Segment Tree
(구간트리)

㈜한컴에듀케이션 이주현

개 요

- 목표 : 구간에 대한 질의에 대하여 효율적으로 답하는 것.
- 적용되는 문제 : 구간의 합, 구간의 최소값, 구간의 최대값등
  구간의 정보를 묻는 문제 해결에 사용한다.
- idea : 구간의 정보를 갖는 노드를 이진트리로 만든다.
  저장 공간은 배열의 길이보다 크거나 같은
  가장 가까운 2의 제곱수의 2배 크기가 필요하다.
- 구현방법은 매우 다양하다.
  기본적으로 1차원 배열과 재귀를 이용하는 것으로 설명한다.

다음 문제를 생각해보자.

[jungol 1726 구간의 최대값]

❖ N개의 정수로 이뤄진 수열을 입력 받는다.
(1 <= N <= 50,000)

❖ 다음 Q개의 질의에 대한 답을 구하여 출력한다.
(1 <= Q <= 200,000)

❖각 질의는 수열의 임의의 연속된 구간의 최대값을 구하여 답하는 것이다.

입력 예

index		
	8	4 *=> N, Q*
1	3	
2	7	
3	4	
4	1	
5	5	
6	2	
7	6	
8	8	
	1	7
	4	6
	2	8
	2	7

출력 예

7
5
8
7

입력 예

index

1
2
3
4
5
6
7
8

8 4 *=> N, Q*

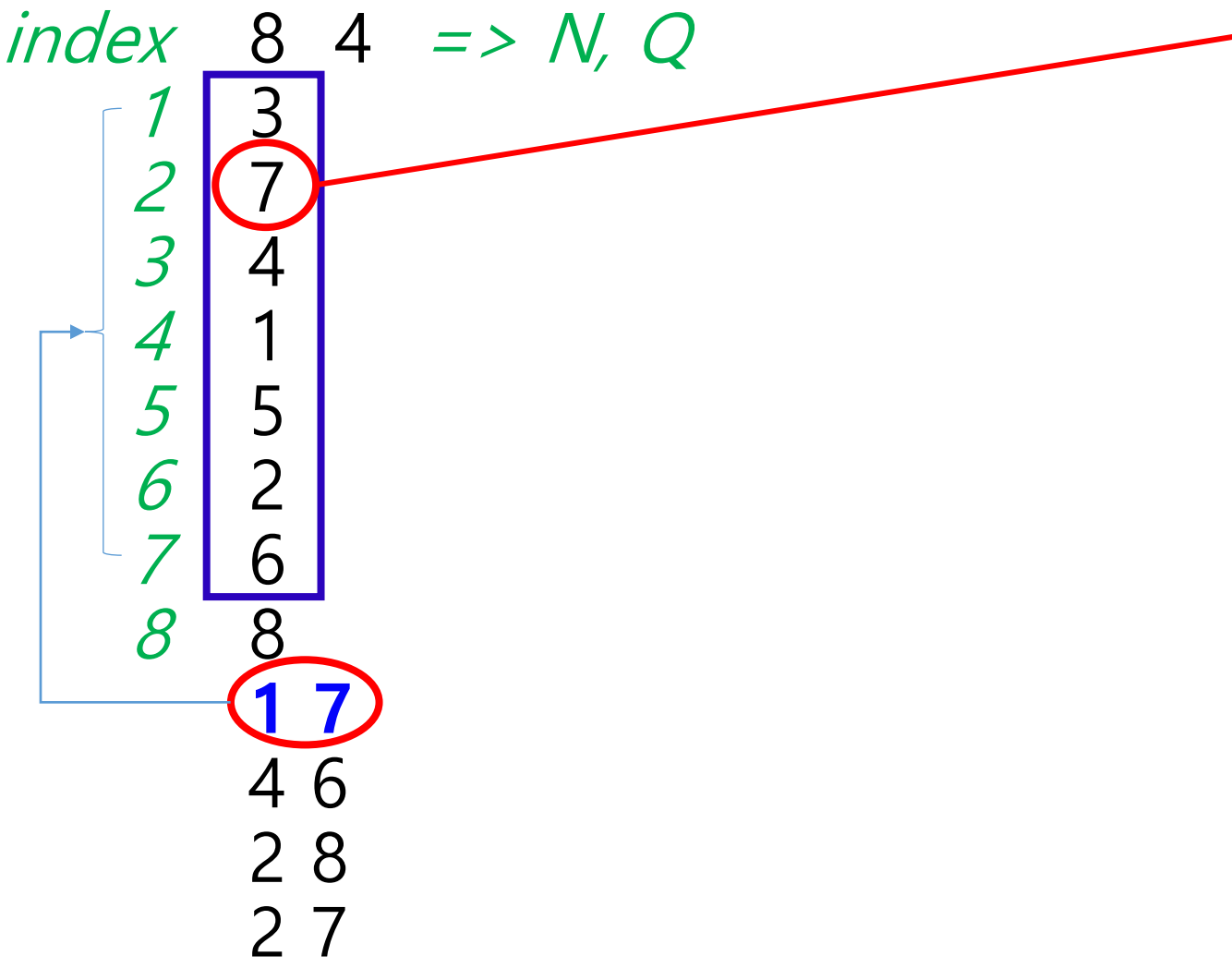
3
7
4
1
5
2
6
8

1 7

4 6
2 8
2 7

출력 예

7
5
8
7



입력 예

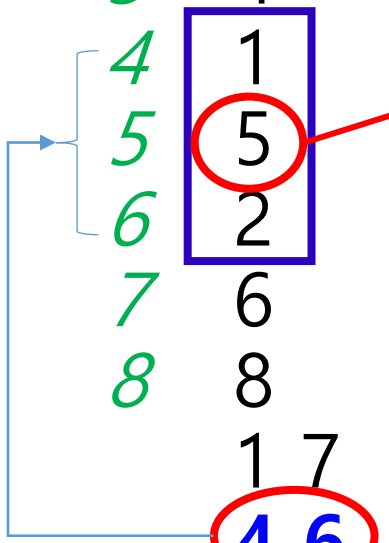
index

1
2
3
4
5
6
7
8

8 4 *=> N, Q*
3
7
4
1
5
2
6
8
1 7
4 6
2 8
2 7

출력 예

7
5
8
7



입력 예

index

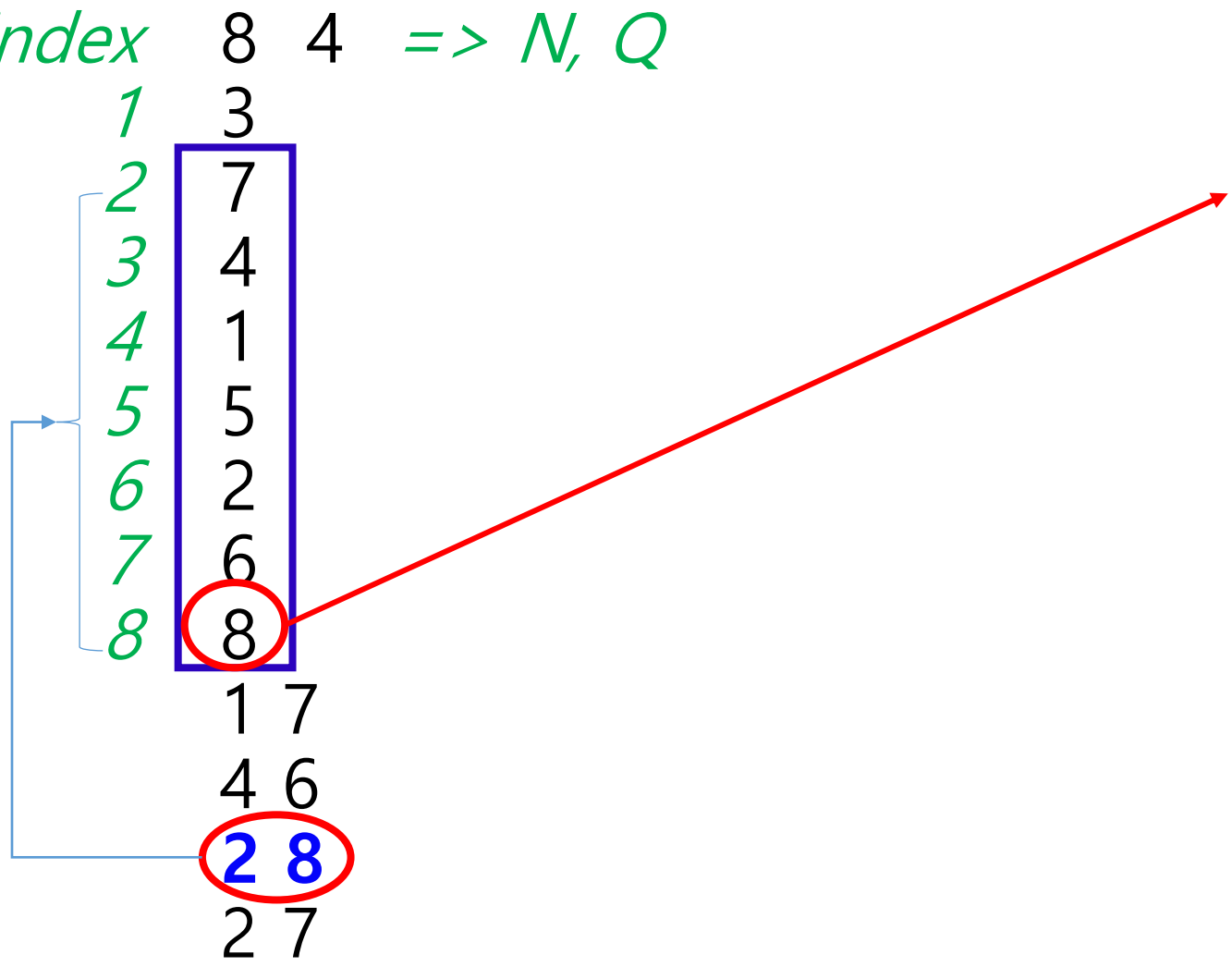
1
2
3
4
5
6
7
8

8 4 *=> N, Q*

3
7
4
1
5
2
6
8
1 7
4 6
2 8
2 7

출력 예

7
5
8
7



입력 예

index

1
2
3
4
5
6
7
8

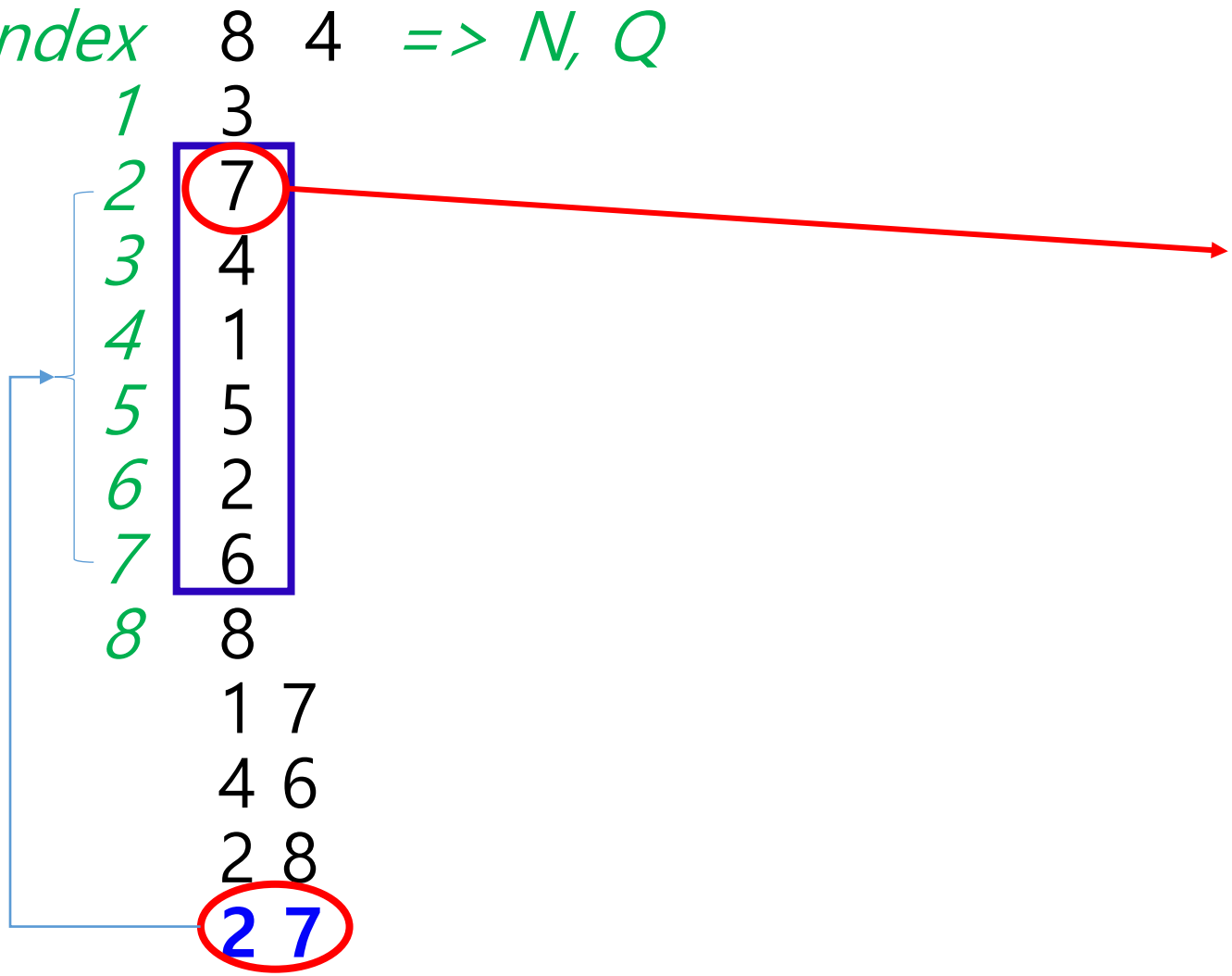
8 4 *=> N, Q*

3
7
4
1
5
2
6
8

1 7
4 6
2 8
2 7

출력 예

7
5
8
7



jungol 1726 구간의 최대값 해법 탐구

- 단순 검색 방법 :
  수열의 길이 N = 50,000 이고 질의 수 Q = 200,000 라면
  시간복잡도는 O(N * Q) = 50000 * 200000 = 100억 ?
  => 시간복잡도 상으로 **시간초과가 예상**된다.

jungol 1726 구간의 최대값 해법 탐구 cont.

- 단순 검색 방법 :
  수열의 길이 N = 50,000 이고 질의 수 Q = 200,000 라면
  시간복잡도는 O(N * Q) = 50000 * 200000 = 100억 ?
  => 시간복잡도 상으로 **시간초과가 예상**된다.

- 중복된 질의를 골라 중복을 배제하는 방법 :
  각 질의에 대한 결과를 출력하는데 가능한 질의 수는
  약 12억5천만개 나오므로 이 또한 쉽지 않다.

jungol 1726 구간의 최대값 해법 탐구 cont.

- 단순 검색 방법 :
  수열의 길이 N = 50,000 이고 질의 수 Q = 200,000 라면
  시간복잡도는 O(N * Q) = 50000 * 200000 = 100억 ?
  => 시간복잡도 상으로 **시간초과가 예상**된다.

- 중복된 질의를 골라 중복을 배제하는 방법 :
  각 질의에 대한 결과를 출력하는데 가능한 질의 수는
  약 12억5천만개 나오므로 이 또한 쉽지 않다.

- 각 질의에 대하여 탐색 공간을 줄이는 방법 :
  하나의 질의에 대하여 최대 50000번 탐색한다.
  이를 최대 16번 이하로 줄일수 있다.
  => 구간트리를 이용하여 ^^

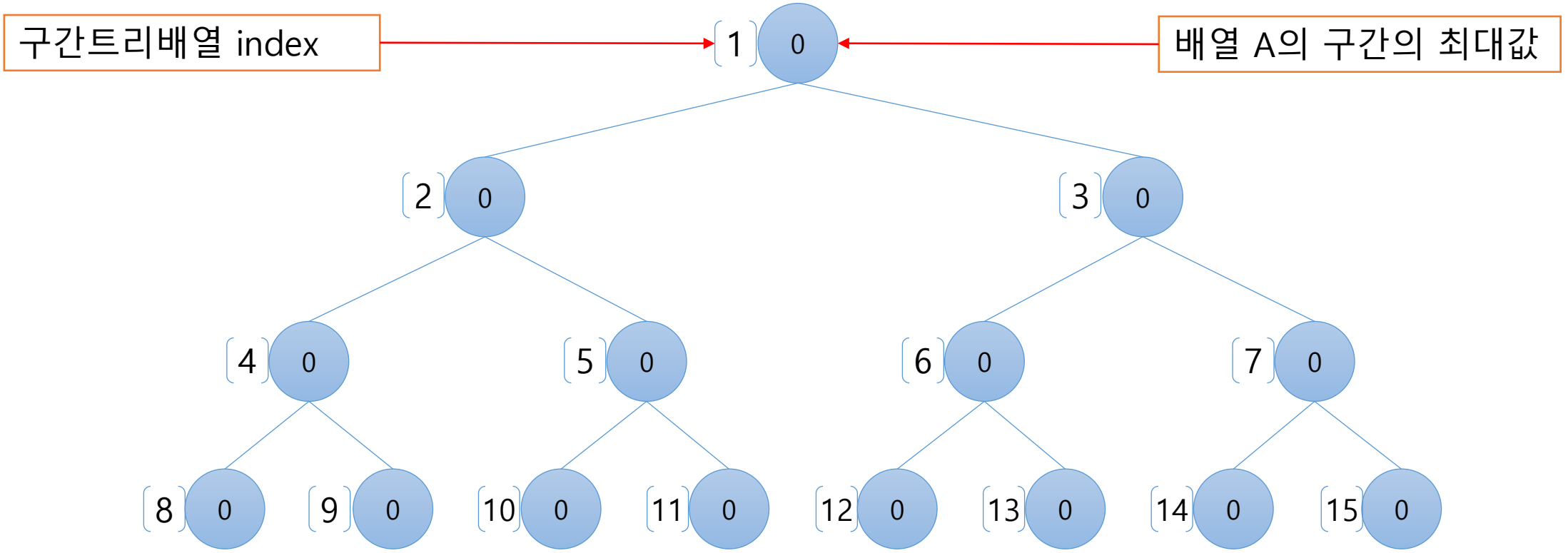
1. 구간트리 만들기(build or update)

index	1	2	3	4	5	6	7	8
A[i]	3	7	4	1	5	2	6	8

- ❖ 위 표와 같이 주어진 입력이 A[] 에 저장되어 있다고 할때 이를 이용하여 구간트리를 만들어보자.
- ❖ 2진 트리를 DFS – post order(깊이우선탐색 후위순회)한다.
  : 재귀함수로 깊이 우선 탐색을 하는데
  왼쪽 오른쪽 child 중에 탐색이 가능한 곳을 탐색한 후
  현재 노드의 값을 업데이트한다.

구간트리 Build

build(구간트리노드번호, 트리구간 시작, 트리구간 끝)

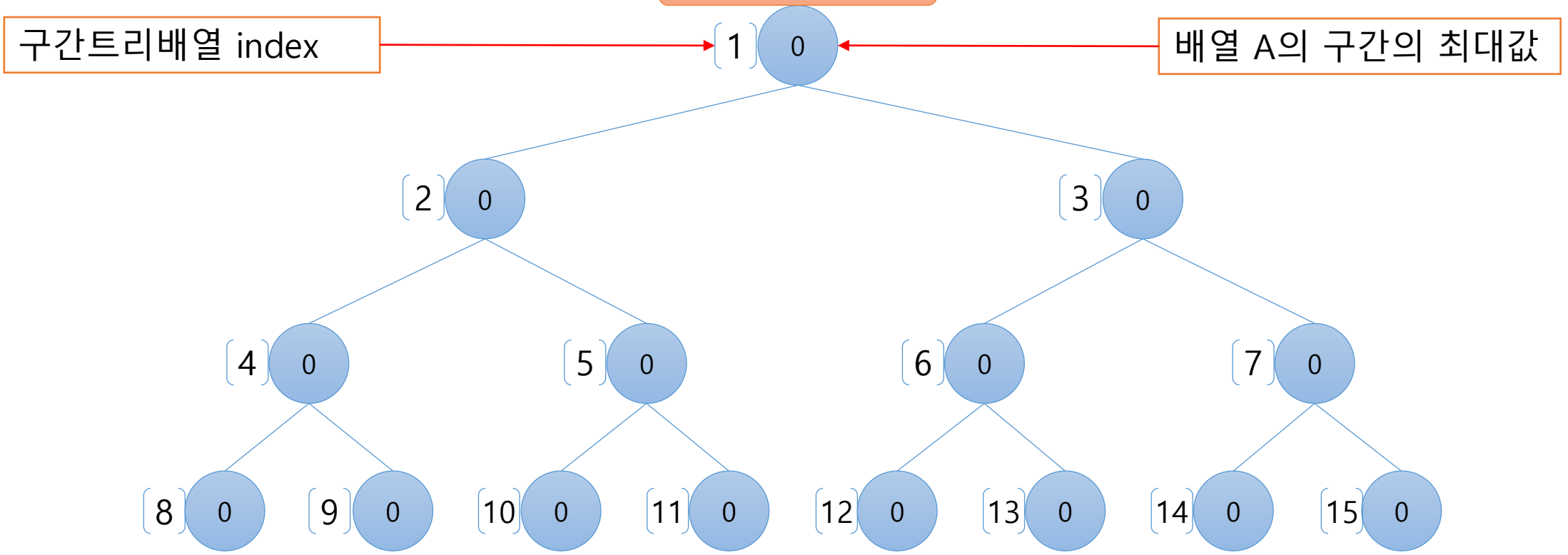


index	1	2	3	4	5	6	7	8
A[i]	3	7	4	1	5	2	6	8

구간트리 Build - cont.

build(구간트리노드번호, 트리구간 시작, 트리구간 끝)

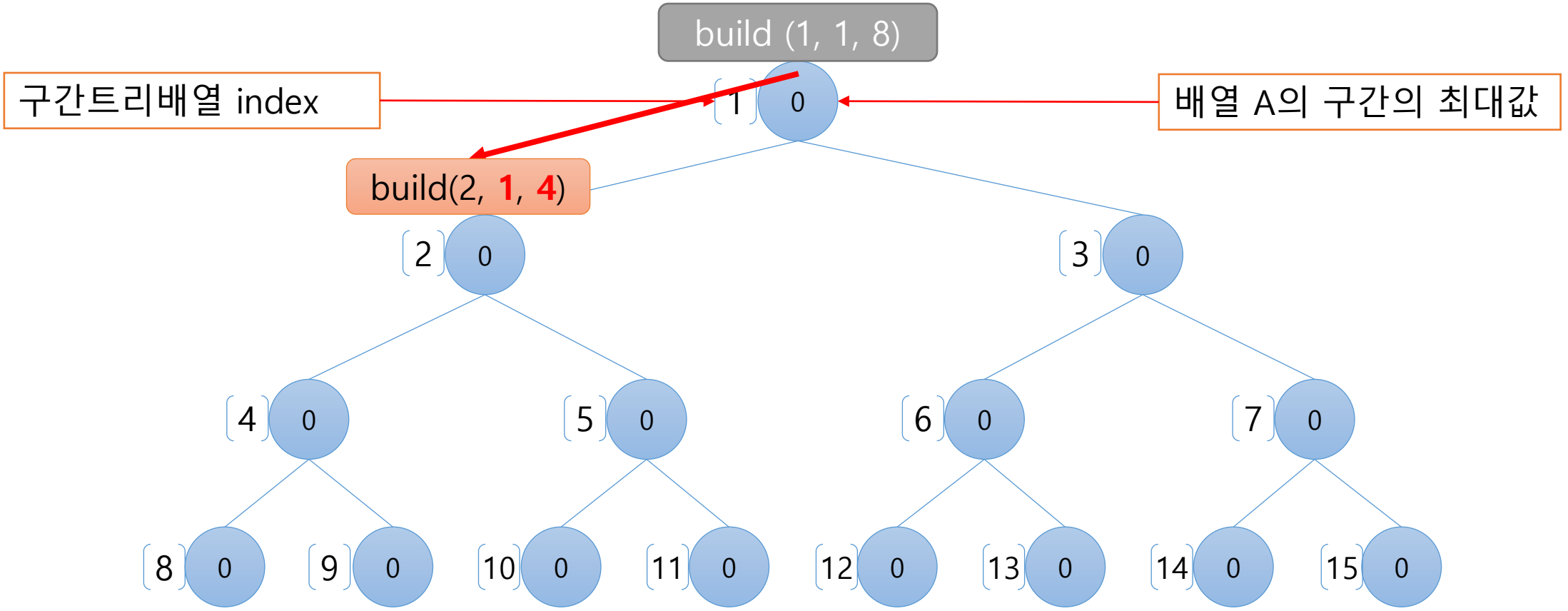
build(1, **1**, **8**)



index	1	2	3	4	5	6	7	8
A[i]	3	7	4	1	5	2	6	8

구간트리 Build - cont.

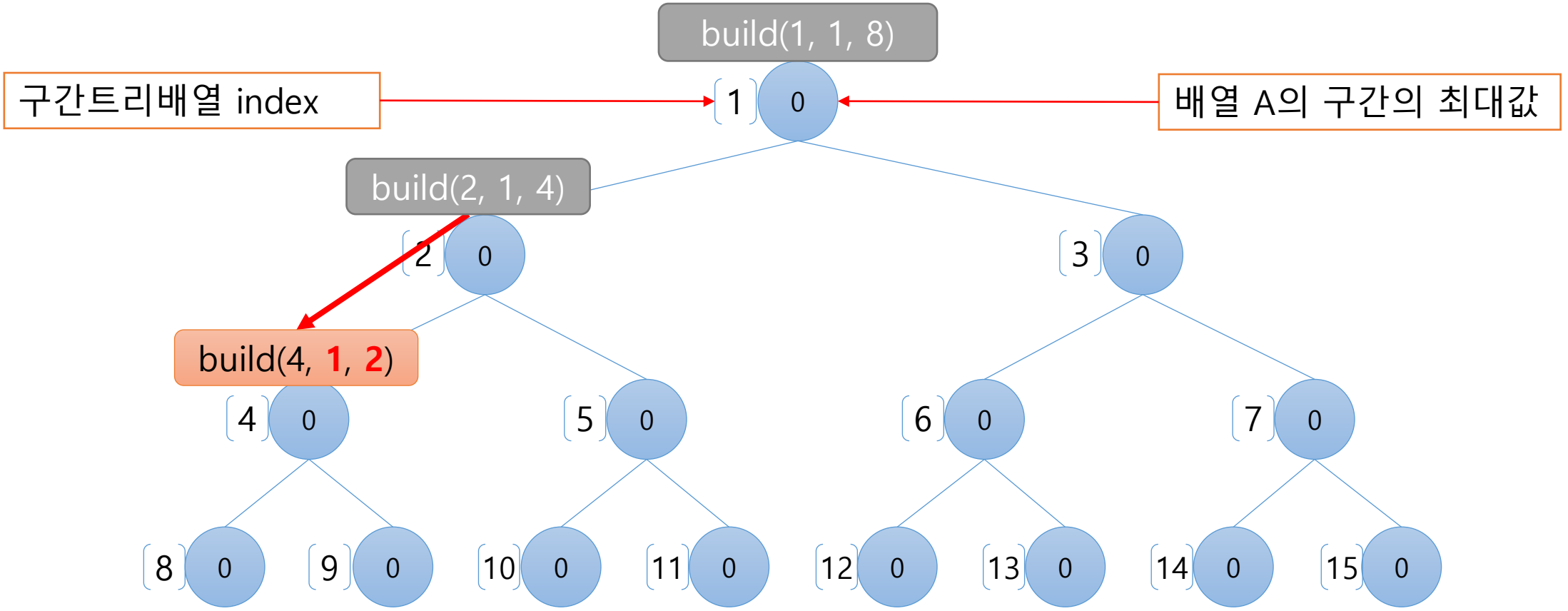
build(구간트리노드번호, 트리구간 시작, 트리구간 끝)



index	1	2	3	4	5	6	7	8
A[i]	3	7	4	1	5	2	6	8

구간트리 Build - cont.

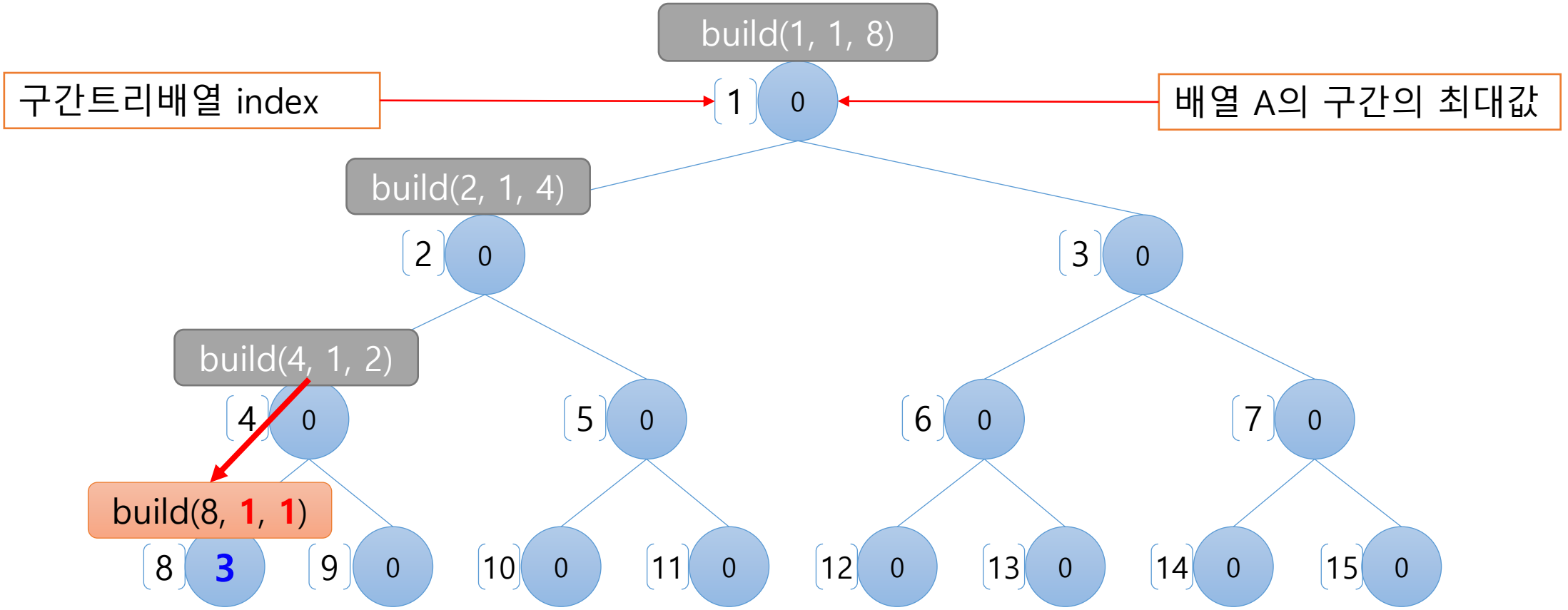
build(구간트리노드번호, 트리구간 시작, 트리구간 끝)



index	1	2	3	4	5	6	7	8
A[i]	3	7	4	1	5	2	6	8

구간트리 Build - cont.

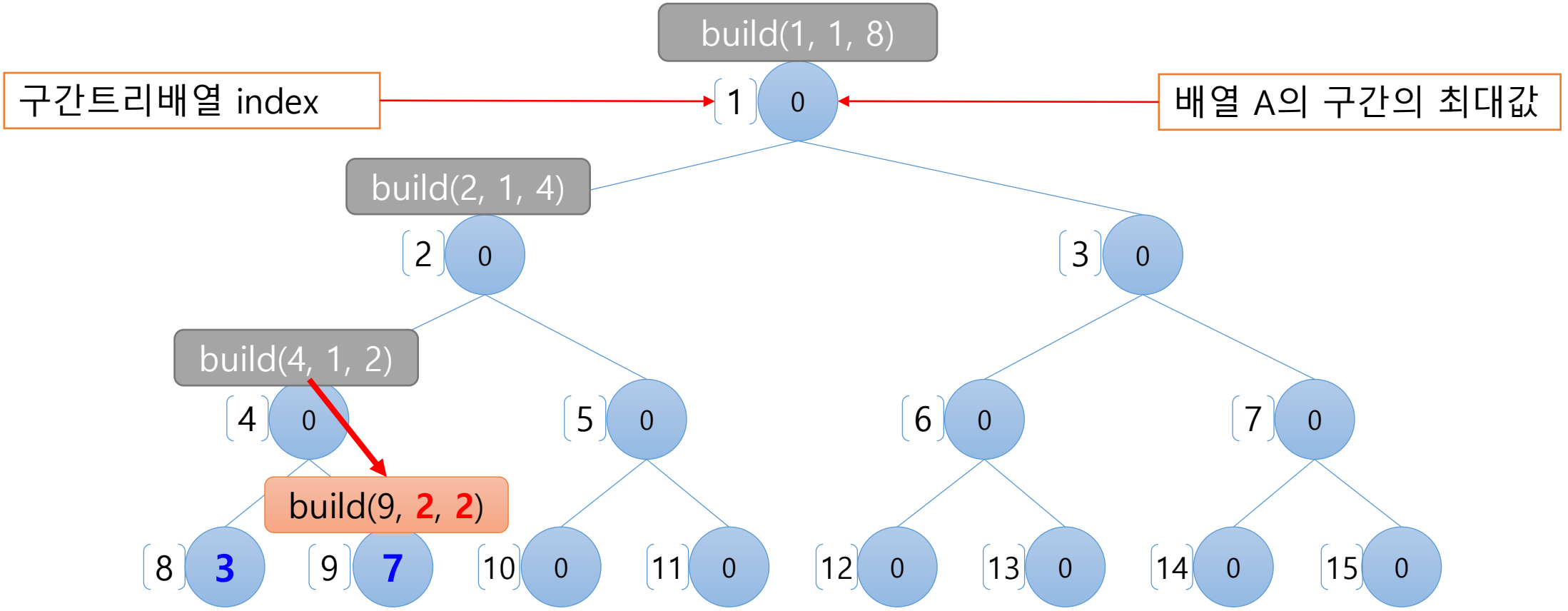
build(구간트리노드번호, 트리구간 시작, 트리구간 끝)



index	1	2	3	4	5	6	7	8
A[i]	3	7	4	1	5	2	6	8

구간트리 Build - cont.

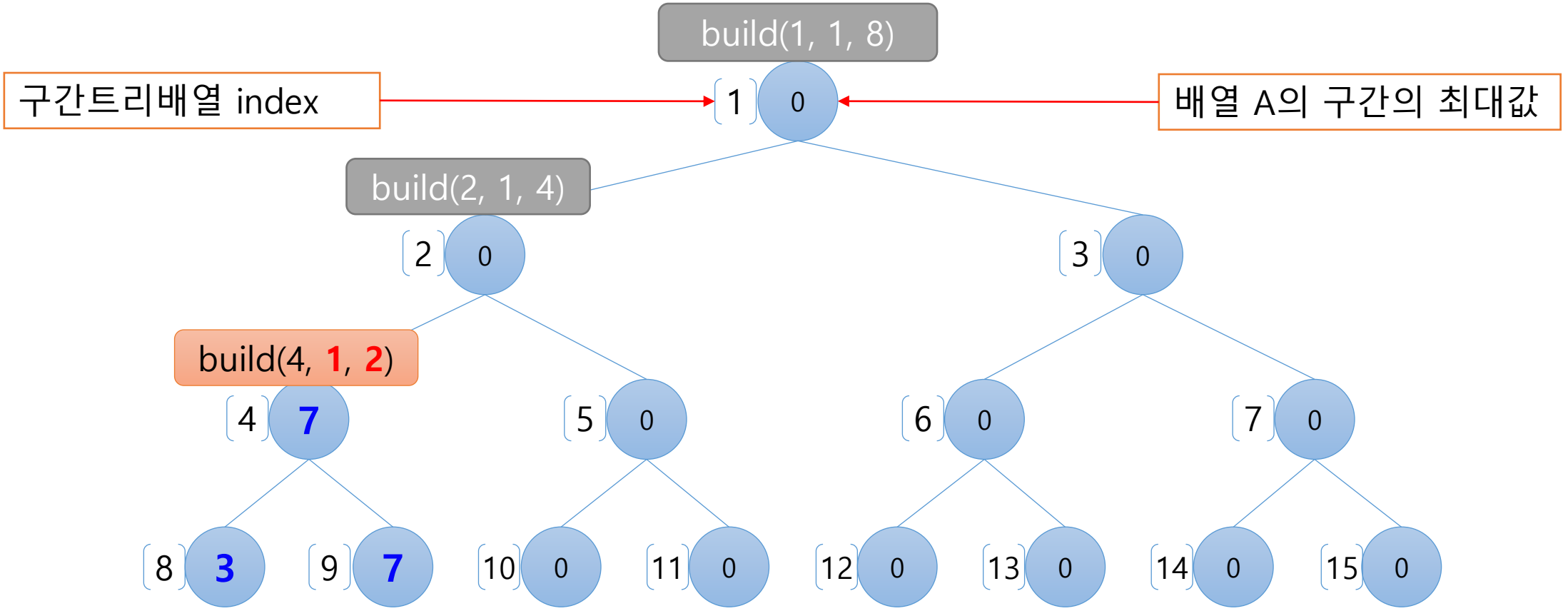
build(구간트리노드번호, 트리구간 시작, 트리구간 끝)



index	1	2	3	4	5	6	7	8
A[i]	3	7	4	1	5	2	6	8

구간트리 Build - cont.

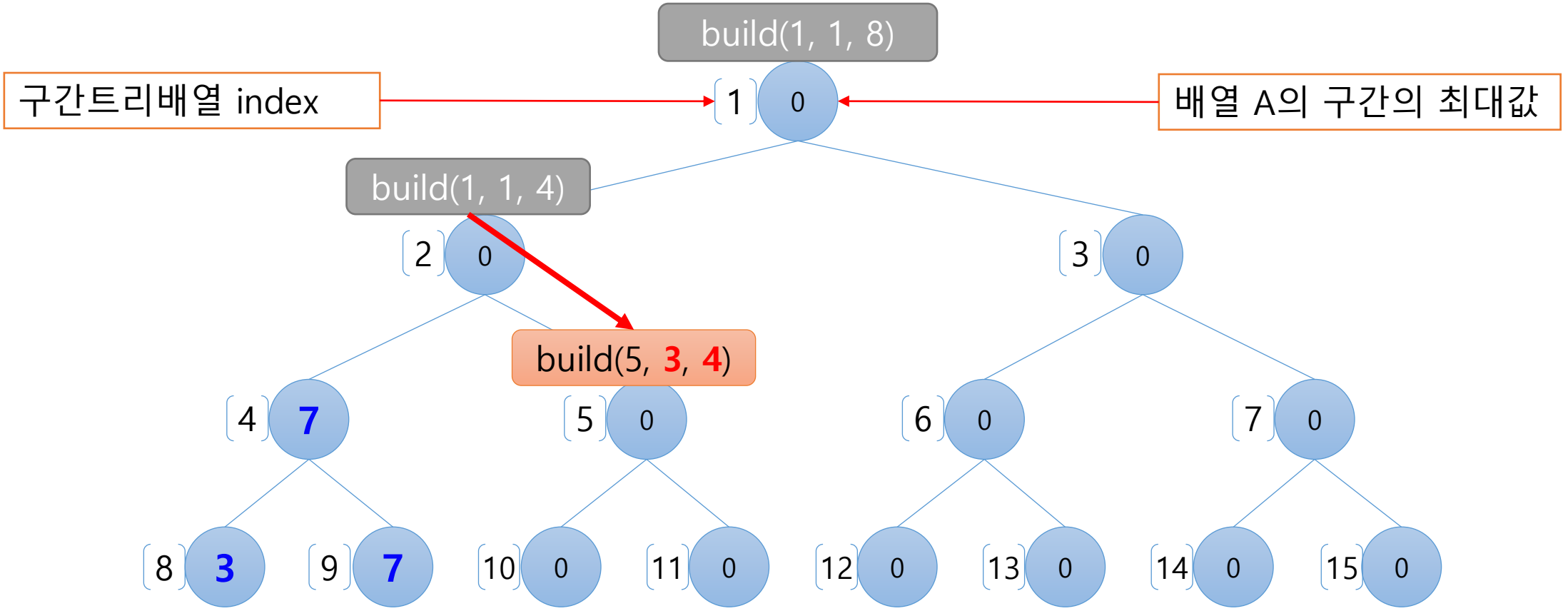
build(구간트리노드번호, 트리구간 시작, 트리구간 끝)



index	1	2	3	4	5	6	7	8
A[i]	3	7	4	1	5	2	6	8

구간트리 Build - cont.

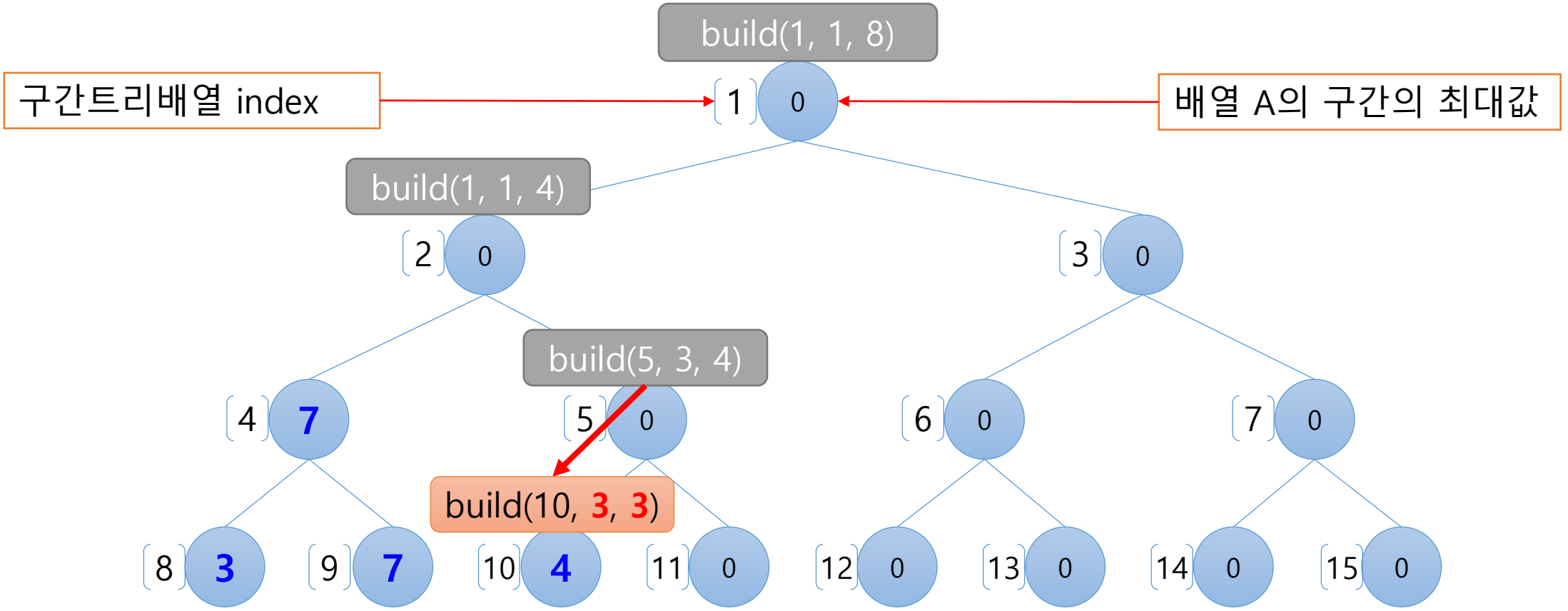
build(구간트리노드번호, 트리구간 시작, 트리구간 끝)



index	1	2	3	4	5	6	7	8
A[i]	3	7	4	1	5	2	6	8

구간트리 Build - cont.

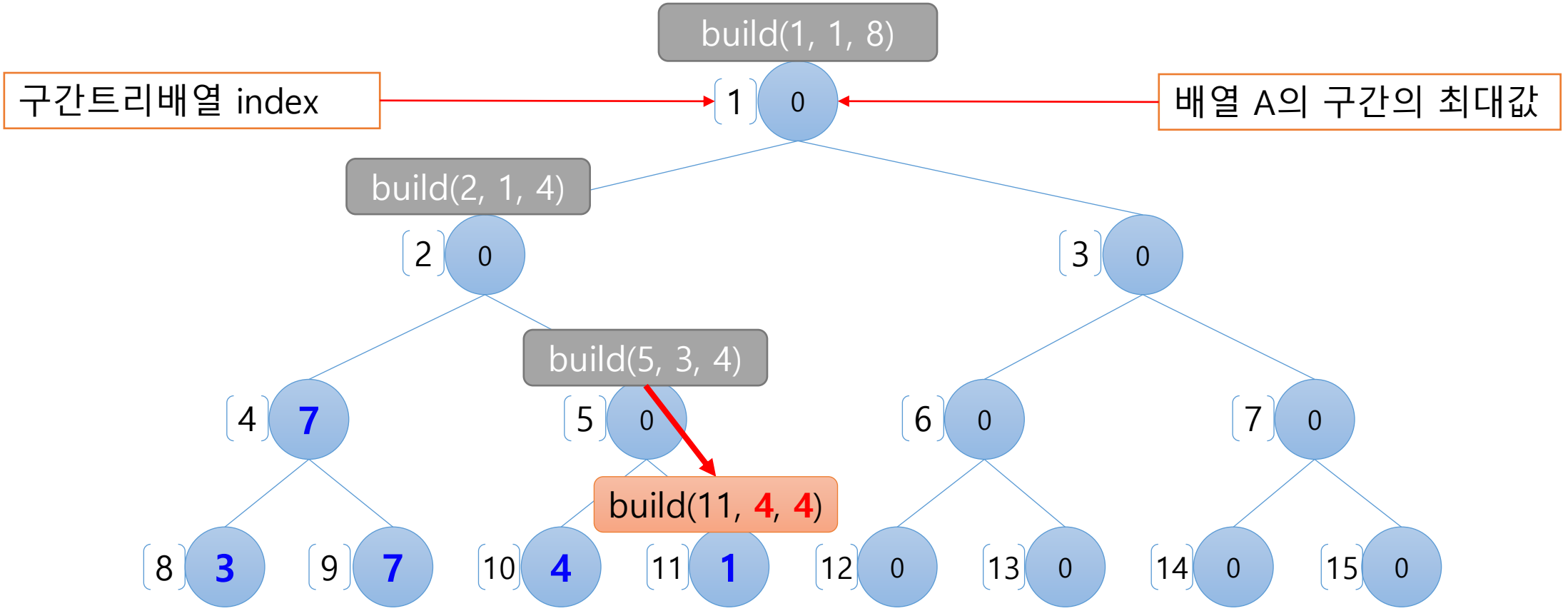
build(구간트리노드번호, 트리구간 시작, 트리구간 끝)



index	1	2	3	4	5	6	7	8
A[i]	3	7	4	1	5	2	6	8

구간트리 Build - cont.

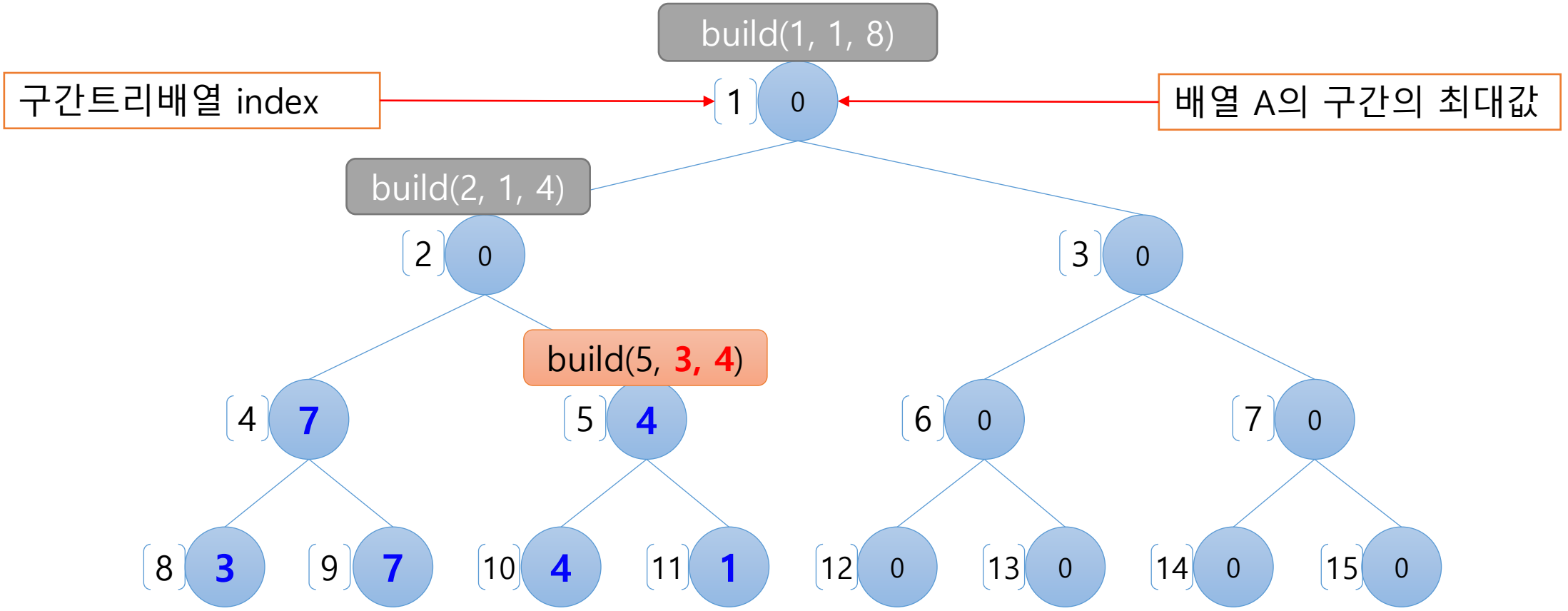
build(구간트리노드번호, 트리구간 시작, 트리구간 끝)



index	1	2	3	4	5	6	7	8
A[i]	3	7	4	1	5	2	6	8

구간트리 Build - cont.

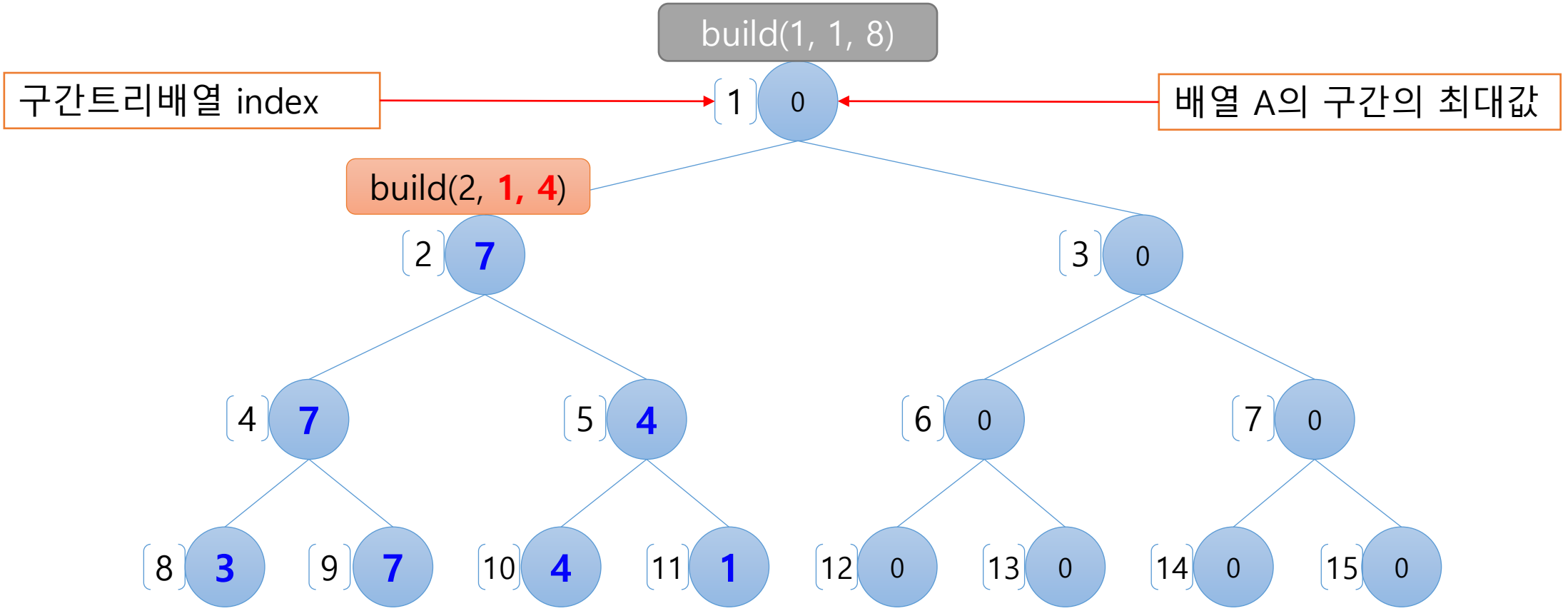
build(구간트리노드번호, 트리구간 시작, 트리구간 끝)



index	1	2	3	4	5	6	7	8
A[i]	3	7	4	1	5	2	6	8

구간트리 Build - cont.

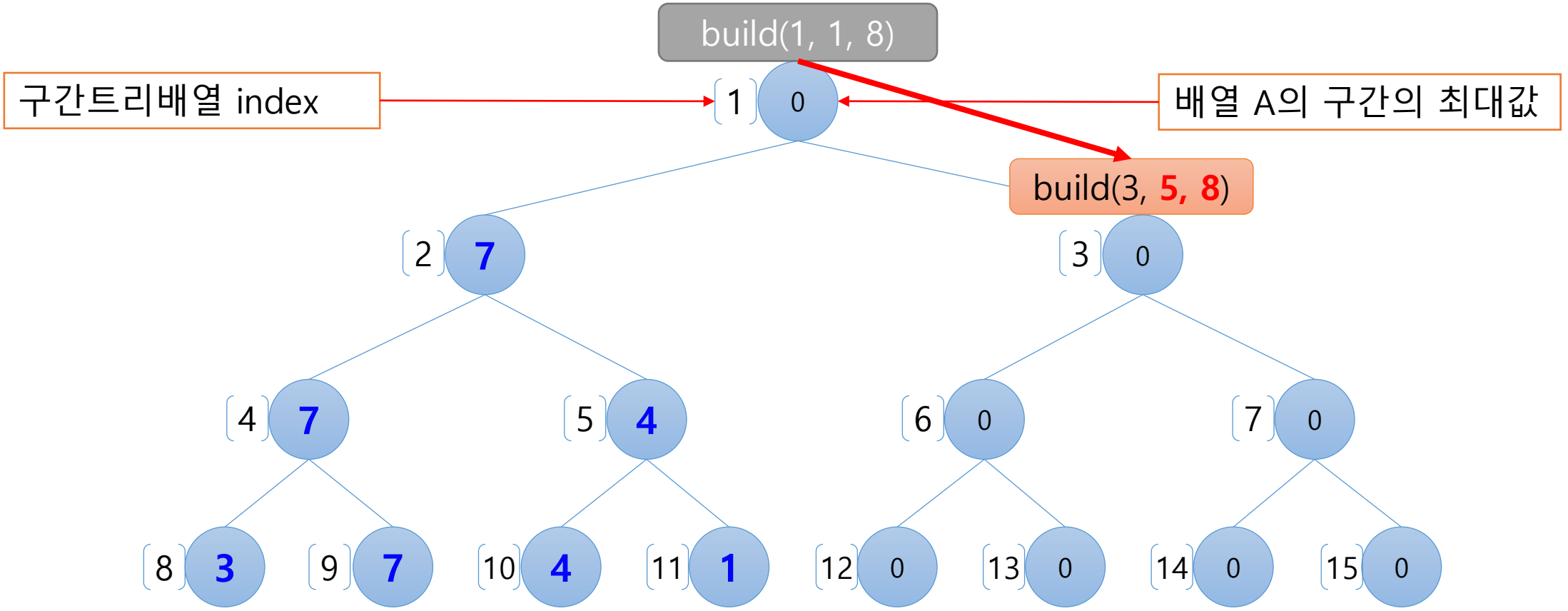
build(구간트리노드번호, 트리구간 시작, 트리구간 끝)



index	1	2	3	4	5	6	7	8
A[i]	3	7	4	1	5	2	6	8

구간트리 Build - cont.

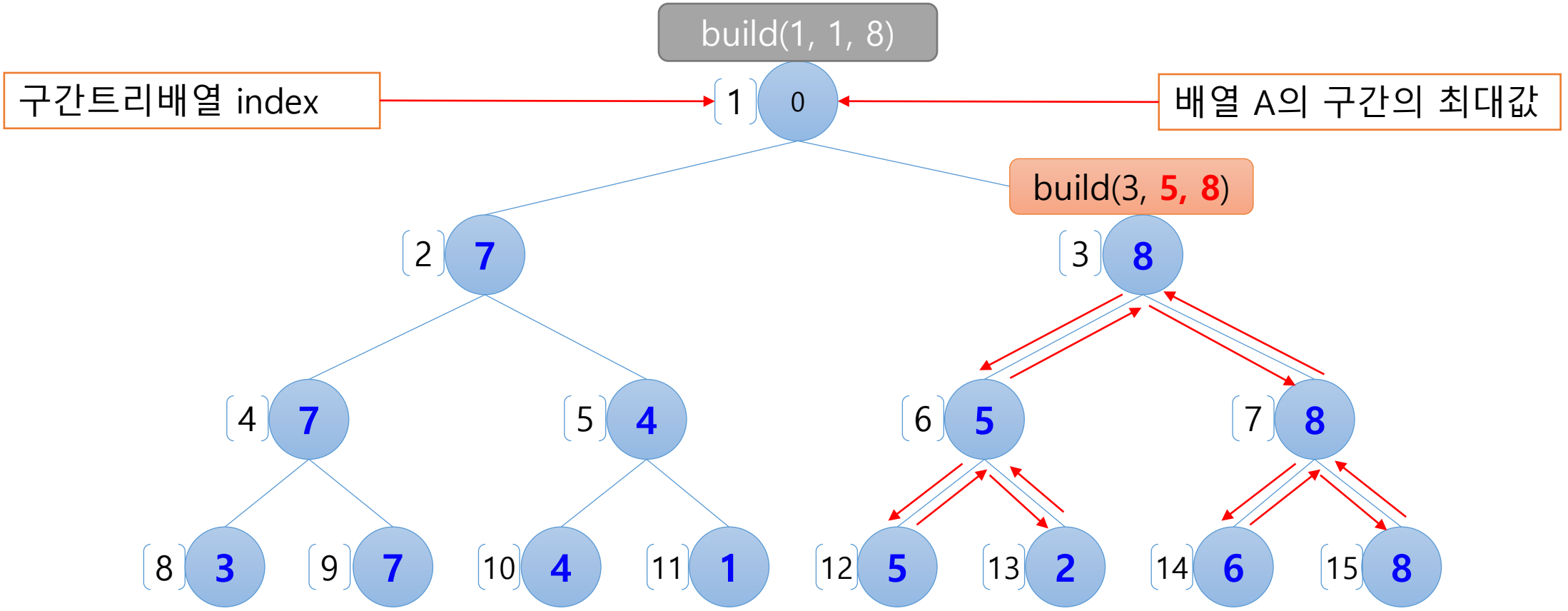
build(구간트리노드번호, 트리구간 시작, 트리구간 끝)



index	1	2	3	4	5	6	7	8
A[i]	3	7	4	1	5	2	6	8

구간트리 Build - cont.

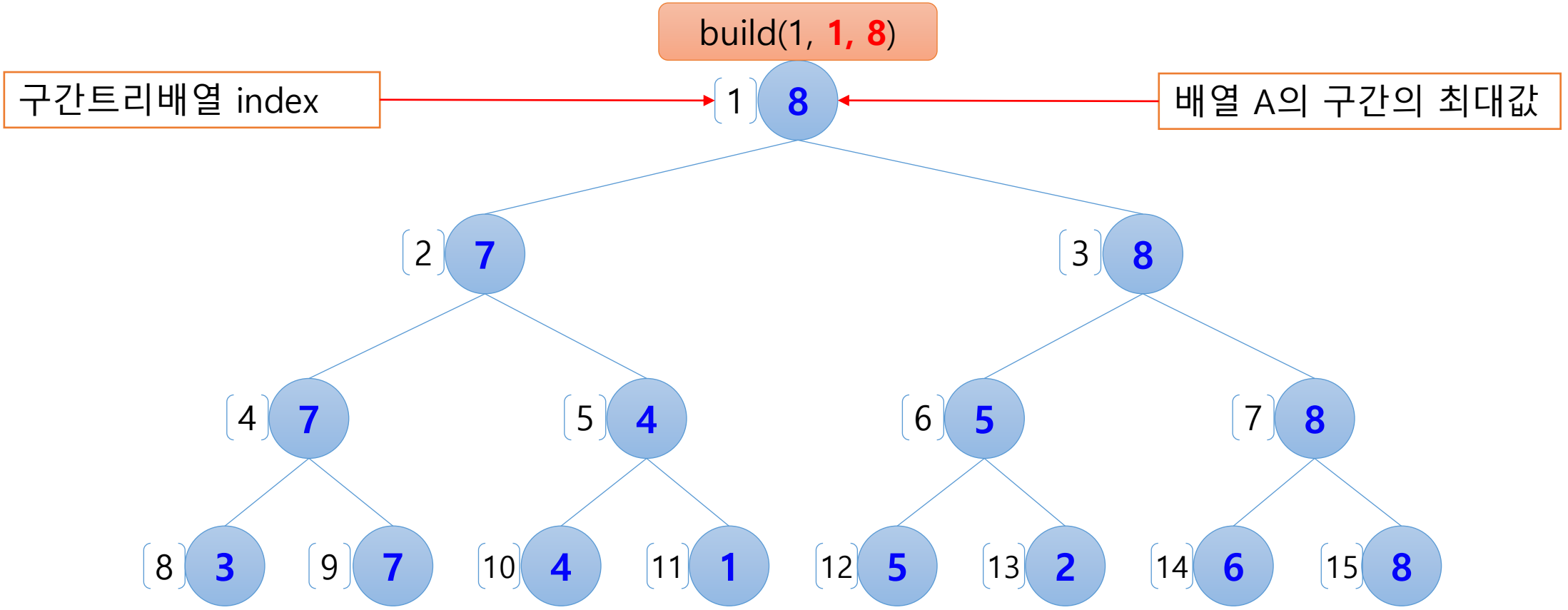
build(구간트리노드번호, 트리구간 시작, 트리구간 끝)



index	1	2	3	4	5	6	7	8
A[i]	3	7	4	1	5	2	6	8

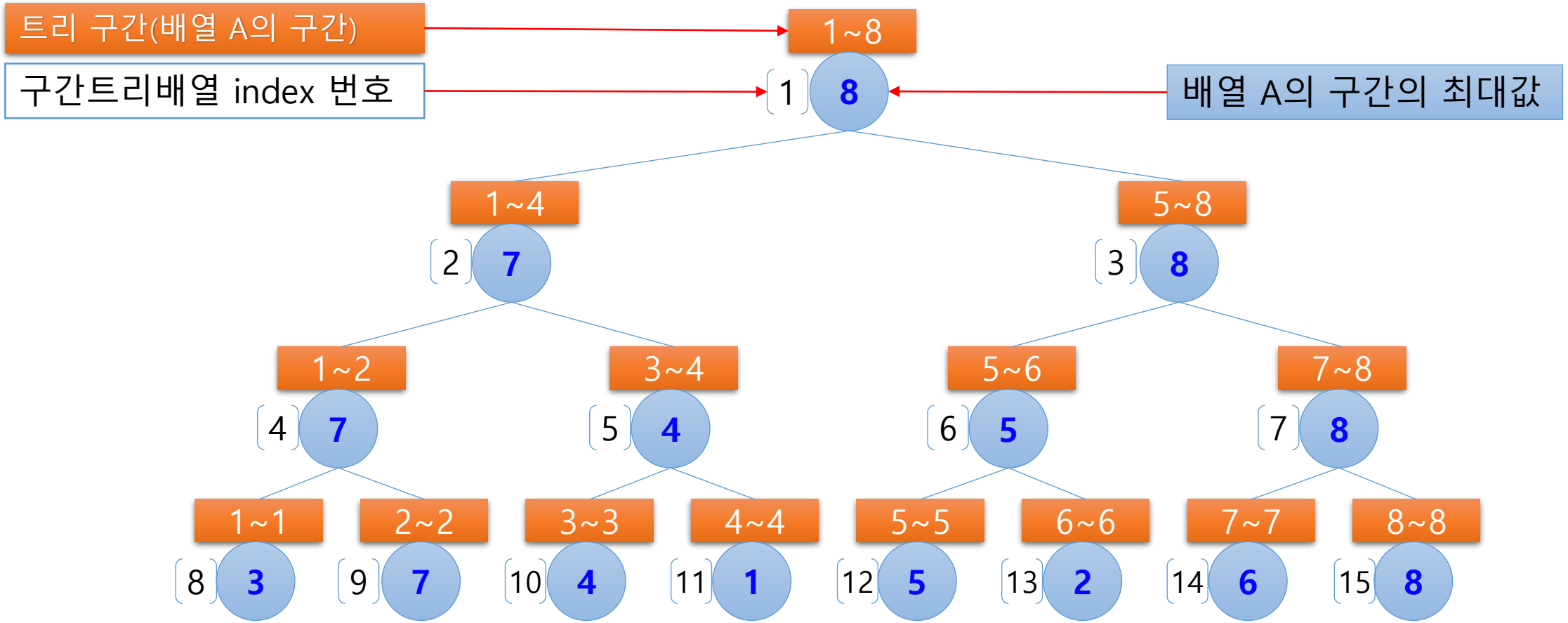
구간트리 Build - cont.

build(구간트리노드번호, 트리구간 시작, 트리구간 끝)



index	1	2	3	4	5	6	7	8
A[i]	3	7	4	1	5	2	6	8

구간트리 Build 완성 후의 모습



index	1	2	3	4	5	6	7	8
A[i]	3	7	4	1	5	2	6	8

Segment Tree – build code

```
/// build (트리노드번호, 구간시작, 구간 끝)
inline int Max(int a, int b){ return a>b ? a : b; }
void build (int now, int s, int e){
    if (s == e){                /// s == e인 경우 : base condition
        tree[now] = A[s];  /// scanf("%d", A + s);
        return;
    }
    int lch = now * 2, rch = lch + 1, m = (s + e) / 2;
    build(lch, s, m);                                   /// 왼쪽(앞쪽) 구간 트리 만들기
    build(rch, m + 1, e);                               /// 오른쪽(뒤쪽) 구간 트리 만들기
    tree[now] = Max(tree[lch], tree[rch]);    /// 현재 노드 구간 업데이트
}

void buildTree(){
    build(1, 1, N);
}
```

Segment Tree –update code sample

```
/// update(트리노드번호, 구간시작, 구간 끝, 목표 위치, 업데이트할 값)
inline int Max(int a, int b){ return a>b ? a : b; }
void update(int now, int s, int e, int tgIdx, int val){
    if (s == e){ /// s == e == tgIdx 인 경우
        tree[now] = val;
        return;
    }
    int lch = now * 2, rch = lch + 1, m = (s + e) / 2;
    if ( tgIdx <= m) update(lch, s, m, tgIdx, val); /// 찾는 위치가 왼쪽(앞쪽)에 있는 경우
    else update(rch, m + 1, e, tgIdx, val);          /// 찾는 위치가 오른쪽(뒤쪽)에 있는 경우
    tree[now] = Max(tree[lch], tree[rch]);          /// 현재 노드의 구간의 최대값 업데이트
}

void buildTree(){   /// update() 함수를 이용하여 구간 트리를 만들 수도 있다.
    for (int i=1; i<=N; ++i){
        update(1, 1, N, i, A[i]);
    }
}
```

2. 질의에 답하기 (query) – 구간의 최대값

❖ 구간트리가 완성된 후에 질의에 답해 보자.

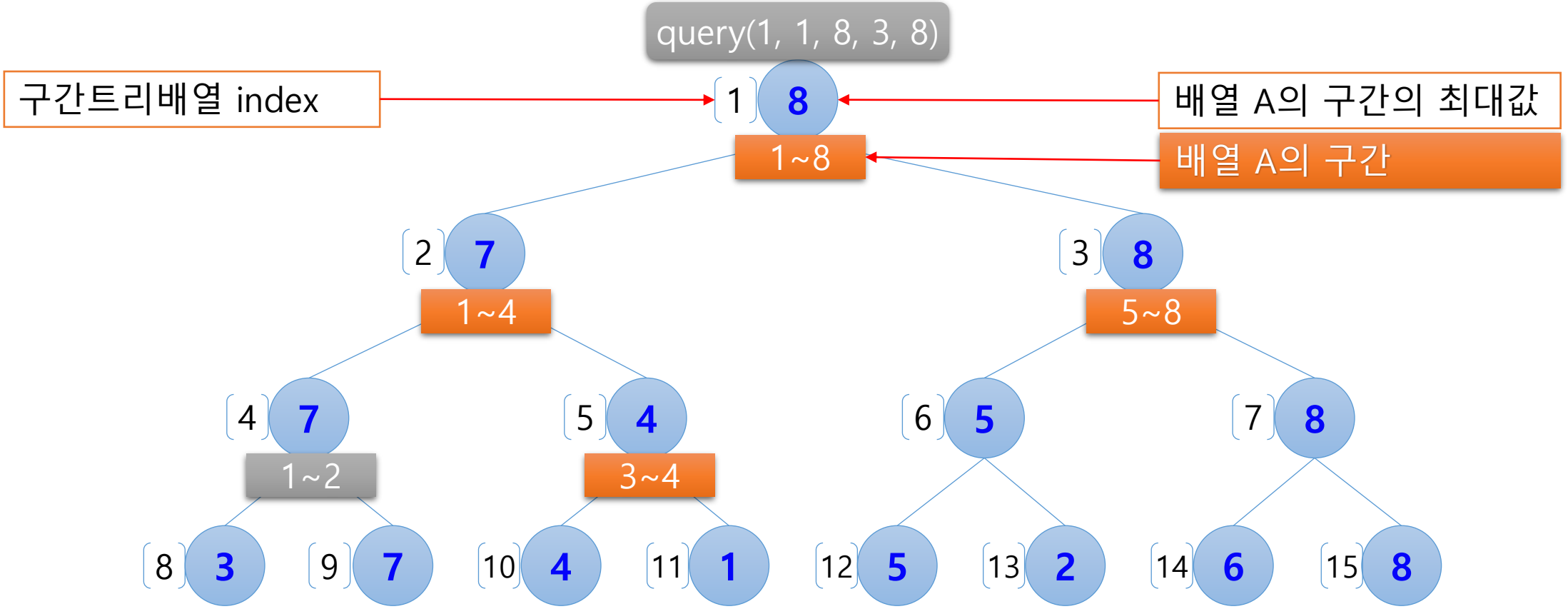
❖ query(트리 노드번호, 트리구간 시작, 트리구간 끝, 질의구간 시작, 질의구간 끝) 형태로 함수가 구성된다.

❖ 구간트리의 노드를 탐색할 때 각 노드는 다음 3가지 상태중 한 가지가 된다.

1. 트리구간이 질의 구간과 겹치는 부분이 없는 경우 : 최소값 반환
2. 트리구간이 질의 구간에 포함되는 경우 : 구간의 최대값 반환
3. 일부만 겹치거나 쿼리 구간이 트리구간보다 작아 트리구간에 포함되는 경우 : 구간을 나누어 재귀 호출

구간트리 – 질의에 답하기 예) getmax(A[3] ~ A[8])

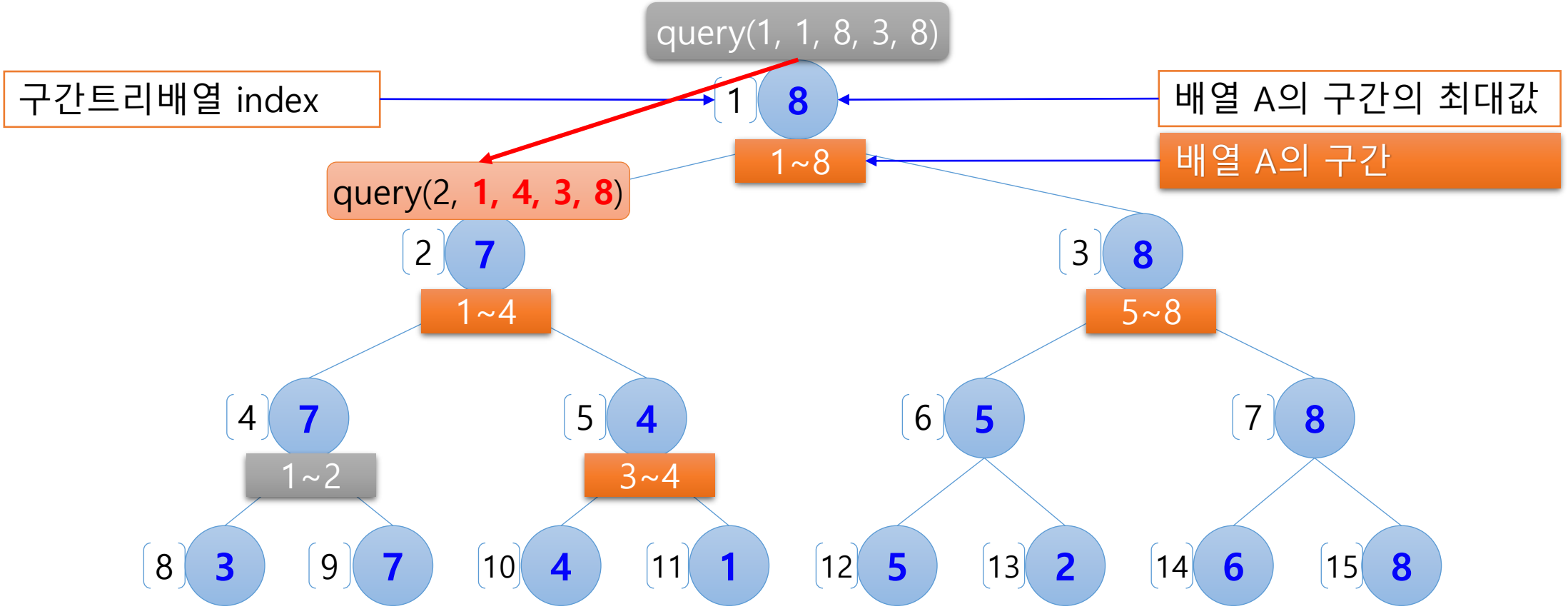
query(구간트리노드번호, 트리구간 시작, 트리구간 끝, 질의구간 시작, 질의구간 끝)



index	1	2	3	4	5	6	7	8
A[i]	3	7	4	1	5	2	6	8

구간트리 – 질의에 답하기 예) getmax(A[3] ~ A[8])

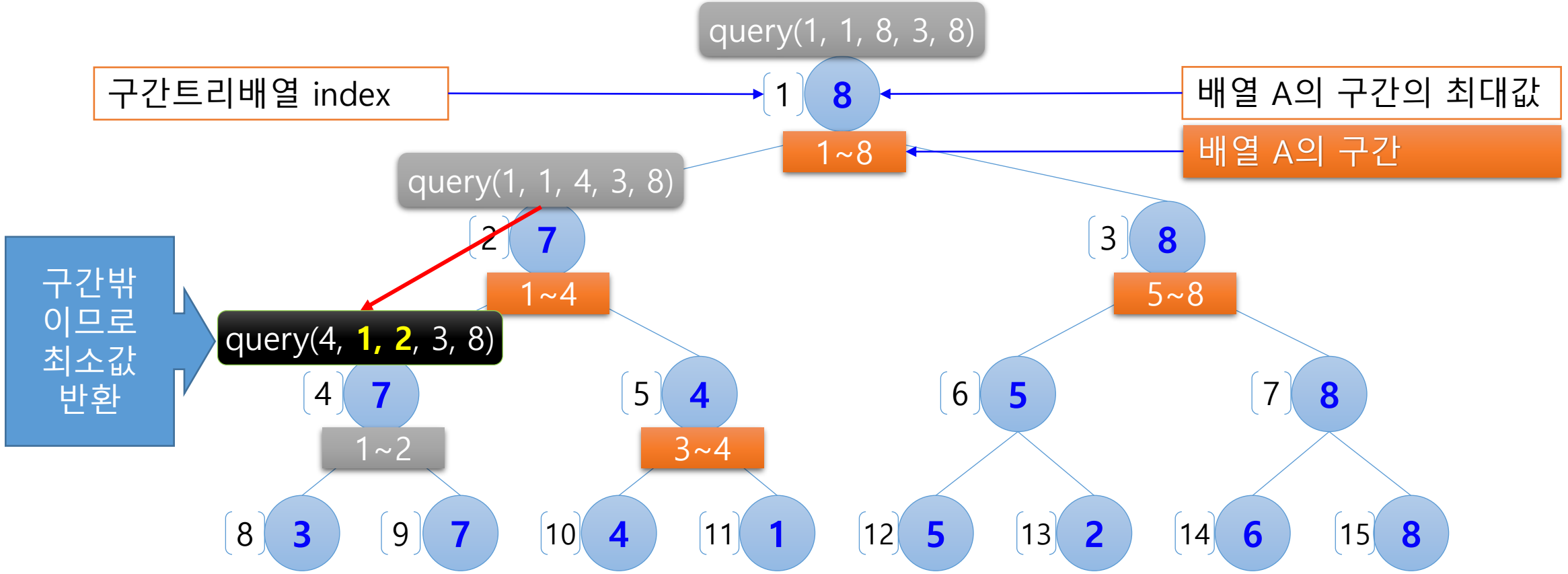
query(구간트리노드번호, 트리구간 시작, 트리구간 끝, 질의구간 시작, 질의구간 끝)



index	1	2	3	4	5	6	7	8
A[i]	3	7	4	1	5	2	6	8

구간트리 – 질의에 답하기 예) getmax(A[3] ~ A[8])

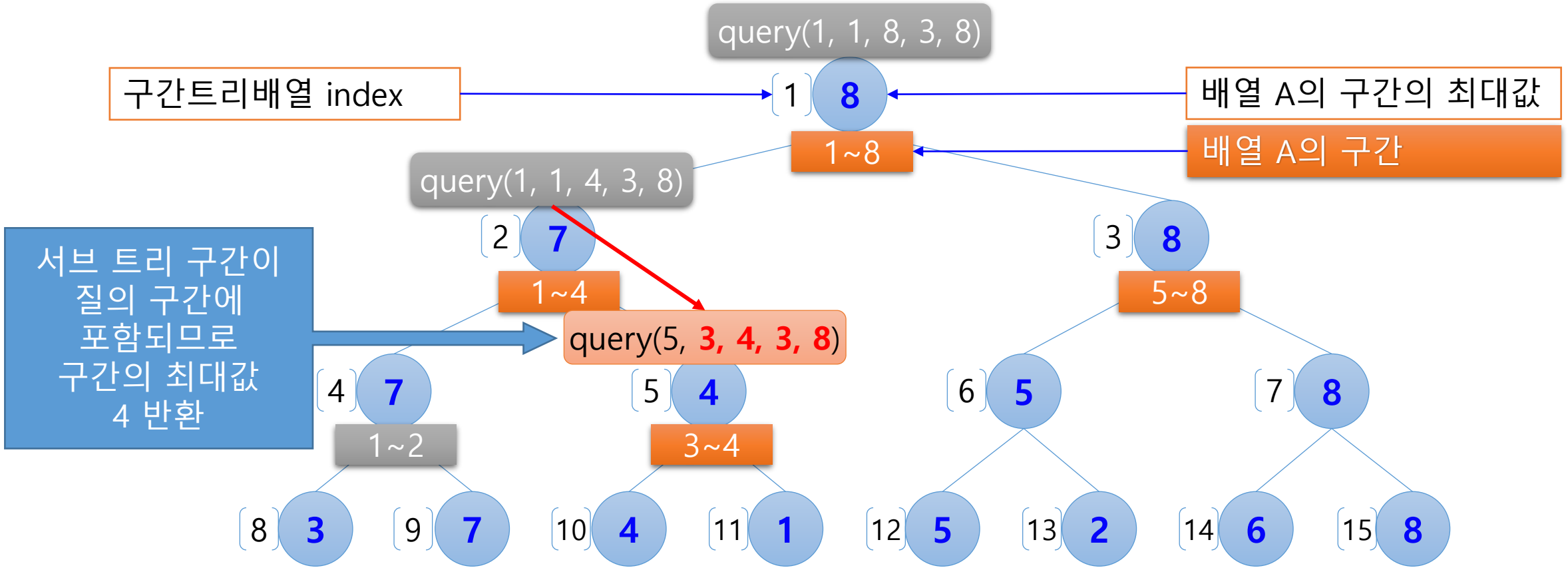
query(구간트리노드번호, 트리구간 시작, 트리구간 끝, 질의구간 시작, 질의구간 끝)



index	1	2	3	4	5	6	7	8
A[i]	3	7	4	1	5	2	6	8

구간트리 – 질의에 답하기 예) getmax(A[3] ~ A[8])

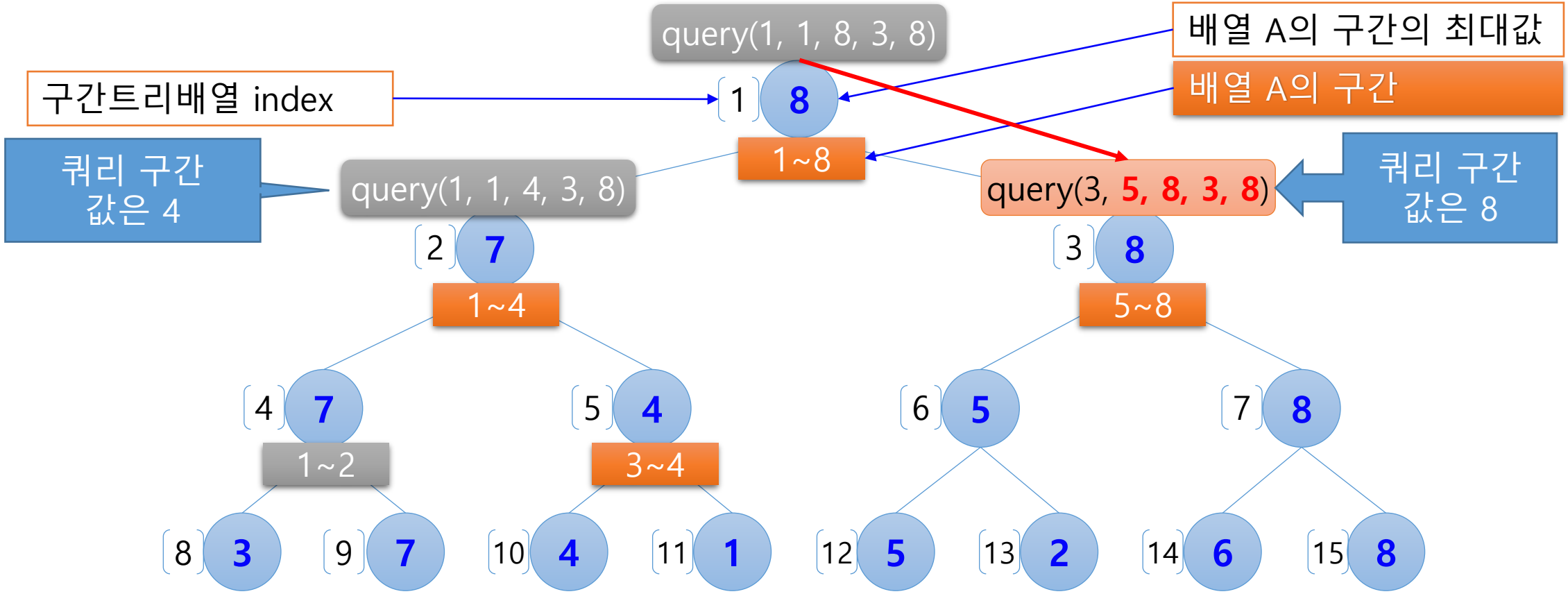
query(구간트리노드번호, 트리구간 시작, 트리구간 끝, 질의구간 시작, 질의구간 끝)



index	1	2	3	4	5	6	7	8
A[i]	3	7	4	1	5	2	6	8

구간트리 – 질의에 답하기 예) getmax(A[3] ~ A[8])

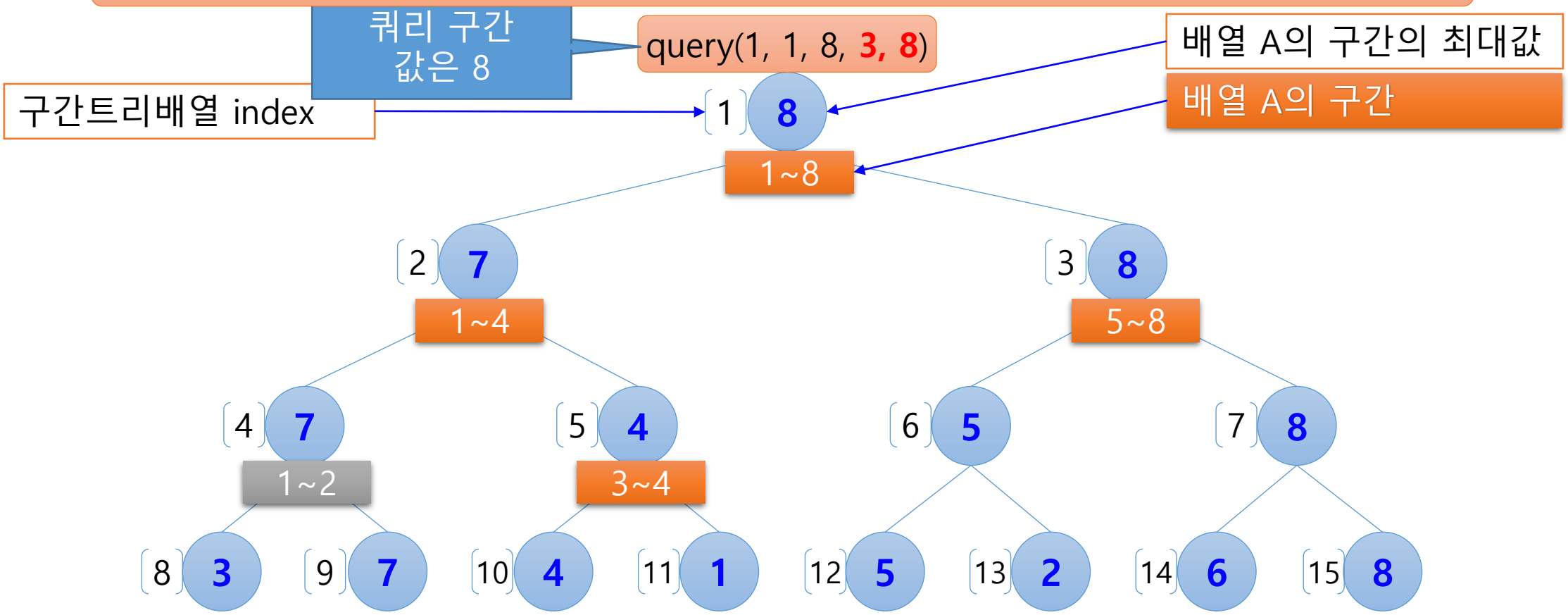
query(구간트리노드번호, 트리구간 시작, 트리구간 끝, 질의구간 시작, 질의구간 끝)



index	1	2	3	4	5	6	7	8
A[i]	3	7	4	1	5	2	6	8

구간트리 – 질의에 답하기 예) getmax(A[3] ~ A[8])

query(구간트리노드번호, 트리구간 시작, 트리구간 끝, 질의구간 시작, 질의구간 끝)



index	1	2	3	4	5	6	7	8
A[i]	3	7	4	1	5	2	6	8

Segment Tree – query code sample

```
int query(int now, int s, int e, int fs, int fe){
    if (e < fs || fe < s) return 0;
    if (fs <= s && e <= fe) return tree[now];
    int lch = now * 2, rch = lch + 1, m = (s + e) / 2;
    int left = query(lch, s, m, fs, fe);
    int right = query(rch, m + 1, e, fs, fe);
    return Max(left, right);
}
```

관련문제

- ❖ jungol 1726 구간의 최대값1
- ❖ jungol 3238 구간의 최대값2
- ❖ jungol 3239 구간의 최소값
- ❖ jungol 2615 공장 : 구간 합
- ❖ jungol 2469 줄세우기 : 구간 합
- ❖ SEC – api 문제
  - Jungol 3240 회원 참여도 분석1 : 구간의 최대 최소
  - Jungol 3219 회원 참여도 분석2 : 구간의 최대 최소
  - Jungol 3107 CD홀더 : 구간의 최대 최소

고찰

- ❖ Segment Tree를 링크드리스트 트리(동적 및 myAlloc)로 구현 해보기
- ❖ Indexed 트리
  : 포화이진트리를 만들어 구간트리문제를 푸는 방법이다.
- ❖ BIT(Binary Indexed Tree – Fenwick Tree)
  : bitwise 특성을 이용하여 N개의 공간만을 사용하여
  구간합을 효율적으로 업데이트 및 쿼리에 응답할 수 있다.
  구간합 문제에서는 탐색시간, 저장공간, 코드 구현에 있어
  탁월한 성능을 발휘한다.
- ❖ *RMQ(Range Minimum Query – 구간 최소 질의) 와*
  LCA(Least Common Ancestor – 최근접 공통 조상)
- ❖ *Lazy Propagation(게으른 전파)*

감사합니다.^^