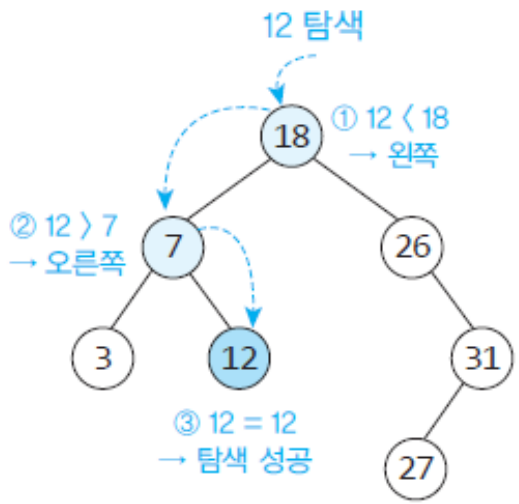
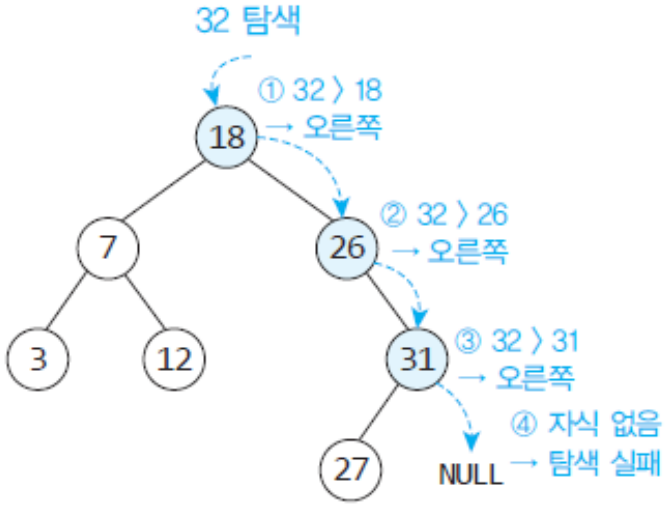
탐색 연산

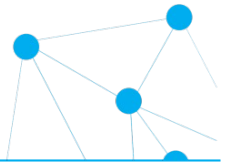


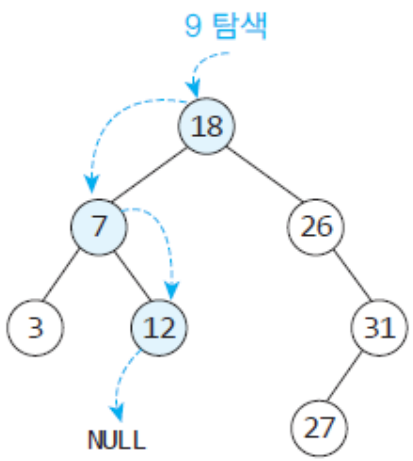


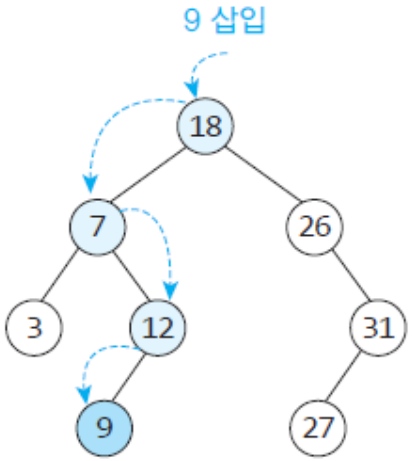
알고리즘 7.10 이진 탐색 트리의 탐색 알고리즘

```
01  search(root, key)
02      if root = NULL :
03          return NULL
04      if KEY(root) = key :                    // 루트의 키가 key와 같으면 탐색 성공
05          return root
06      else if KEY(root) < key :               // 루트의 키가 key보다 작으면
07          return search(root.right , key)     // 왼쪽 서브 트리 탐색
08      else :                                  // 루트의 키가 key보다 크면
09          return search(root.left  , key)     // 오른쪽 서브 트리 탐색
```

삽입 연산







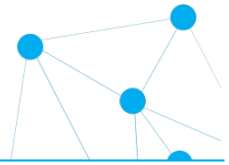
알고리즘 7.11 이진 탐색 트리의 삽입 알고리즘

```
01  insert(root, n)
02      if KEY(n) < KEY(root) :              // root보다 키가 작으면→ 왼쪽
03          if root.left = NULL :            // 왼쪽 자식이 없으면
04                root.left ← n              // n이 왼쪽 자식
05          else insert(root.left, n)        // 있으면 왼쪽 서브 트리에 삽입
06      else if KEY(n) > KEY(root) :         // root보다 키가 크면→ 오른쪽
07          if root.right = NULL :           // 오른쪽 자식이 없으면
08                root.right ← n             // n이 오른쪽 자식
09          else : insert(root.right , n)    // 있으면 오른쪽 서브 트리에 삽입
10      else :                               // 중복된 키가 있음.
11          delete_node(n)                   // 노드 n 삭제
```

알고리즘 7.14 최대 힙의 삭제 연산

$O(log_2 n)$

```
01  heap_pop()
02      root ← A[1]                          // 삭제할 루트 노드 저장
03      A[1] ← A[heap_size]                  // 말단 노드를 루트에 복사
04      heap_size ← heap_size - 1            // 노드의 수 감소
05      i ← 1                                // 루트의 위치
06      while LEFT(i) ≤ heap_size :          // 자식 노드가 남아 있을 때까지
07          if LEFT(i) < heap_size and KEY(LEFT(i)) < KEY(RIGHT(i)) :
08              child ← RIGHT(i)                                  자식 위치
09          else child ← LEFT(i)                                  쪽 자식 위치
10
11          if KEY(i) > KEY(child) : break   // 자식보다 크면 제자리 찾았음
12          else :                           // 자식이 더 크면
13              A[i] ↔ A[child]              // 자식과 교환
14              i ← child                    // 자식 위치로 내려옴
15      return root                          // 저장해 둔 루트를 반환
```



```c
//---------------------------------------------------------------------
// 코드 7.11 배열이 최대 힙인지 검사하기

int is_max_heap(HNode arr[], int len)
{
    for (int i = 1; i <= len / 2; i++)
        if (arr[i] < arr[LEFT(i)] || (RIGHT(i) <= len && arr[i] < arr[RIGHT(i)]))
            return 0;    // 크기 조건이 맞지 않음 -> 최대 힙이 아님
    return 1;            // 모든 노드에서 크기 조건 만족 -> 최대힙 맞음
}
```

```c
// 코드 7.11 테스트
int a[] = { 0, 9, 7, 6, 5, 4, 3, 2, 2, 1, 3 }; // 최대힙 맞음
int b[] = { 0, 9, 7, 6, 5, 3, 3, 2, 2, 1, 4 }; // 최대힙 아님
printf("a[]: 최대힙 %s\n", is_max_heap(a, 10) ? "맞음" : "아님");
printf("b[]: 최대힙 %s\n", is_max_heap(b, 10) ? "맞음" : "아님");
```