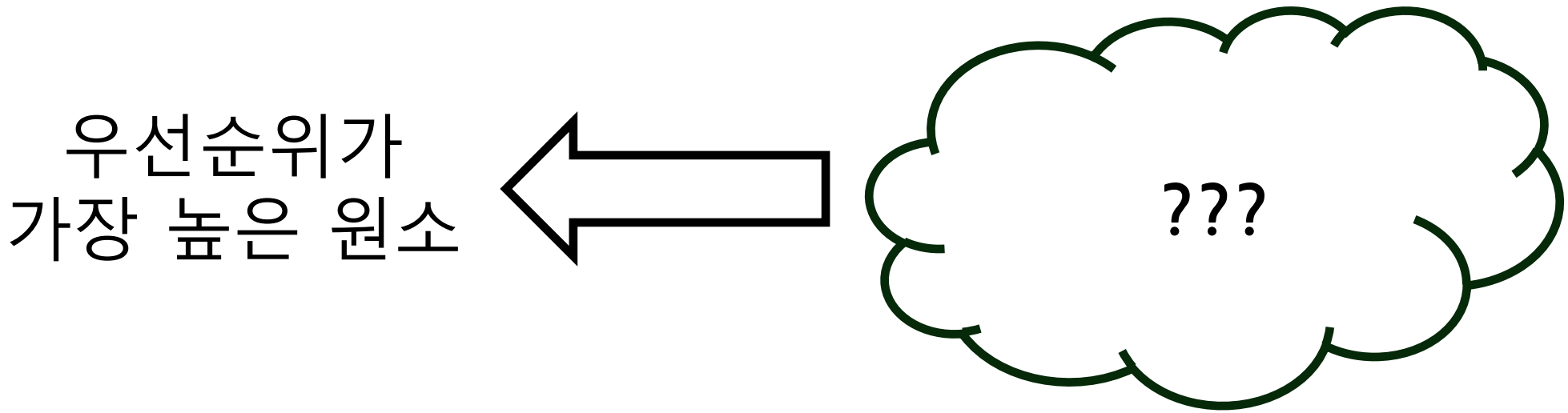
Heap & Priority Queue

즐겁고 알찬 자료구조 튜터링

Priority Queue

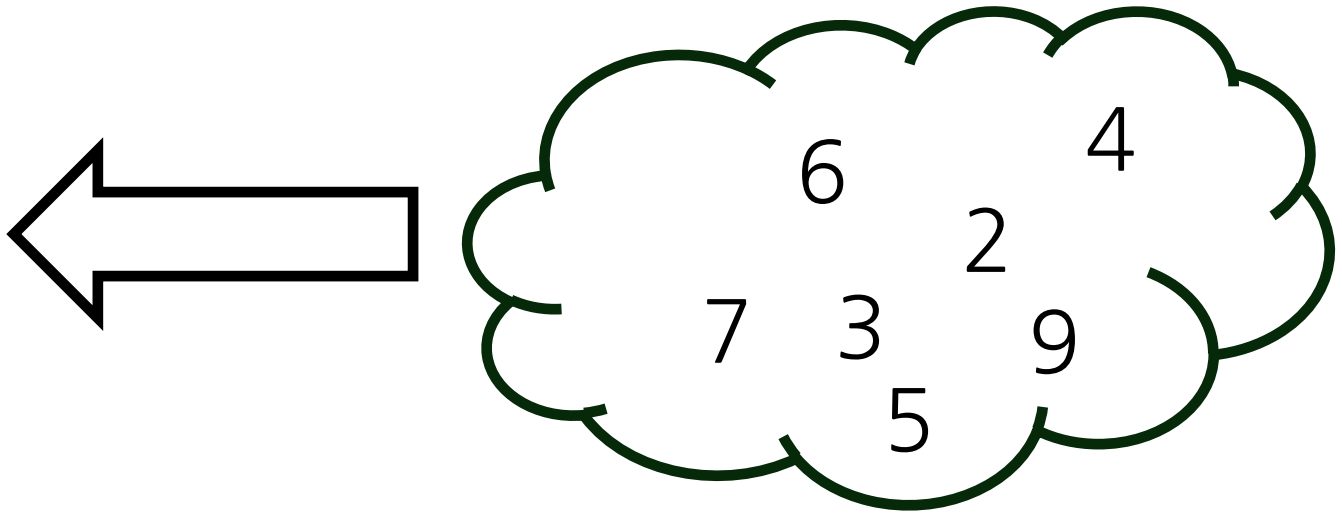
Priority Queue

- 우선순위가 가장 높은 원소가 먼저 참조 및 삭제



Priority Queue

2



Simple Priority Queue

```
1 template <typename T>
2 class SimplePriorityQueue {
3 private:
4     T *arr;
5     int capacity;
6     int n;
```

```
1 SimplePriorityQueue(int cap) {
2     capacity = cap;
3     arr = new T[capacity];
4     n = 0;
5 }
```

Simple Priority Queue

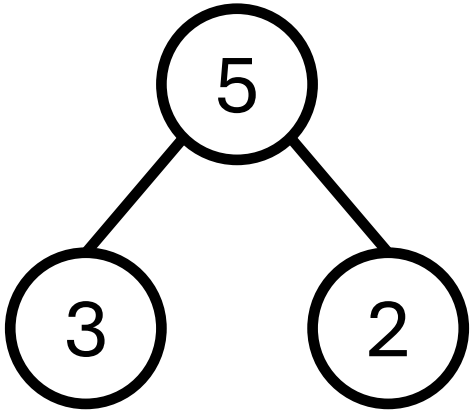
```
1 void push(const T e) {
2     arr[n++] = e;
3 }
4
5 void pop() {
6     n--;
7 }
```

```
1  T top() const {
2      T ret = arr[0];
3      for (int i = 1; i < n; i++) {
4          if (arr[i] < ret) {
5              ret = arr[i];
6          }
7      }
8
9      return ret;
10 }
```

Heap

Heap

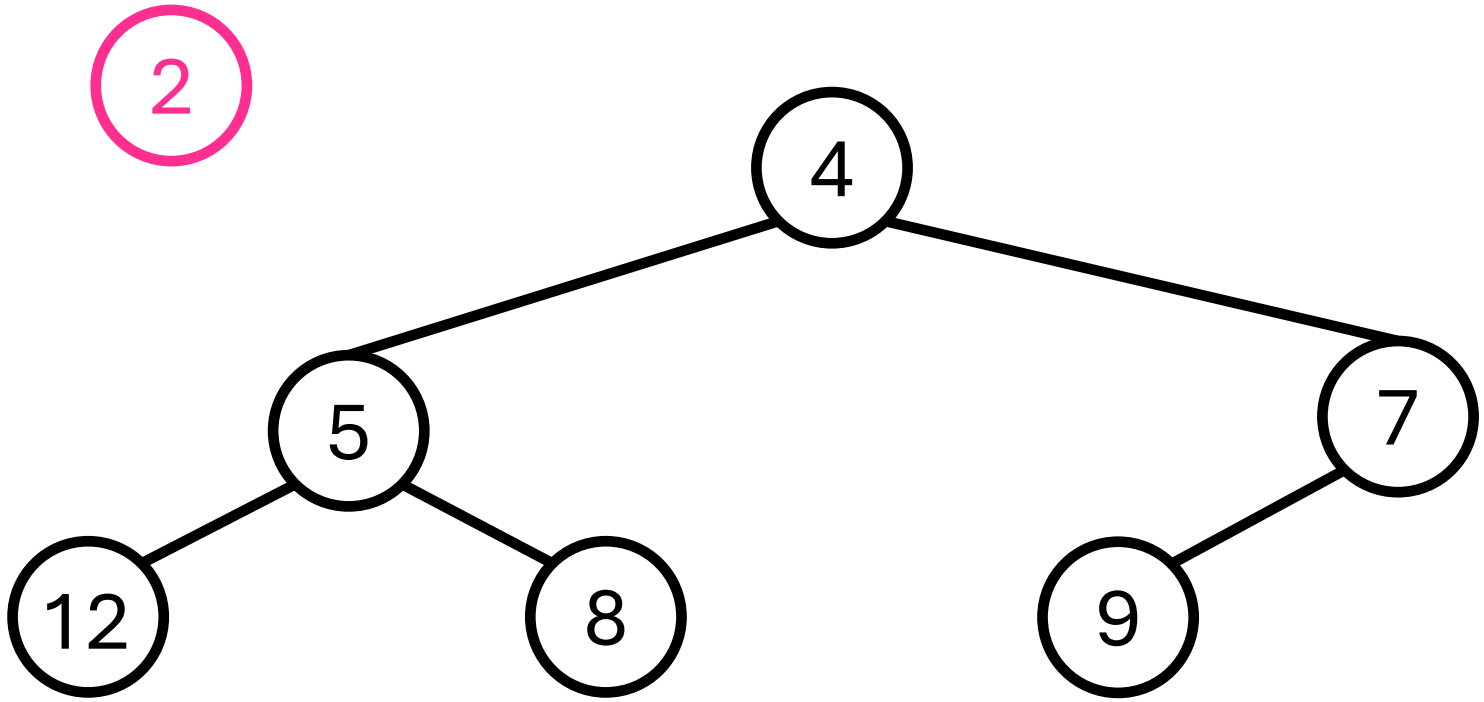
- 우선순위를 만족하고 있는 완전 이진 트리
  - 최대 힙 : 각 노드의 키 값이 그 자식의 키 값보다 작지 않은 트리
  - 최소 힙 : 각 노드의 키 값이 그 자식의 키 값보다 크지 않은 트리



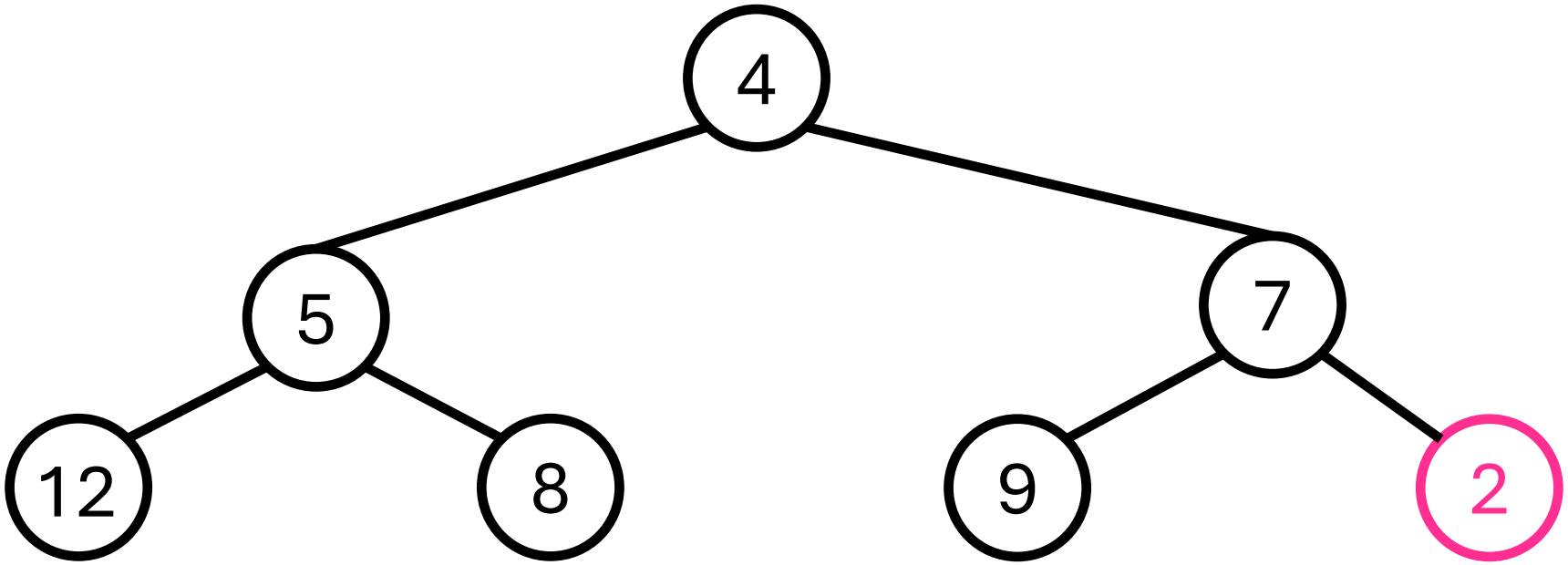
Insert Heap (Upheap)

- 완전 이진 트리를 유지하기 위해 삽입할 원소를 마지막 노드에 저장
  - 노드가 n개인 완전 이진 트리에서 n+1번 노드에 임시로 저장
- 완전 이진 트리 내에서 우선순위를 만족하기 위한 제자리 찾기
  - 현재 위치에서 부모와 비교하여 크기를 확인

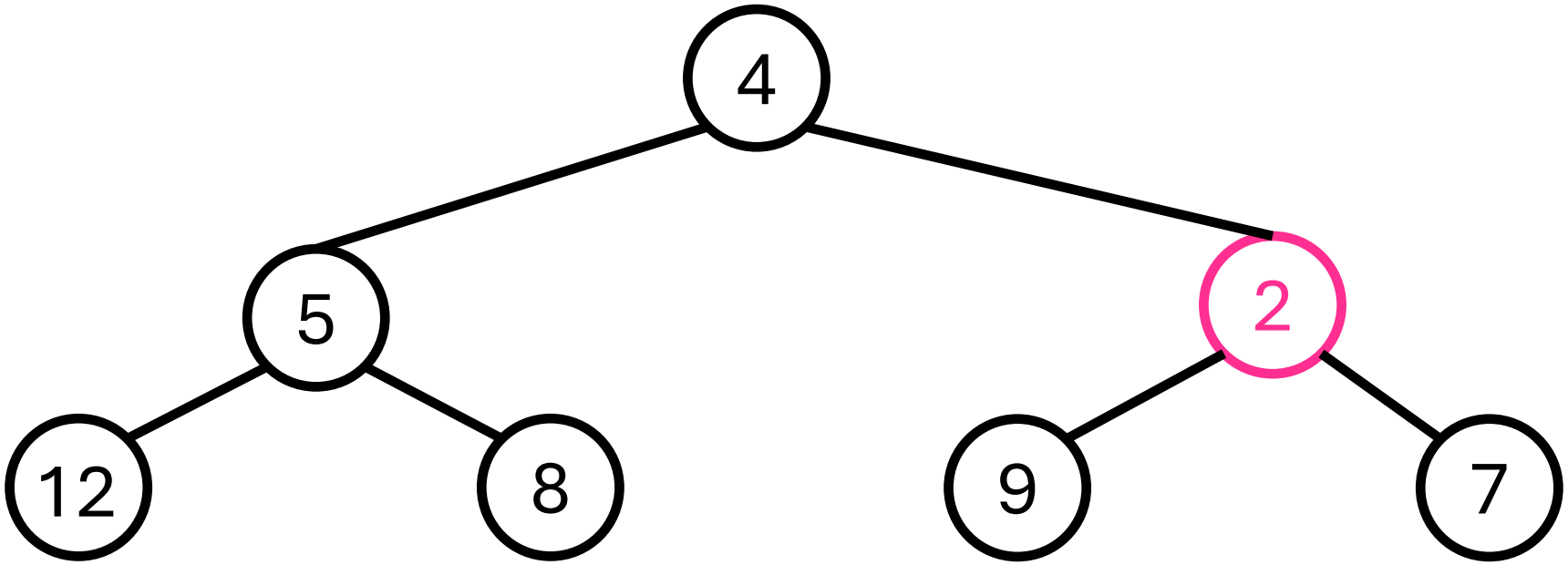
Insert Heap (Upheap)



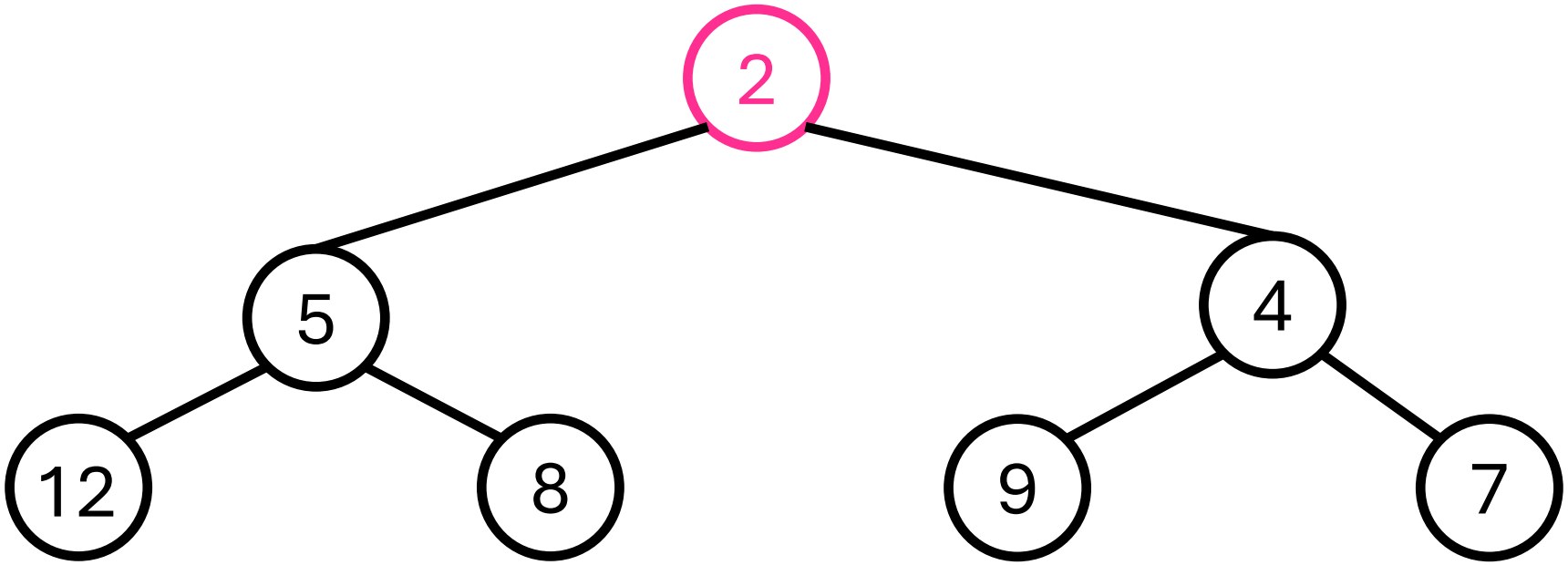
Insert Heap (Upheap)



Insert Heap (Upheap)



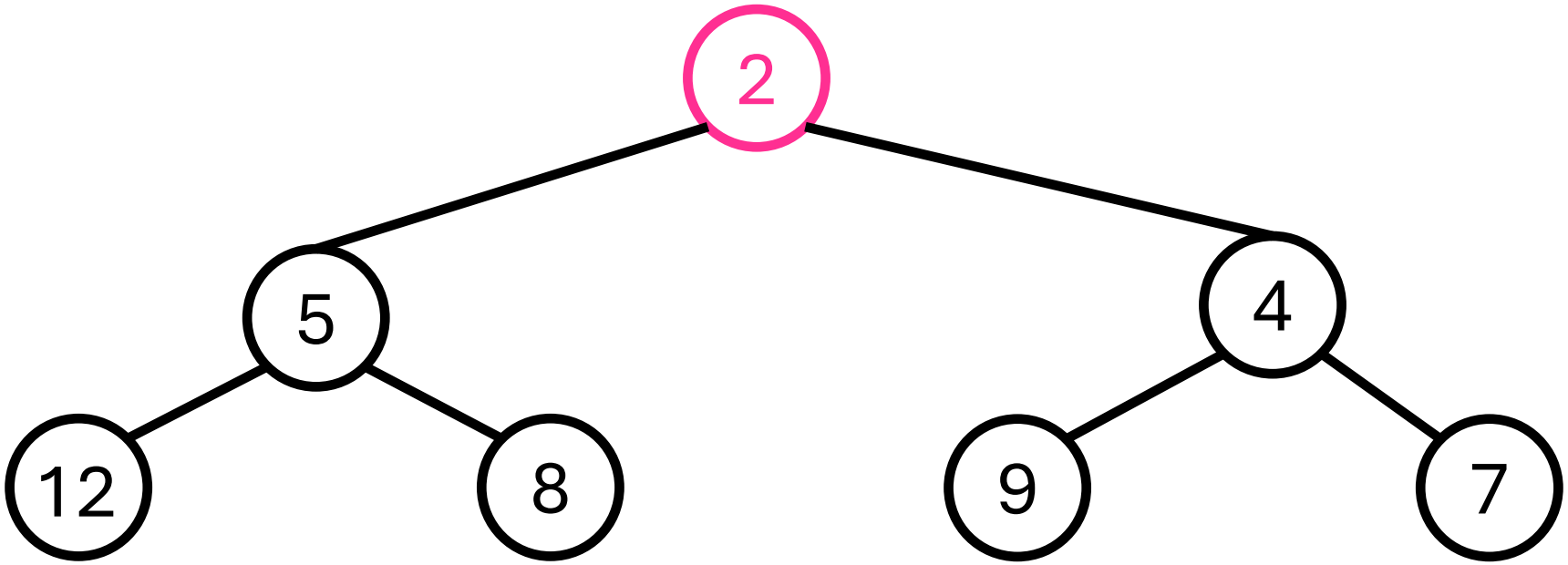
Insert Heap (Upheap)



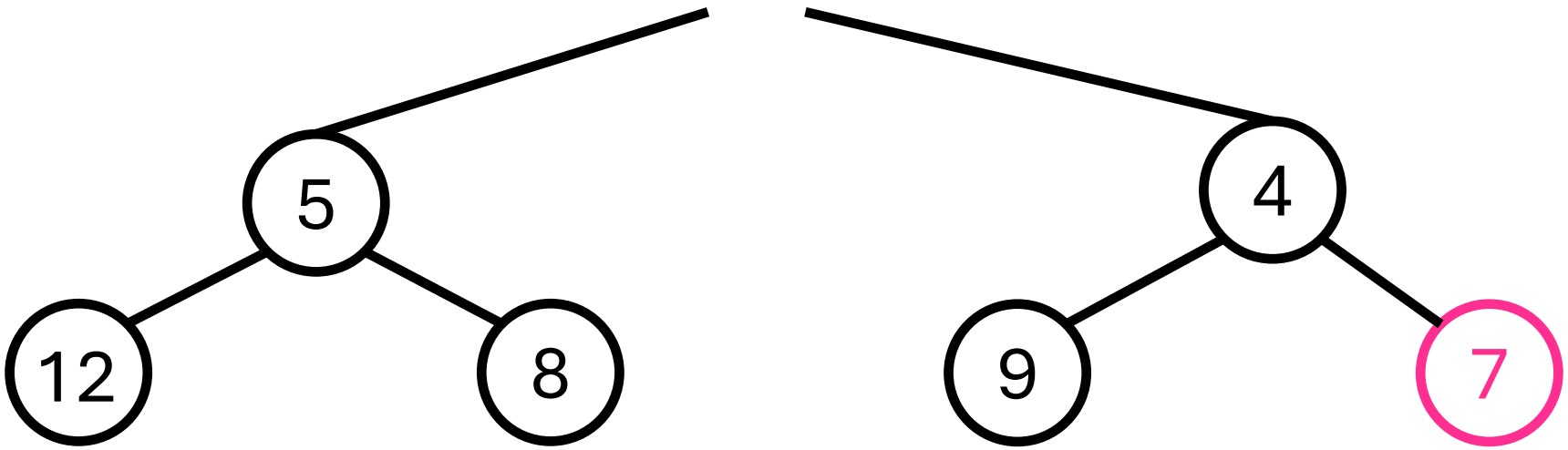
Delete Heap (Downheap)

- 힙에서 우선순위가 가장 높은 노드는 루트 노드이기 때문에 루트 노드를 삭제
- 가장 마지막 번호인 노드를 루트 노드로 이동
  - n번 노드의 번호를 1번으로 변경
- 완전 이진 트리 내에서 우선순위를 만족하기 위한 제자리 찾기
  - 현재 위치에서 자식과 비교하여 크기를 확인

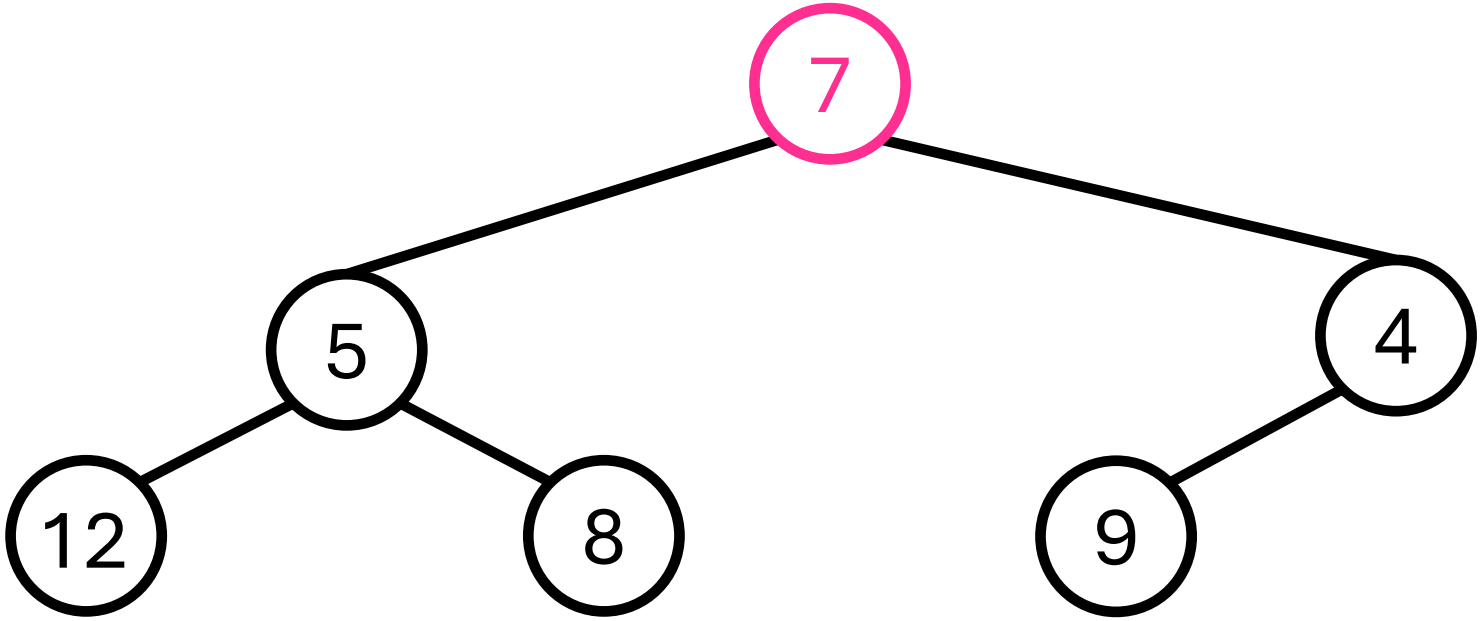
Delete Heap (Downheap)



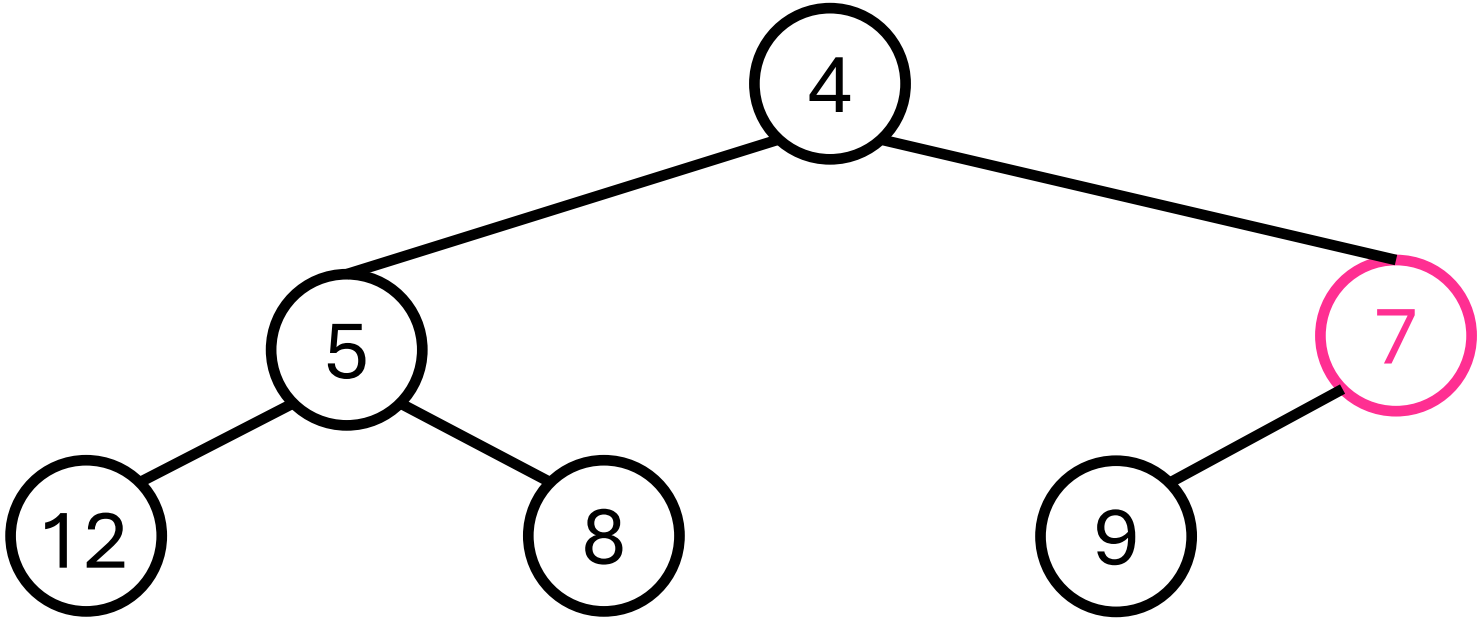
Delete Heap (Downheap)



Delete Heap (Downheap)



Delete Heap (Downheap)



Comparator

Comparator

- 우선순위 결정??
  - 기본 자료형은 비교 연산자 사용이 가능하기 때문에 자동적으로 처리 (int, double, string)
  - 사용자 정의형은 연산자 오버로딩을 통해 비교 연산이 가능하도록 함
  - 자료구조를 선언할 때 comparator를 등록

Functor (function object)

- 함수처럼 동작하는 객체
- 함수처럼 동작하기 위해 객체가 ( ) 연산자를 오버로딩
- 함수 객체를 사용하면 함수가 상태를 가질 수 있음

Functor (function object)

```
1 int less(int a, int b) {
2     return a < b;
3 }
4
5 struct Less {
6     bool operator()(int a, int b) {
7         return a < b;
8     }
9 };
```

Functor (function object)

```
1 int less(int a, int b) {
2     return a < b;
3 }
4
5 struct Less {
6     bool operator()(int a, int b) {
7         return a < b;
8     }
9 };
```

Functor (function object)

```
1 int main() {
2     less(1, 2);
3     Less()(1, 2);
4
5     Less cmp;
6     cmp(1, 2);
7     // cmp.operator()(1, 2);
8 }
```

Functor (function object)

Template parameters

`T`

Type of the elements.

Aliased as member type `priority_queue::value_type`.

`Container`

Type of the internal ***underlying container*** object where the elements are stored.

Its `value_type` shall be `T`.

Aliased as member type `priority_queue::container_type`.

`Compare`

A binary predicate that takes two elements (of type `T`) as arguments and returns a `bool`.

The expression `comp(a,b)`, where *comp* is an object of this type and *a* and *b* are elements in the container, shall return `true` if *a* is considered to go before *b* in the ***strict weak ordering*** the function defines.

The `priority_queue` uses this function to maintain the elements sorted in a way that preserves ***heap properties*** (i.e., that the element popped is the last according to this ***strict weak ordering***).

This can be a function pointer or a function object, and defaults to `less`<T>, which returns the same as applying the ***less-than operator*** (a<b).