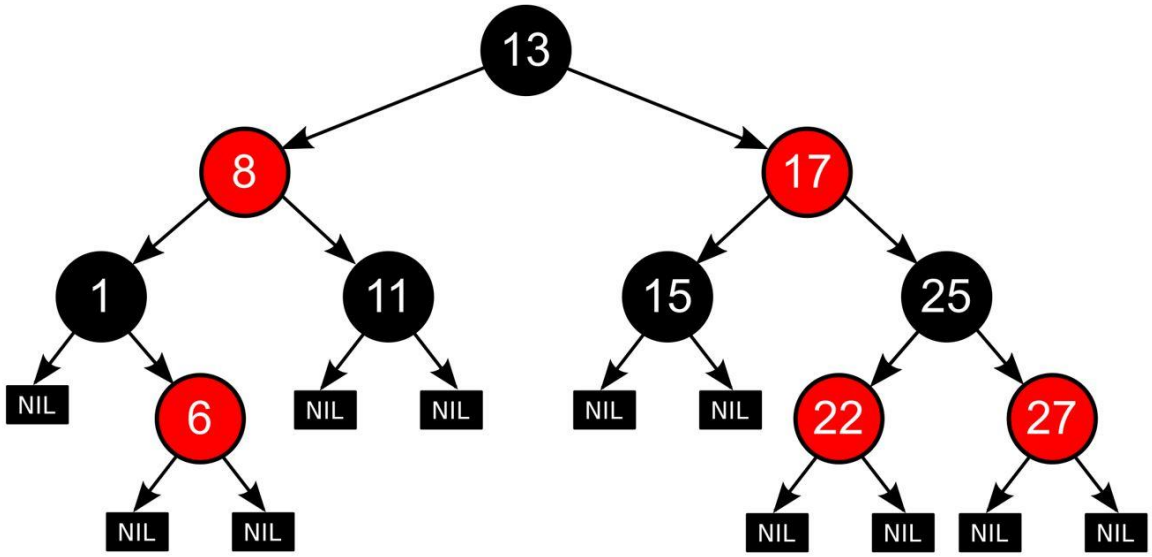
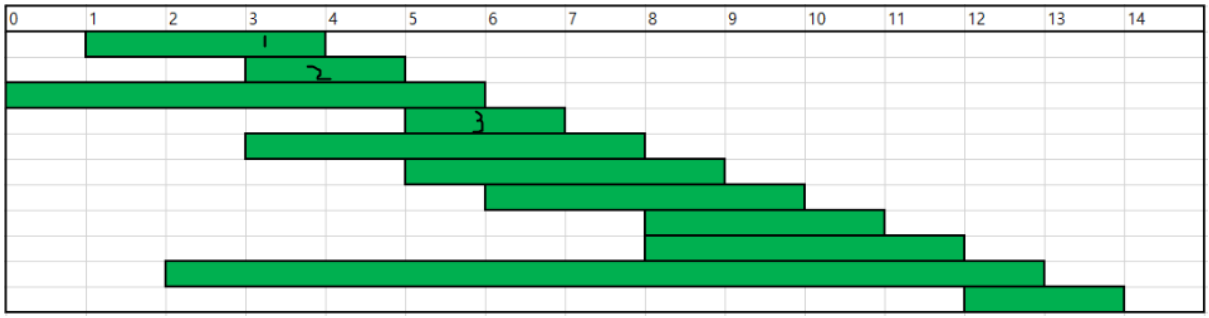


BR Tree의 구조 - TreeSet을 사용하면 자동으로 저 형태로 저장한다.



백준 1931 그림으로 나타낸 회의 시간표 (끝나는 시간 오름차순 정렬)

1->3 을 선택해도 다음 회의 시간은 7시 이후가 되어야 하고, 회의 개수는 2개이고

2->3을 선택해도 다음 회의 시간은 7시 이후가 되어야 하고, 회의 개수는 2개이다.

따라서 둘 중 당장 끝나는 시간이 빠른 회의를 선택하는 것이 반드시 이득이다.

그러므로 이 문제는 Greedy Algorithm을 사용해 풀 수 있다.

Sorting	장점	단점
버블 정렬	- 구현이 쉽다	- 시간이 오래 걸리고 비효율적이다.
선택 정렬	- 구현이 쉽다 - 비교하는 횟수에 비해 교환이 적게 일어난다.	- 시간이 오래 걸려서 비효율적이다.
퀵 정렬	- 실행시간 O(NlogN)으로 빠른 편이다	- Pivot에 의해서 성능의 차이가 심하다. - 최악의 경우 $O(N^2)$이 걸리게 된다.
힙 정렬	- 항상 O(NlogN)으로 빠른 편이다.	- 실제 시간이 다른 O(NlogN)이 정렬법들에 비해서 오래걸린다.
병합 정렬	- 항상 O(NlogN)으로 빠른 편이다.	- 추가적인 메모리 공간을 필요로 한다.
삽입 정렬	- 최선의 경우 O(N)으로 굉장히 빠른 편이다. - 성능이 좋아서 다른 정렬법에 일부로 사용됨.	- 최악의 경우$O(N^2)$ 으로, 데이터의 상태에 따라서 성능 차이가 심하다.
쉘 정렬	- 삽입정렬보다 더 빠른 정렬법이다.	- 설정하는 '간격'에 따라서 성능 차이가 심하다.
기수 정렬	- O(N)이라는 말도 안 되는 속도를 갖는다.	- 추가적인 메모리가 '많이' 필요하다. - 데이터 타입이 일정해야 한다. - 구현을 위한 조건이 많이 붙는다.
카운팅 정렬	- O(N)이라는 말도 안 되는 속도를 갖는다.	- 추가적인 메모리 공간이 필요하다. - 일부 값 때문에 메모리의 낭비가 심해질 수 있다.

array

index	0	1	2	3	4	5	6	7	8	9	10	11
value	7	2	3	5	7	1	4	6	7	5	0	1

counting

index	0	1	2	3	4	5	6	7
value	1	2	1	1	1	2	1	3

카운팅 정렬은 이런 식으로 들어 오는 숫자의 value의 범위가 정수이고, 정해져 있을 경우 사용할 수 있다.

Array에 value를 counting의 인덱스에 넣고, ++해주고(counting[array.value]++;), array를 1회 순회하며 모든 value를 count 했으면, counting array를 작은 숫자부터 value의 개수만큼 index를 출력해주기만 하면 정렬이 완료된다.