


# Segment tree

2020 winter 중급

20141284 이기현





- #11659 구간 합 구하기 4 
- N개의 수, M번의 쿼리( $1 \leq N, M \leq 100,000$ )
- $i \sim j$  번째 수의 합을 구하여라



## • #11659 구간 합 구하기 4

- N개의 수, M번의 쿼리( $1 \leq N, M \leq 100,000$ )
- $i \sim j$  번째 수의 합을 구하여라
- prefix sum 전처리  $O(N)$
- $i \sim j$  번째 수의 합을 구하는데  $O(1)$



## • #11659 구간 합 구하기 4

- N개의 수, M번의 쿼리( $1 \leq N, M \leq 100,000$ )
- $i \sim j$  번째 수의 합을 구하여라
- prefix sum 전처리  $O(N)$
- $i \sim j$  번째 수의 합을 구하는데  $O(1)$
- 시간 복잡도  $O(N)$



## • #11659 구간 합 구하기 4

- N개의 수, M번의 쿼리( $1 \leq N, M \leq 100,000$ )

- $i \sim j$  번째 수의 합을 구하여라

k 번째 수를 x로 바꿔라

- prefix sum 전처리  $O(N)$

- $i \sim j$  번째 수의 합을 구하는데  $O(1)$

- 시간 복잡도  $O(N)$



## • #11659 구간 합 구하기 4

- N개의 수, M번의 쿼리( $1 \leq N, M \leq 100,000$ )
- $i \sim j$  번째 수의 합을 구하여라
- prefix sum 전처리  $O(N)$
- $i \sim j$  번째 수의 합을 구하는데  $O(1)$
- 시간 복잡도  $O(N)$

k 번째 수를 x로 바꿔라



K 이후 prefix sum 값 수정  $O(N)$



## • #11659 구간 합 구하기 4

- N개의 수, M번의 쿼리( $1 \leq N, M \leq 100,000$ )
- $i \sim j$  번째 수의 합을 구하여라
- prefix sum 전처리  $O(N)$
- $i \sim j$  번째 수의 합을 구하는데  $O(1)$
- 시간 복잡도  $O(N)$

k 번째 수를 x로 바꿔라

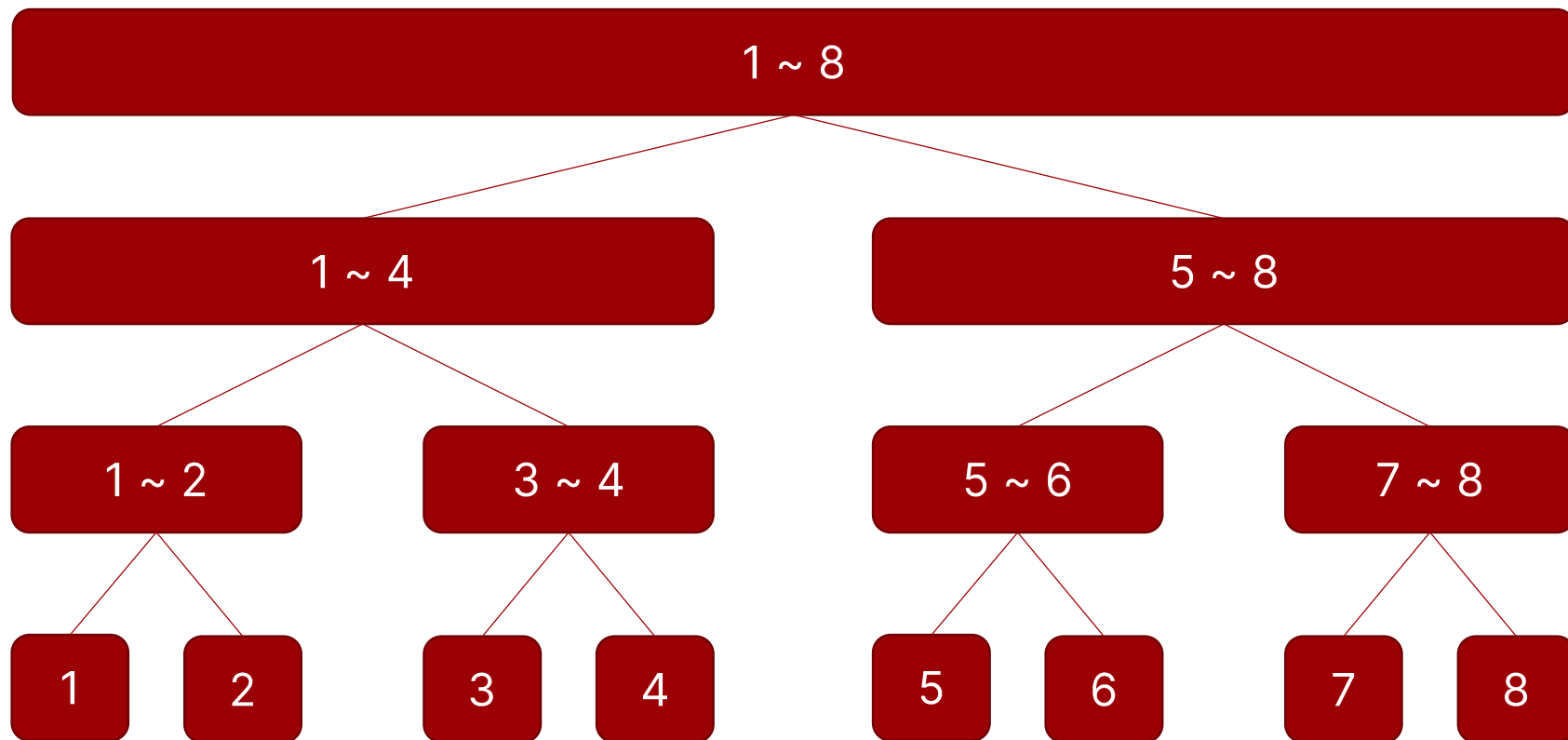


K 이후 prefix sum 값 수정  $O(N)$



시간 복잡도  $O(NM)$

## Segment tree

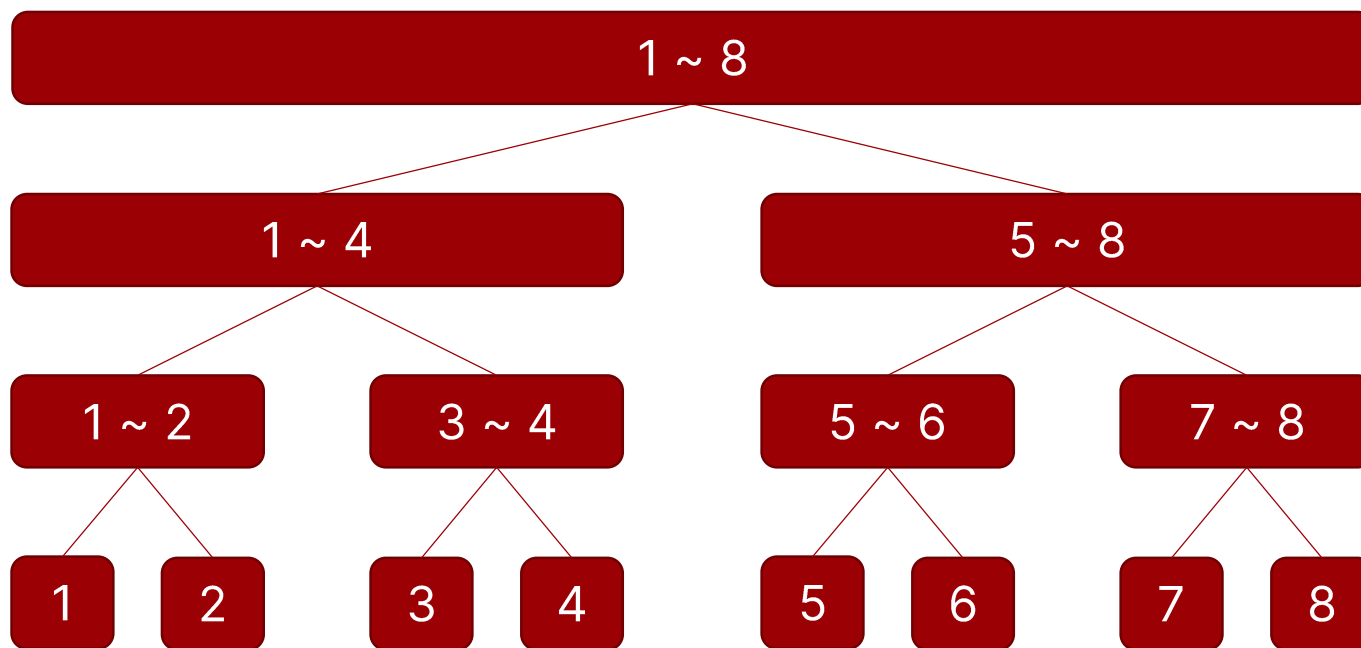






## 세그먼트 트리

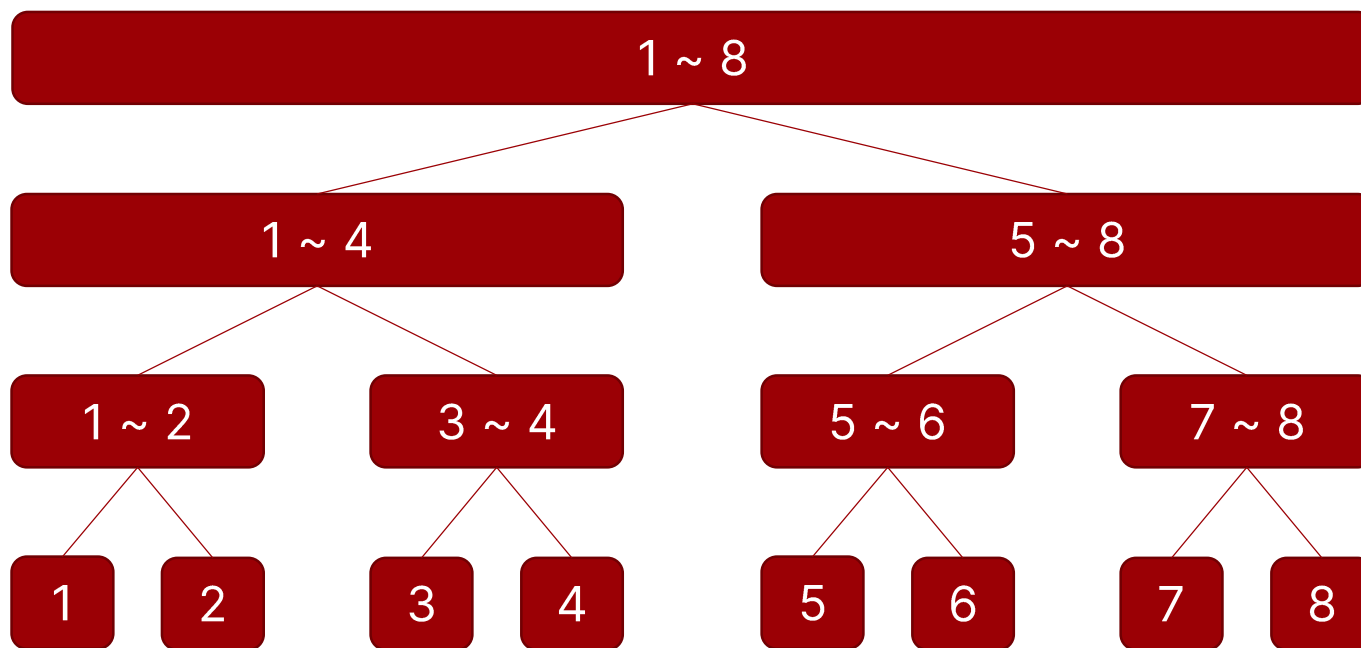
- 각 노드는 '구간' 의 정보





## 세그먼트 트리

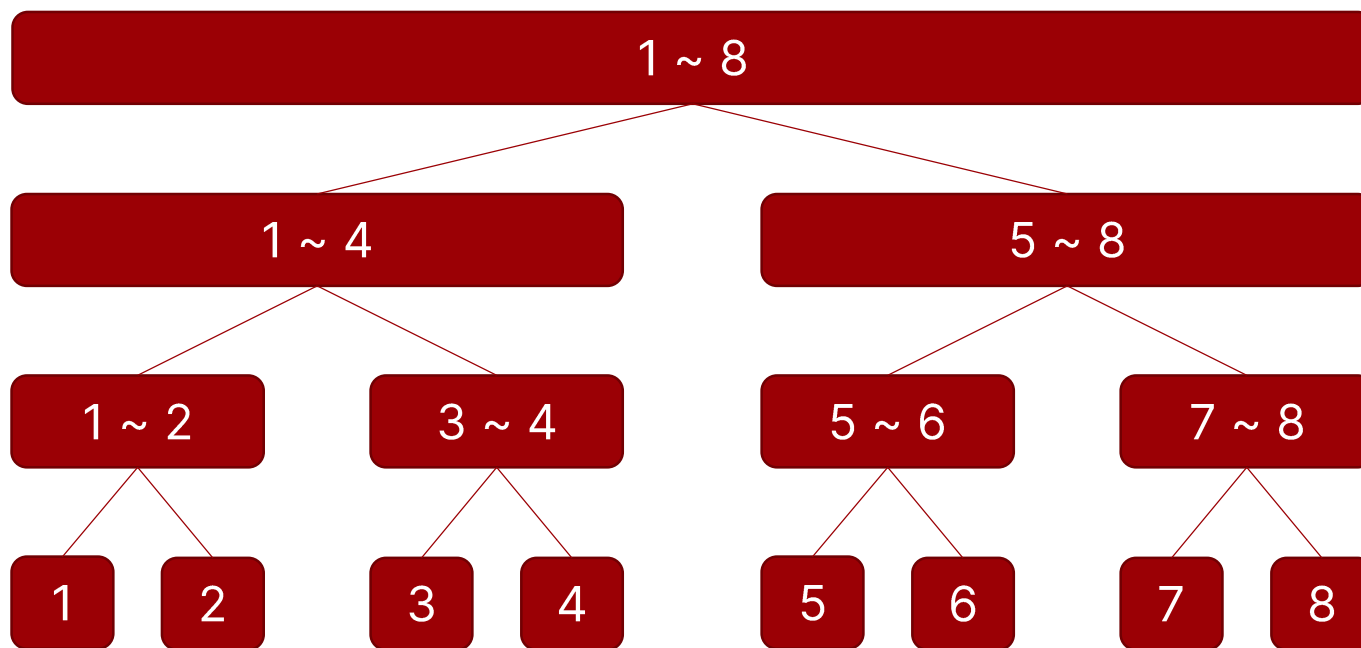
- 각 노드는 '구간' 의 정보
- 부모는 자식의 구간을 포괄





## 세그먼트 트리

- 각 노드는 '구간' 의 정보
- 부모는 자식의 구간을 포괄
- Binary tree (complete 끝)
  - root 의 인덱스  $\rightarrow 1$
  - 부모의 인덱스  $\rightarrow i$
  - 자식의 인덱스  $\rightarrow i * 2, i * 2 + 1$





## 세그먼트 트리 구현

1. 합을 구하는 함수
2. 값을 업데이트 하는 함수
3. 초기화 함수



## 세그먼트 트리 구현 – 합을 구하는 함수

int sum(node, ns, ne, l, r) : l ~ r 번째 수의 합을 구한다

- node : 트리에서 현재 확인하는 구간(ns ~ ne)의 정보를 갖는 노드의 인덱스
- ns, ne : node가 포함하는 정보의 구간
- l, r : 구하고 싶은 구간



## 세그먼트 트리 구현 – 합을 구하는 함수

int sum(node, ns, ne, l, r) : l ~ r 번째 수의 합을 구한다

- node : 트리에서 현재 확인하는 구간(ns ~ ne)의 정보를 갖는 노드의 인덱스
- ns, ne : node가 포함하는 정보의 구간
- l, r : 구하고 싶은 구간

Case 1.  $[ns, ne] \subseteq [l, r]$                        $\rightarrow$         seg[node] 를 반환



## 세그먼트 트리 구현 – 합을 구하는 함수

int sum(node, ns, ne, l, r) : l ~ r 번째 수의 합을 구한다

- node : 트리에서 현재 확인하는 구간(ns ~ ne)의 정보를 갖는 노드의 인덱스
- ns, ne : node가 포함하는 정보의 구간
- l, r : 구하고 싶은 구간

Case 1.  $[ns, ne] \subseteq [l, r]$  → seg[node] 를 반환

Case 2.  $[ns, ne] \cap [l, r] = \emptyset$  → 0을 반환



## 세그먼트 트리 구현 – 합을 구하는 함수

int sum(node, ns, ne, l, r) : l ~ r 번째 수의 합을 구한다

- node : 트리에서 현재 확인하는 구간(ns ~ ne)의 정보를 갖는 노드의 인덱스
- ns, ne : node가 포함하는 정보의 구간
- l, r : 구하고 싶은 구간

Case 1.  $[ns, ne] \subseteq [l, r]$

→ seg[node] 를 반환

Case 2.  $[ns, ne] \cap [l, r] = \emptyset$

→ 0을 반환

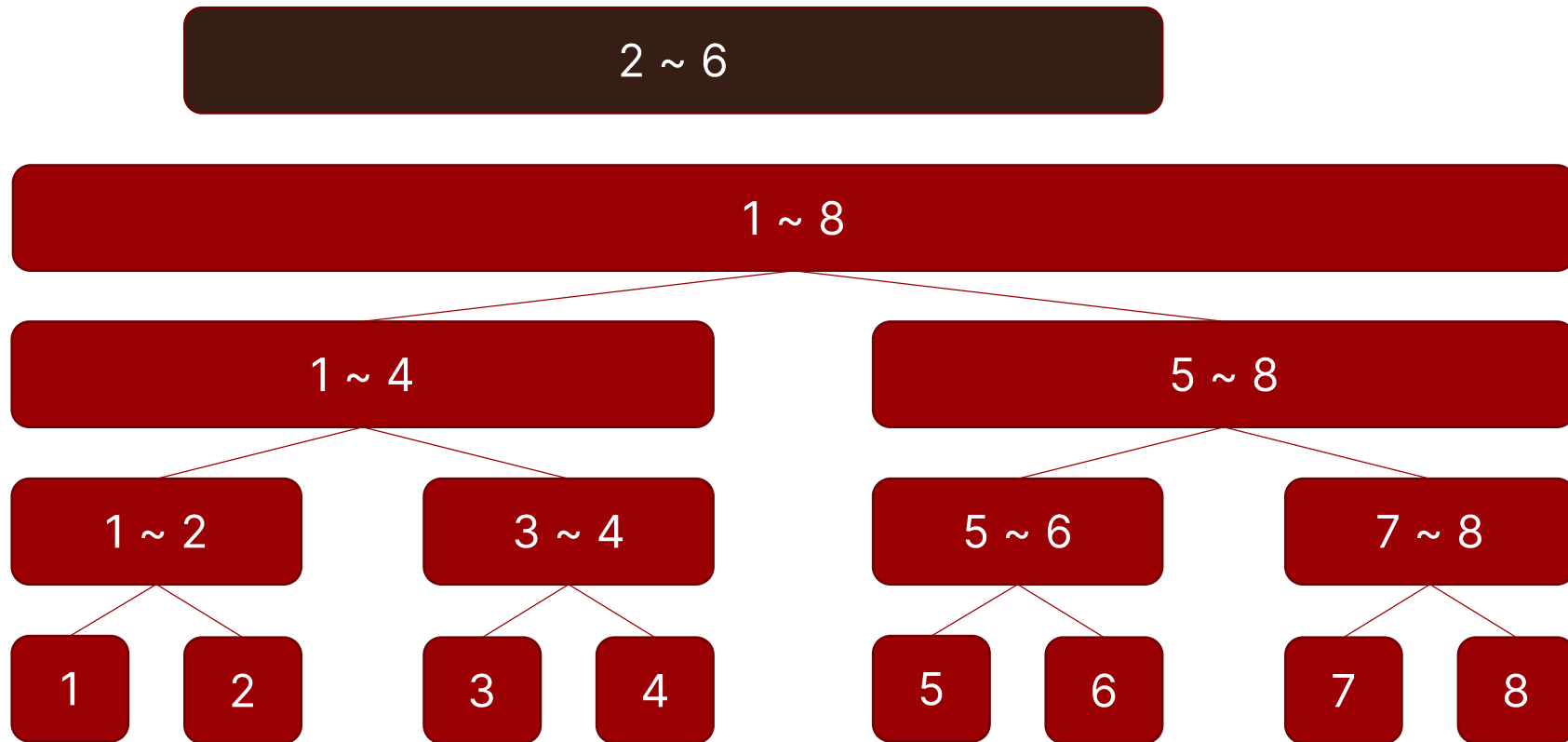
Case 3. Otherwise

→  $\text{sum}(\text{node} * 2, ns, (\text{ns} + \text{ne}) / 2, l, r)$   
+  $\text{sum}(\text{node} * 2 + 1, (\text{ns} + \text{ne}) / 2 + 1, ne, l, r)$  반환



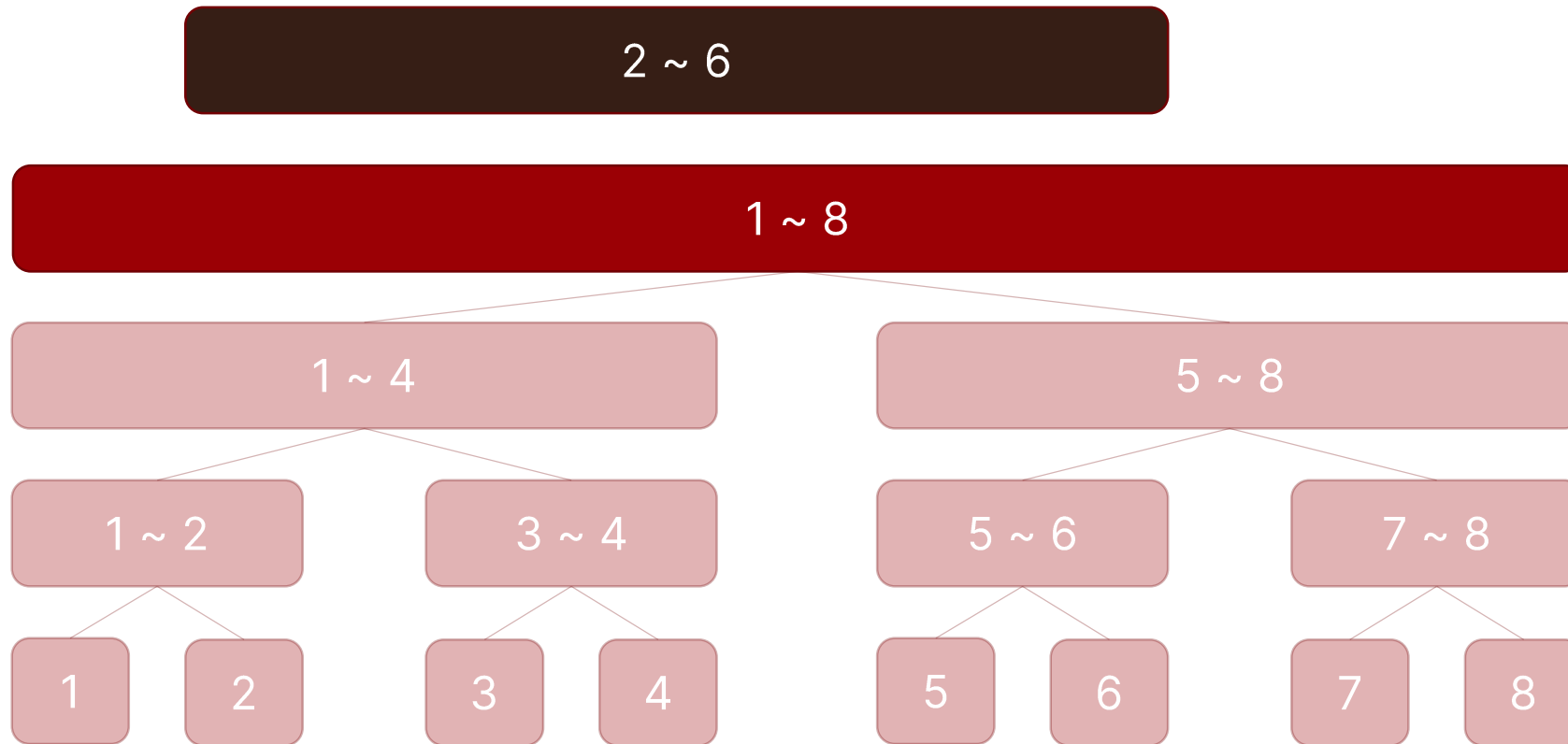


## 세그먼트 트리 구현 – 합을 구하는 함수



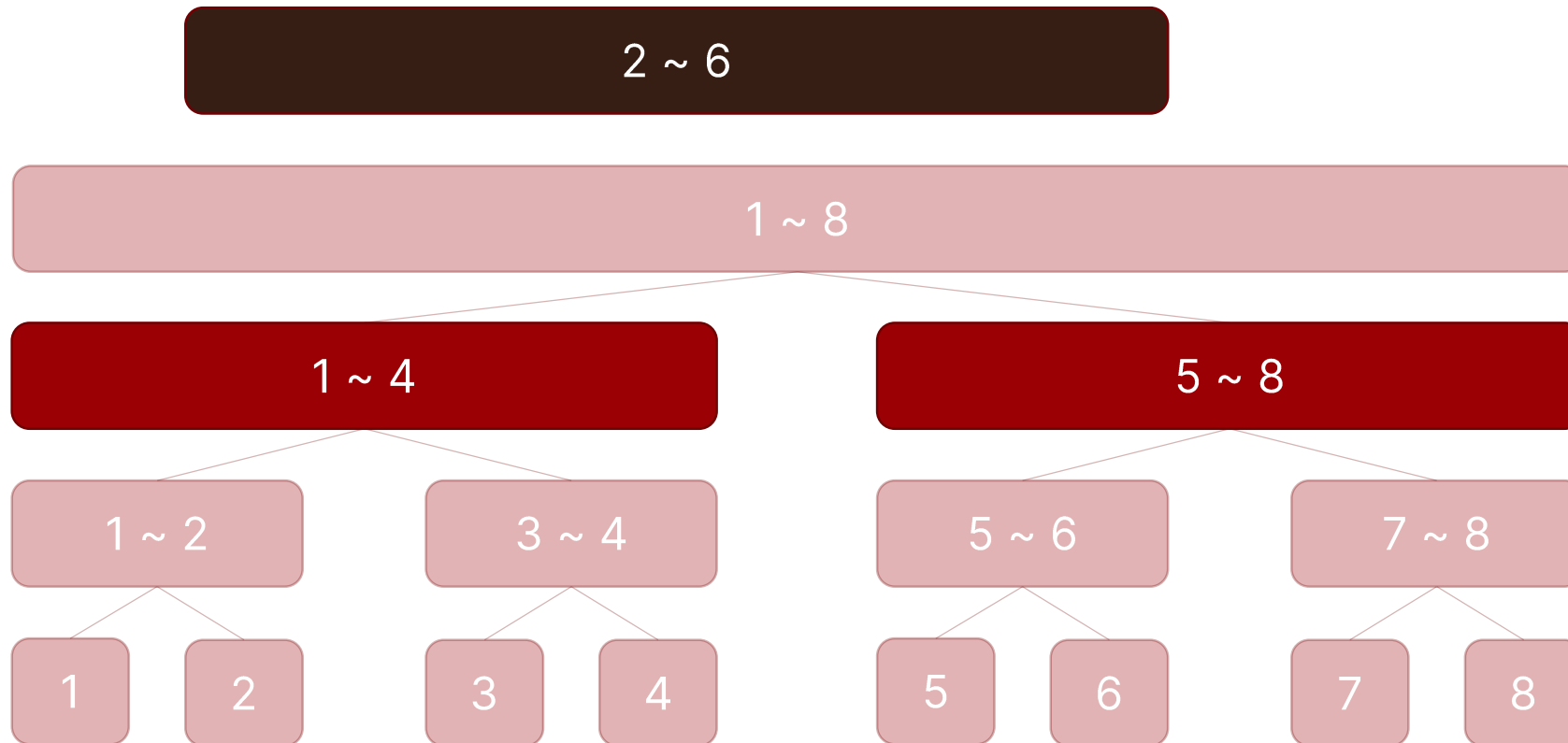


## 세그먼트 트리 구현 – 합을 구하는 함수



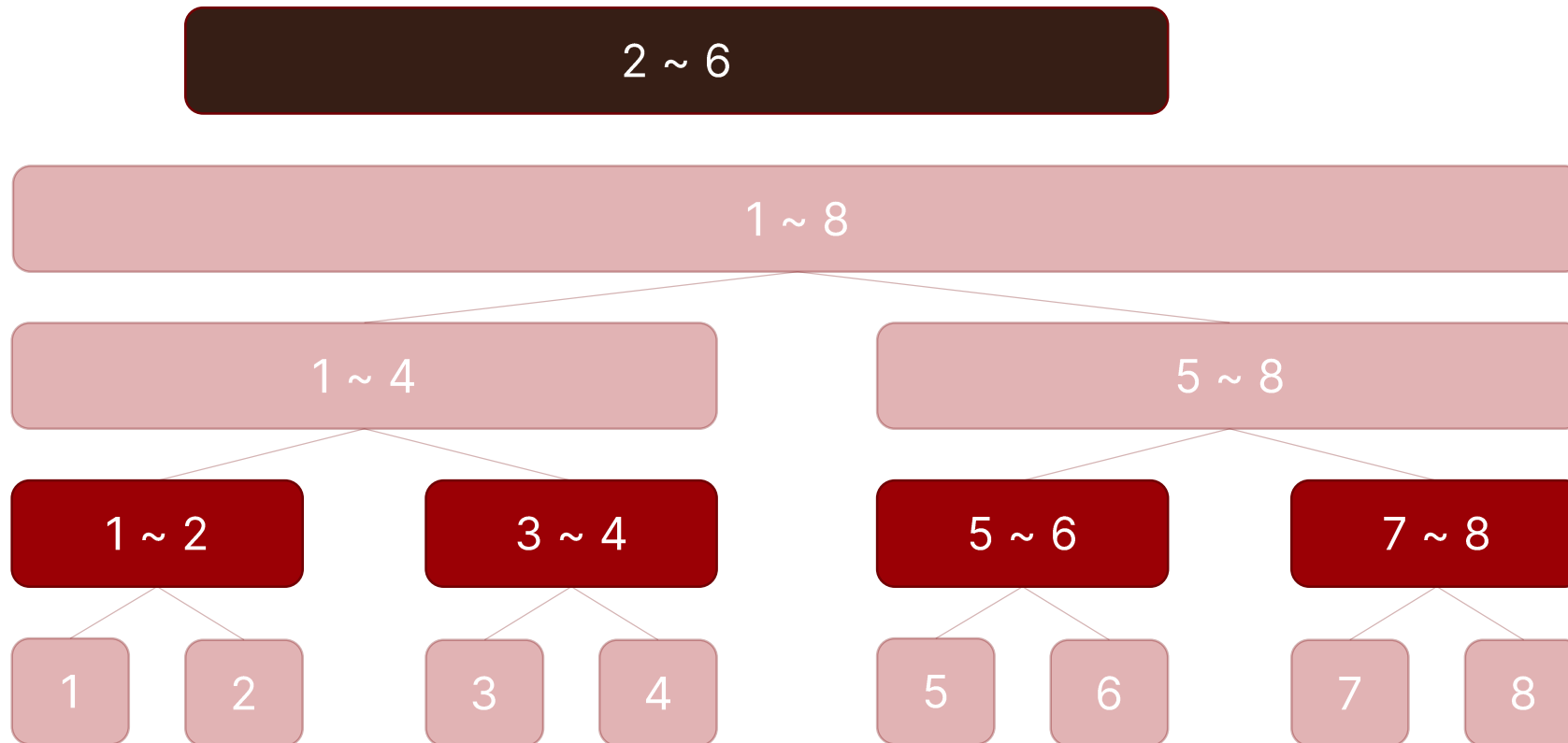


## 세그먼트 트리 구현 – 합을 구하는 함수



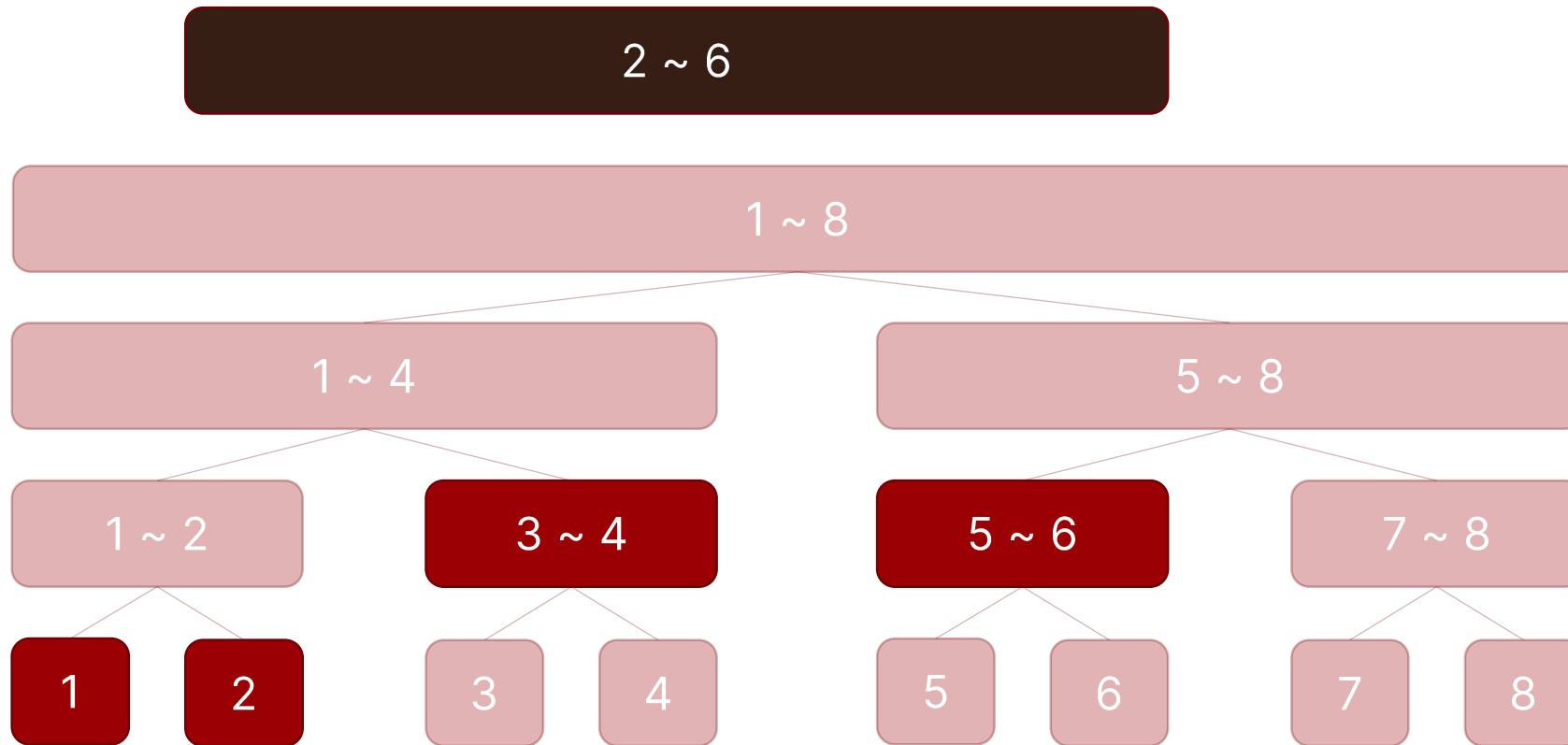


## 세그먼트 트리 구현 – 합을 구하는 함수



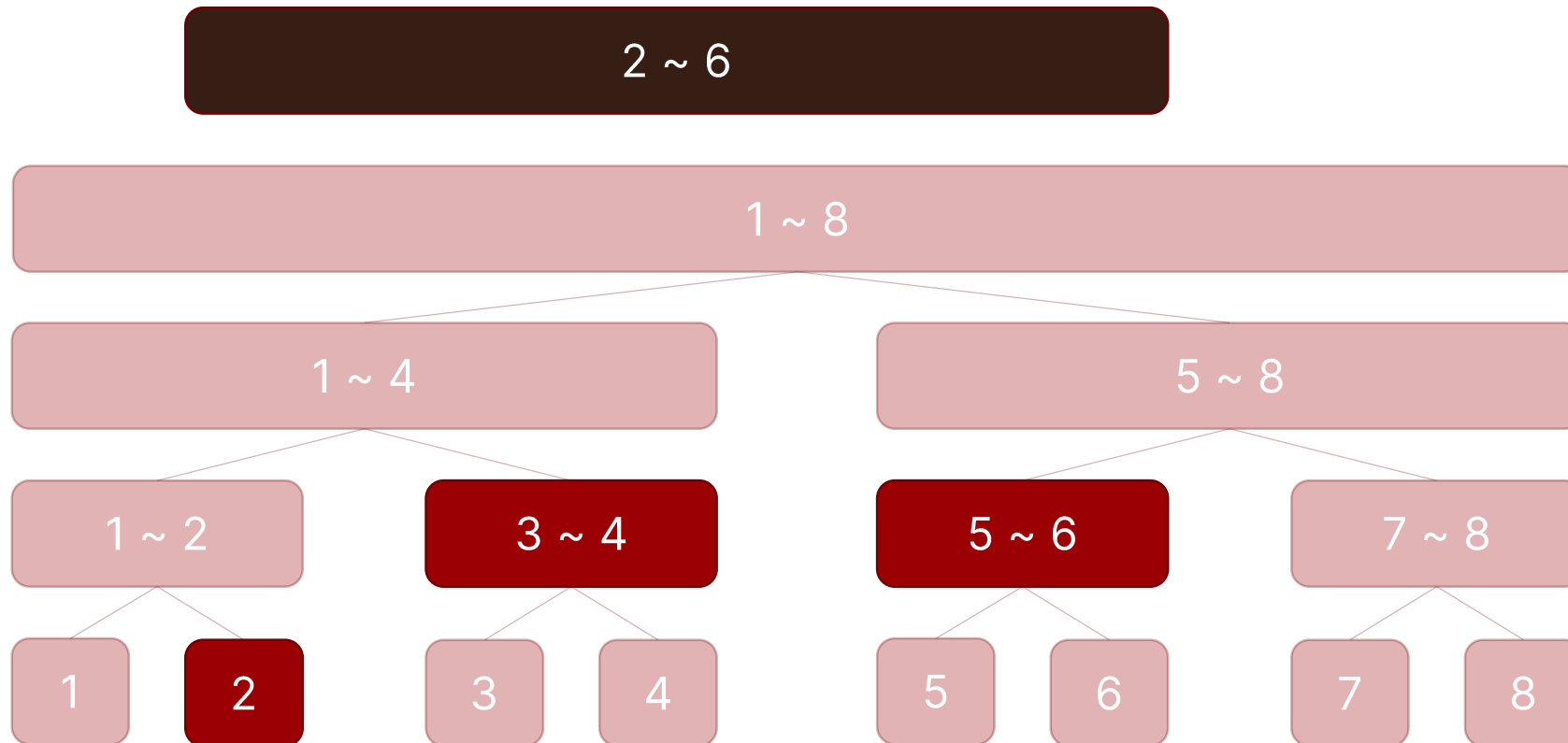


## 세그먼트 트리 구현 – 합을 구하는 함수





## 세그먼트 트리 구현 – 합을 구하는 함수





## 세그먼트 트리 구현 – 합을 구하는 함수

- 21 line  
Case 1.  $[ns, ne] \subseteq [l, r]$
- 22 line  
Case 2.  $[ns, ne] \cap [l, r] = \emptyset$
- 24~26 line  
Case 3. Otherwise

```
20 int sum(int node, int ns, int ne, int l, int r) {  
21     if (l <= ns && ne <= r) return item[node];  
22     else if (ne < l || r < ns) return 0;  
23     else {  
24         int mid = (ns + ne) / 2;  
25         return sum(2 * node, ns, mid, l, r)  
26             + sum(2 * node + 1, mid + 1, ne, l, r);  
27     }  
28 }
```



## 세그먼트 트리 구현 – 값을 업데이트하는 함수

`void update(node, ns, ne, idx, x) : idx 번째 수를 x로 바꾼다`

- `node` : 트리에서 현재 확인하는 구간(`ns ~ ne`)의 정보를 갖는 노드의 인덱스
- `ns, ne` : `node`가 포함하는 정보의 구간
- `idx` : 수열에서 바꿀 수의 인덱스





## 세그먼트 트리 구현 – 값을 업데이트하는 함수

`void update(node, ns, ne, idx, x) : idx 번째 수를 x로 바꾼다`

- `node` : 트리에서 현재 확인하는 구간(`ns ~ ne`)의 정보를 갖는 노드의 인덱스
- `ns, ne` : `node`가 포함하는 정보의 구간
- `idx` : 수열에서 바꿀 수의 인덱스

Step 1. `ns == ne (= leaf node)`       $\rightarrow$       `seg[node] = x`



## 세그먼트 트리 구현 – 값을 업데이트하는 함수

void update(node, ns, ne, idx, x) : idx 번째 수를 x로 바꾼다

- node : 트리에서 현재 확인하는 구간(ns ~ ne)의 정보를 갖는 노드의 인덱스
- ns, ne : node가 포함하는 정보의 구간
- idx : 수열에서 바꿀 수의 인덱스

Step 1.  $ns == ne$  (= leaf node)       $\rightarrow$        $seg[node] = x$

Step 2-1.  $idx \leq (ns+ne)/2$        $\rightarrow$        $update(node*2, ns, (ns+ne)/2, idx, x)$

Step 2-2. otherwise       $\rightarrow$        $update(node*2+1, (ns+ne)/2+1, ne, idx, x)$



## 세그먼트 트리 구현 – 값을 업데이트하는 함수

void update(node, ns, ne, idx, x) : idx 번째 수를 x로 바꾼다

- node : 트리에서 현재 확인하는 구간(ns ~ ne)의 정보를 갖는 노드의 인덱스
- ns, ne : node가 포함하는 정보의 구간
- idx : 수열에서 바꿀 수의 인덱스

Step 1. ns == ne (= leaf node)

→ seg[node] = x

Step 2-1. idx ≤ (ns+ne)/2

→ update(node\*2, ns, (ns+ne)/2, idx, x)

Step 2-2. otherwise

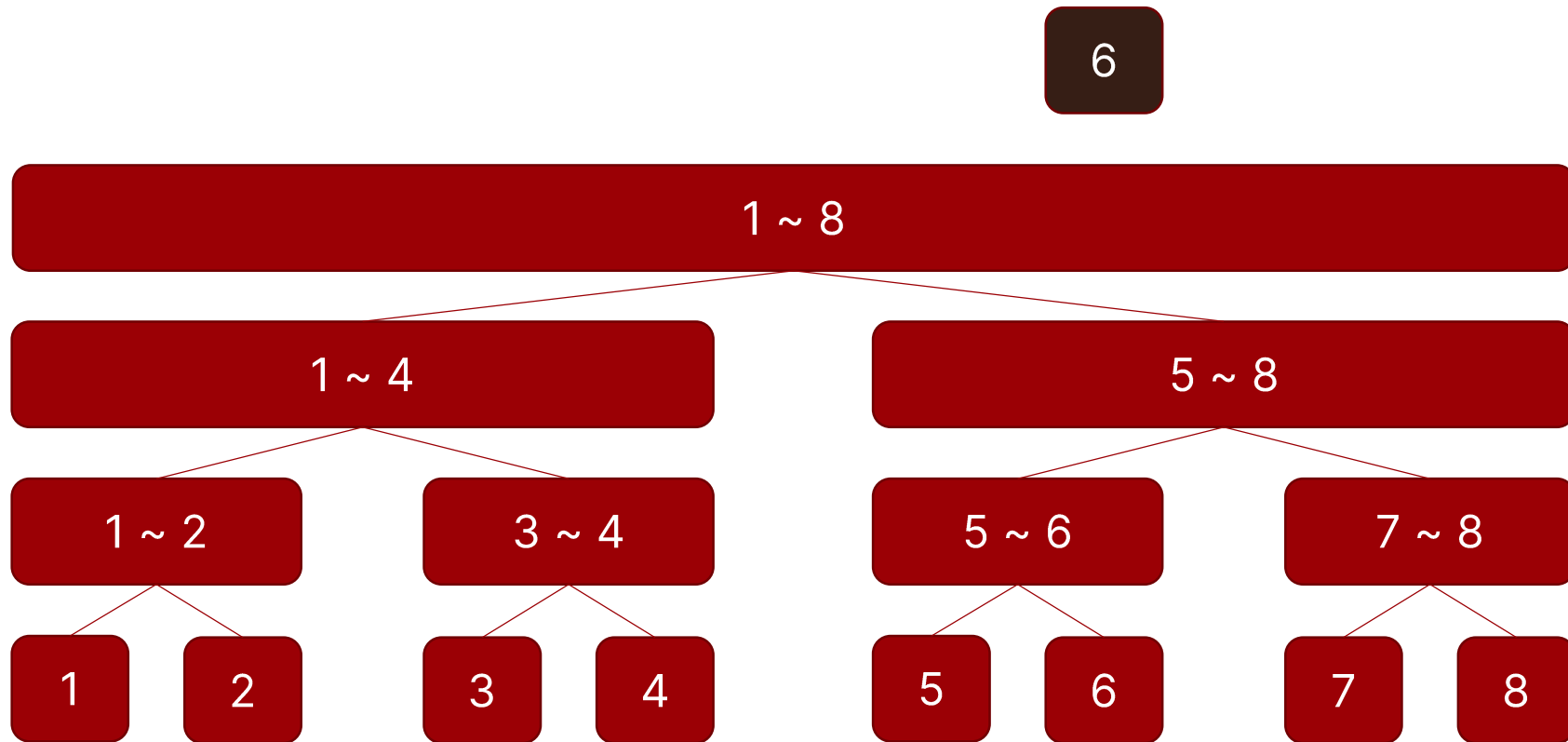
→ update(node\*2+1, (ns+ne)/2+1, ne, idx, x)

Step 3. 자식 갱신 후

→ seg[node] = seg[node\*2] + seg[node\*2+1]

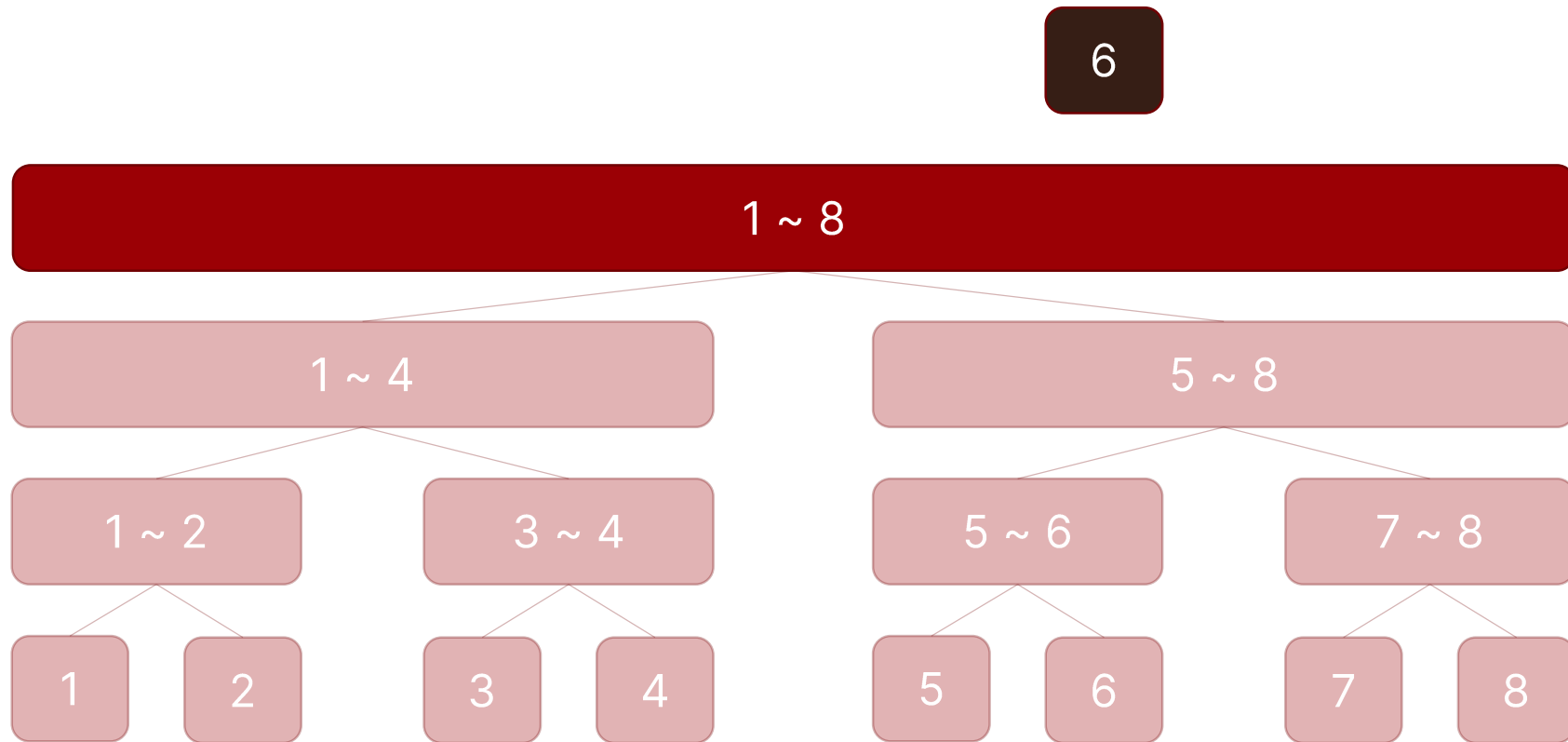


## 세그먼트 트리 구현 – 값을 업데이트하는 함수



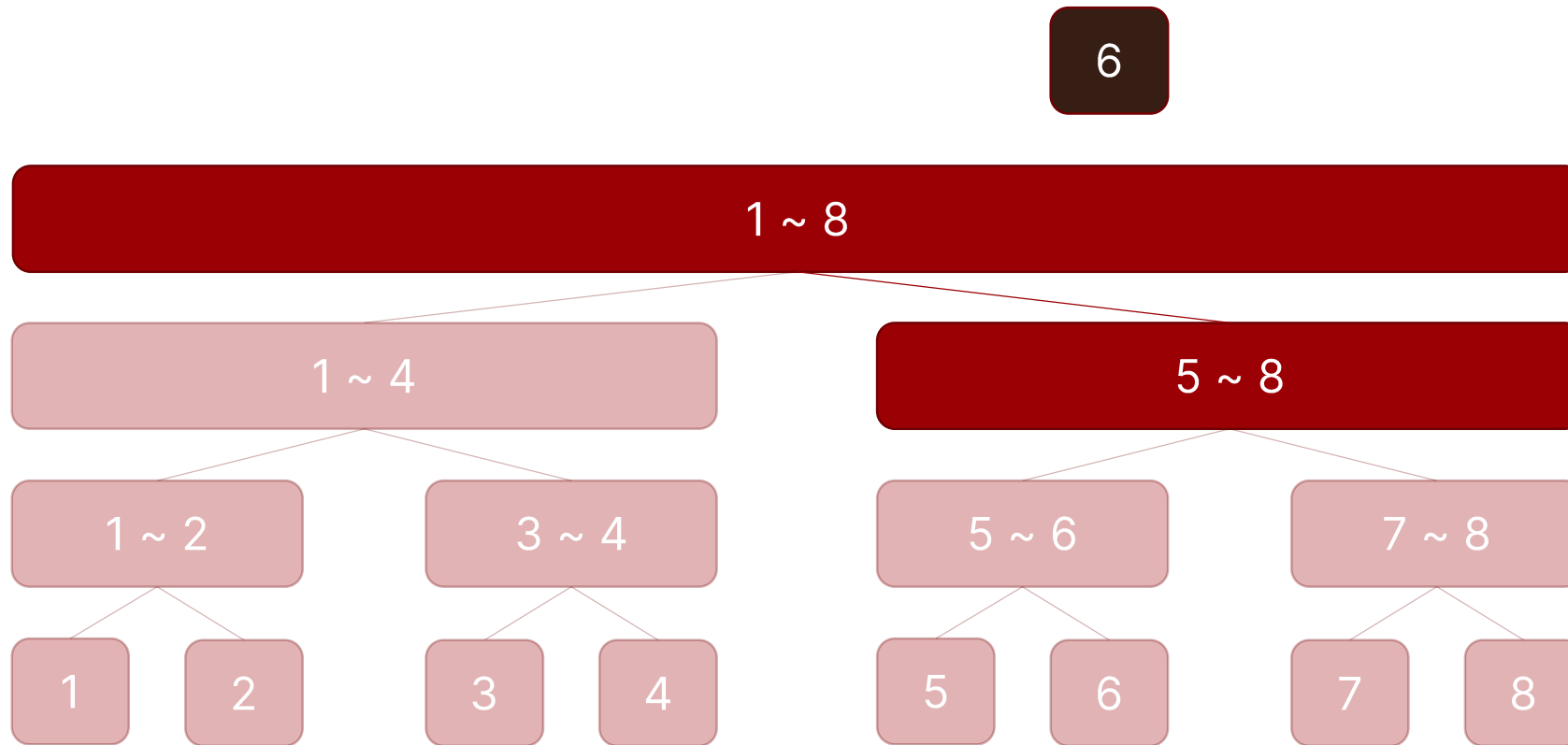


## 세그먼트 트리 구현 – 값을 업데이트하는 함수



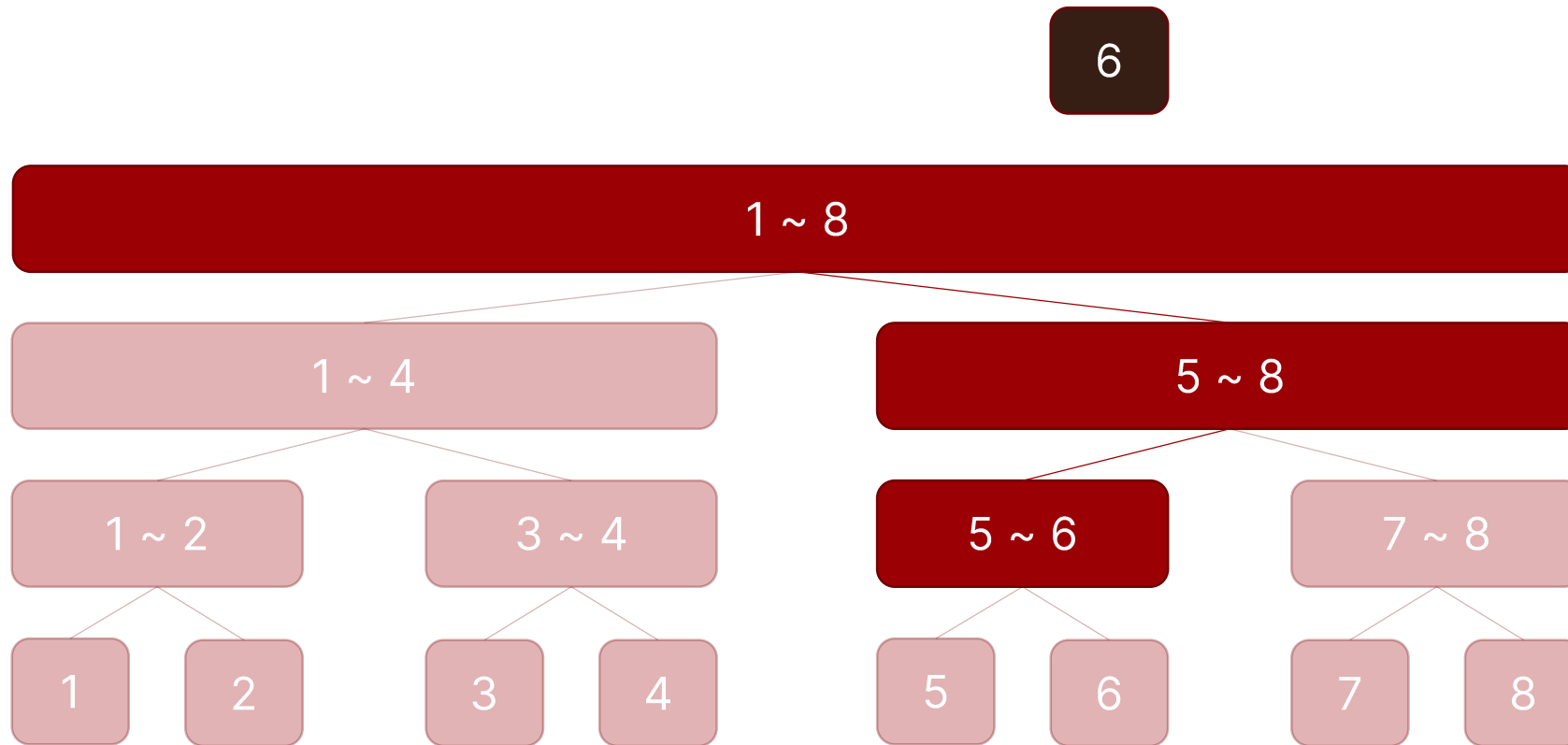


## 세그먼트 트리 구현 – 값을 업데이트하는 함수



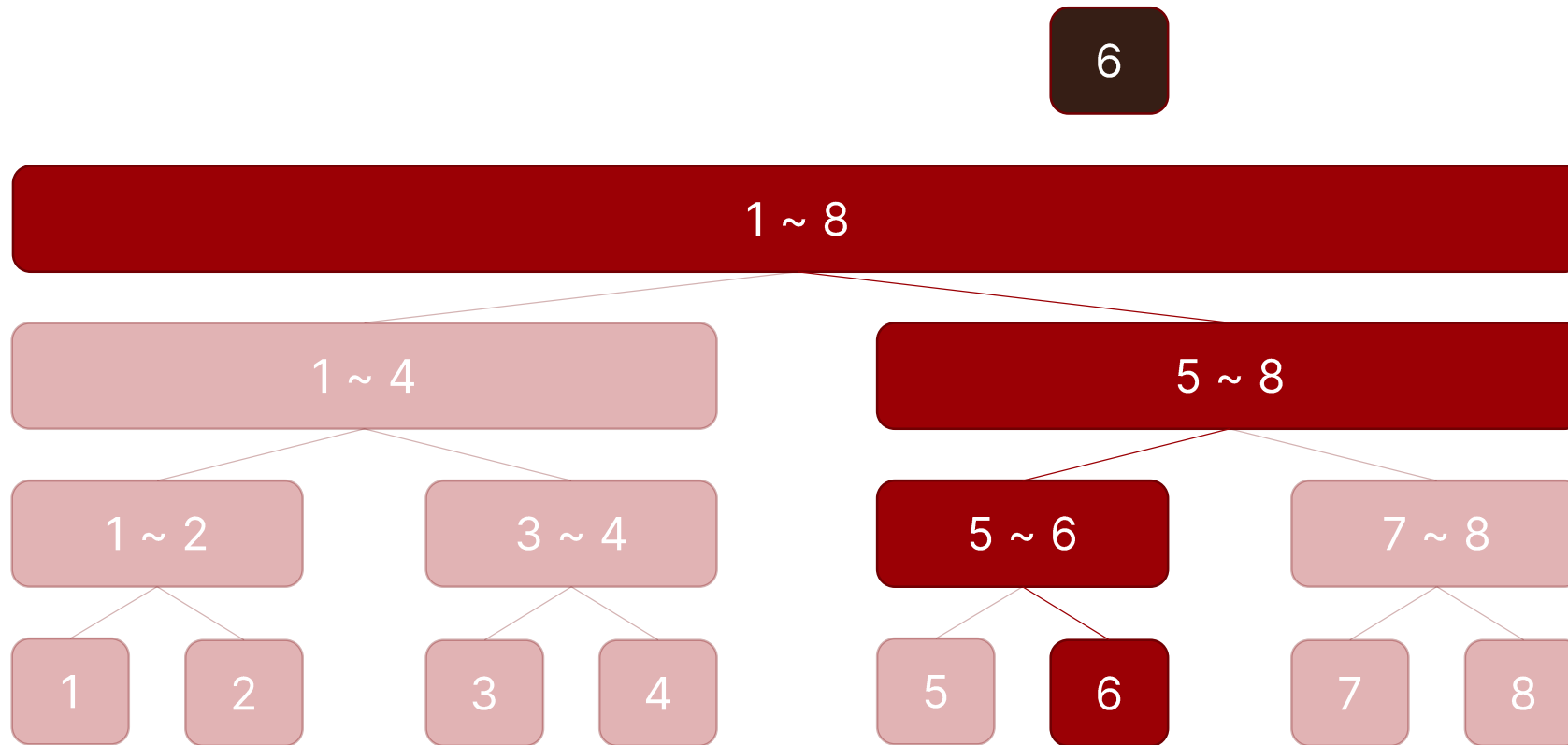


## 세그먼트 트리 구현 – 값을 업데이트하는 함수





## 세그먼트 트리 구현 – 값을 업데이트하는 함수







## 세그먼트 트리 구현 – 값을 업데이트하는 함수

- 30 line  
Step 1.  $ns == ne$  (= leaf node)
- 33 line  
Step 2-1.  $idx \leq (ns+ne)/2$
- 34 line  
Step 2-2.  $idx > (ns+ne)/2$
- 36 line  
Step 3. 자식 갱신 후

```
29 void update(int node, int ns, int ne, int idx, int x) {  
30     if (ns == ne) item[node] = x;  
31     else {  
32         int mid = (ns + ne) / 2;  
33         if (idx <= mid) update(node * 2, ns, mid, idx, x);  
34         else update(node * 2 + 1, mid + 1, ne, idx, x);  
35  
36         item[node] = item[node * 2] + item[node * 2 + 1];  
37     }  
38 }
```



## 세그먼트 트리 구현 – 합을 구하는 함수

int sum(l, r) : l ~ r 번째 수의 합을 구한다

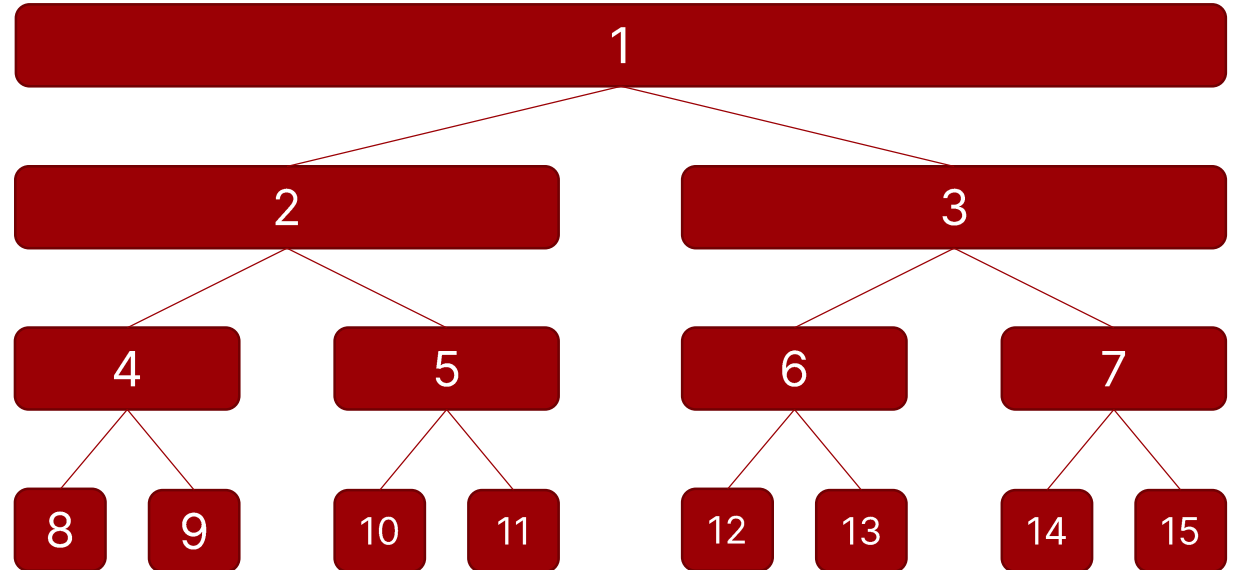
- l, r : 구하고 싶은 구간



## 세그먼트 트리 구현 – 합을 구하는 함수

int sum(l, r) : l ~ r 번째 수의 합을 구한다

- l, r : 구하고 싶은 구간
- $i \% 2 == 0 \rightarrow \text{seg}[i] = \text{왼쪽 자식}$
- $i \% 2 == 1 \rightarrow \text{seg}[i] = \text{오른쪽 자식}$

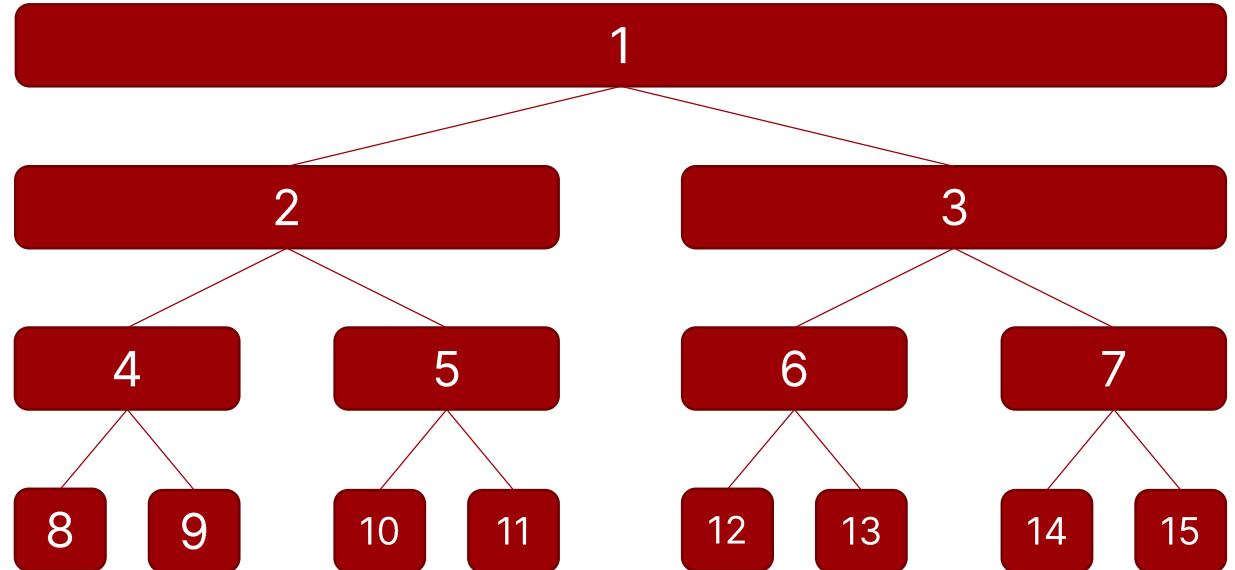




## 세그먼트 트리 구현 – 합을 구하는 함수

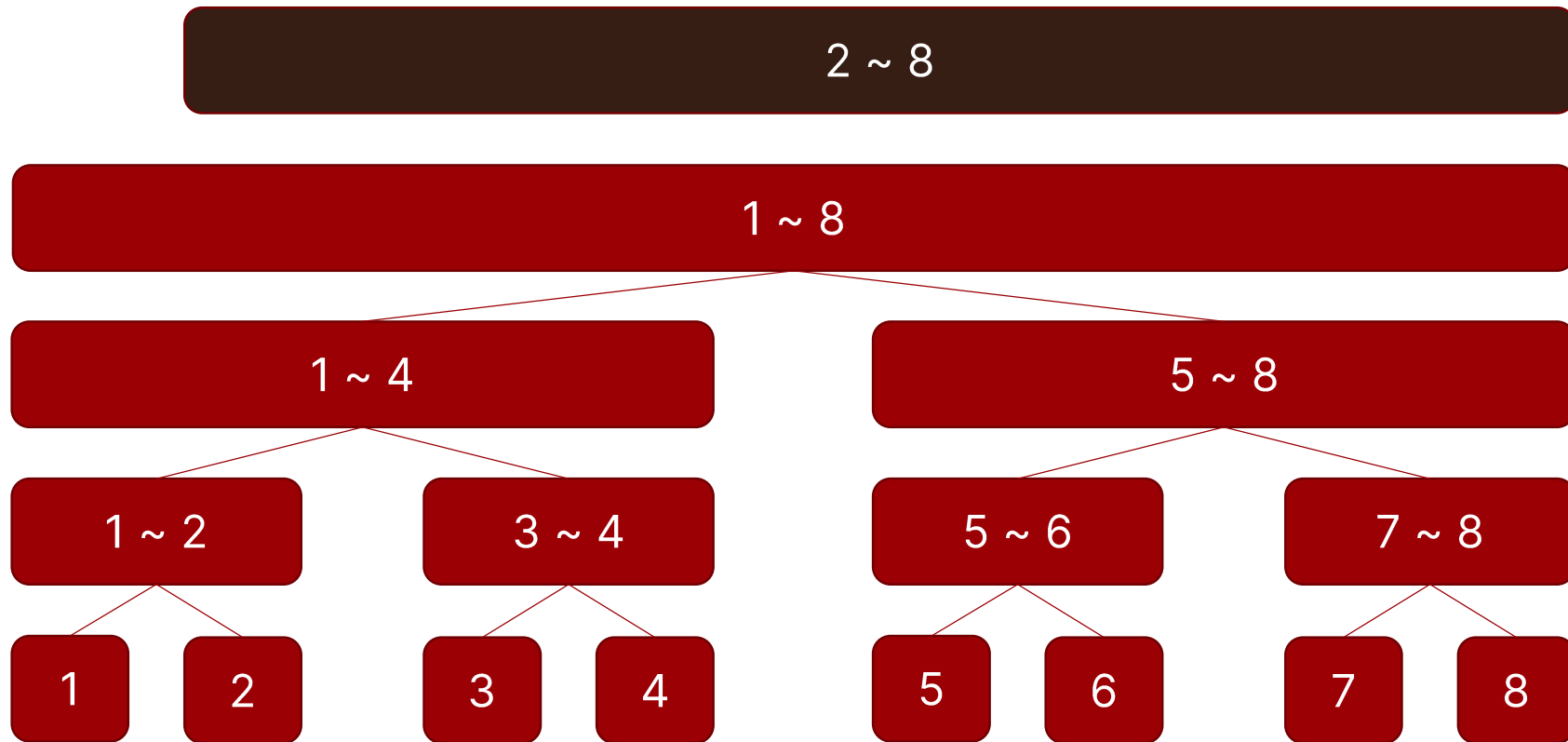
int sum(l, r) : l ~ r 번째 수의 합을 구한다

- l, r : 구하고 싶은 구간
- l → 왼쪽자식 → 부모로 올라가기
- l → 오른쪽자식 → 더하고 넘기기
- r → 오른쪽자식 → 부모로 올라가기
- r → 왼쪽자식 → 더하고 넘기기



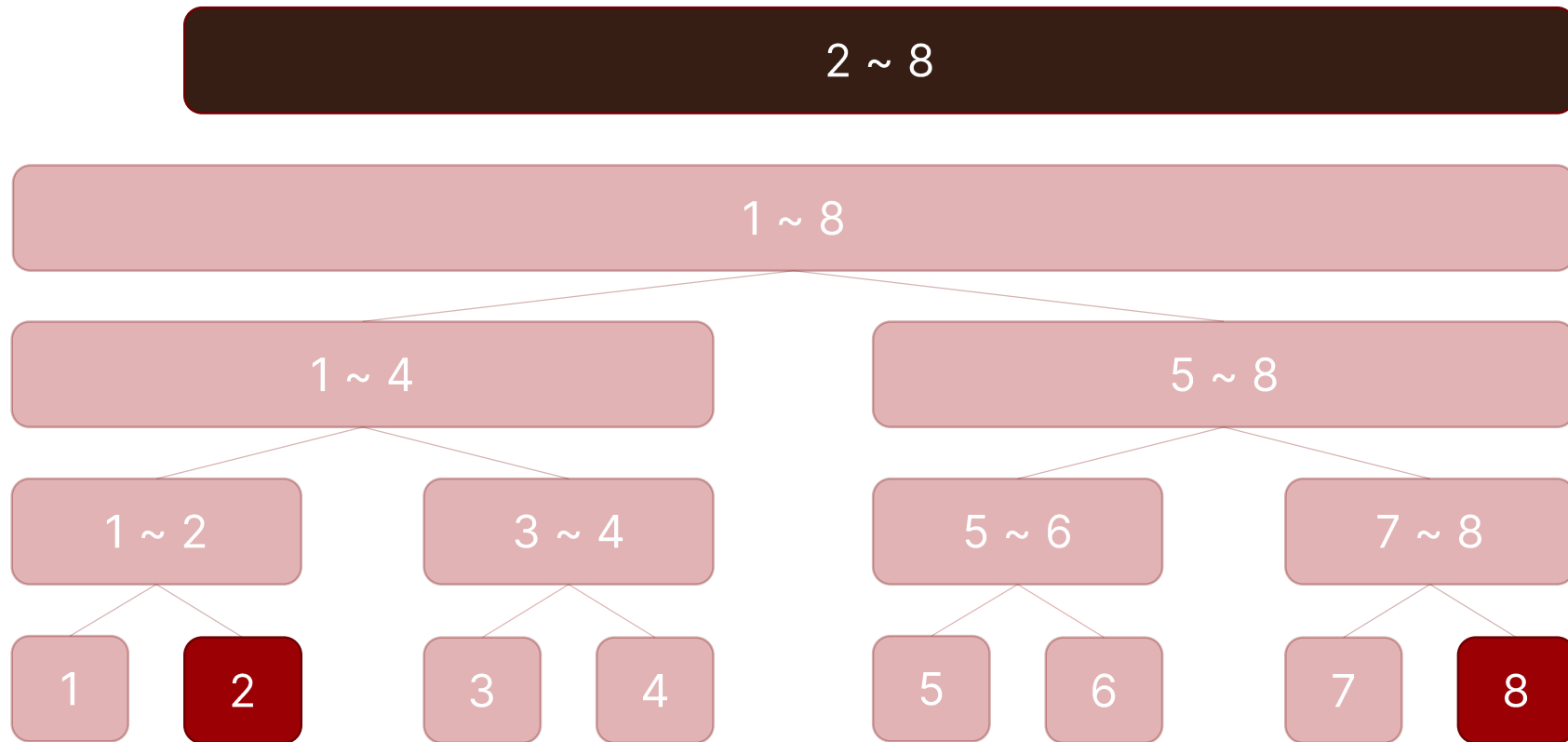


## 세그먼트 트리 구현 – 합을 구하는 함수



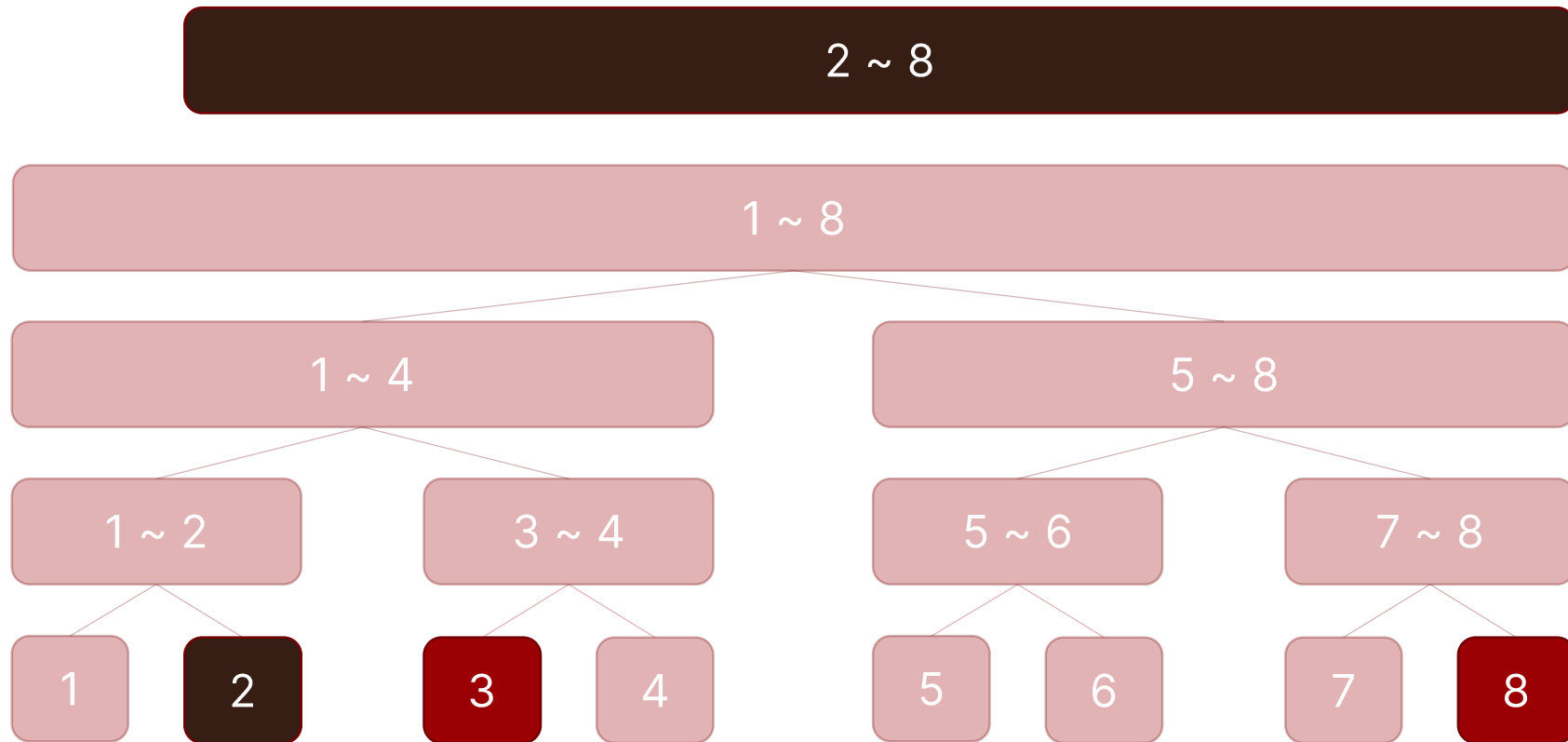


## 세그먼트 트리 구현 – 합을 구하는 함수



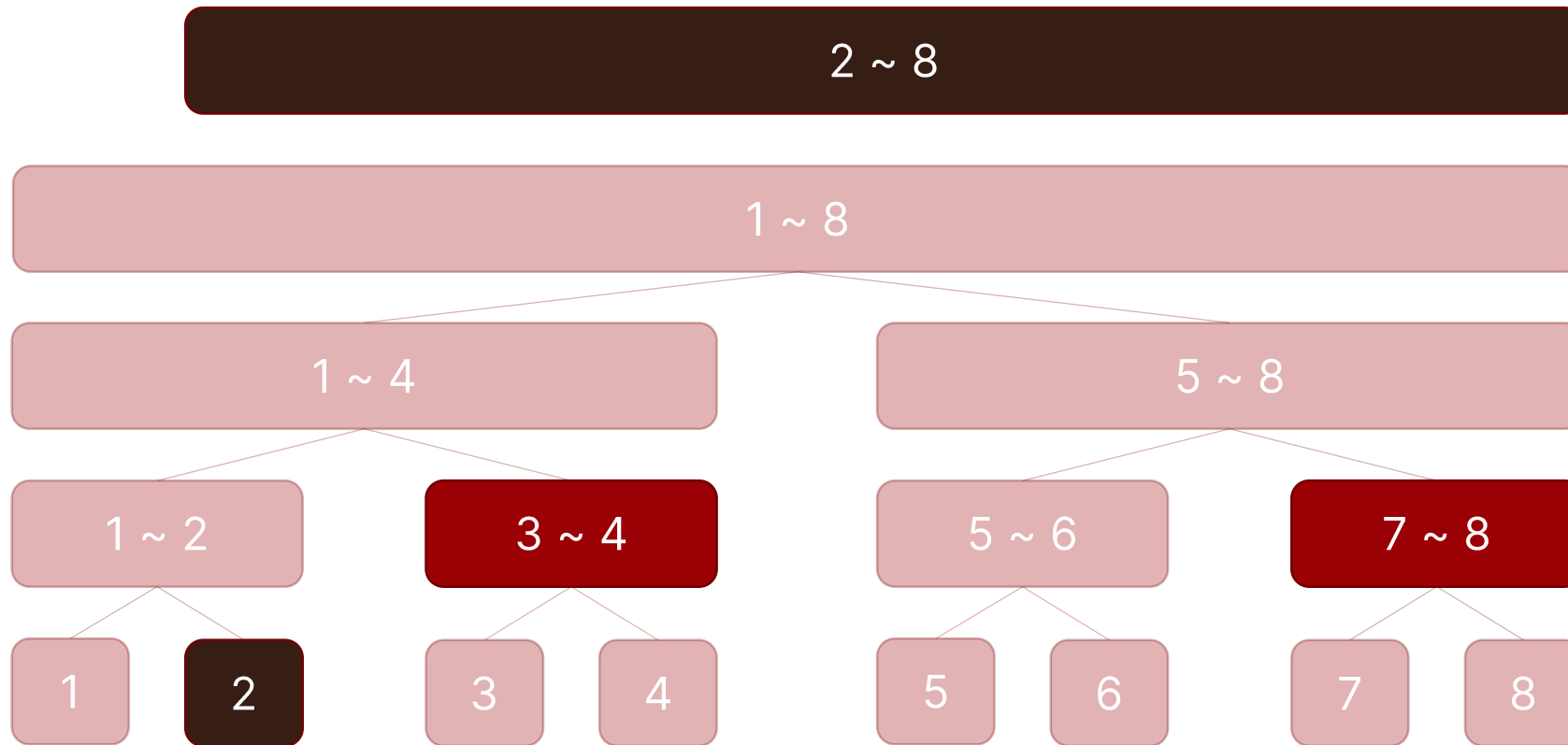


## 세그먼트 트리 구현 – 합을 구하는 함수





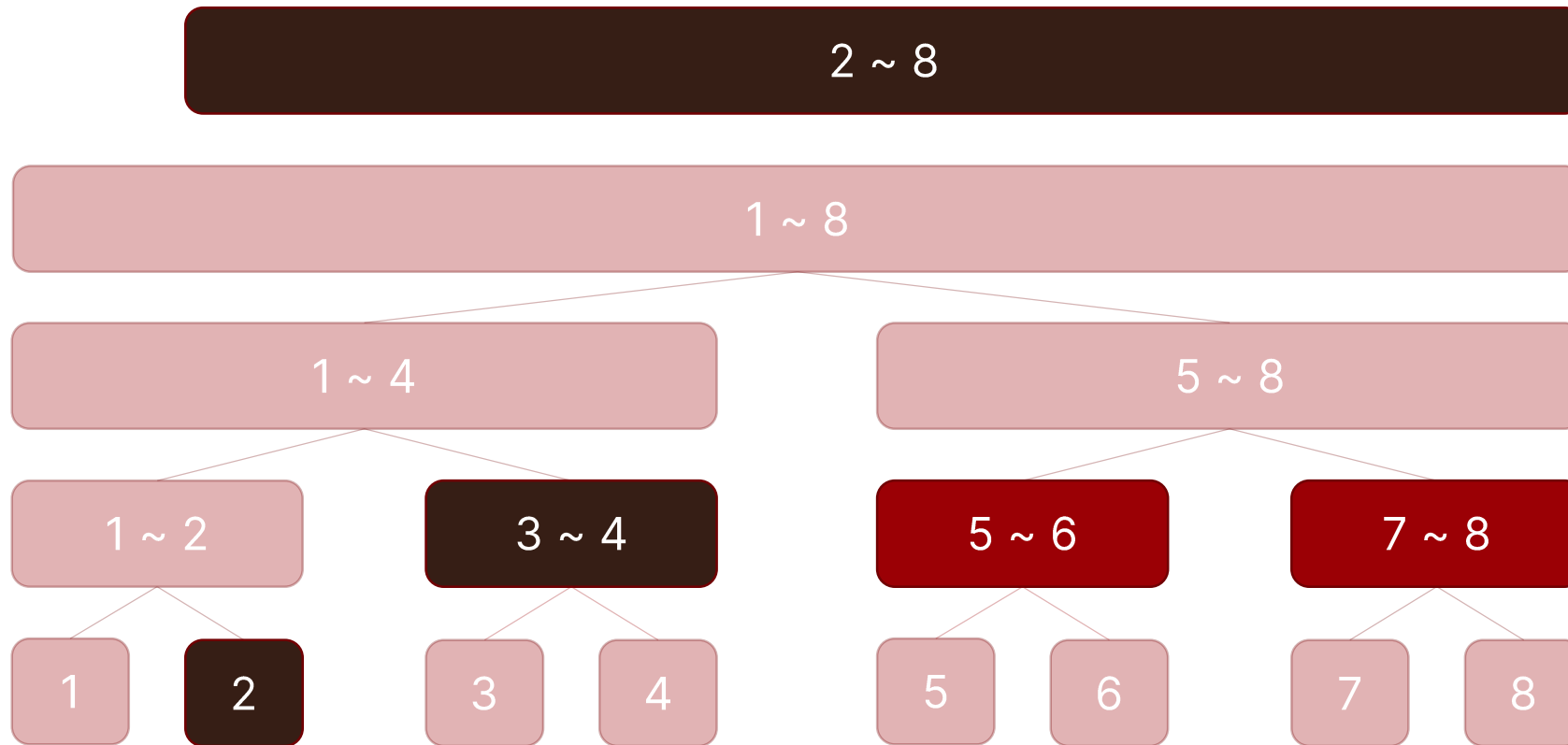
## 세그먼트 트리 구현 – 합을 구하는 함수





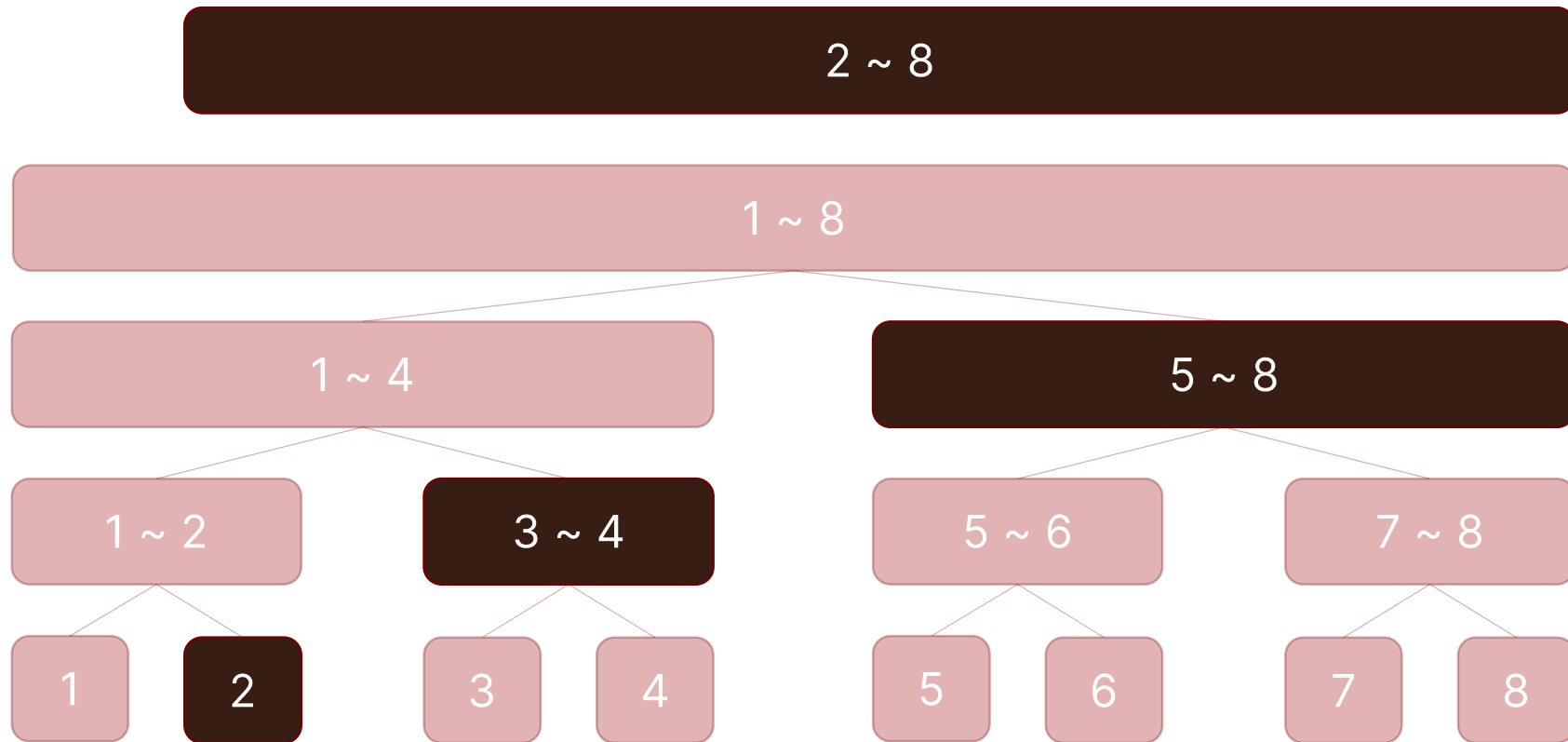


## 세그먼트 트리 구현 - 합을 구하는 함수





## 세그먼트 트리 구현 – 합을 구하는 함수





## 세그먼트 트리 구현 – 합을 구하는 함수

- 22 line  
수열 index → 세그먼트 트리의 index
- 24~25 line  
좌측 경계가 우측자식이면 더해주고 경계를 밀기
- 26~27 line  
우측 경계가 좌측자식이면 더해주고 경계를 밀기
- 28 line  
좌측, 우측 경계를 부모로 올리기

```
20 int sum(int l, int r) {  
21     int res = 0;  
22     l += half, r += half;  
23     while (l <= r) {  
24         if (l % 2)  
25             res += item[l++];  
26         if (!(r % 2))  
27             res += item[r--];  
28         l >>= 1; r >>= 1;  
29     }  
30     return res;  
31 }
```



## 세그먼트 트리 구현 – 값을 업데이트하는 함수

void update(idx, x) : idx 번째 수를 x로 바꾼다

- idx : 수열에서 바꿀 수의 인덱스



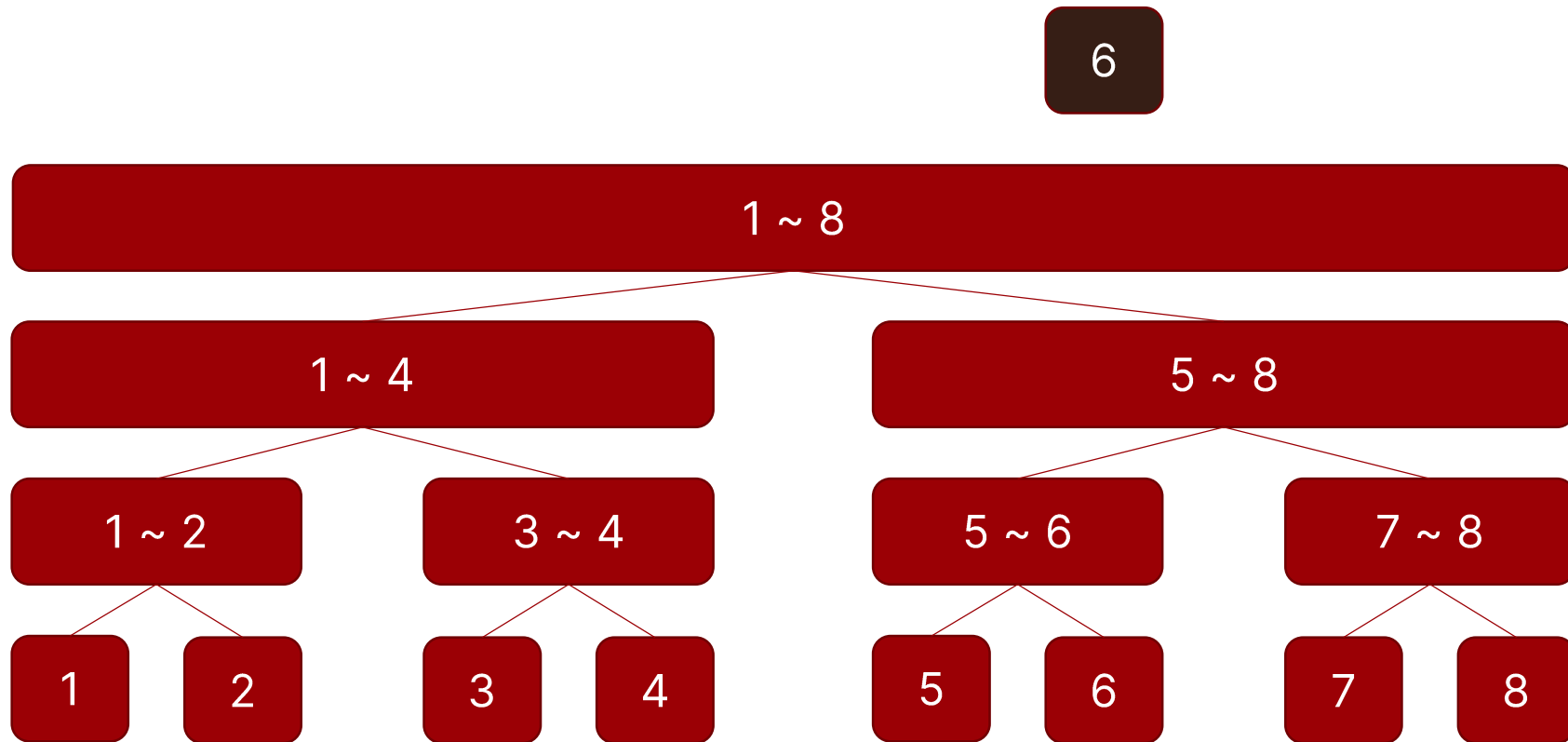
## 세그먼트 트리 구현 – 값을 업데이트하는 함수

void update(idx, x) : idx 번째 수를 x로 바꾼다

- idx : 수열에서 바꿀 수의 인덱스
- idx 번째의 값이 각 노드의 구간 내에 존재하면 모두 수정

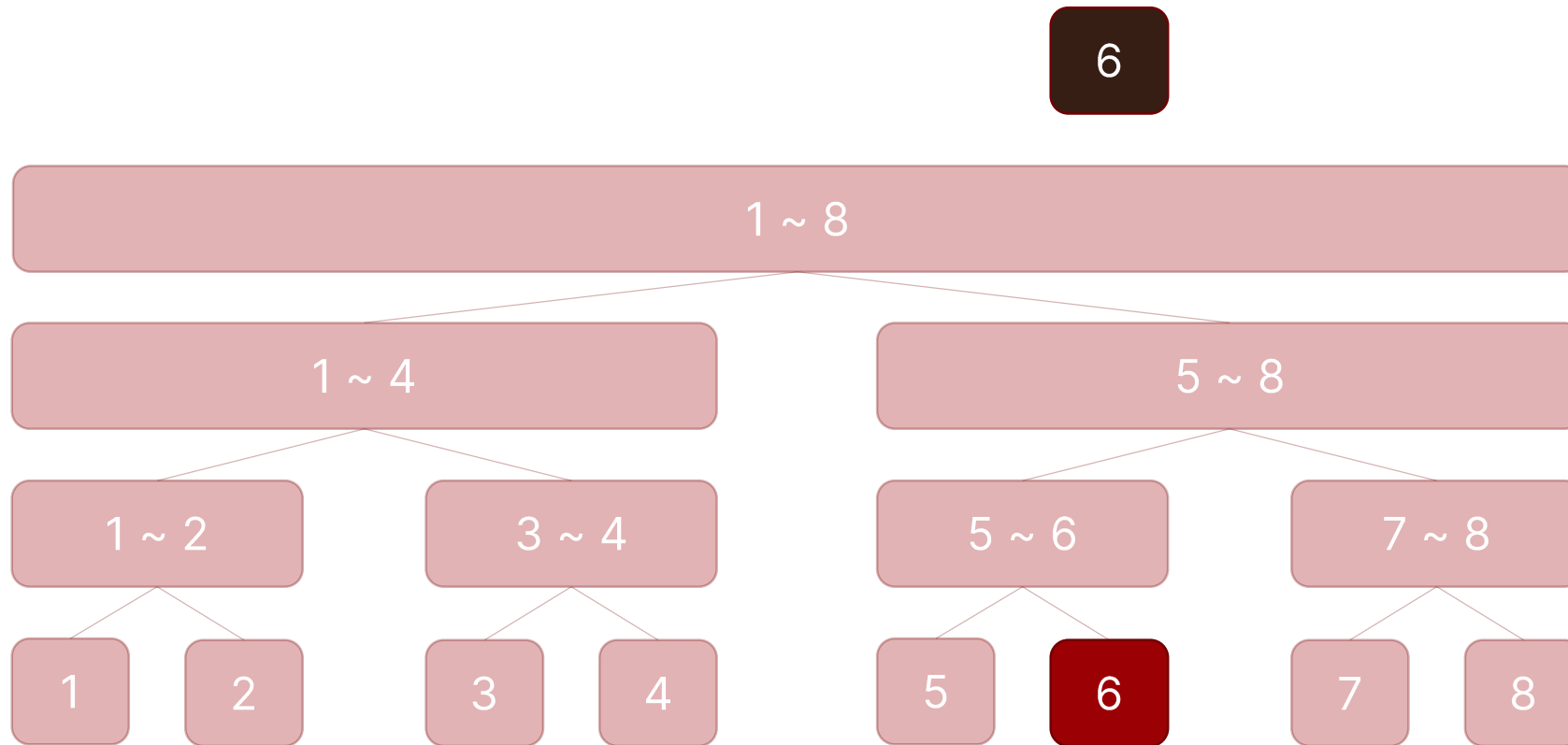


## 세그먼트 트리 구현 – 값을 업데이트하는 함수



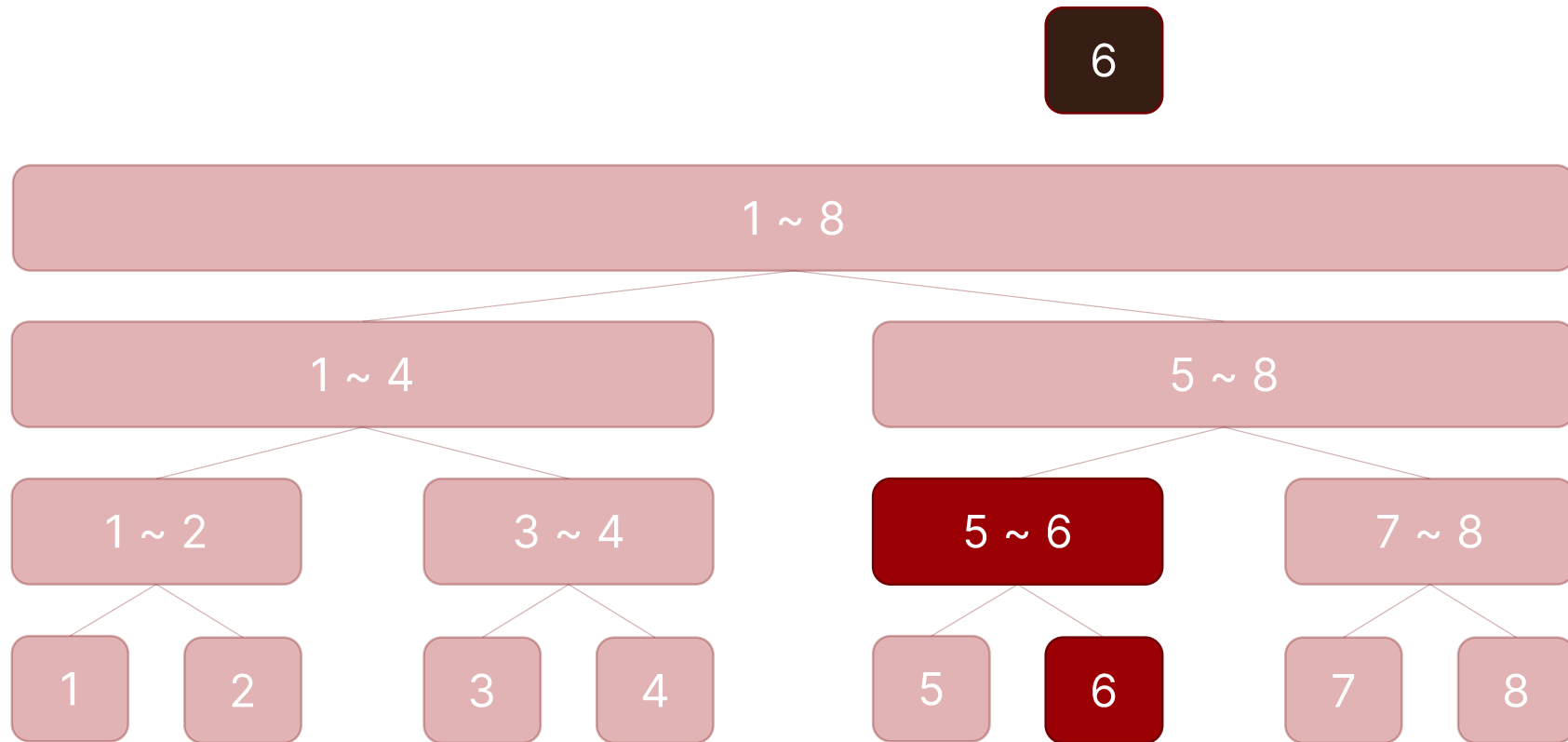


## 세그먼트 트리 구현 – 값을 업데이트하는 함수





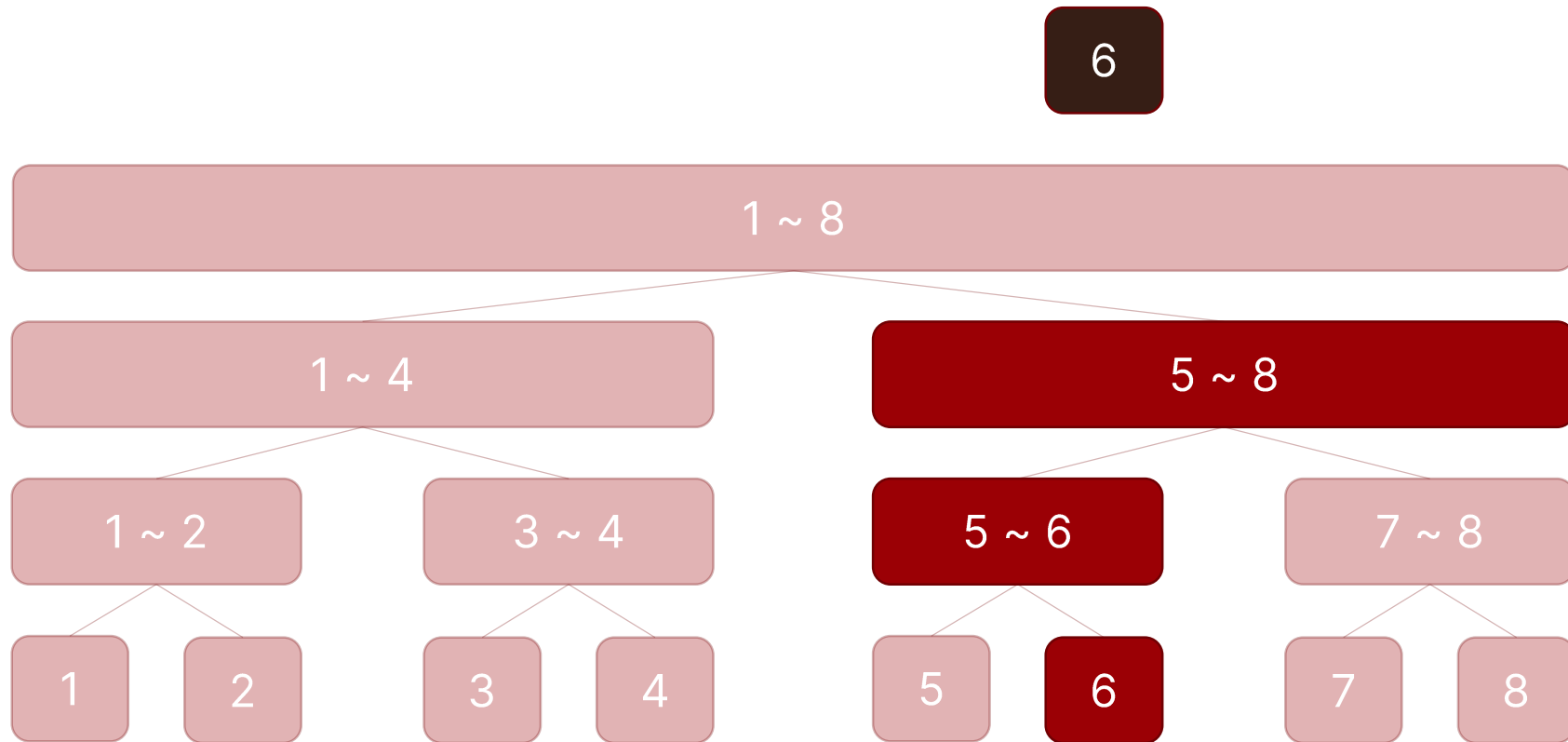
## 세그먼트 트리 구현 – 값을 업데이트하는 함수





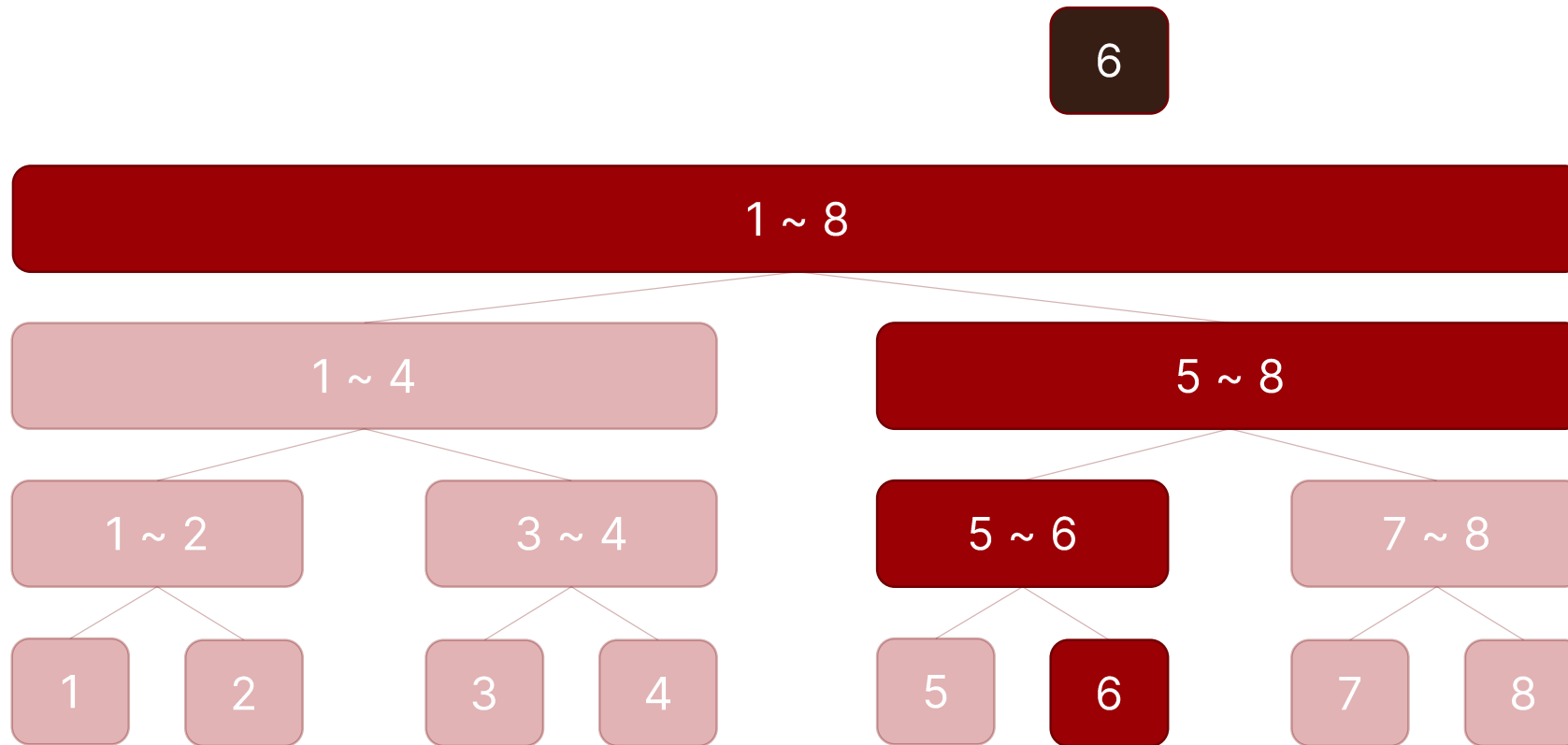


## 세그먼트 트리 구현 – 값을 업데이트하는 함수





## 세그먼트 트리 구현 – 값을 업데이트하는 함수





## 세그먼트 트리 구현 – 값을 업데이트하는 함수

- 33 line  
수열 index → 세그먼트 트리의 index
- 34 line  
세그먼트 트리의 leaf값 수정
- 36~39 line  
idx 번째 수를 포함하는 모든 노드 값 수정

```
32 void update(int idx, int x) {  
33     idx += half;  
34     item[idx] = x;  
35     idx >>= 1;  
36     while (idx) {  
37         item[idx] = item[idx * 2] + item[idx * 2 + 1];  
38         idx >>= 1;  
39     }  
40 }  
41 };
```



## 세그먼트 트리 구현 - 초기화 함수

- 세그먼트 트리의 생김새 = 완전 이진 트리 (또는 포화)
- $n$ 개의 값을 넣고 남은 부분은?



## 세그먼트 트리 구현 - 초기화 함수

- 세그먼트 트리의 생김새 = 완전 이진 트리 (또는 포화)
- $n$ 개의 값을 넣고 남은 부분은?

⇒ 0과 같이 덧셈 결과 값에 영향을 미치지 않는 값으로 설정



## 세그먼트 트리 구현 - 초기화 함수

- 9~10 line
  - $2n$  사이즈의 공간 할당
- 13~14 line
  - n개의 수열 값 저장
- 15~16 line
  - n개의 수열 값 이외의 값에는 0 저장
- 17~18 line
  - 모든 부모 노드는 자식 노드의 합

```
5 struct seg {  
6     int n, half;  
7     vector<int> item;  
8     seg(int n) : n(n) {  
9         for (half = 1; half < n; half <= 1);  
10        item.resize(half * 2);  
11    }  
12    void init(vector<int>& arr) {  
13        for (int i = 0; i < n; ++i)  
14            item[i + half] = arr[i];  
15        for (int i = n; i < half; ++i)  
16            item[i + half] = 0;  
17        for (int i = half - 1; i; --i)  
18            item[i] = item[i * 2] + item[i * 2 + 1];  
19    }
```



## 세그먼트 트리 구현

1. 합을 구하는 함수  $\rightarrow O(\log n)$
2. 값을 업데이트 하는 함수  $\rightarrow O(\log n)$
3. 초기화 함수  $\rightarrow O(n)$



# 세그먼트 트리

- 구간합 이외에도 누적할 수 있는 값이라면 가능
  - 구간곱
  - 최솟값 → RMQ (Range Minimum Query)
  - 최댓값
  - 벡터곱
  - ...
- 구간에 대해서 update → Lazy Propagation
  - Top down 방식으로 가능
  - 고급 스터디에서 다룸





## #2042 구간 합 구하기

- N개의 수, M번의 변경 쿼리, K번의 합쿼리
- 합쿼리가 들어올 때,  $i \sim j$ 의 합을 출력하여라
- $1 \leq N \leq 1,000,000, 1 \leq M, K \leq 10,000$



# LIS (Longest Increasing Subsequence)

- 가장 긴 증가하는 부분수열



# LIS (Longest Increasing Subsequence)

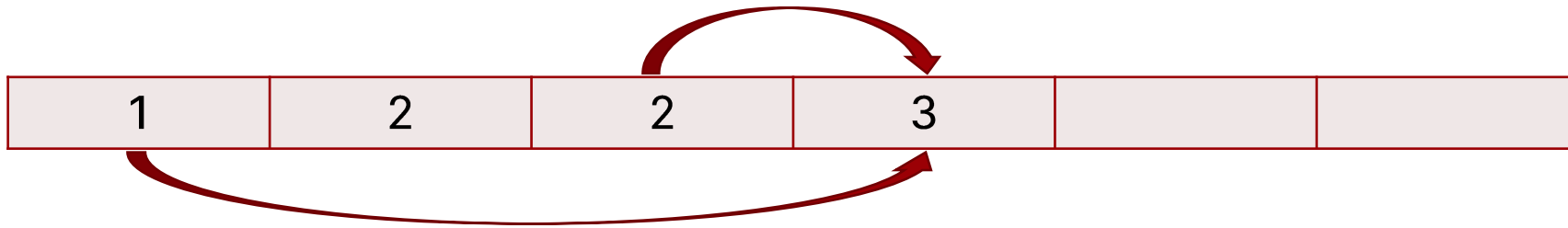
- 가장 긴 증가하는 부분수열
- Dynamic programming  $\rightarrow O(n^2)$
- Lower bound  $\rightarrow O(n * \log n)$



# LIS (Longest Increasing Subsequence)

seq = 

10	30	15	25	20	50
----	----	----	----	----	----



LIS = 

1	2	2	3	3	4
---	---	---	---	---	---




## LIS (Longest Increasing Subsequence)

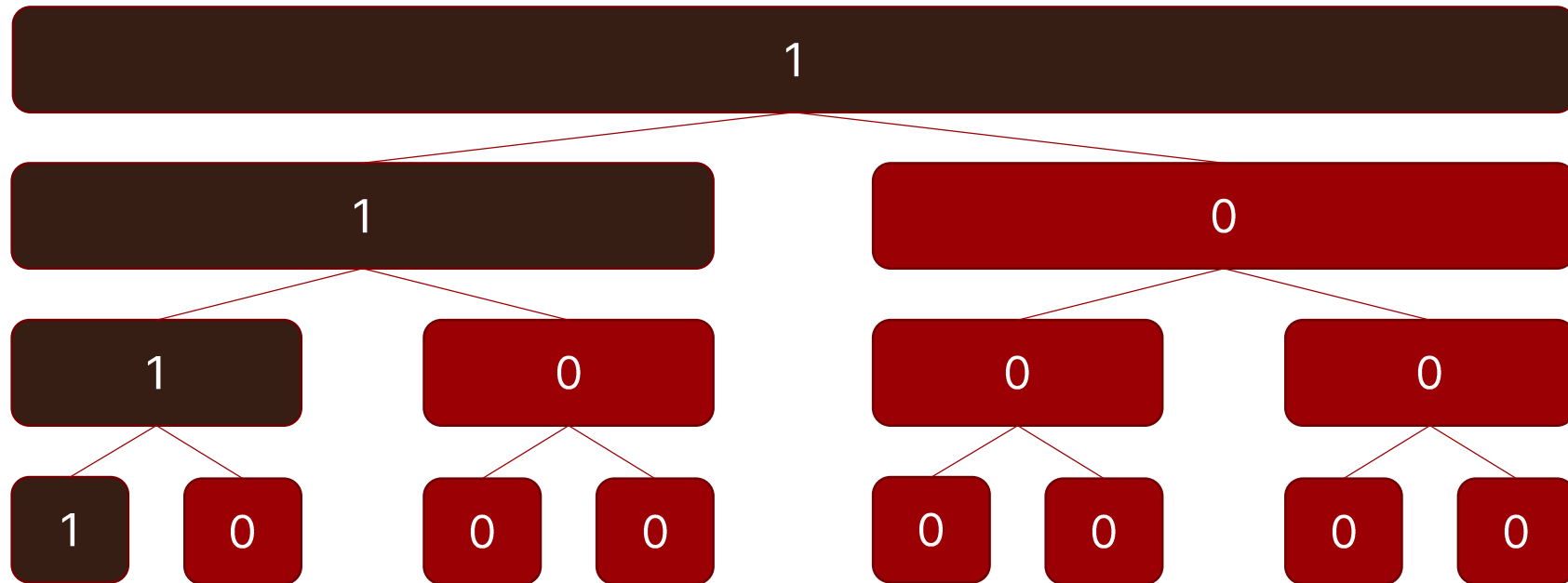
- 현재 수보다 앞에 있고, 작은 값을 갖는 수로 끝나는 LIS 길이 중에서 비교
- 값이 작은 것부터 LIS 갱신
  - 현재 index보다 작은 index들의 LIS중 최대를 뽑아내기 (구간의 최대)
  - 구한 최대 LIS 길이 + 1로 세그먼트 트리에 update



# LIS (Longest Increasing Subsequence)




10	30	15	25	20	50
1					



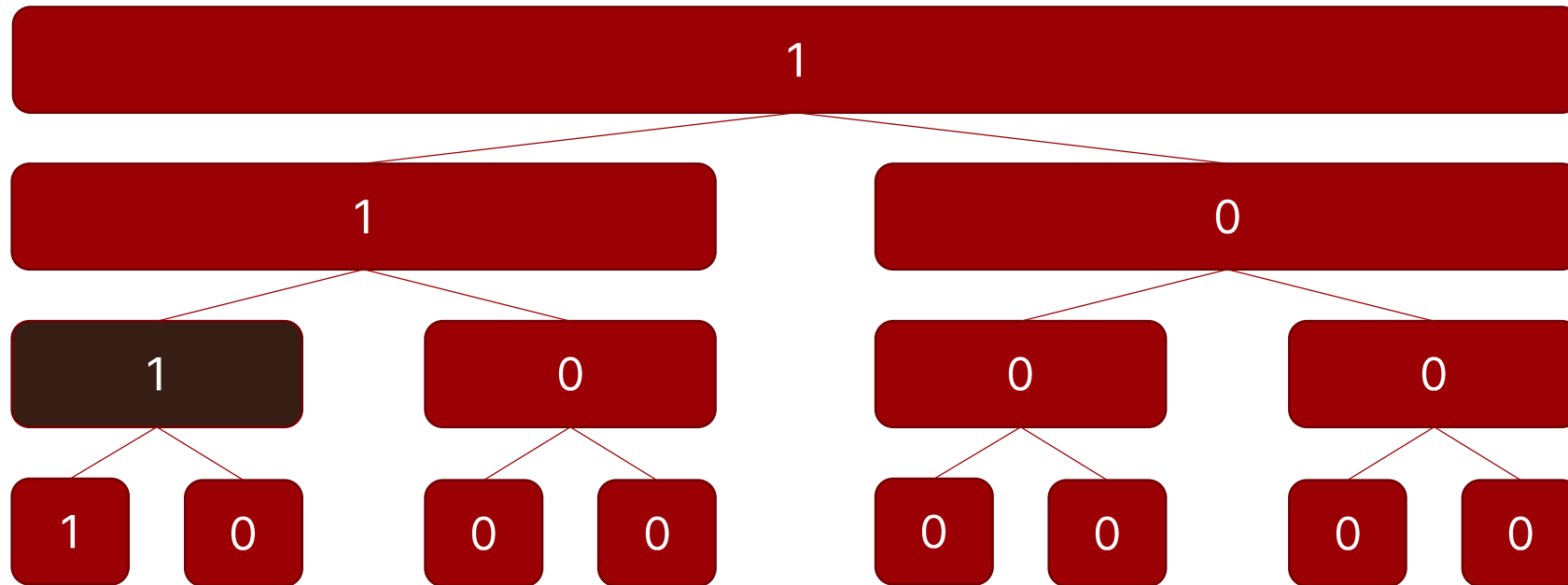


## LIS (Longest Increasing Subsequence)

$$\max(0, 1) = 1$$




10	30	15	25	20	50
1					



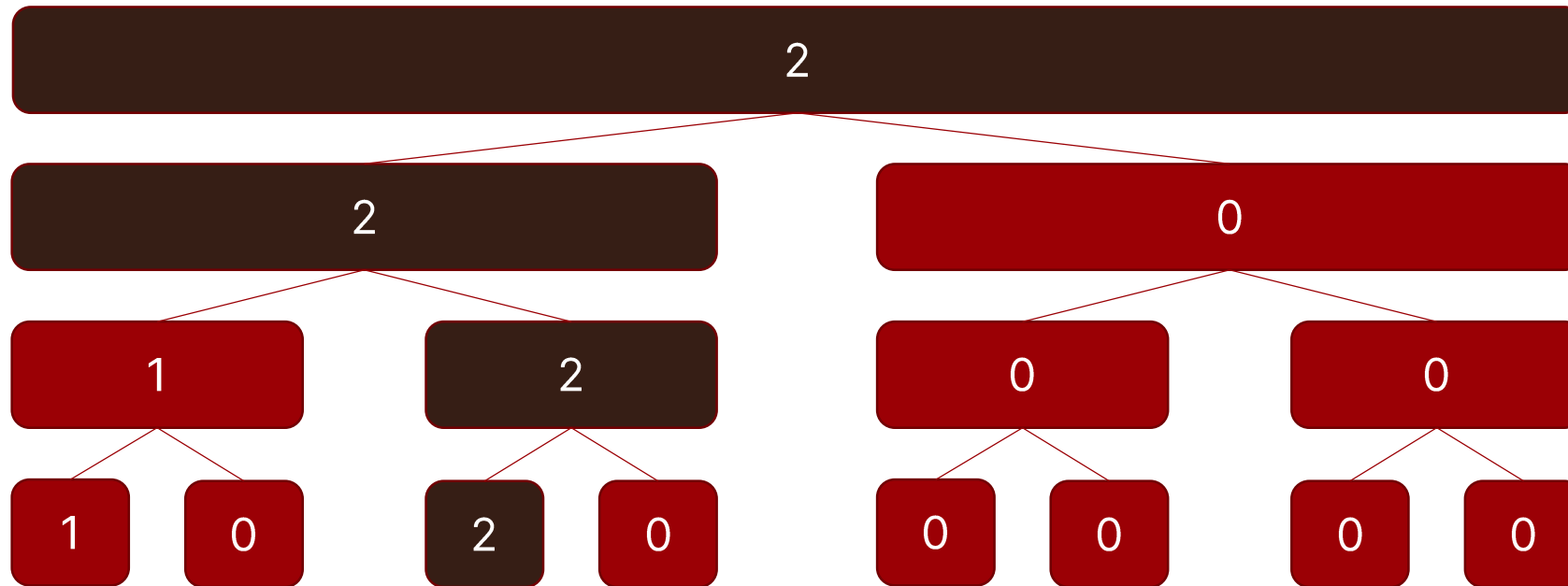


## LIS (Longest Increasing Subsequence)

update(2,2)



10	30	15	25	20	50
1		2			






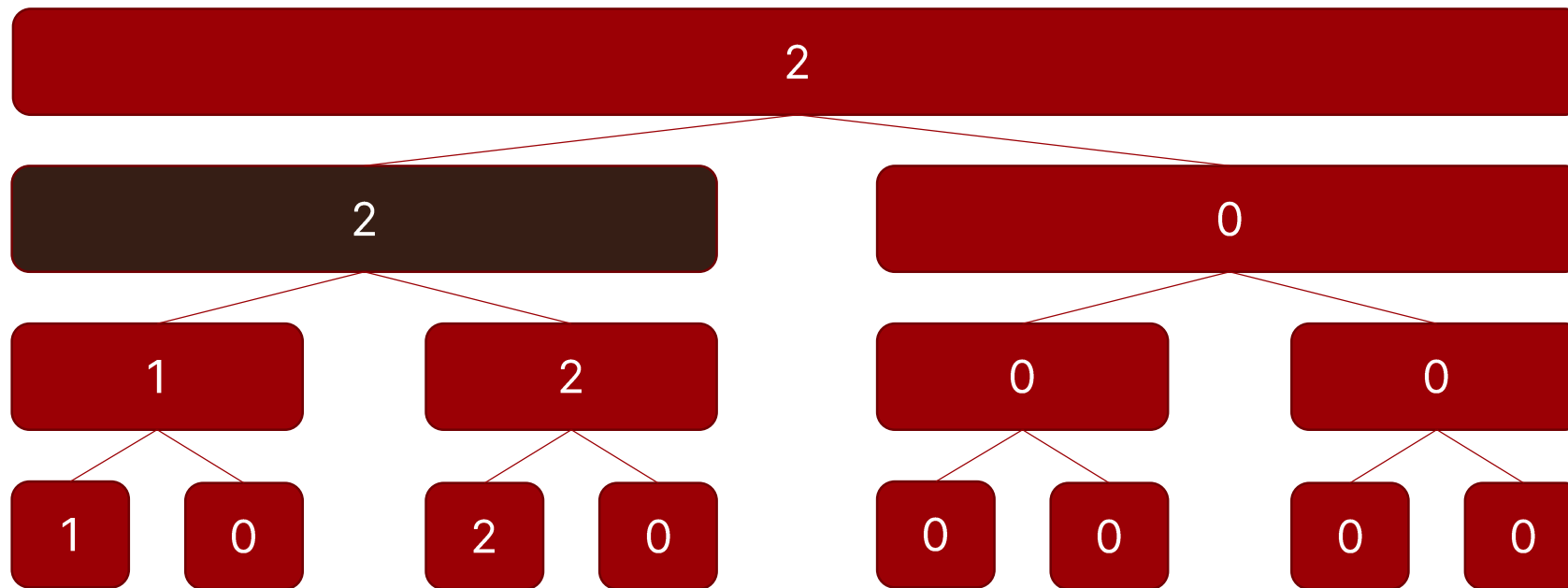


# LIS (Longest Increasing Subsequence)

$$\max(0, 3) = 2$$




10	30	15	25	20	50
1		2			



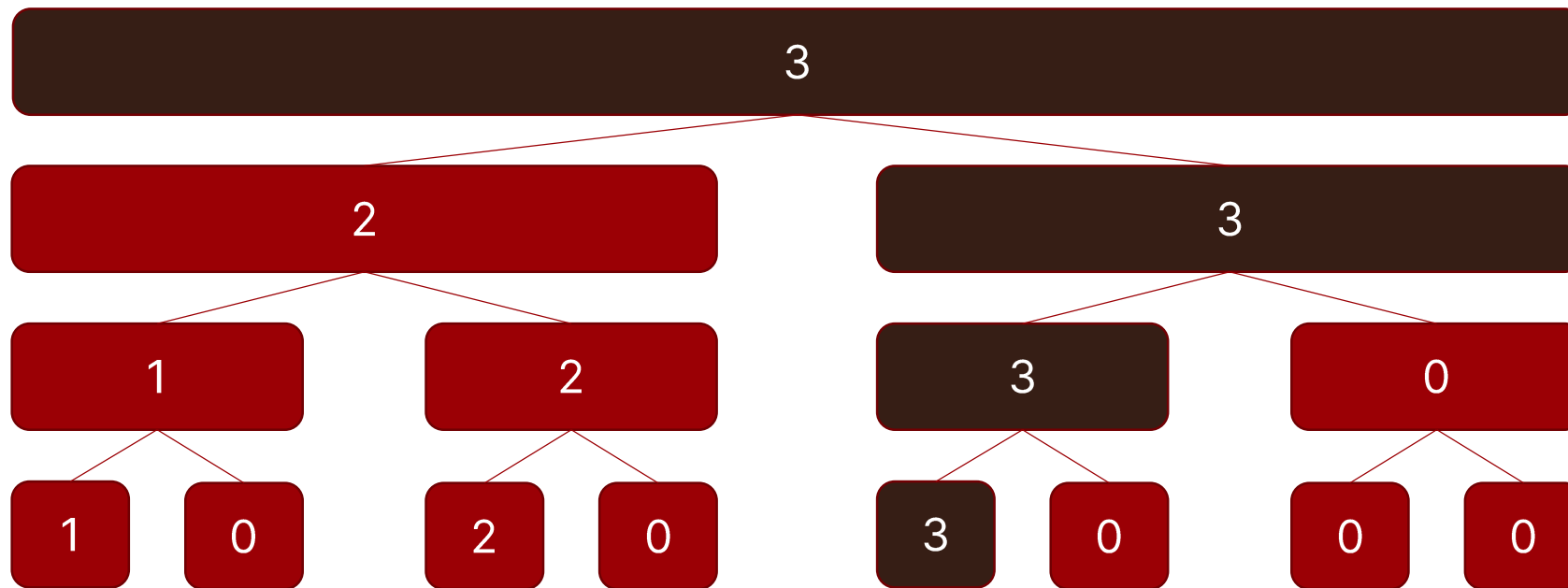


## LIS (Longest Increasing Subsequence)

update(4,3)




10	30	15	25	20	50
1		2		3	



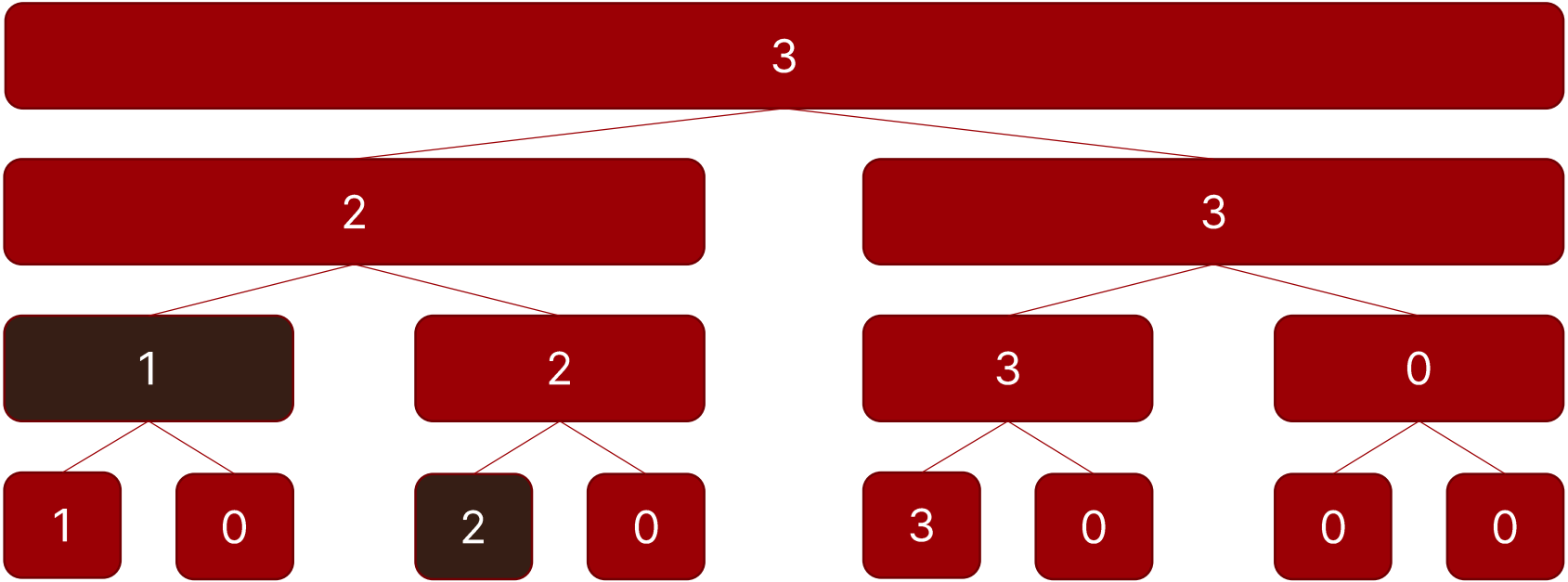


# LIS (Longest Increasing Subsequence)

$\max(0,2) = 2$




10	30	15	25	20	50
1		2		3	



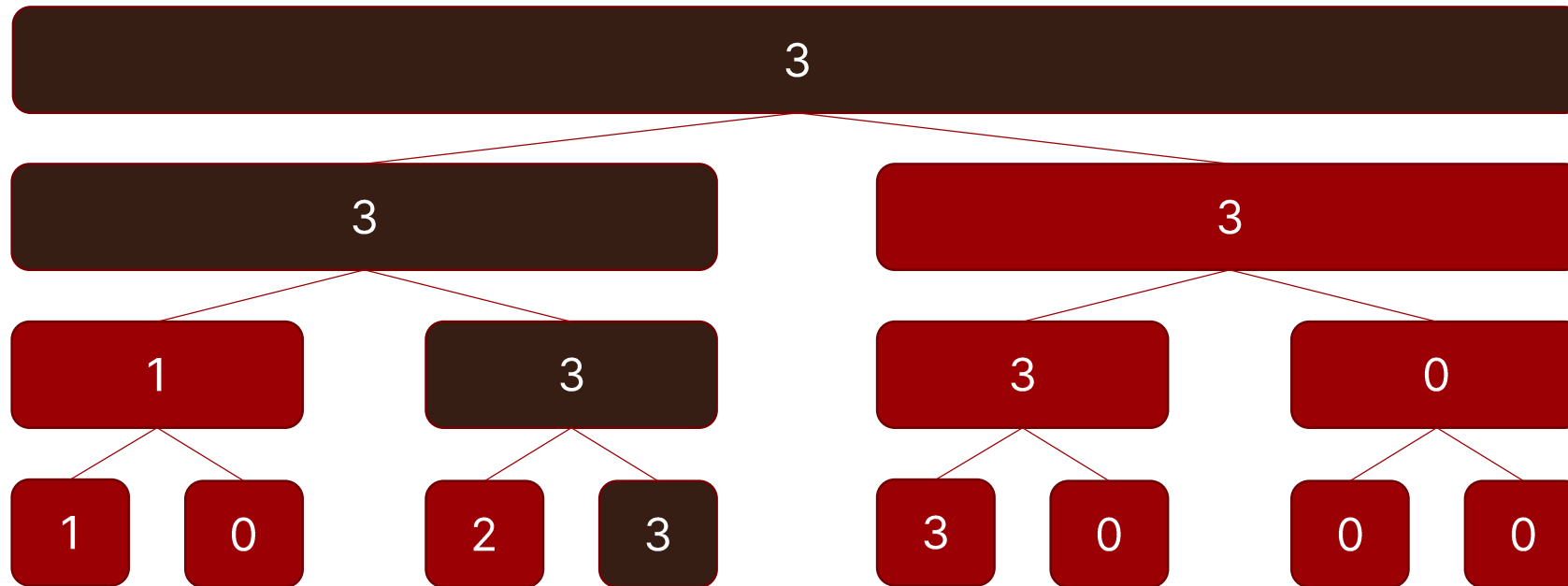


## LIS (Longest Increasing Subsequence)

update(3,3)




10	30	15	25	20	50
1		2	3	3	



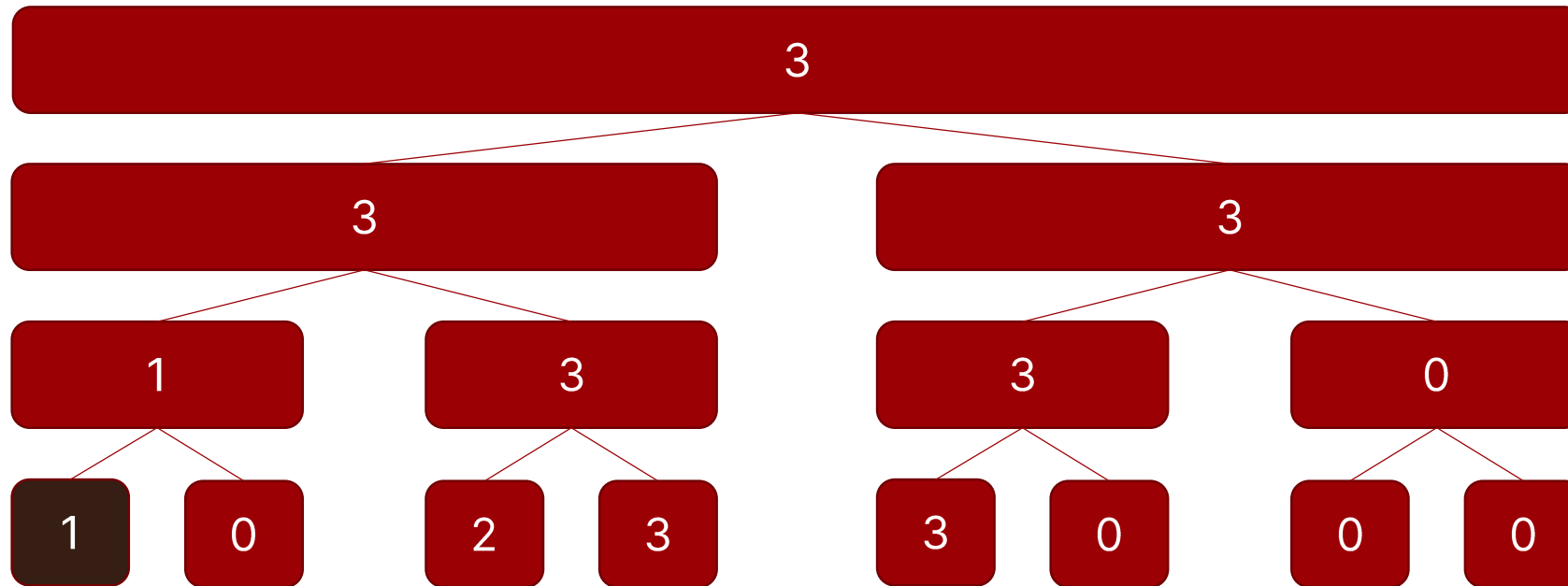


## LIS (Longest Increasing Subsequence)

$$\max(0,0) = 1$$




10	30	15	25	20	50
1		2	3	3	



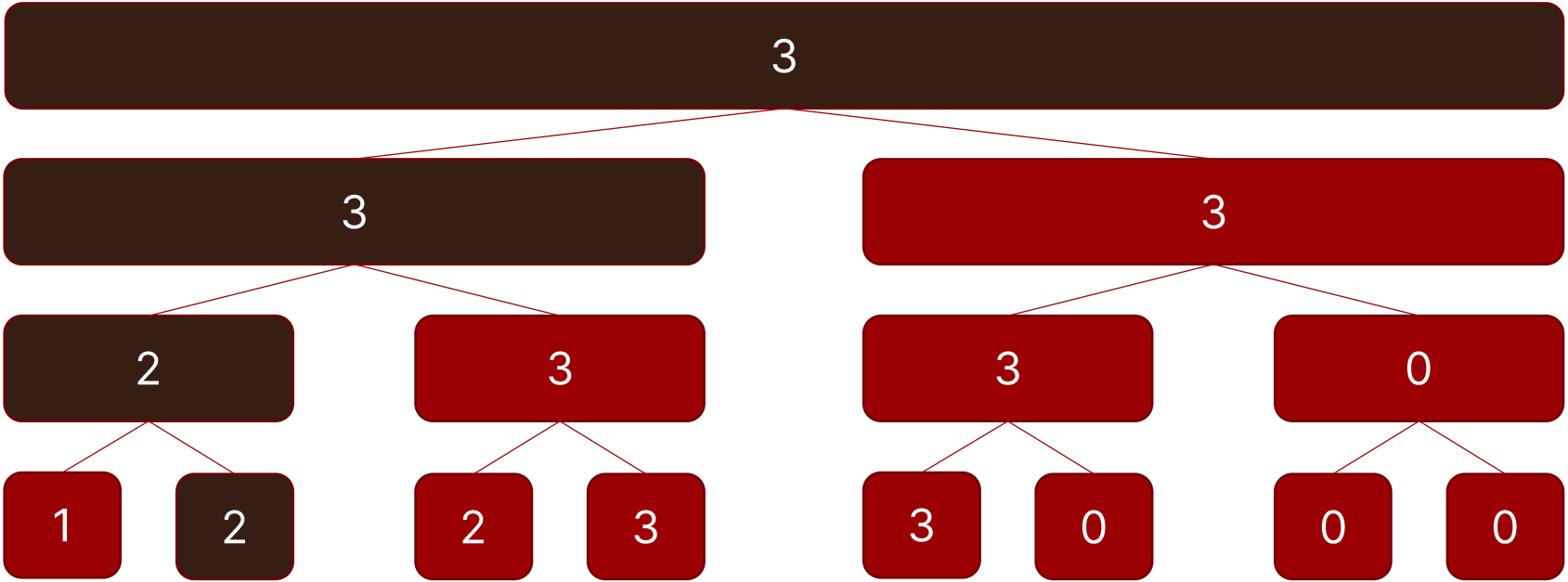


# LIS (Longest Increasing Subsequence)

update(1,2)



10	30	15	25	20	50
1	2	2	3	3	



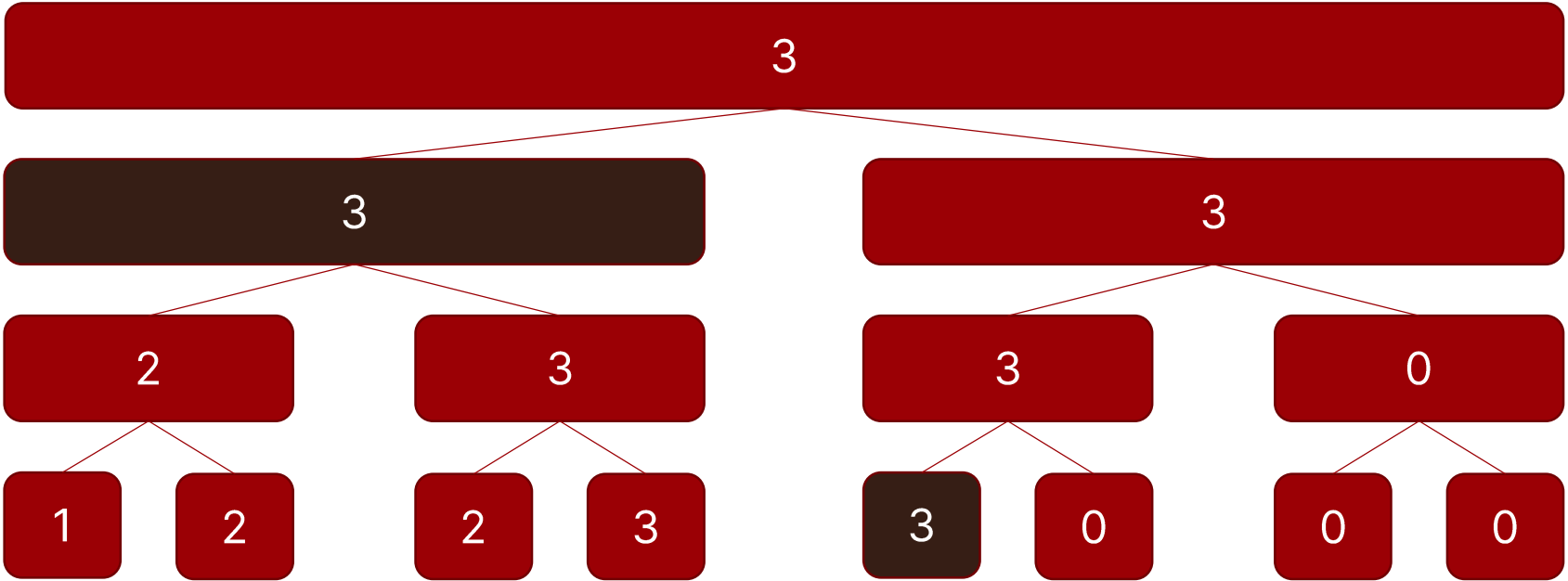


# LIS (Longest Increasing Subsequence)

$\text{max\_}(0,4) = 3$



10	30	15	25	20	50
1	2	2	3	3	



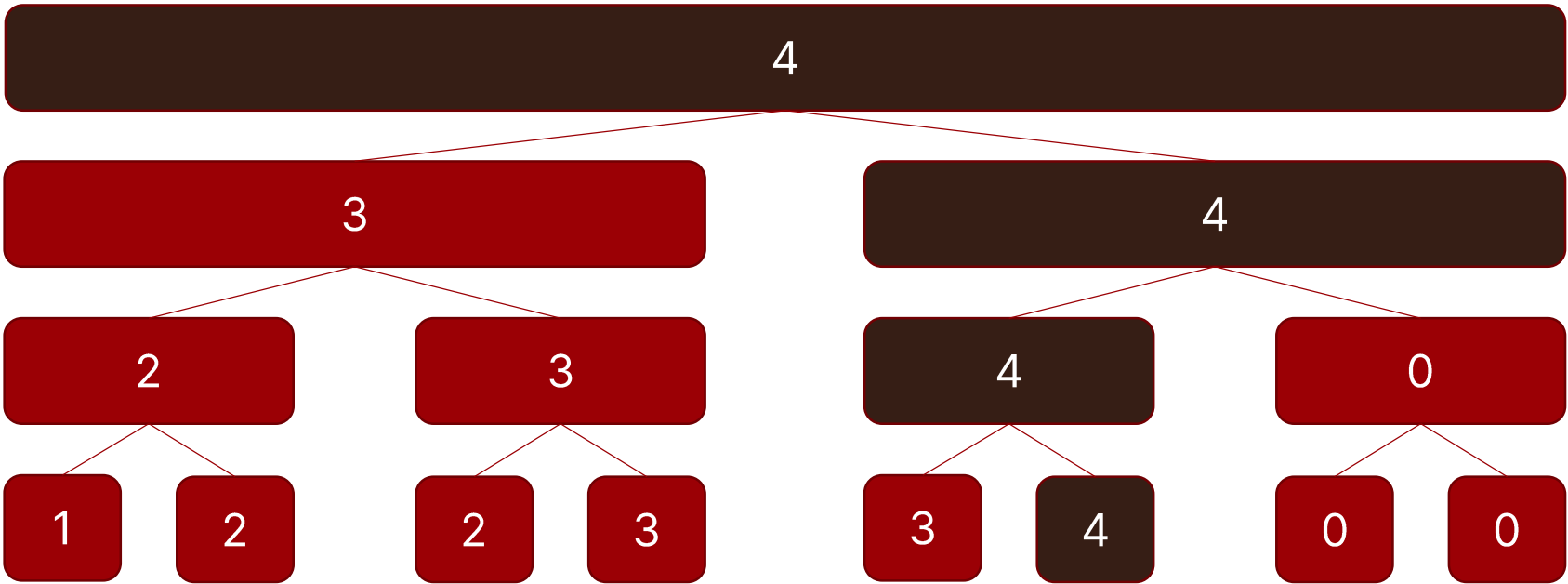


# LIS (Longest Increasing Subsequence)

update(5,4)



10	30	15	25	20	50
1	1	2	3	3	4







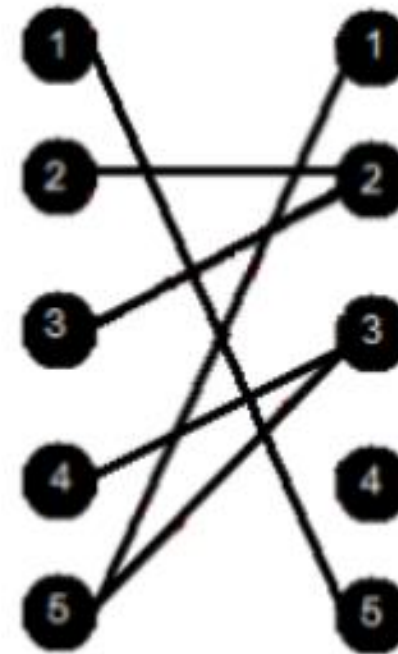
# LIS (Longest Increasing Subsequence)

- 현재 수보다 앞에 있고, 작은 값을 갖는 수로 끝나는 LIS 길이 중에서 비교
- 값이 작은 것부터 LIS 갱신
  - 현재 index보다 작은 index들의 LIS중 최대를 뽑아내기 (구간의 최대, 0~index-1)
  - 구한 최대 LIS 길이 + 1로 세그먼트 트리에 update
- 값이 같다면 인덱스가 큰 것부터 LIS 갱신
  - 인덱스 작은 것부터 갱신하면 같은 수 참조할 수도 있음



## #1615 교차개수세기

- $2N$ 개의 정점,  $M$ 개의 간선을 갖는 이분 그래프
- 교차하는 점의 개수를 구하라
- $1 \leq N \leq 2000, 1 \leq M \leq \frac{N(N-1)}{2}$





## #1615 교차개수세기

- 교차하는 점?



## #1615 교차개수세기

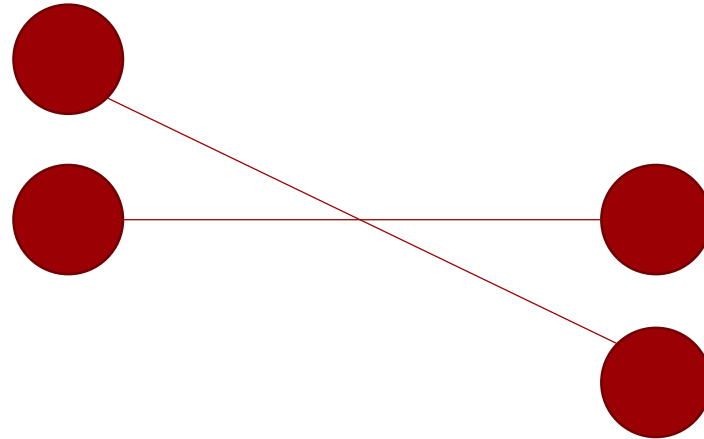
- 교차하는 점?





## #1615 교차개수세기

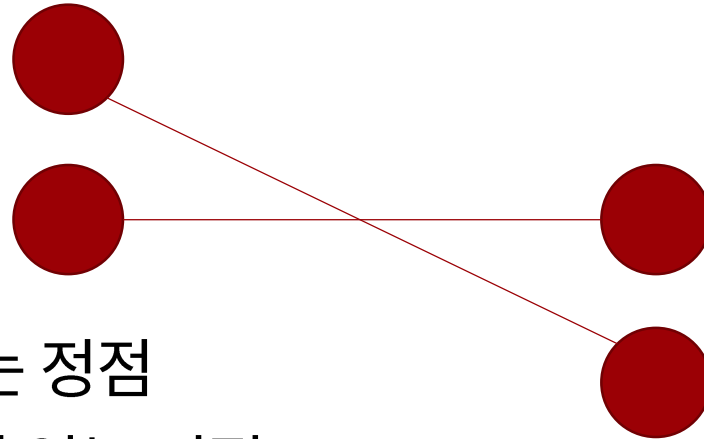
- 교차하는 점?





## #1615 교차개수세기

- 교차하는 점?

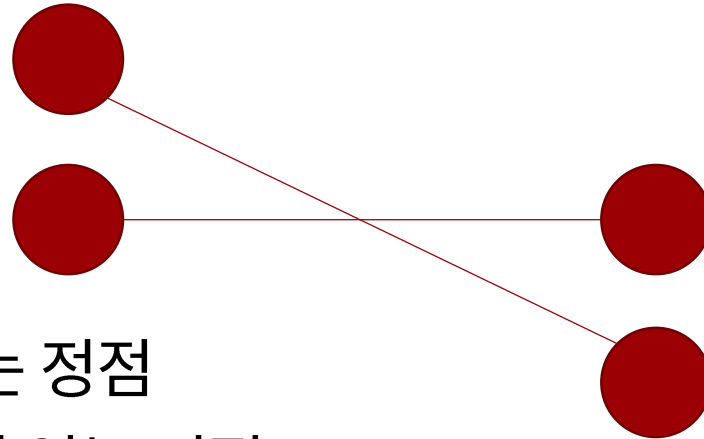


- 한 간선을 기준 왼쪽 그룹에선 위에 있는 정점  
오른쪽 그룹에선 아래에 있는 정점



## #1615 교차개수세기

- 교차하는 점?



- 한 간선을 기준 왼쪽 그룹에선 위에 있는 정점  
오른쪽 그룹에선 아래에 있는 정점

⇒ 현재 간선이 연결하려는 우측 정점보다 아래에 연결된 정점의 개수



## #1615 교차개수세기

- 인접리스트 입력받기

```
90      int n, m;
91      ll res = 0;
92      cin >> n >> m;
93
94      seg S(n);
95      vector<vector<int> > adj(n);
96
97      for (int i = 0; i < m; ++i) {
98          int u, v;
99          cin >> u >> v;
100         adj[u - 1].push_back(v - 1);
101     }
```





## #1615 교차개수세기

- 좌측 그룹의 위쪽 정점부터 간선 연결
- 간선이 연결하는 우측 그룹의 정점 =  $v$
- $v$ 보다 아래 있는 정점에 연결된 간선 개수

```
103     for (int u = 0; u < n; ++u) {  
104         for (auto v : adj[u])  
105             if (v != n - 1)  
106                 res += S.sum(v + 1, n - 1);  
107  
108         for (auto v : adj[u])  
109             S.update(v);  
110     }  
111  
112     cout << res;
```

- 한 정점에서 나가는 여러 간선은 서로 영향 X → 한번에 업데이트 하기



5 2042 구간 합 구하기

5 2357 최솟값과 최댓값

5 14428 수열과 쿼리 16

4 1517 버블 소트

3 3745 오름세

3 1615 교차개수세기

5 2517 달리기

5 1777 순열복원

5 2243 사탕상자

4 1321 군인