Segment Tree

2020.01.11

Segment Tree

- 주어진 배열의 각 구간 정보를 트리로 저장 (구간 내의 최댓/최솟값, 구간 내 모든 수의 합 등등)

→ **O(n)**

- 1) 구간에 대한 쿼리 (주어진 배열에서 구간 [2, 5]에서의 최댓값 찾기)
  2) 배열 내 임의의 원소를 수정한 후 트리에 반영

→ **O(logn)**

Naive Approach

7 2 4 5 1 7 2 9 3 5

> 구간 [2,5]의 최댓값, 구간 [1,3]의 최댓값, 구간 [4,8]의 최댓값, …

➡ 매번 하나하나 체크: O(n) x 쿼리 m개 = **O(mn)**

> 구간 [2,5]의 합, 임의의 원소 수정, 구간 [1,3]의 합, …

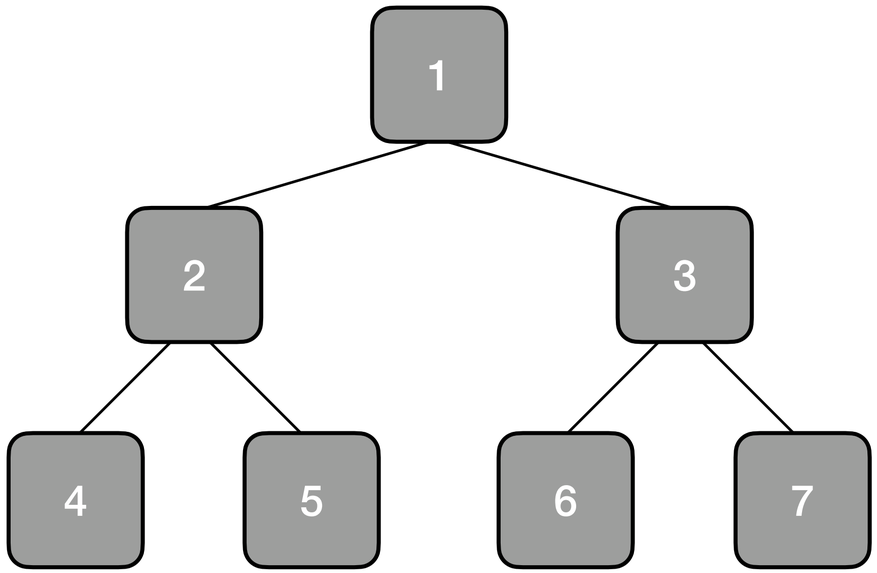
➡ 업데이트가 포함되어 구간합 활용이 어려움

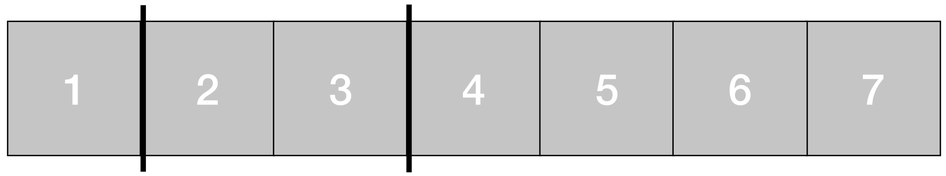
➡ 꿋꿋이 구간합 사용/매번 하나하나 체크: **O(mn)**

Prerequisite

Heap

내용은 전혀 관련 없지만, 구현 시 자료구조 활용 방식이 동일





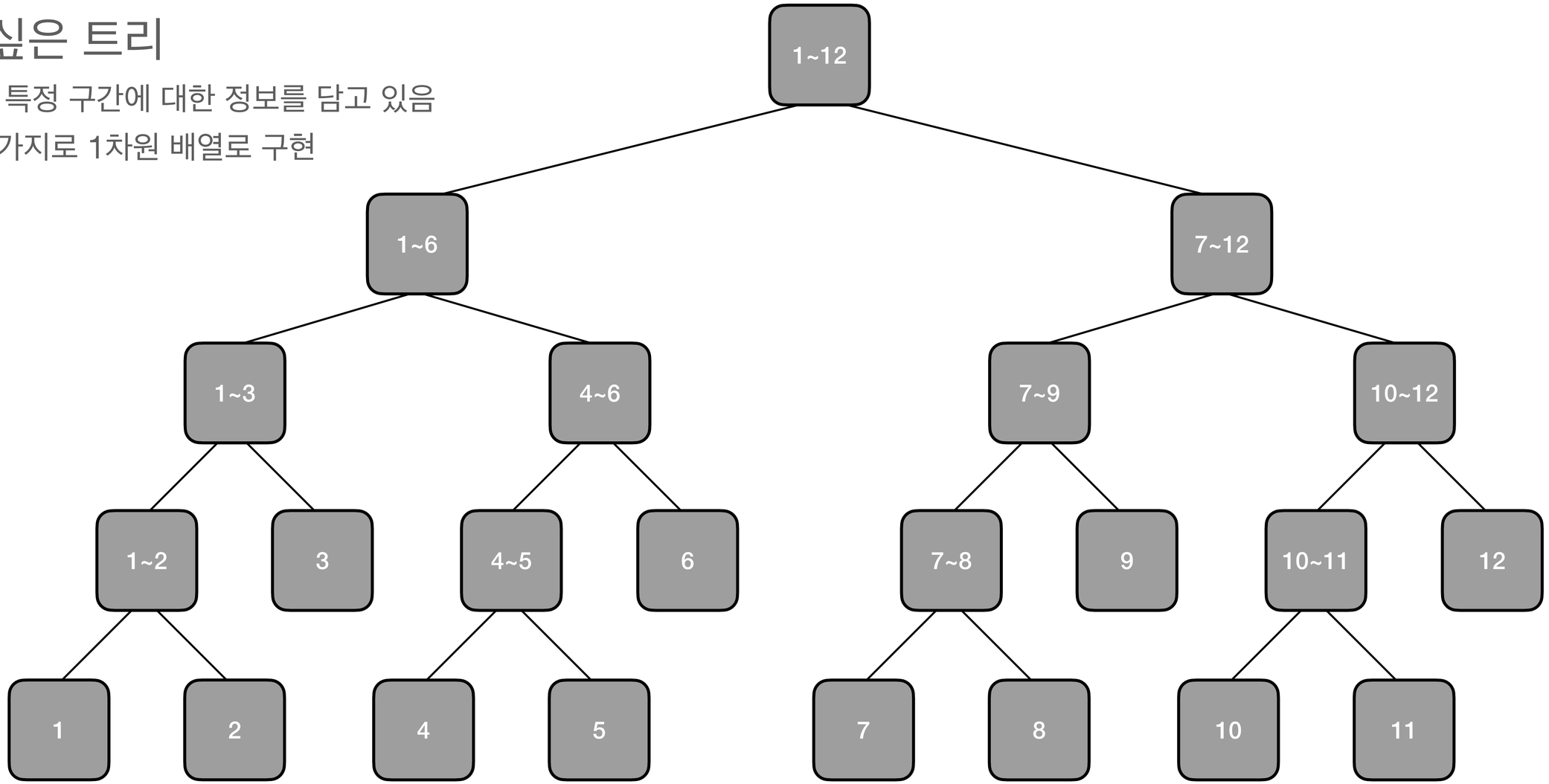
→ k번 노드의 자식 노드: **2k, 2k+1**

*** 이번 내용에서는 편의상 인덱스를 1부터 시작하는 걸로**

How it works

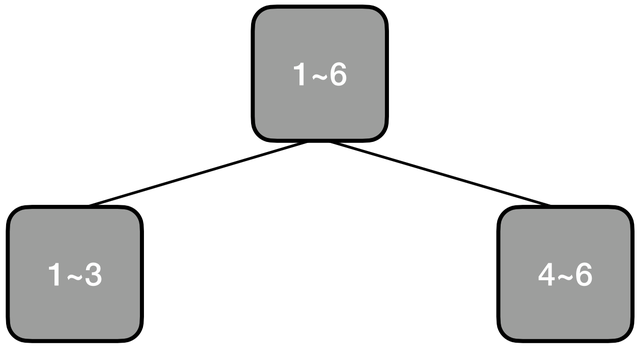
만들고 싶은 트리

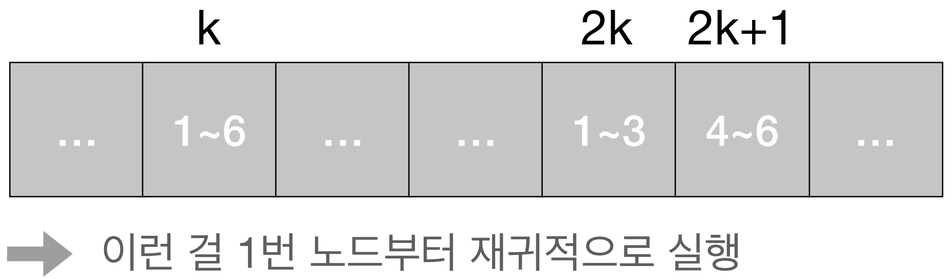
- 각 노드가 특정 구간에 대한 정보를 담고 있음
- 힙과 마찬가지로 1차원 배열로 구현



How it works

Construction



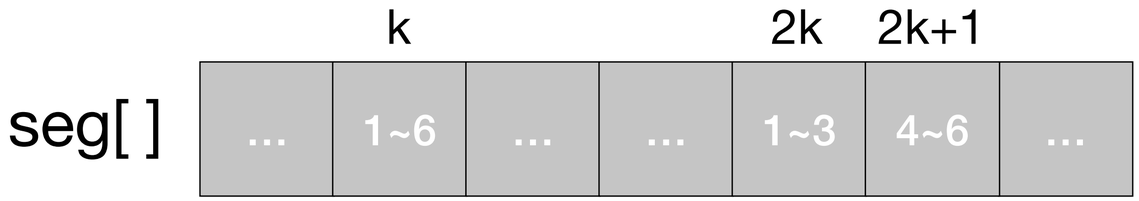


이런 걸 1번 노드부터 재귀적으로 실행

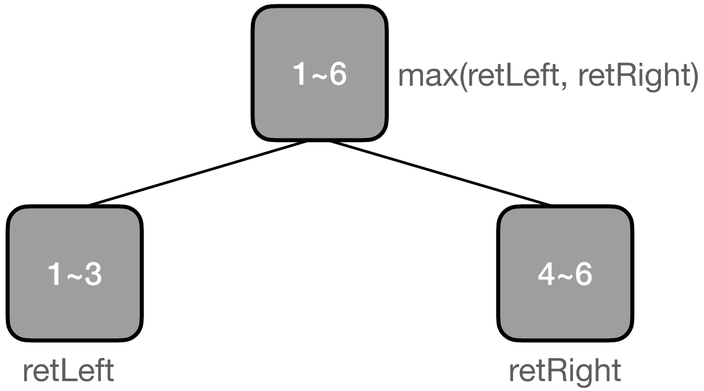
How it works

Construction

A[ ] **7 2 4 5 1 7 2 9 3 7 5 8**



➡ 이런 걸 1번 노드부터 재귀적으로 실행



construct(int node, int nodeLeft, int nodeRight)

➡ node: 노드 번호 **(for seg[ ])**
nodeLeft, nodeRight: 그 노드가 나타내는 범위 **(for A[ ])**

1) nodeLeft == nodeRight일 경우,
요구사항에 맞는 값 **seg[node]**에 저장 후 반환
(최댓/최솟값, 구간합 등 일반적인 경우에는 A[nodeLeft])

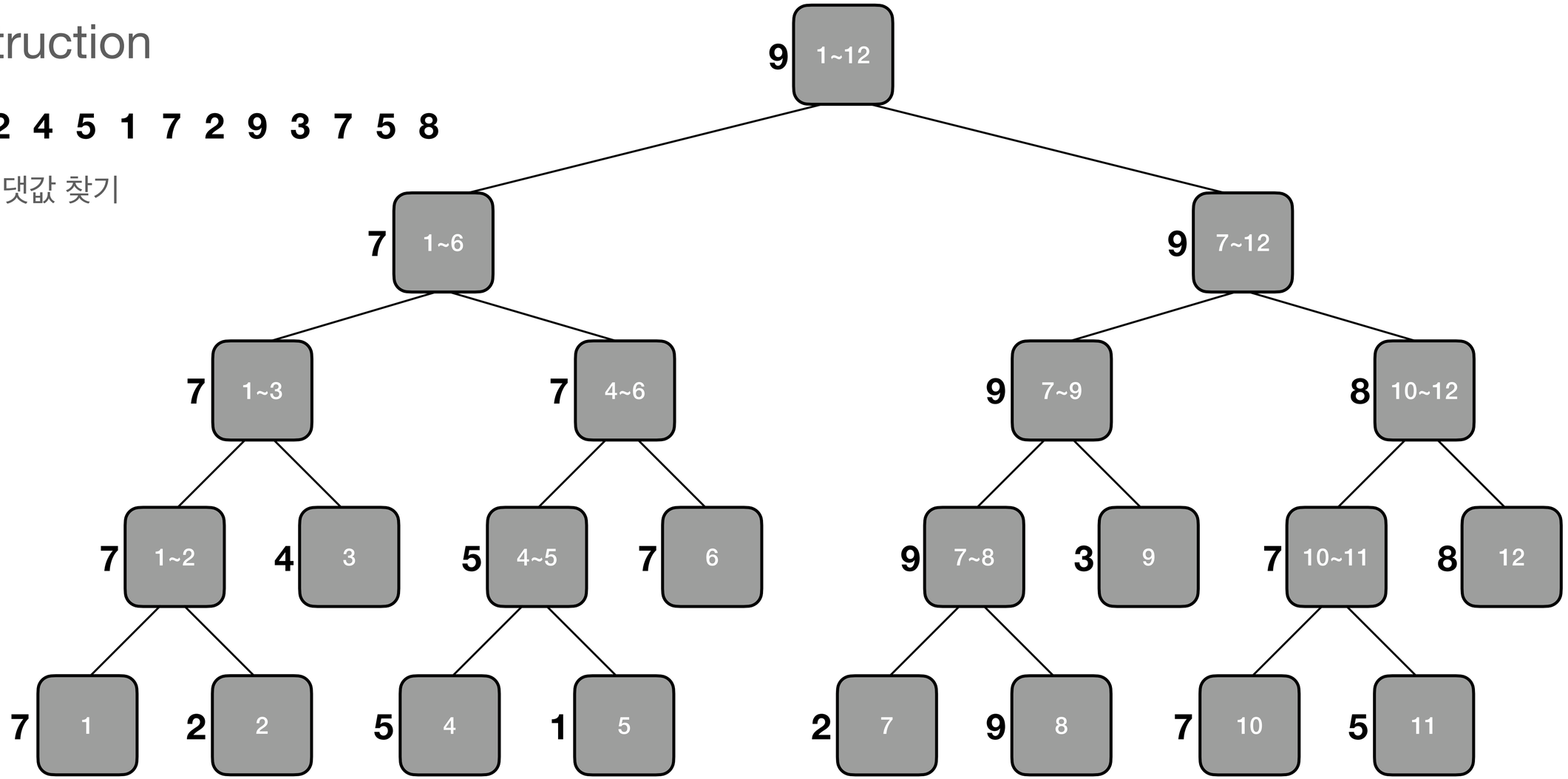
2) nodeLeft != nodeRight일 경우,
mid = (nodeLeft+nodeRight) / 2 로 하여
retLeft = **construct(2*node, nodeLeft, mid)**,
retRight = **construct(2*node+1, mid+1, nodeRight)**
를 구한 뒤 요구사항에 맞게 값을 **seg[node]**에 저장 후 반환함
(최댓값의 경우 max(retLeft, retRight))

How it works

Construction

A[ ] **7 2 4 5 1 7 2 9 3 7 5 8**

각 구간 별 최댓값 찾기



How it works

Construction

세그먼트 트리의 각 노드에 대해 1번 씩만 탐색하기 때문에 노드가 몇 개 필요한지(= seg[ ]의 크기)를 알면 됨

n이 2의 제곱수일 경우: 세그먼트 트리는 full binary tree이므로 노드의 수는 2n-1

n이 2의 제곱수가 아닐 경우, 세그먼트 트리의 높이는 h = floor(logn) ~~(트리 높이를 0부터 세던가..? 기억이 안나네)~~
seg[ ] 의 크기를 2^(h+1)-1 로 하면 됨

TL;DR

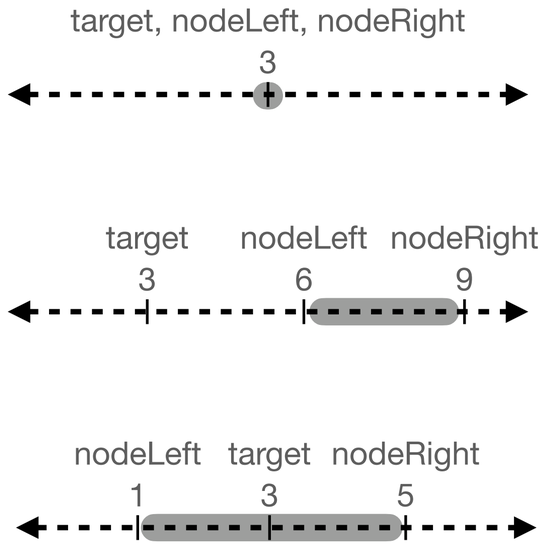
➡ **O(n)**

➡ seg[ ]의 크기를 **4n**으로 잡으면 모든 경우가 커버 가능하다 카더라

How it works

Update

O(logn)



update(int node, int nodeLeft, int nodeRight, int target, int value)

node: 노드 번호 **(for seg[ ])**
nodeLeft, nodeRight: 그 노드가 나타내는 범위 **(for A[ ])**
target, value: 변경할 인덱스**(for A[ ])**와 새로운 값

1) nodeLeft == nodeRight == target일 경우,
value를 **seg[node]**에 저장 후 반환

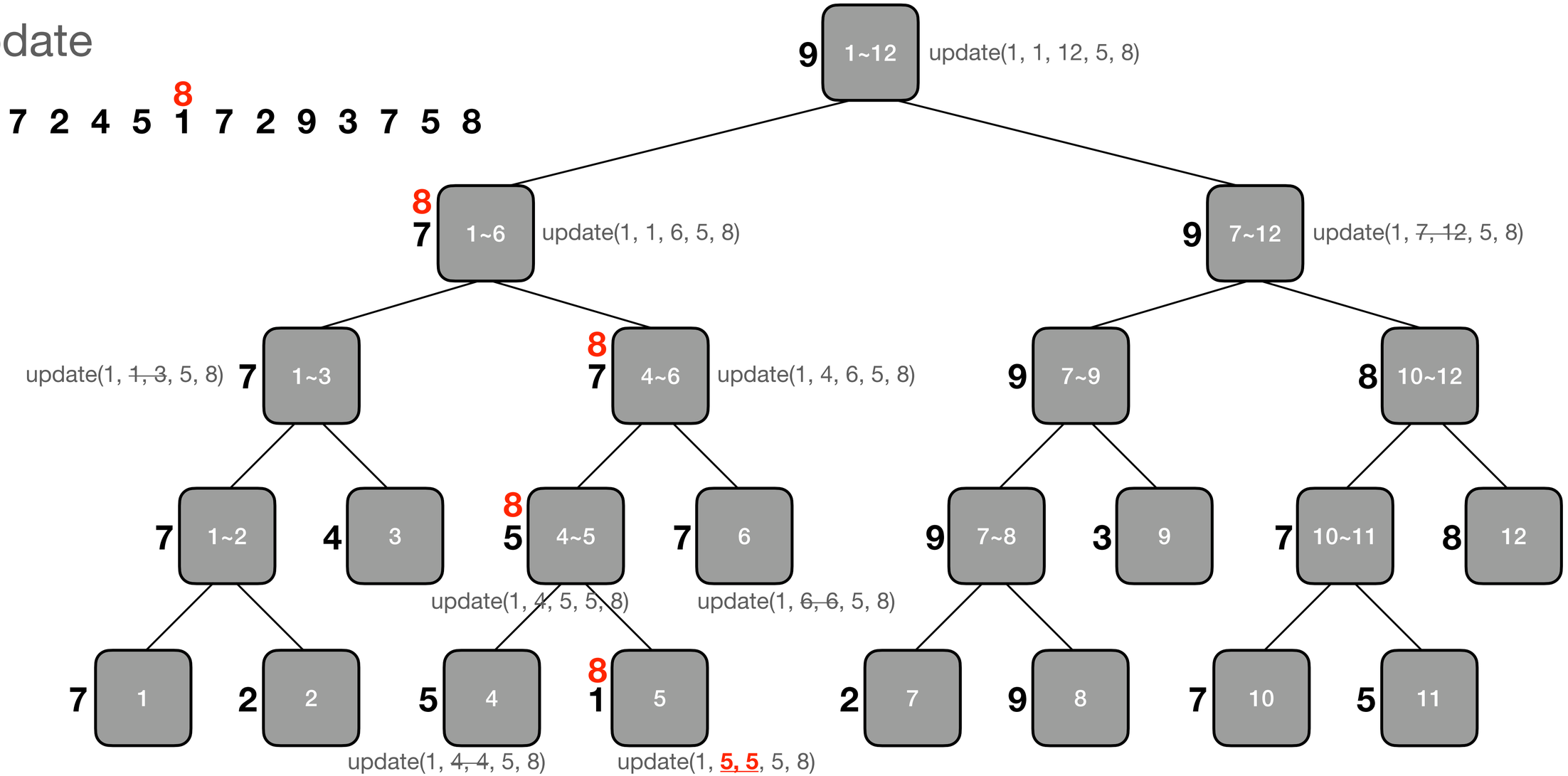
2) target < nodeLeft || nodeRight < target (target이 해당 노드 범위에 포함X),
여기엔 볼 일 없으므로 기존 값 그대로 반환

3) 그 외 (target이 [nodeLeft, nodeRight] 에 포함됨)
mid = (nodeLeft+nodeRight) / 2 로 하여
retLeft = **update(2*node, nodeLeft, mid, target, value)**,
retRight = **update(2*node+1, mid+1, nodeRight, target, value)**
를 구한 뒤 요구사항에 맞게 값을 **seg[node]**에 저장 후 반환함

How it works

Update

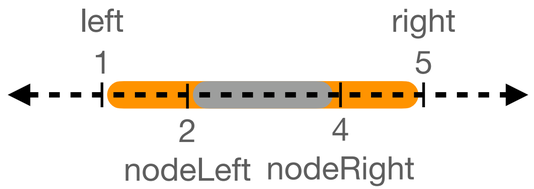
A[ ] 7 2 4 5 8 1 7 2 9 3 7 5 8

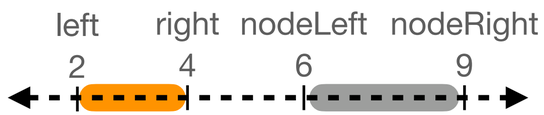


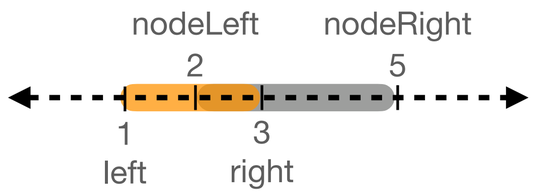
How it works

Query

→ O(logn)







query(int node, int nodeLeft, int nodeRight, int left, int right)

→ node: 노드 번호 **(for seg[ ])**
nodeLeft, nodeRight: 그 노드가 나타내는 범위 **(for A[ ])**
left, right: 쿼리에서 묻는 구간 **(for A[ ])**

1) left <= nodeLeft && nodeRight <= right (해당 노드 범위가 쿼리 구간 안에 포함)
seg[node]를 즉시 반환

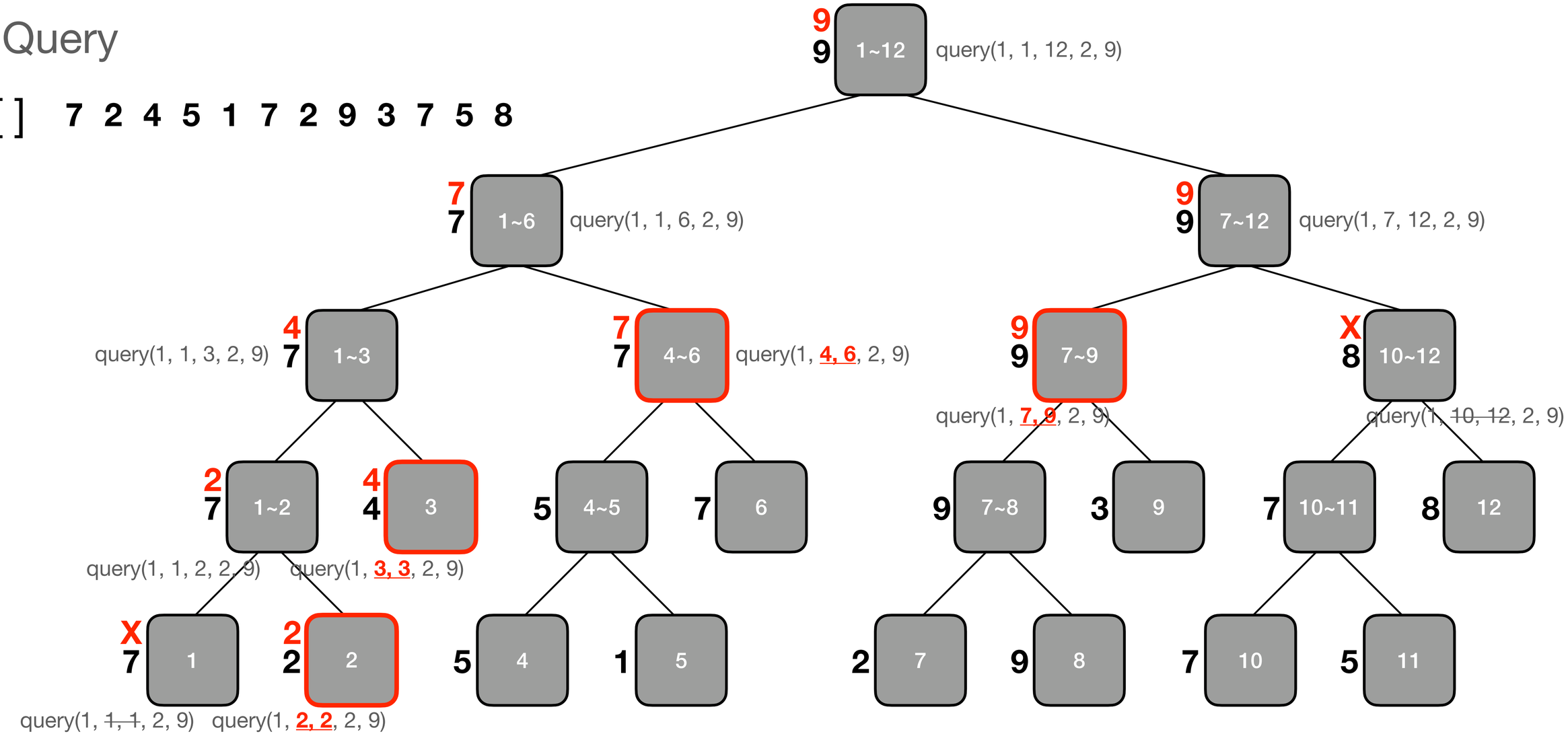
2) right < nodeLeft || nodeRight < left (쿼리 구간이 해당 노드 범위와 겹침X)
여기엔 볼 일 없으므로 결과값에 영향을 미치지 않는 쓰레기값 반환
(최댓값을 구할 경우 여기서는 아주아주 작은 값 반환)

3) 그 외 (구간이 [nodeLeft, nodeRight] 에 걸침)
mid = (nodeLeft+nodeRight) / 2 로 하여
retLeft = **query(2*node, nodeLeft, mid, left, right)**,
retRight = **query(2*node+1, mid+1, nodeRight, left, right)**
를 구한 뒤 요구사항에 맞게 값을 반환함

How it works

Query

A[ ] 7 2 4 5 1 7 2 9 3 7 5 8



Does it work?

Query 연산의 시간복잡도는 왜 O(logn)인가요? 그럴거 같긴 하지만

➡ **그러게요..?(죄송…)**

➡ 아무튼 업데이트가 포함된 m개의 쿼리를 **O(mlogn)**으로 처리할 수 있음…아무튼

Related Problems

- [BOJ 2042] 구간합 구하기
- [BOJ 2357] 최솟값과 최댓값
- [BOJ 14417] 팰린드롬과 쿼리2 (Manacher + Segment Tree)