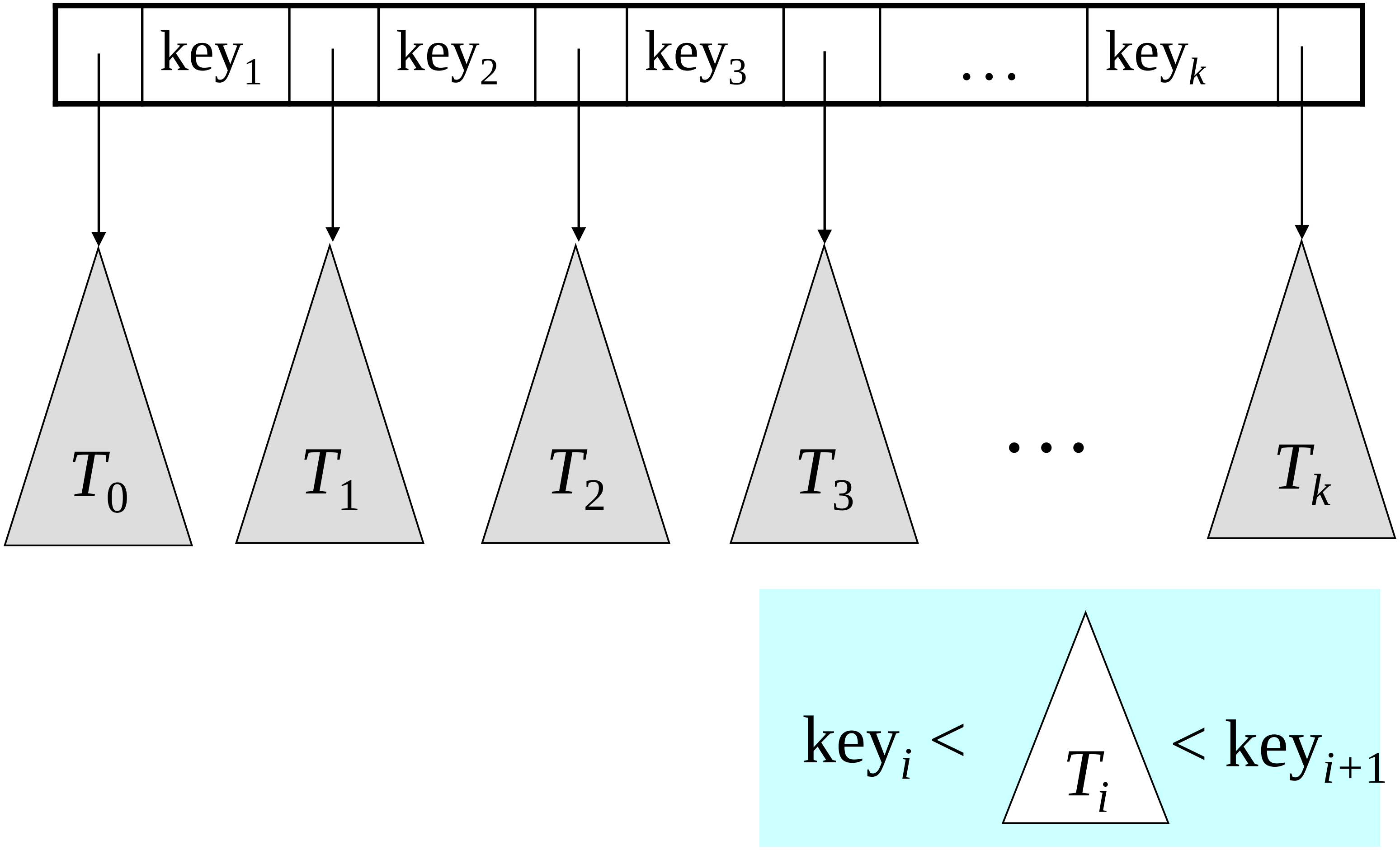
Ch. 14 B-Trees

B-Tree

- 디스크의 접근 단위는 블록(페이지)
- 디스크에 한 번 접근하는 시간은 수십만 명령어의 처리 시간과 맞먹는다
- 검색트리가 디스크에 저장되어 있다면 트리의 높이를 최소화하는 것이 유리하다
- B-트리는 다진검색트리가 균형을 유지하도록 하여 최악의 경우 디스크 접근 횟수를 줄인 것이다

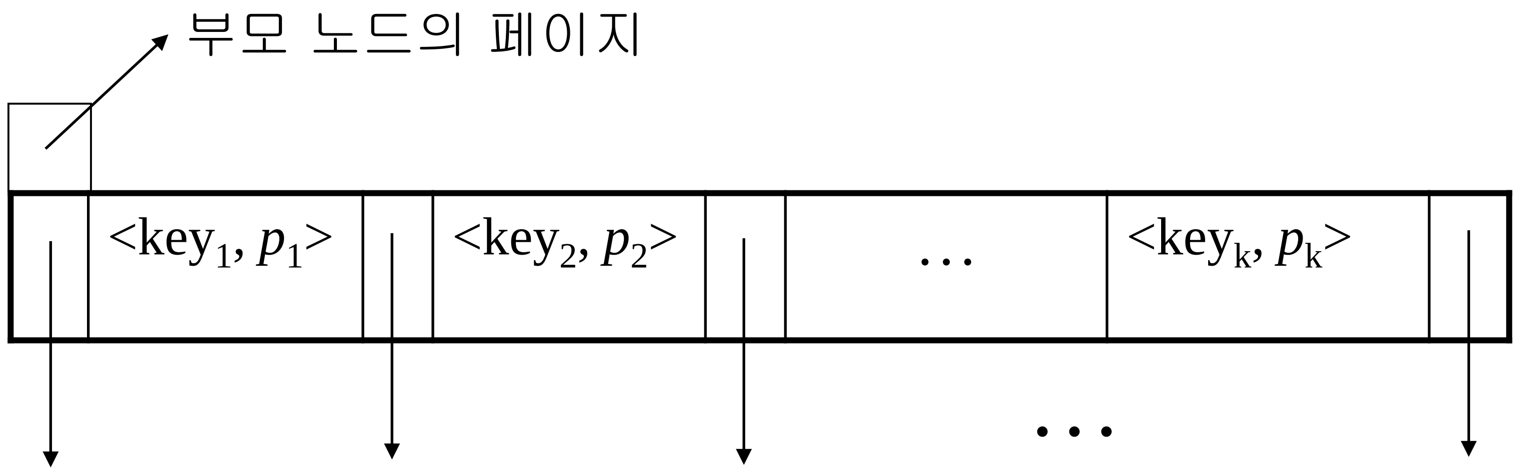
다진검색트리



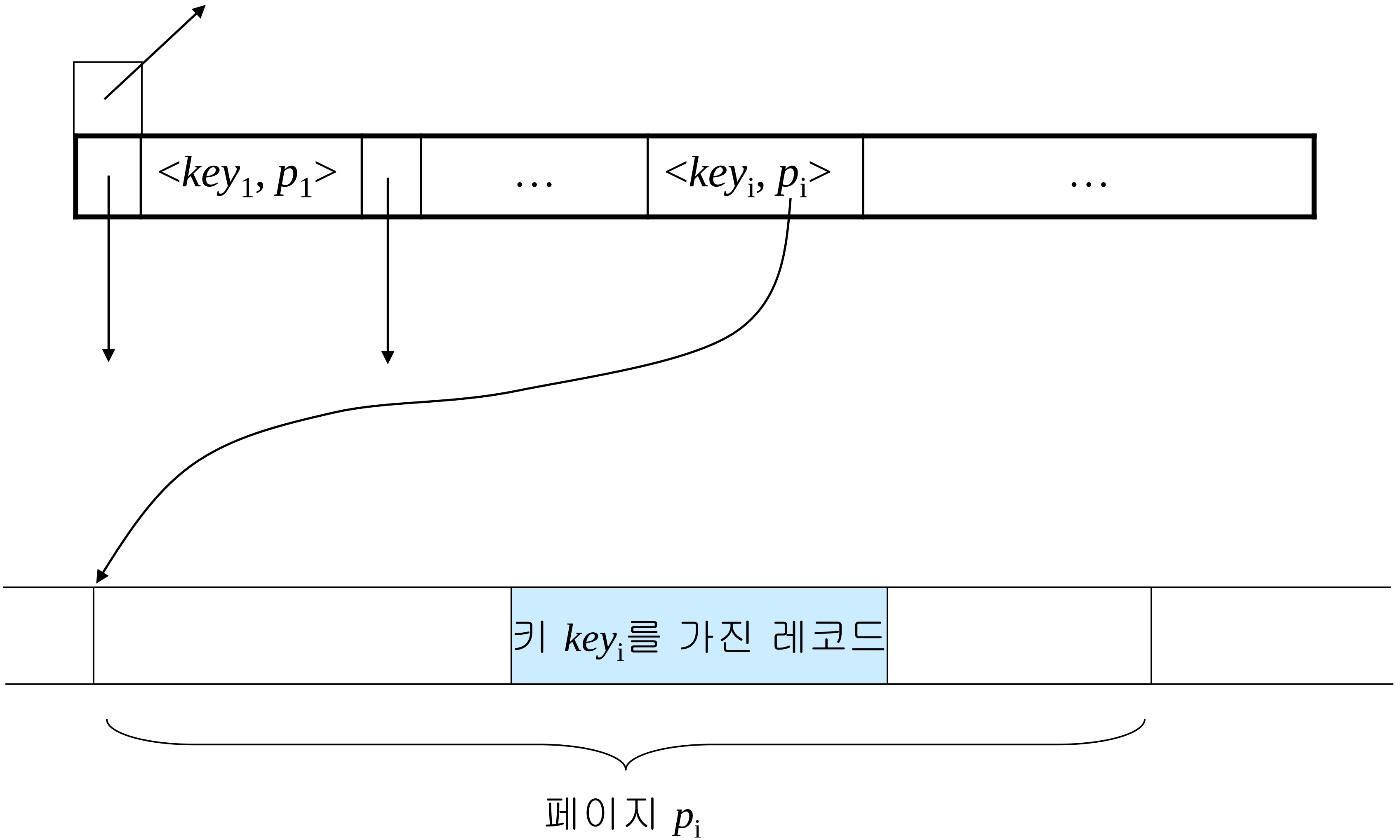
B-트리

- B-트리는 균형잡힌 다진검색트리로 다음의 성질을 만족한다
  - 루트를 제외한 모든 노드는 $\lfloor k/2 \rfloor \sim k$ 개의 키를 갖는다
  - 모든 리프 노드는 같은 깊이를 가진다

B-트리의 노드 구조



B-트리를 통해 레코드에 접근하는 과정

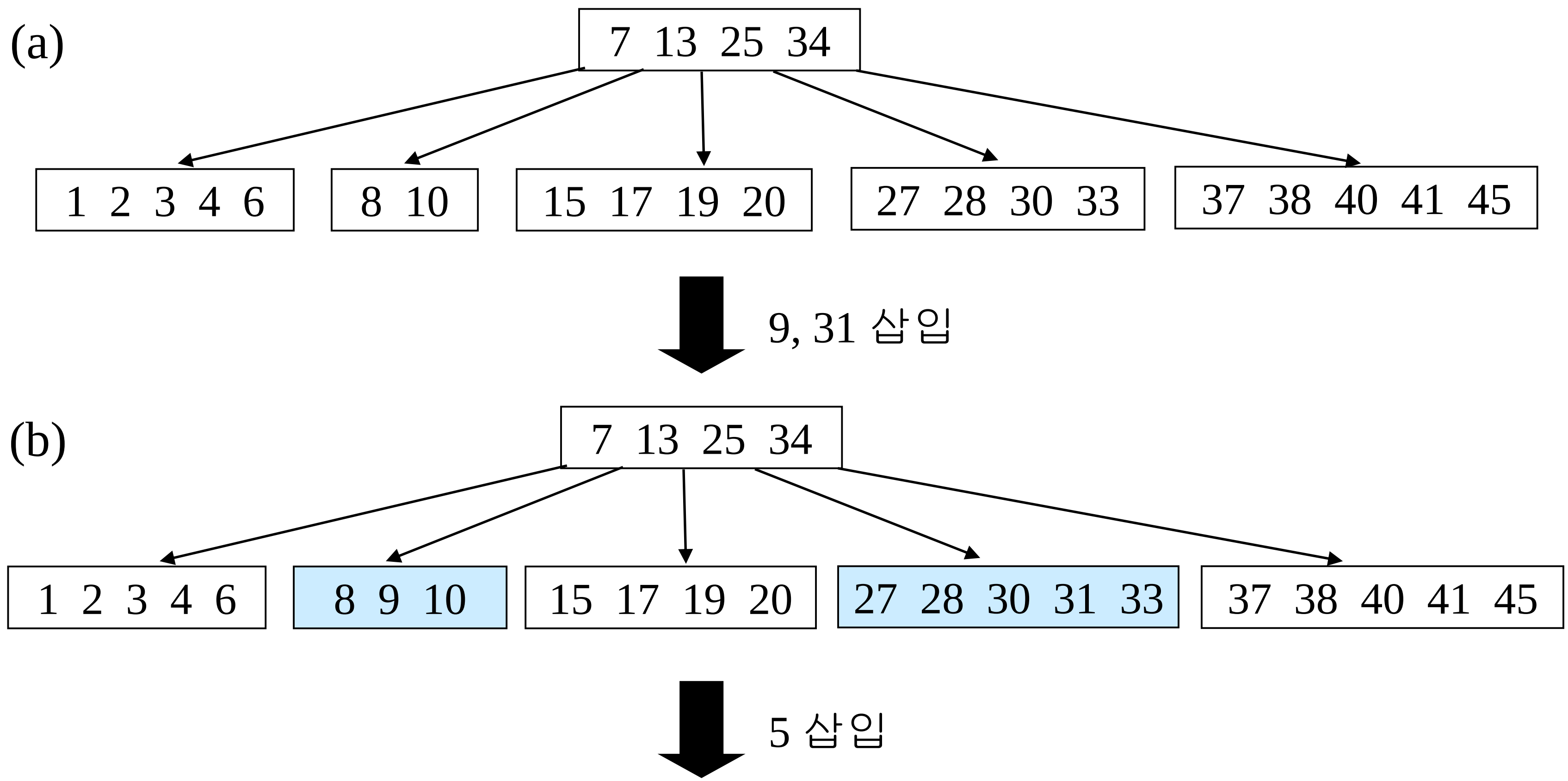


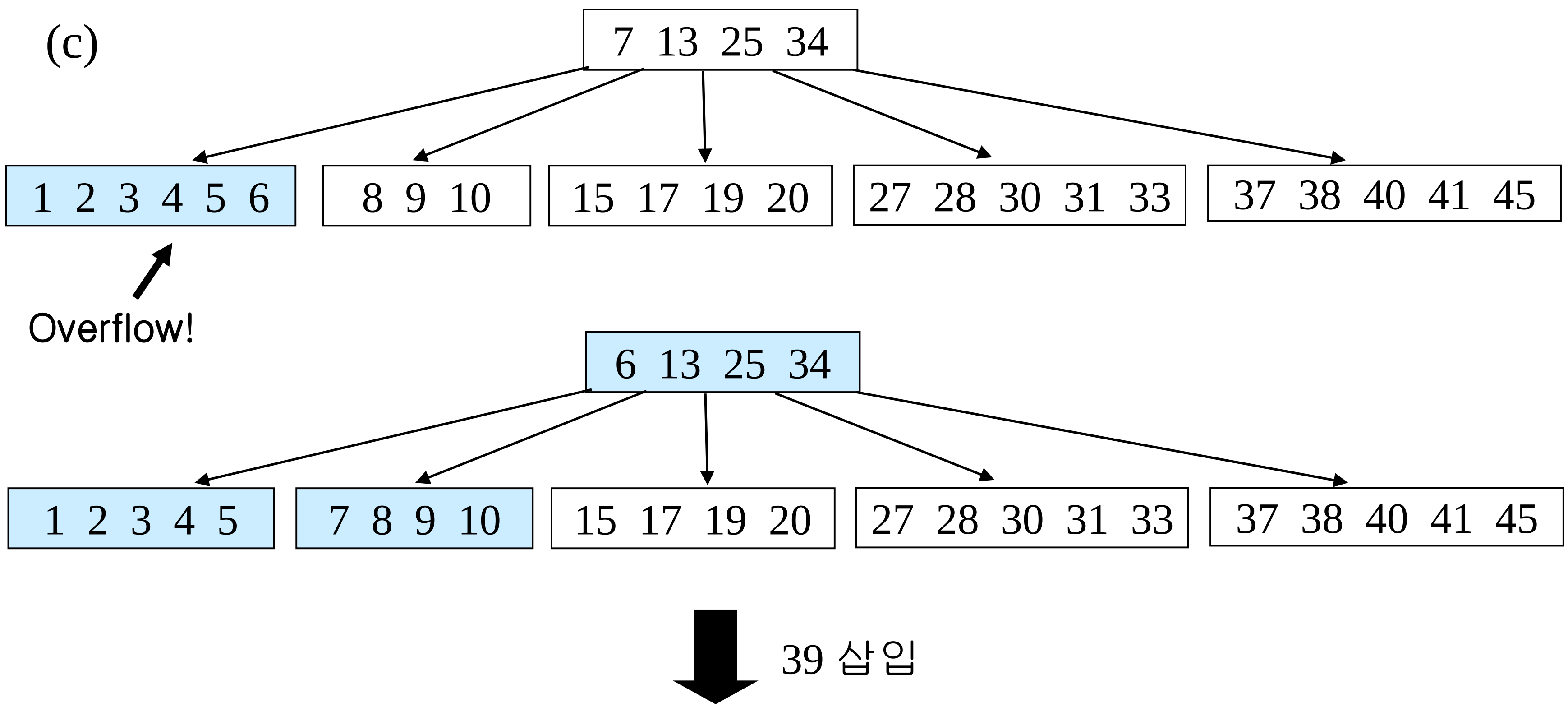
B-트리에서의 삽입

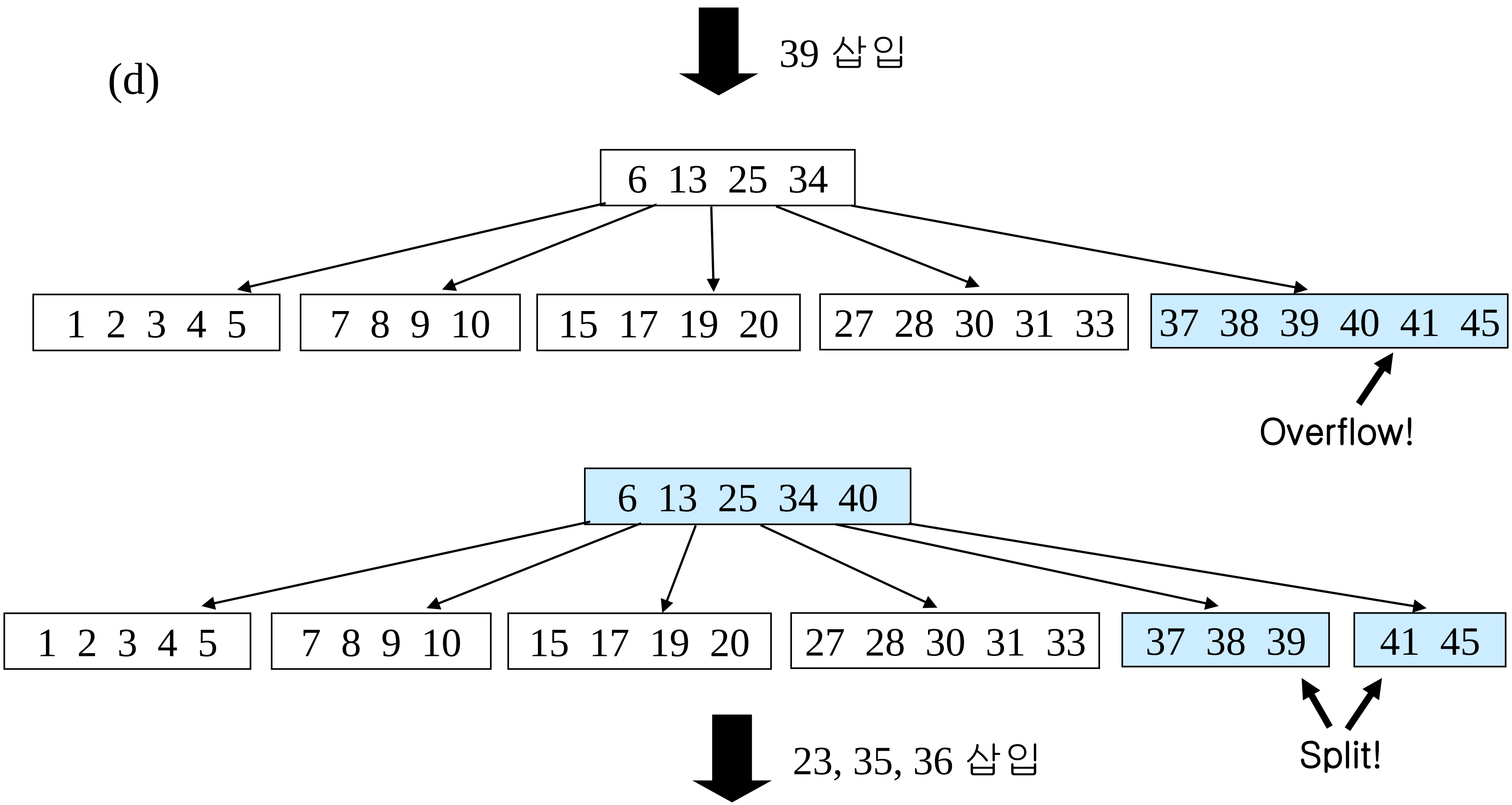
▷ t : 트리의 루트 노드
▷ x : 삽입하고자 하는 키

```
BTreeInsert(t, x)
{
        x를 삽입할 리프 노드 r을 찾는다;
        x를 r에 삽입한다;
        if (r에 오버플로우 발생) then clearOverflow(r);
}
clearOverflow(r)
{
    if (r의 형제 노드 중 여유가 있는 노드가 있음) then {r의 남는 키를 넘긴다};
    else {
            r을 둘로 분할하고 가운데 키를 부모 노드로 넘긴다;
            if (부모 노드 p에 오버플로우 발생) then clearOverflow(p);
    }
}
```

B-트리에서 삽입의 예

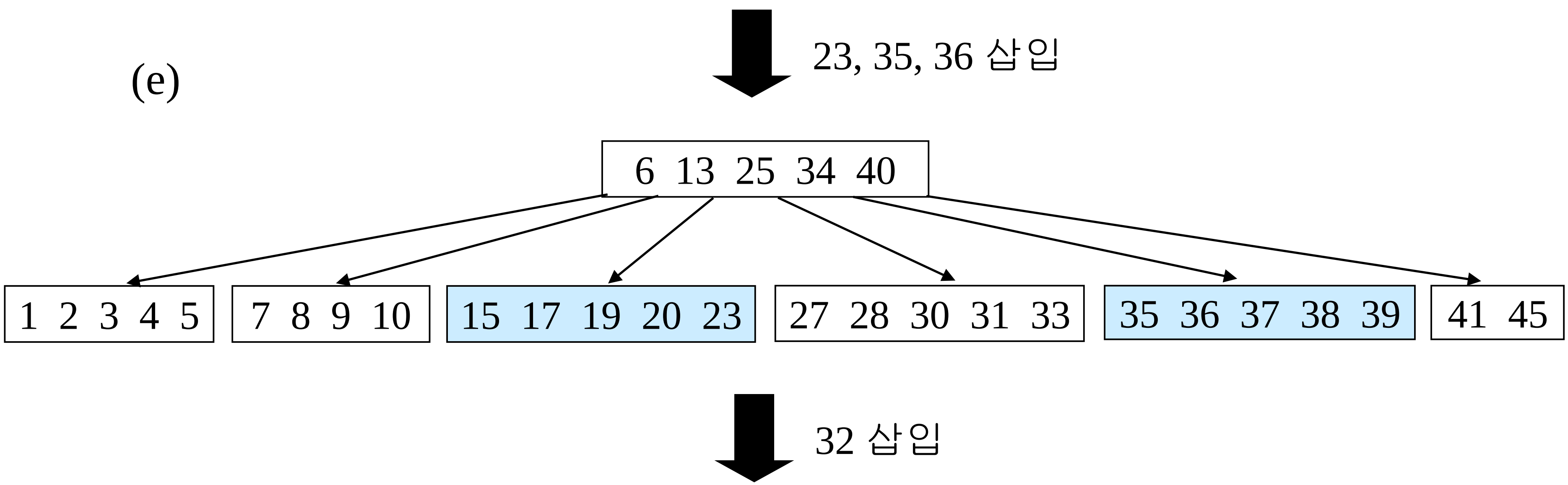

(a)
7 13 25 34
1 2 3 4 6
8 10
15 17 19 20
27 28 30 33
37 38 40 41 45
9, 31 삽입
(b)
7 13 25 34
1 2 3 4 6
8 9 10
15 17 19 20
27 28 30 31 33
37 38 40 41 45
5 삽입

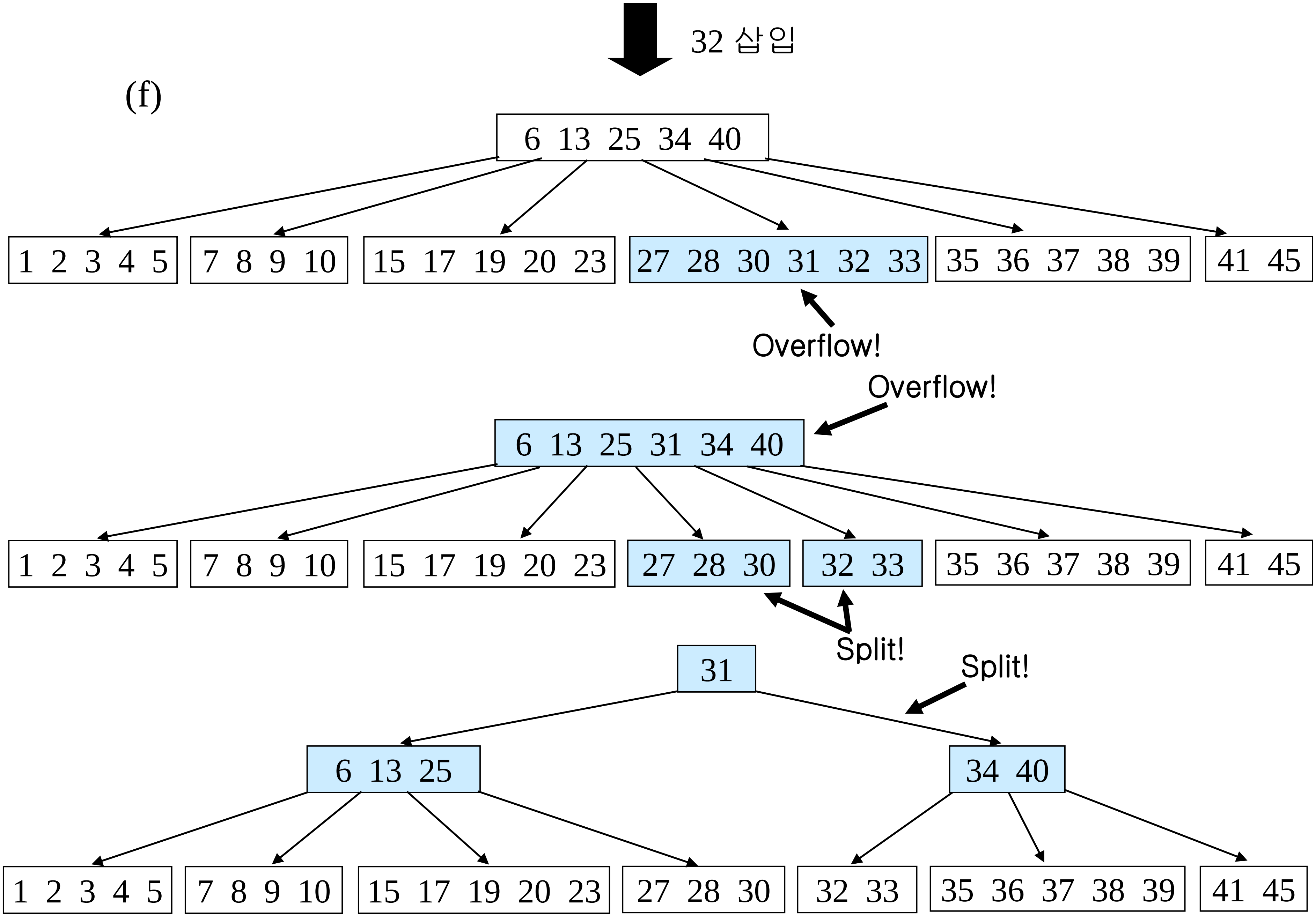

(c)
7 13 25 34
1 2 3 4 5 6
8 9 10
15 17 19 20
27 28 30 31 33
37 38 40 41 45
Overflow!
6 13 25 34
1 2 3 4 5
7 8 9 10
15 17 19 20
27 28 30 31 33
37 38 40 41 45
39 삽입



(d)
39 삽입
6 13 25 34
1 2 3 4 5
7 8 9 10
15 17 19 20
27 28 30 31 33
37 38 39 40 41 45
Overflow!
6 13 25 34 40
1 2 3 4 5
7 8 9 10
15 17 19 20
27 28 30 31 33
37 38 39
41 45
Split!
23, 35, 36 삽입

(e)

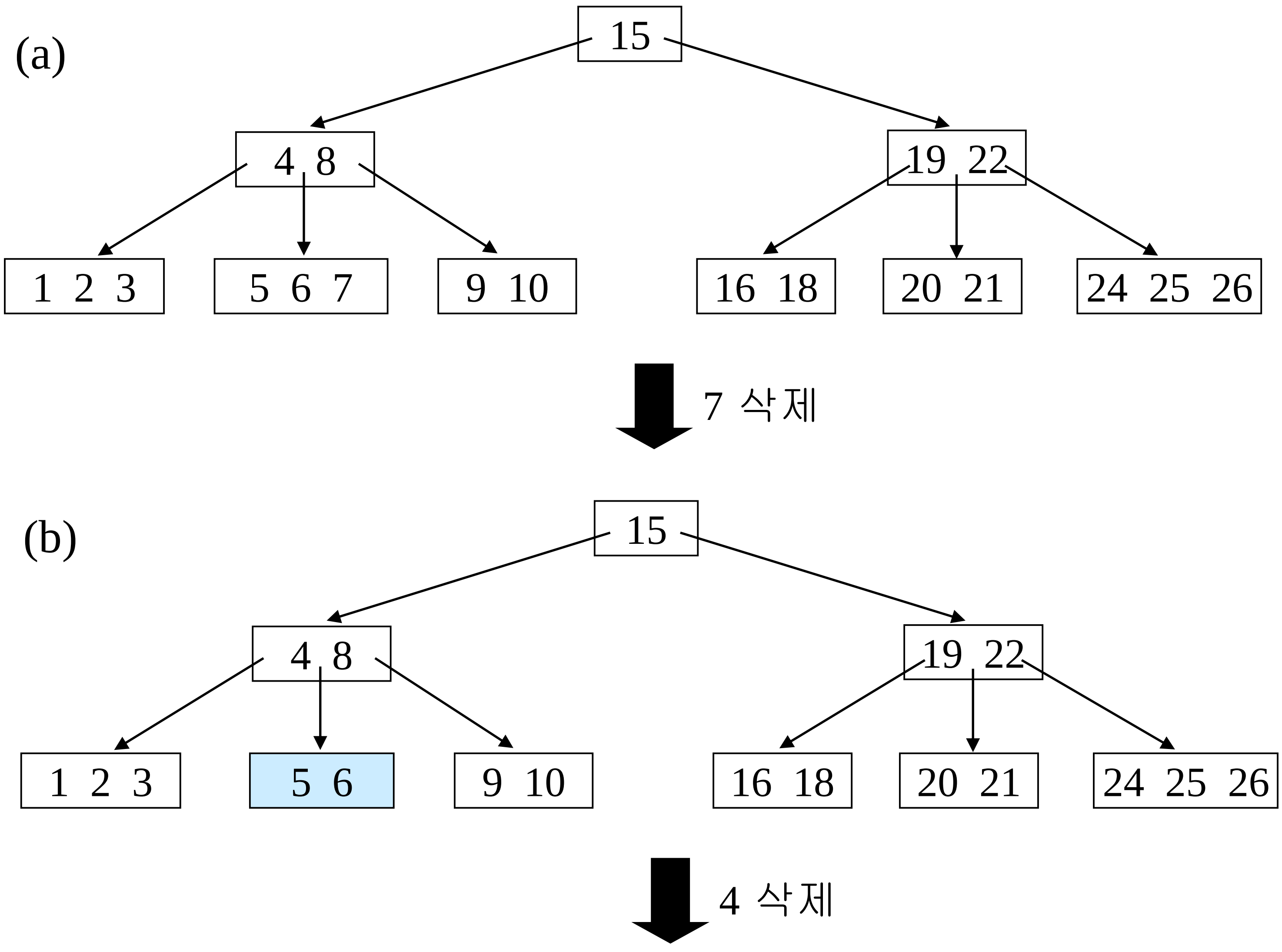

23, 35, 36 삽입
6 13 25 34 40
1 2 3 4 5
7 8 9 10
15 17 19 20 23
27 28 30 31 33
35 36 37 38 39
41 45
32 삽입

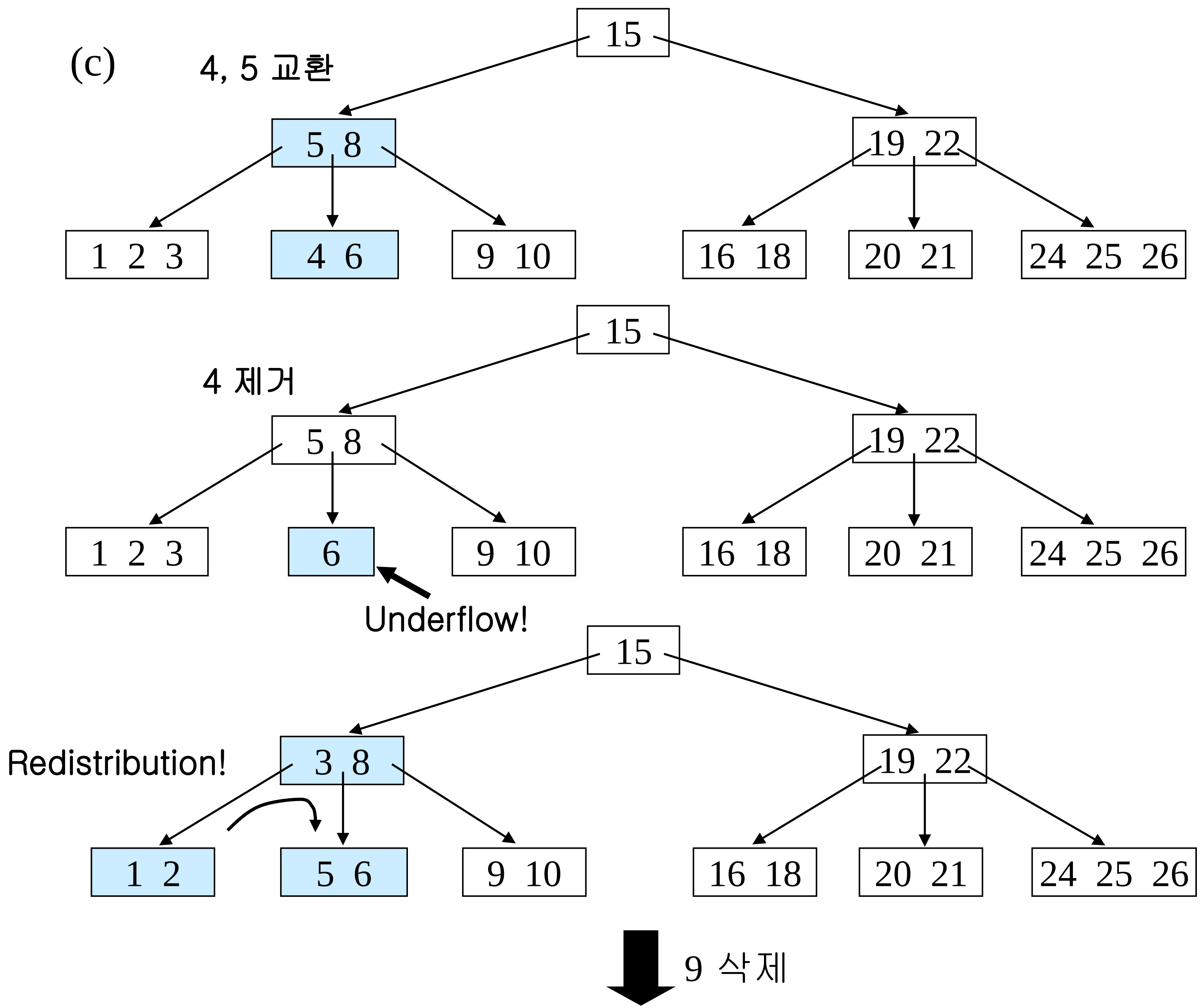

32 삽입
(f)
6 13 25 34 40
1 2 3 4 5
7 8 9 10
15 17 19 20 23
27 28 30 31 32 33
35 36 37 38 39
41 45
Overflow!
Overflow!
6 13 25 31 34 40
1 2 3 4 5
7 8 9 10
15 17 19 20 23
27 28 30
32 33
35 36 37 38 39
41 45
Split!
Split!
31
6 13 25
34 40
1 2 3 4 5
7 8 9 10
15 17 19 20 23
27 28 30
32 33
35 36 37 38 39
41 45

B-트리에서의 삭제

```
BTreeDelete(t, x, v)                                    ▷ t : 트리의 루트 노드
{                                                       ▷ x : 삭제하고자 하는 키
    if (v가 리프 노드 아님) then {                          ▷ v : x를 갖고 있는 노드
        x의 직후원소 y를 가진 리프 노드를 찾는다;
        x와 y를 맞바꾼다;
    }
    리프 노드에서 x를 제거하고 이 리프 노드를 r이라 한다;
    if (r에서 언더플로우 발생) then clearUnderflow(r);
}
clearUnderflow(r)
{
    if ( r의 형제 노드 중 키를 하나 내놓을 수 있는 여분을 가진 노드가 있음)
        then { r이 키를 넘겨받는다;}
        else {
                r의 형제 노드와 r을 합병한다;
                if (부모 노드 p에 언더플로우 발생) then clearUnderflow(p);
        }
}
```

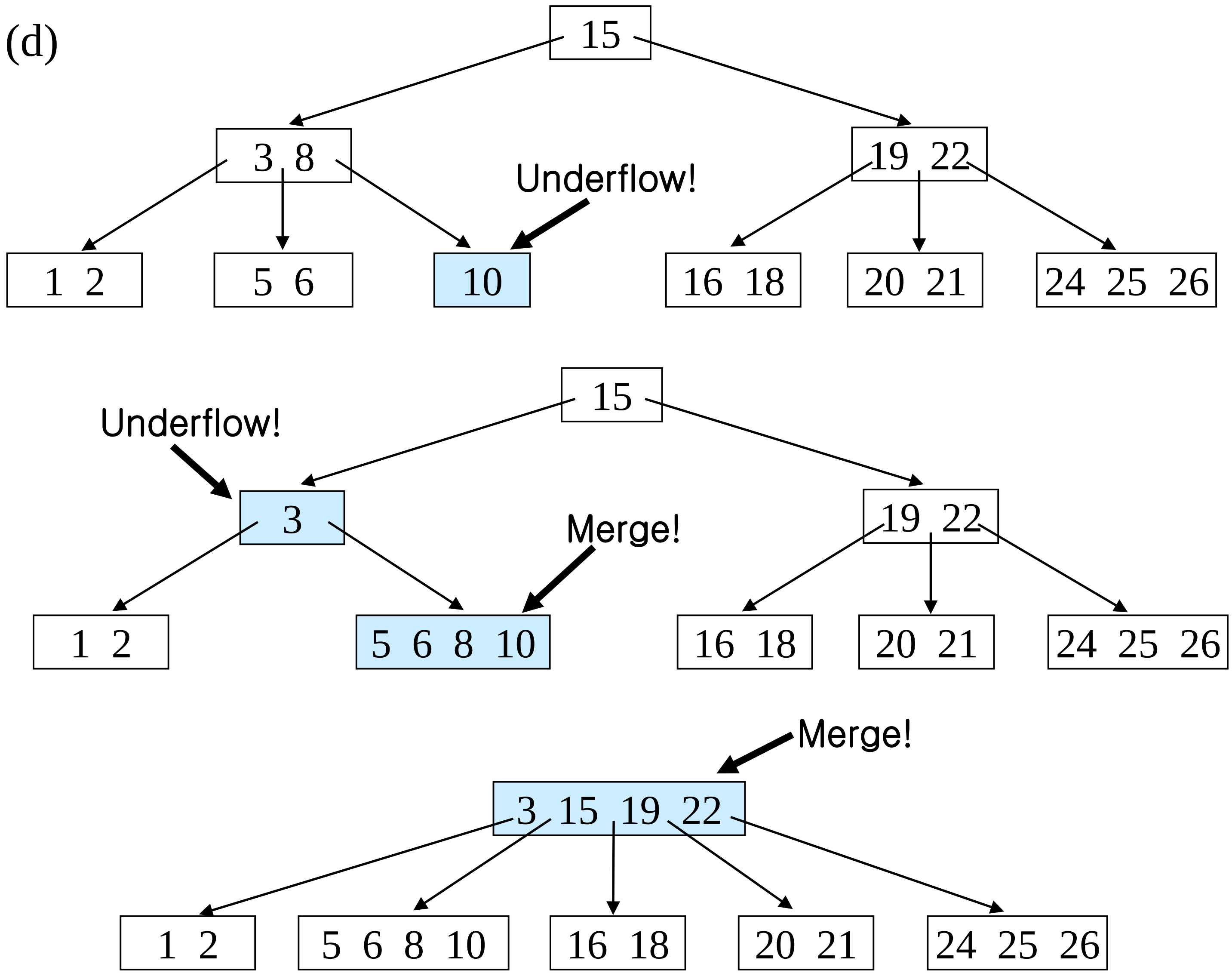
B-트리에서 삭제의 예


(a)
15
4 8
19 22
1 2 3
5 6 7
9 10
16 18
20 21
24 25 26
7 삭제
(b)
15
4 8
19 22
1 2 3
5 6
9 10
16 18
20 21
24 25 26
4 삭제



(c)
4, 5 교환
15
5 8
19 22
1 2 3
4 6
9 10
16 18
20 21
24 25 26
4 제거
15
5 8
19 22
1 2 3
6
9 10
16 18
20 21
24 25 26
Underflow!
15
Redistribution!
3 8
19 22
1 2
5 6
9 10
16 18
20 21
24 25 26
9 삭제

(d)


15
3 8
19 22
Underflow!
1 2
5 6
10
16 18
20 21
24 25 26
15
Underflow!
3
Merge!
19 22
1 2
5 6 8 10
16 18
20 21
24 25 26
Merge!
3 15 19 22
1 2
5 6 8 10
16 18
20 21
24 25 26