코딩테스트 대비

자료구조 & 알고리즘 정리본

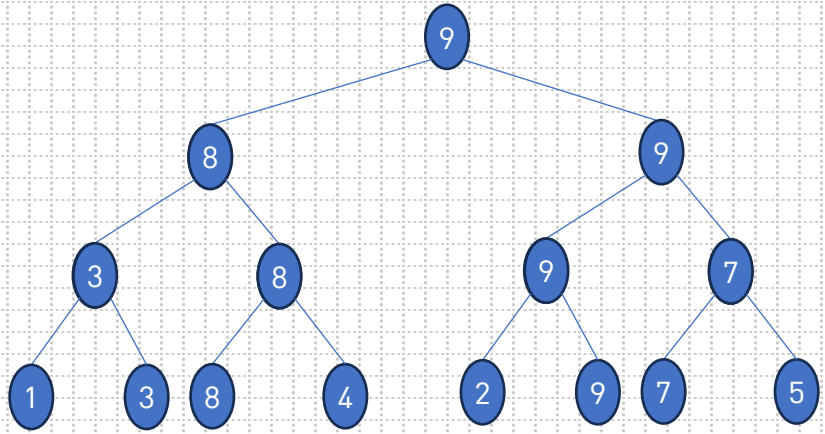
- 세그먼트 트리 -

---

devholic

세그먼트 트리 Segment Tree

- 구간의 합, 최솟값, 최댓값 등을 빠르게 구할 때 사용
  - 기존 구간 합 알고리즘의 단점: 값이 변경될 시 전부 갱신 필요
  - 반면 세그먼트 트리는 트리 구조를 활용해 빠른 시간 (O(logN))안에 갱신 가능
- 시간 복잡도
  - N개의 데이터에 대한 데이터의 변경: O(NlogN)
  - 구간 합 계산: O(logN) (이 점에서, 값이 변경되지 않는다면 구간 합을 이용하는 게 낫다는 결론을 내릴 수 있다. 구간 합은 누적 합 배열을 이용해 O(1)에 구할 수 있기 때문이다.)
  - 최소(최대)값 조회: 저장이 완료되었을 시 O(1) (단, 저장까지 O(NlogN) 소요)
- 공간 복잡도: O(logN) (높이) x O(N) (너비) = O(NlogN)
- 특징
  - 완전 이진 트리 구조를 띠어야 함
  - 원래 주어진 숫자 N보다 크거나 같은 2^M을 찾아야 함
  - 필요한 트리 배열의 크기는 2^M * 2 (1번 인덱스부터 시작되도록)
    - ex: N이 5일 시 필요한 크기는 8개 (M: 3), 따라서 16개 필요
  - 트리 배열의 초기값은 문제의 요구사항에 따라 다름
    - 최솟값 배열일 경우: INF
    - 최댓값, 누적 합 배열일 경우: 0



세그먼트 트리의 최댓값 트리 예시

예시 문제: 구간 합 구하기 (백준 / 골드 1)

문제 링크: https://www.acmicpc.net/problem/2042

- N의 크기: 5
- 따라서 N보다 큰 2^M은 8 (M = 3), 트리의 길이는 2^M * 2 = 16
- [1 2 3 4 5]로 입력 받음

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

입력값이 저장된 배열

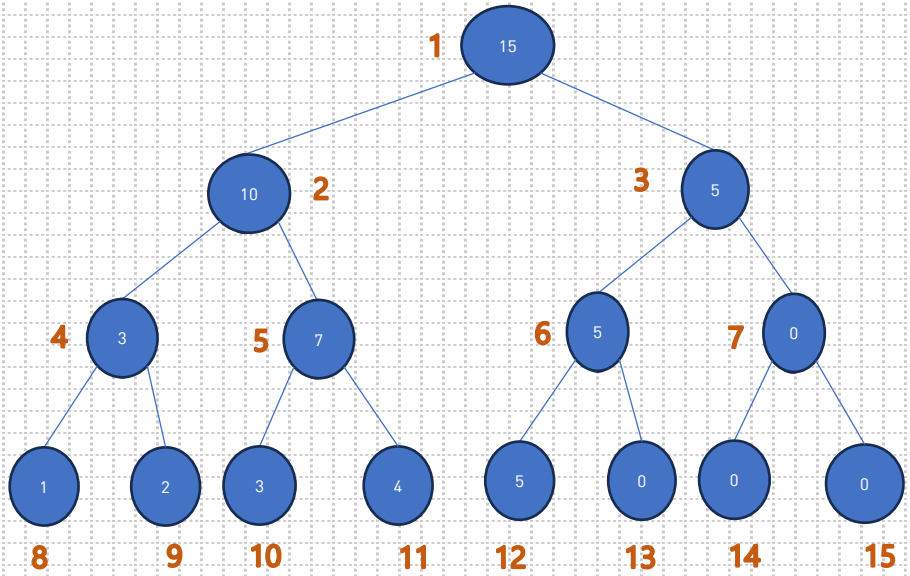
index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

트리의 초기 상태

예시 문제: 구간 합 구하기 (백준 / 골드 1)

문제 링크: https://www.acmicpc.net/problem/2042

- 2^M이 필요했던 이유
  입력 값 배열의 i번째 인덱스가 트리 상에 어디에 위치하는지 파악하기 위함!
  (입력의 i번째 값 = 트리의 2^M – 1번째의 값)
  입력 값이 저장된 배열의 값들은 트리의 리프 노드에 위치하도록 구현



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

입력값이 저장된 배열

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
value	0	15	10	5	3	7	5	0	1	2	3	4	5	0	0	0

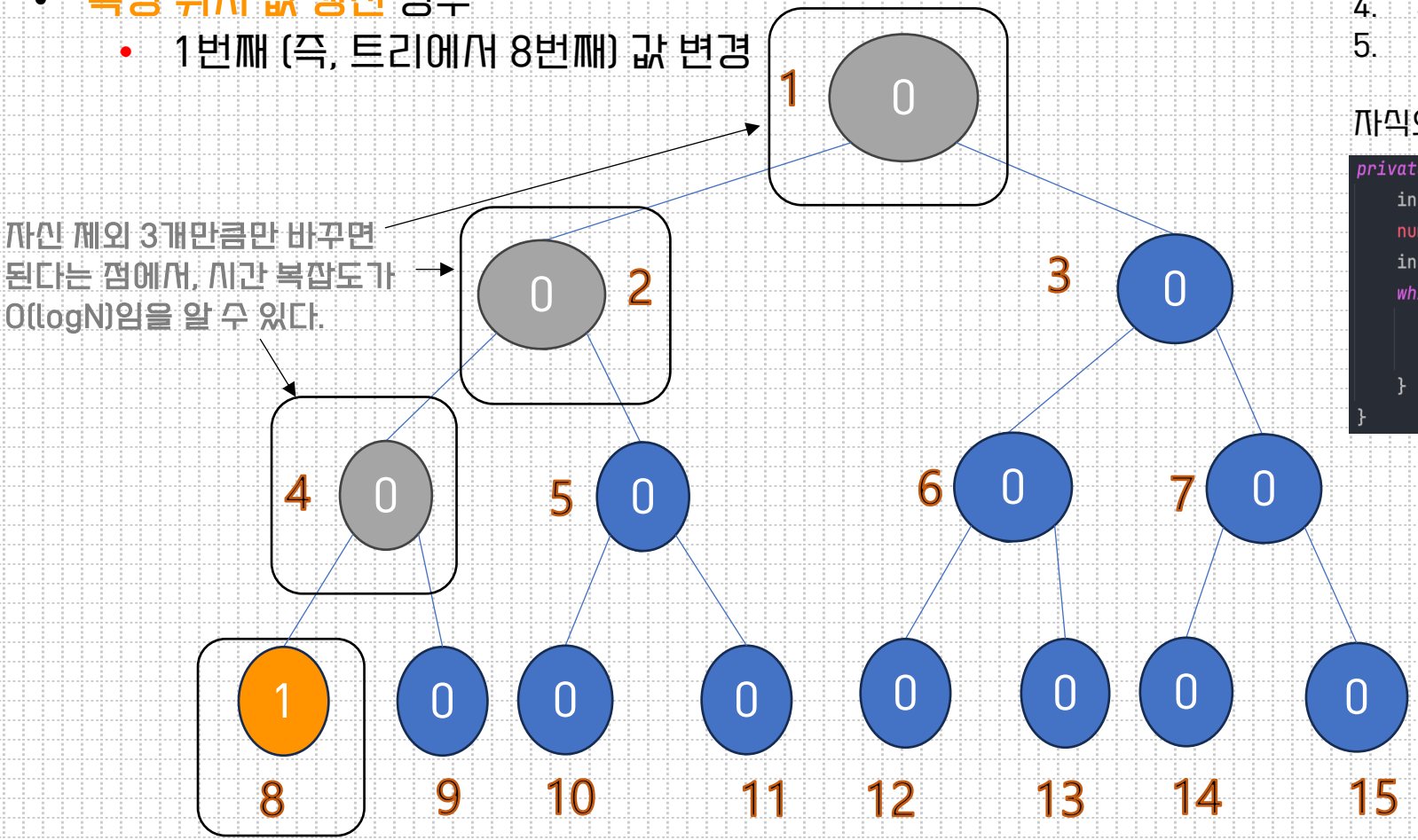
세그먼트 트리

1번째는 8번째에 위치, 2번째는 9번째에 위치.. i번째는 i + 7번째에 위치함을 알 수 있다.
그런데 이 7이라는 숫자는 2^M – 1과 같다!

예시 문제: 구간 합 구하기 (백준 / 골드 1)

문제 링크: https://www.acmicpc.net/problem/2042

- **특정 위치 값 갱신** 경우
  - 1번째 (즉, 트리에서 8번째) 값 변경



1. 인덱스 += 트리 기본 인덱스 (7)
2. 8번째에 값 할당
3. 8번째 값 + 9번째 값을 4번째 값 (부모)에 할당
4. 4번째 값 + 5번째 값을 2번째 값 (부모)에 할당
5. 2번째 값 + 3번째 값을 1번째 값 (부모)에 할당

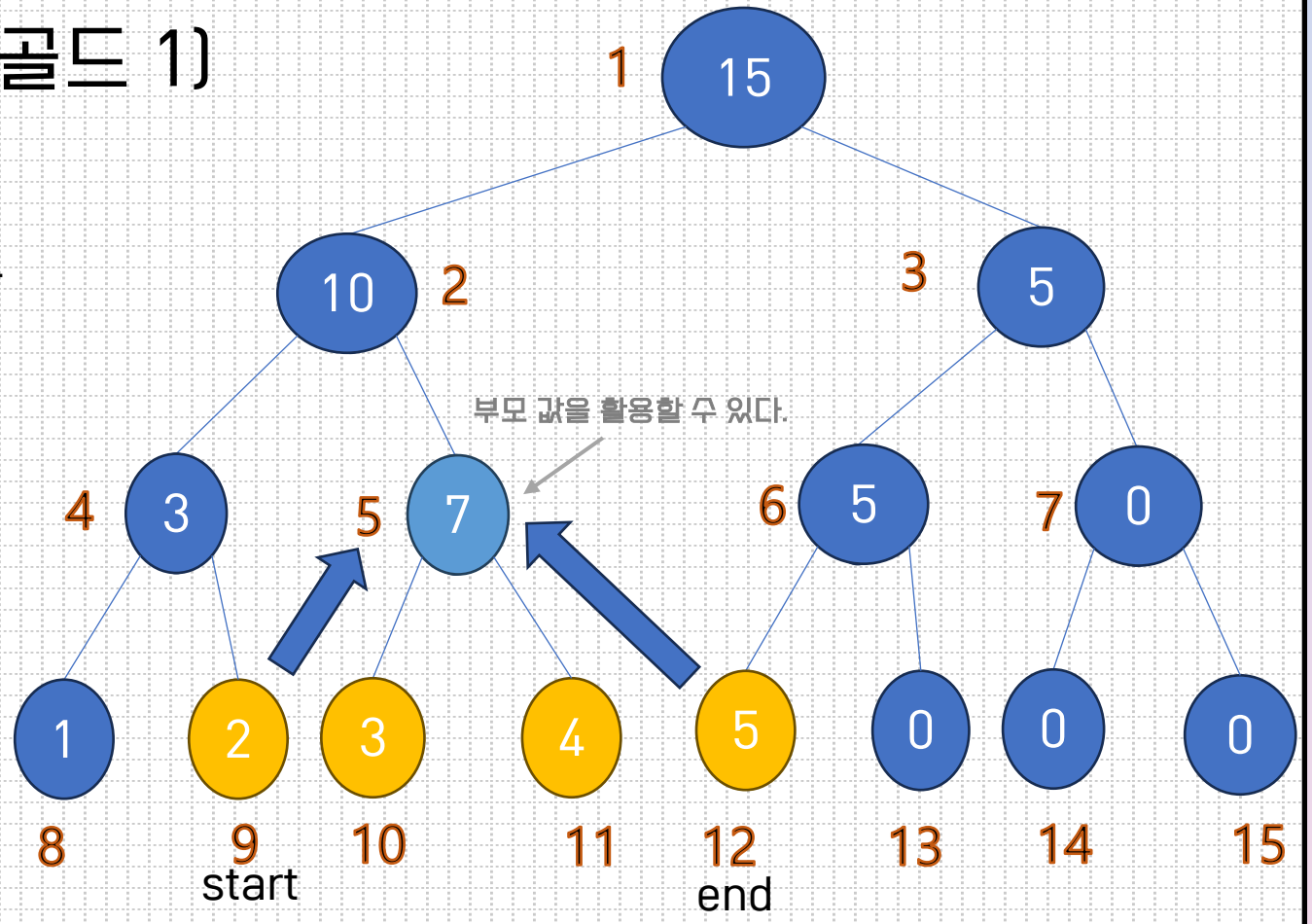
자식의 인덱스 / 2 (나머지 버림) = 부모의 인덱스

```
private static void update(int index, long value) {
    index += treeIndex;
    numbers[index] = value;
    index /= 2;
    while (index >= 1) {
        numbers[index] = numbers[index * 2] + numbers[index * 2 + 1];
        index /= 2;
    }
}
```

예시 문제: 구간 합 구하기 (백준 / 골드 1)

문제 링크: https://www.acmicpc.net/problem/2042

- **특정 구간 합 계산** 경우
  - 2번째 ~ 5번째 (트리에서 9번째 ~ 12번째)의 합
  - start와 end를 각각 9, 12로 잡는다.
  - start가 홀수라면, 우측에 있는 자식 노드이다.
    - 전체 result에 tree[start]를 담는다.
  - end가 짝수라면, 좌측에 있는 자식 노드이다.
    - 전체 result에 tree[end]를 담는다.
  - start는 다음 부모로 점프한다.
    - 사이에 있는 부모를 활용할 수 있다.
    - 다음 부모는 (start + 1) / 2와 같다.
  - end 또한 다음 부모로 점프한다.
    - 다음 부모는 (end - 1) / 2와 같다.
  - 이것을 start <= end일 때 까지 반복한다.
    - 같을 때 해야 하는 이유는 그러지 않을 경우 공통된 tree[5]로 넘어왔는데 더하지 못하고 끝나버리기 때문이다.
    - start가 홀수이든 end가 짝수이든 둘 중에 한 경우에는 포함되게 된다.



이 또한 최대 트리의 높이만큼 진행되기 때문에 O(logN)임을 알 수 있다.

예시 문제: 구간 합 구하기 (백준 / 골드 1) 전체 코드는 이곳 참고!

문제 링크: https://www.acmicpc.net/problem/2042

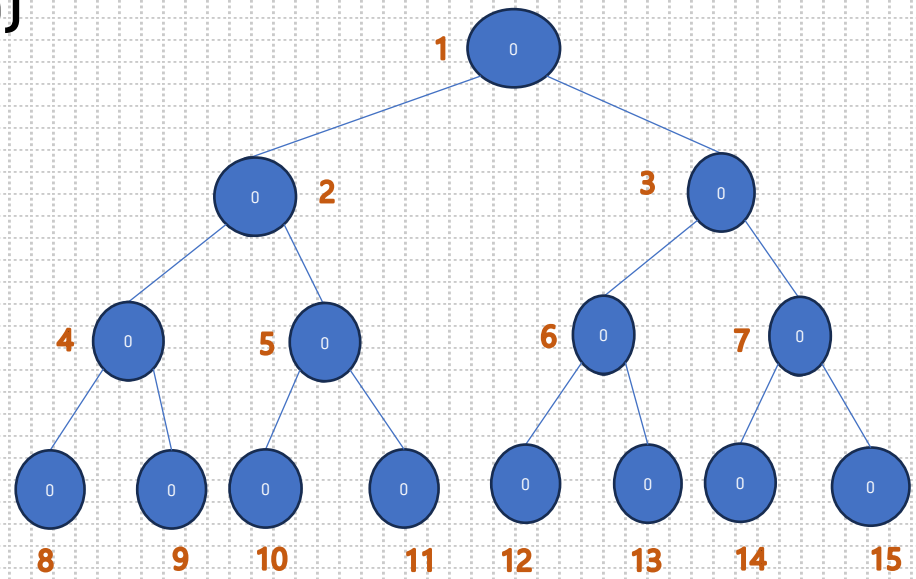
특정 구간 합 계산 코드

```java
private static long sum(int start, int end) {
    start += treeIndex;
    end += treeIndex;

    long sum = 0;
    while (start <= end) {
        if (start % 2 != 0) {
            sum += numbers[start];
        }
        if (end % 2 == 0) {
            sum += numbers[end];
        }
        start = (start + 1) / 2;
        end = (end - 1) / 2;
    }

    return sum;
}
```

심화 문제: 부분배열 고르기 (백준 / 플래티넘 5)

문제 링크: https://www.acmicpc.net/problem/2104

- **최솟값 인덱스 트리** 구현 – 초기 설정
  - 숫자 배열의 0번째 요소에 INF를 할당한다.
  - 트리 배열의 값을 전부 0으로 할당한다.



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

numbers

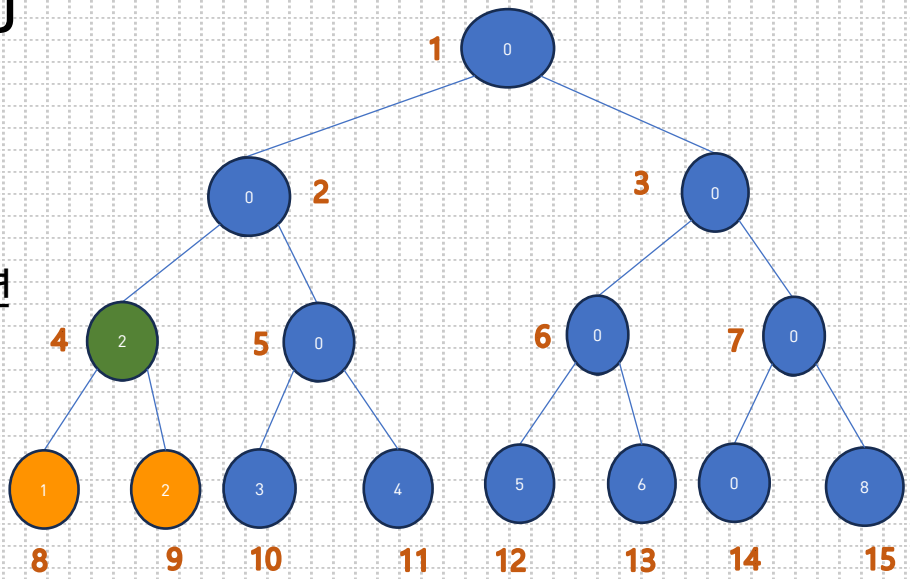
index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

positions

심화 문제: 부분배열 고르기 (백준 / 플래티넘 5)

문제 링크: https://www.acmicpc.net/problem/2104

- 최솟값 인덱스 트리 구현 – 값 갱신
  - 트리의 리프 노드들은 해당 인덱스를 넣는다.
    - 0번 인덱스의 실제 값 (INF)과 해당 인덱스의 실제 값을 비교하면 당연히 해당 인덱스의 실제 값이 더 작기 때문이다.
  - numbers[positions[index * 2]], numbers[positions[index * 2 + 1]] 중 값이 더 작은 인덱스를 positions[index]에 할당한다.



index = 4일 때,
numbers[positions[8]] = numbers[1] = 3
numbers[positions[9]] = numbers[2] = 1
따라서 positions[4] = positions[9] = 1

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

numbers

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
value	0	0	0	0	0	0	0	0	1	2	3	4	5	6	0	0

positions

심화 문제: 부분배열 고르기 (백준 / 플래티넘 5)

문제 링크: https://www.acmicpc.net/problem/2104

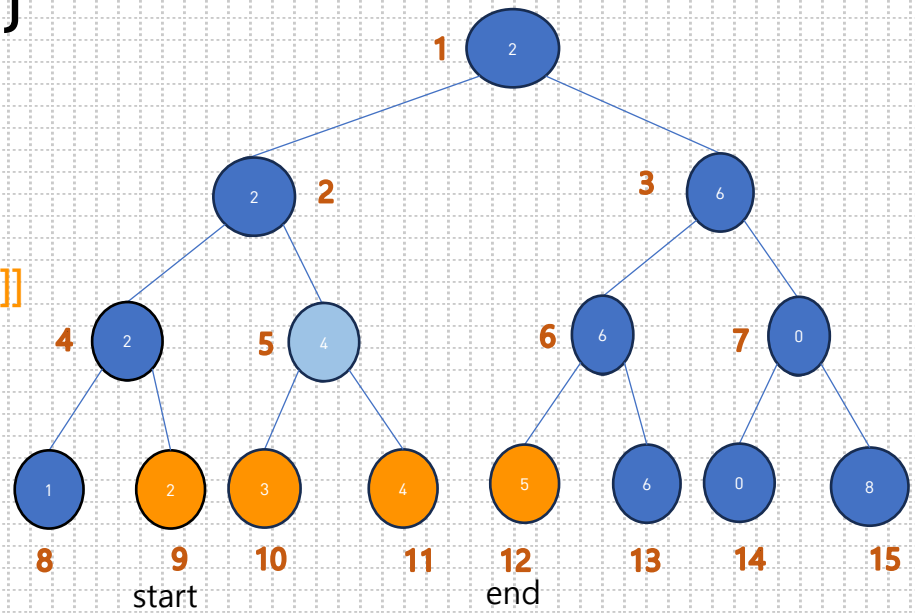
- 최솟값 인덱스 트리 구현 – 값 갱신 코드

```
private static void update(int index) {
    index += treeIndex;
    positions[index] = index - treeIndex;
    index /= 2;
    while (index >= 1) {
        if (numbers[positions[index * 2]] < numbers[positions[index * 2 + 1]]) {
            positions[index] = positions[index * 2];
        } else {
            positions[index] = positions[index * 2 + 1];
        }
        index /= 2;
    }
}
```

심화 문제: 부분배열 고르기 (백준 / 플래티넘 5)

문제 링크: https://www.acmicpc.net/problem/2104

- 최솟값 인덱스 트리 구현 – start ~ end 사이의 최솟값 인덱스 계산
  - 초기 인덱스 (결과값) 0으로 설정
  - start가 홀수이면서 numbers[인덱스] > numbers[positions[start]] 라면 인덱스를 positions[start]로 갱신
  - end가 짝수이면서 numbers[인덱스] > numbers[positions[end]] 라면 인덱스를 positions[end]로 갱신
  - 그 외 과정은 2042 문제에서의 구간 합 코드와 동일



numbers[0] > numbers[positions[9]] = INF > 1, 갱신
numbers[2] > numbers[positions[12]] = 1 > 5, 갱신 X
이후 로직에 의해 반복

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

numbers

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
value	0	2	2	6	2	4	6	0	1	2	3	4	5	6	0	0

positions

심화 문제: 부분배열 고르기 (백준 / 플래티넘 5) 전체 코드는 이곳 참고!

문제 링크: https://www.acmicpc.net/problem/2104

- 최솟값 인덱스 트리 구현 – start ~ end 사이의 최솟값 인덱스 계산 코드

```
private static int getMinIndex(int start, int end) {
    start += treeIndex;
    end += treeIndex;
    if (start == end) {
        return start;
    }
    int index = 0;
    while (start <= end) {
        if (start % 2 != 0) {
            if (numbers[index] > numbers[positions[start]]) {
                index = positions[start];
            }
        }
        if (end % 2 == 0) {
            if (numbers[index] > numbers[positions[end]]) {
                index = positions[end];
            }
        }
        start = (start + 1) / 2;
        end = (end - 1) / 2;
    }
    return index;
}
```

Reference

- [wikidocs] 세그먼트 트리: https://wikidocs.net/209446
- [백준] 구간 합 구하기: https://www.acmicpc.net/problem/2042
- [백준] 부분배열 고르기: https://www.acmicpc.net/problem/2104