이주현

Heap : Priority Queue

(주) 한컴에듀케이션

Max Heap
: Concept

Max Heap

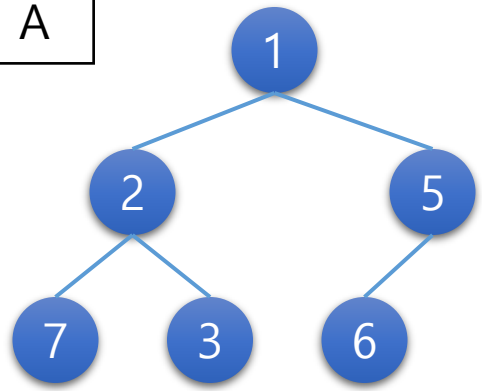
다음 2가지 특성을 만족하는 경우 **최대 힙(max heap)**이라 한다.

1. **완전이진트리 (complete binary tree)**
   : 이진 트리의 원소가 왼쪽부터 채워지는 트리를 말한다.
   : 왼쪽부터 채우는데 더 이상 채울 수 없는 경우
   새로운 레벨의 왼쪽부터 채운다.
2. **부모 노드의 값 >= 자식 노드의 값**
   max heap visualization

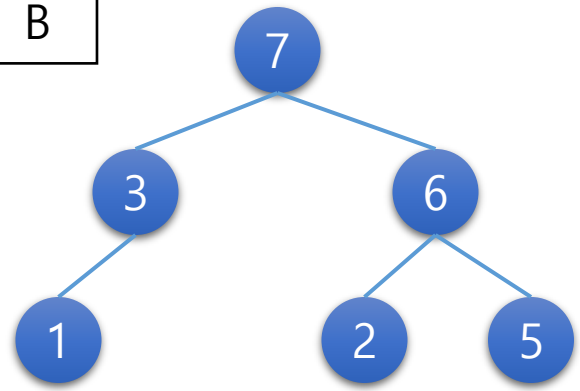
cf. min_heap : 부모노드의 값 <= 자식노드의 값
   min heap visualization
   heap : "쌓아놓은 더미(대충)"
   stack : "쌓아놓은 더미(차곡차곡)"

다음중 Max Heap인 것은?

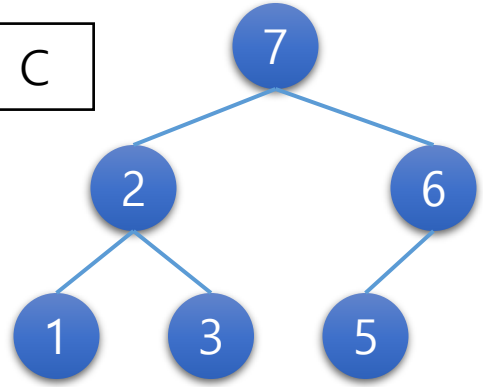
A



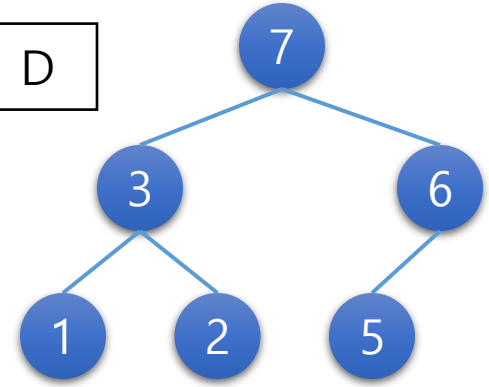
B



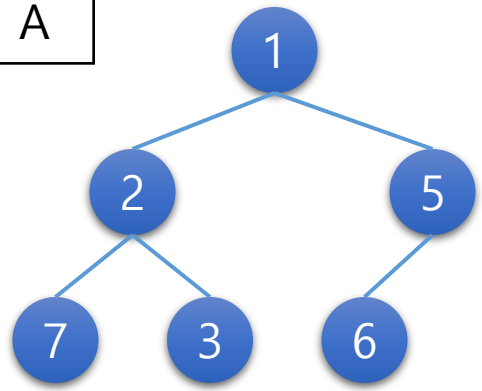
C



D

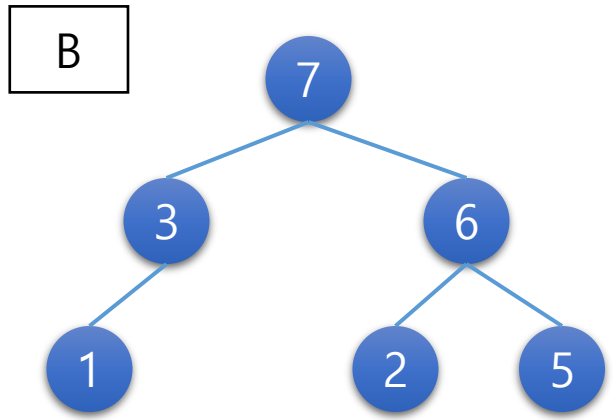


A



1. 완전이진트리 속성을 만족한다.
2. 부모노드 값 >= 자식노드 값 특성을 만족하지 못하고 있다.

따라서 **최대 힙이 아니다**. => 오리려 최소 힙을 만족한다.



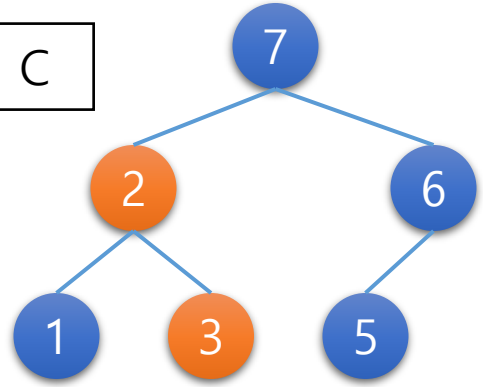
1. 완전이진트리 속성을 만족하지 않는다.
   따라서 **최대 힙이 아니다.**

1. 완전이진트리 속성을 만족한다.
2. 최대 힙(max heap)은 부모 노드 값 >= 자식 노드 값
   특성을 만족해야 한다.

   그런데 아래 트리의 별색 노드들이 이 특성을 만족하지 않는다.

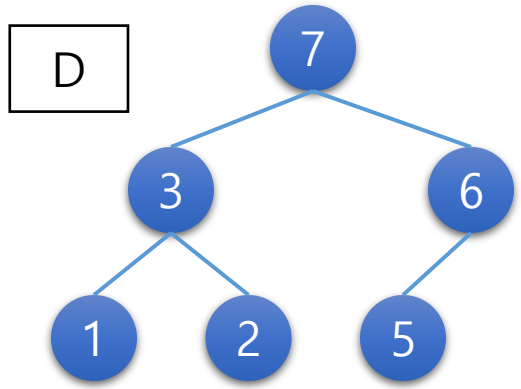
   따라서 **최대 힙이 아니다.**

C



1. 완전이진트리 속성을 만족한다.
2. 부모 노드 값 >= 자식 노드 값 특성을 만족한다.

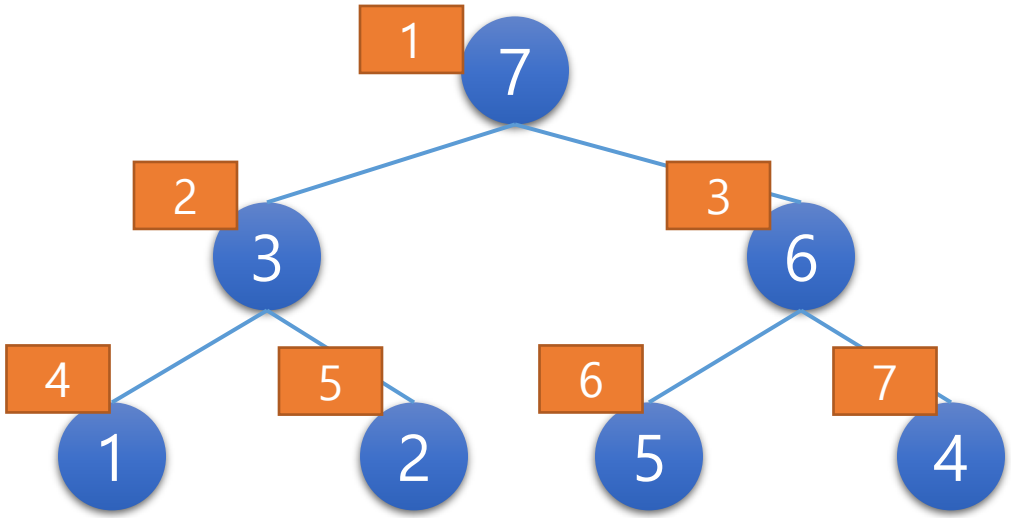
   두 특성을 모두 만족하므로 **최대 힙(max heap)이다.**



Heap을 구현한 Tree 에서 노드 번호와 배열의 인덱스

MAX HEAP :

1. 완전이진트리 (complete binary tree)
2. 부모 노드의 값 >= 자식 노드의 값

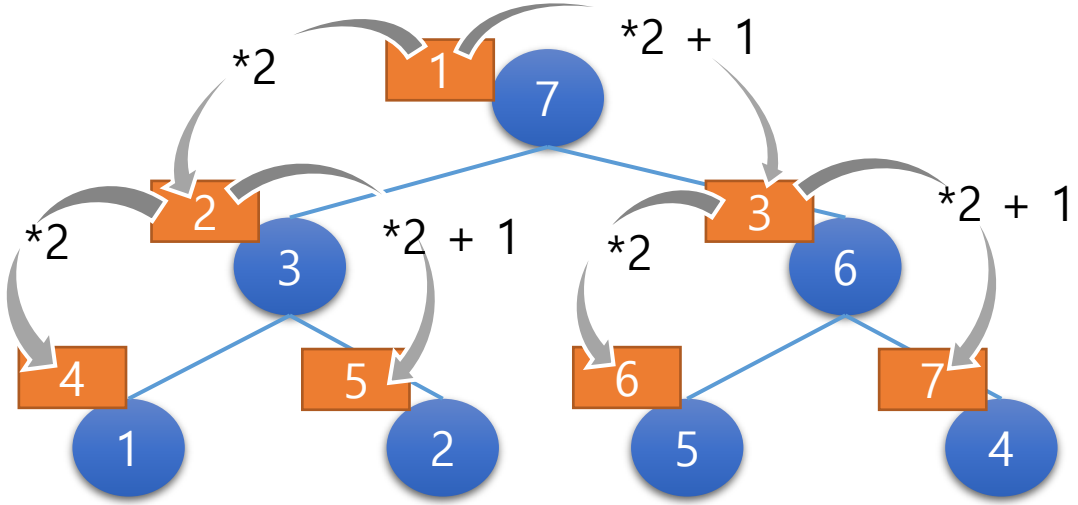


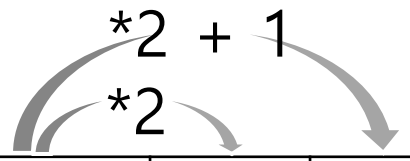
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
	7	**3**	**6**	**1**	**2**	**5**	**4**

Parent와 Child의 노드 번호 관계식

트리의 임의의 인덱스 p에 대하여
다음을 만족한다.

1. p의 left child 번호 => p* 2
2. p의 right child 번호 => p * 2 + 1
3. p의 parent번호 => p / 2





0	1	2	3	4	5	6	7
	7	**3**	**6**	**1**	**2**	**5**	**4**

Max Heap
: push heap

Heap에 원소를
추가하기

Heap(max heap)에 원소를 추가하기

2, 1, 3, 6, 4, 5, 7 을 차례로 heap에 추가하는 예를 살펴보자.

힙은 비어있다.
따라서 hn = 0;

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*

2,1,3,6,4,5,7 을 차례로 추가하기

2를 heap에 추가한다.

먼저 heap[]의 1번 index에 2을 대입한다.
hn=0 으로 시작하므로 heap[++hn] = 2;

0	1	2	3	4	5	6	7
	2						

2,1,3,6,4,5,7 을 차례로 추가하기

힙에 저장된 2 에 대하여 힙 특성 유지시키기

push_heap()함수를 호출하여 heap 특성을 유지시킨다.
push_heap();

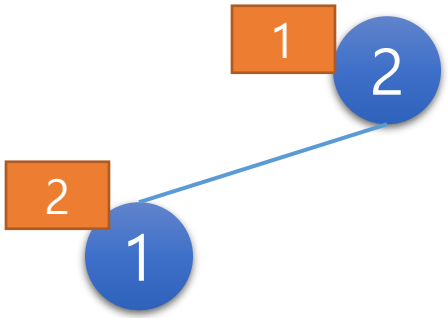
원소가 하나뿐이므로
더 이상 할 일이 없다.

0	1	2	3	4	5	6	7
	2						

2,1,3,6,4,5,7 을 차례로 추가하기

1을 heap에 추가한다.

먼저 heap[]의 2번 index에 1을 대입한다.
heap[++hn] = 1; /// heap[2] = 1;



0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
	2	**1**					

2,1,3,6,4,5,7 을 차례로 추가하기

힙에 저장된 2, 1 에 대하여 힙 특성 유지시키
기

push_heap()함수를 호출하여 heap 특성을 유지시킨다.

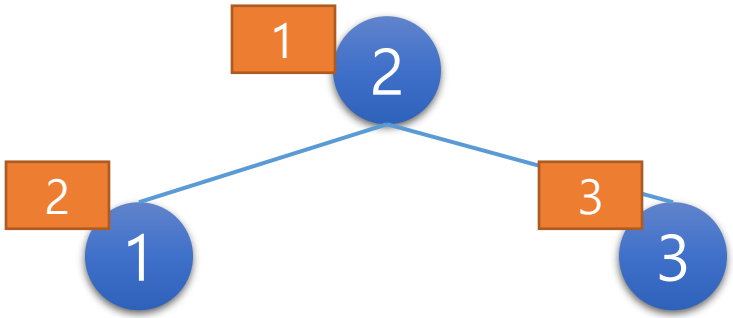
```
push_heap();
```

0	1	2	3	4	5	6	7
	2	1					

2,1,3,6,4,5,7 을 차례로 추가하기

3을 heap에 추가한다.

먼저 heap[]의 3번 index에 3을 대입한다.
heap[++hn] = 3; /// heap[3] = 3;

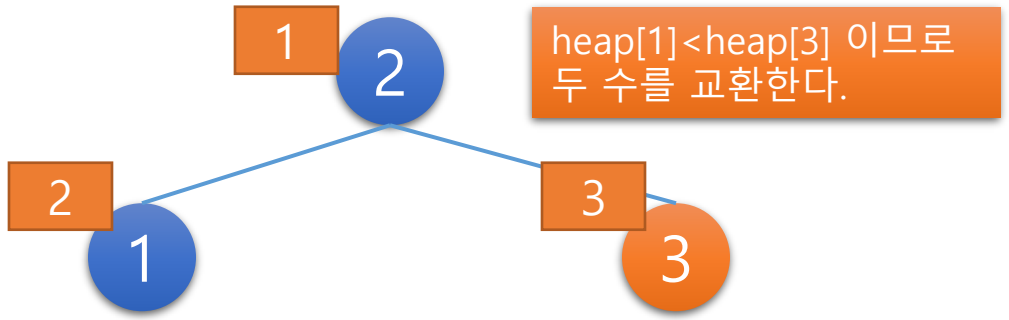


0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
	2	1	**3**				

2,1,3,6,4,5,7 을 차례로 추가하기

힙에 저장된 2, 1, 3 에 대하여 힙 특성 유지시키기

push_heap()함수를 호출하여 heap 특성을 유지시킨다.
push_heap(); 시작

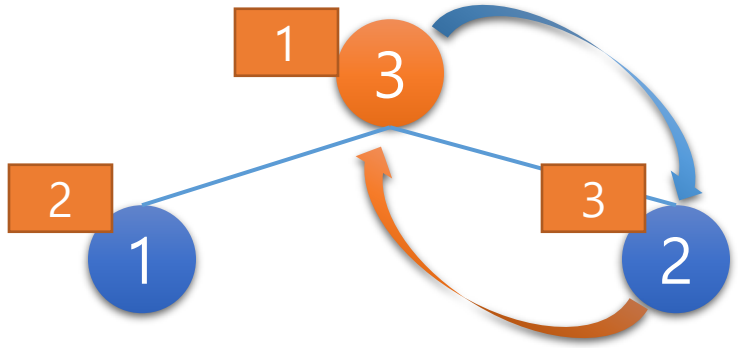


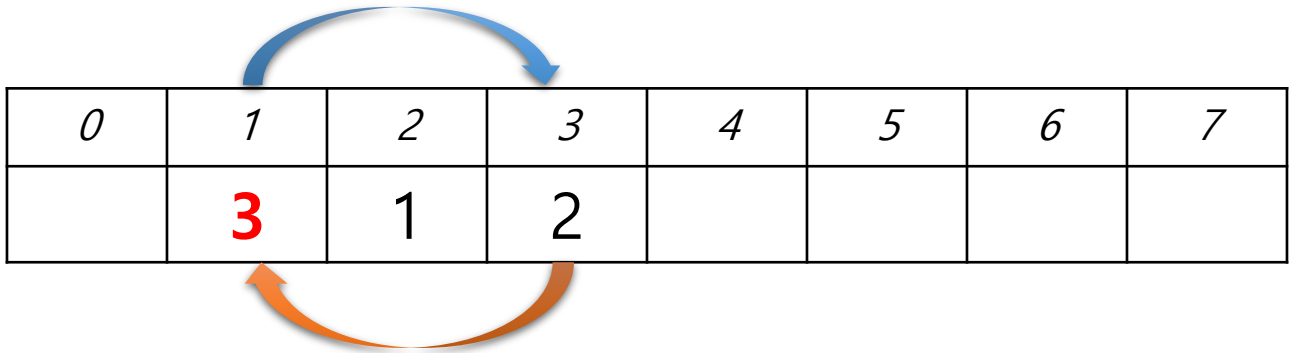
0	1	2	3	4	5	6	7
	2	1	**3**				

2,1,3,6,4,5,7 을 차례로 추가하기

힙에 저장된 2, 1, 3 에 대하여 힙 특성 유지시키기

push_heap(); 진행중
swap(heap[1], heap[3]);



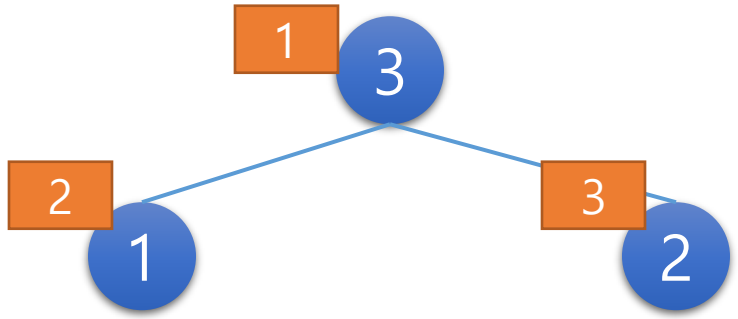


0	1	2	3	4	5	6	7
	3	1	2				

2,1,3,6,4,5,7 을 차례로 추가하기

힙이 3, 1, 2 로 힙 특성이 유지되었다.

push_heap(); 종료

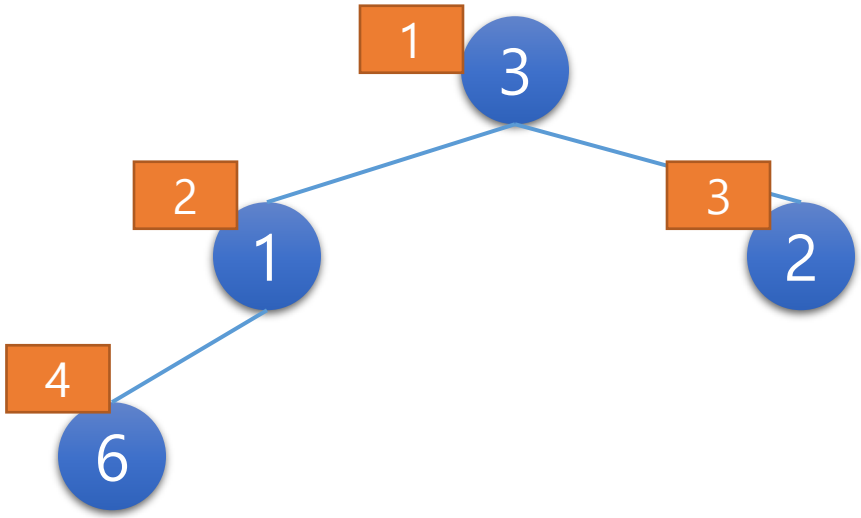


0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
	3	1	2				

2,1,3,6,4,5,7 을 차례로 추가하기

6을 heap에 추가한다.

먼저 heap[]의 4번 index에 6을 대입한다.
heap[++hn] = 6; /// heap[4] = 6;

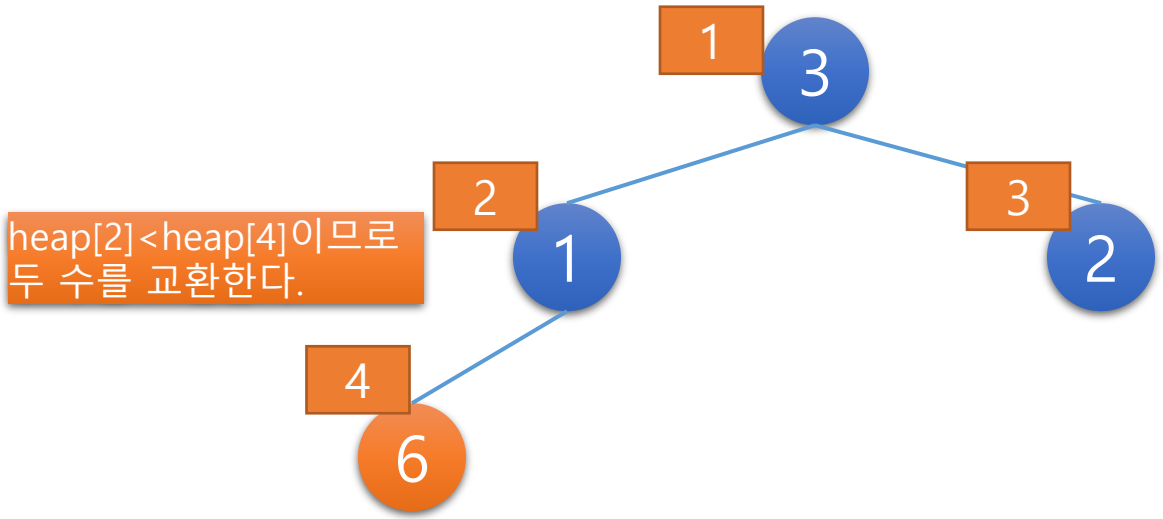


0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
	3	1	2	**6**			

2,1,3,6,4,5,7 을 차례로 추가하기

힙에 저장된 3, 1, 2, 6 에 대하여 힙 특성 유지시키기

push_heap()함수를 호출하여 heap 특성을 유지시킨다.
push_heap(); 시작

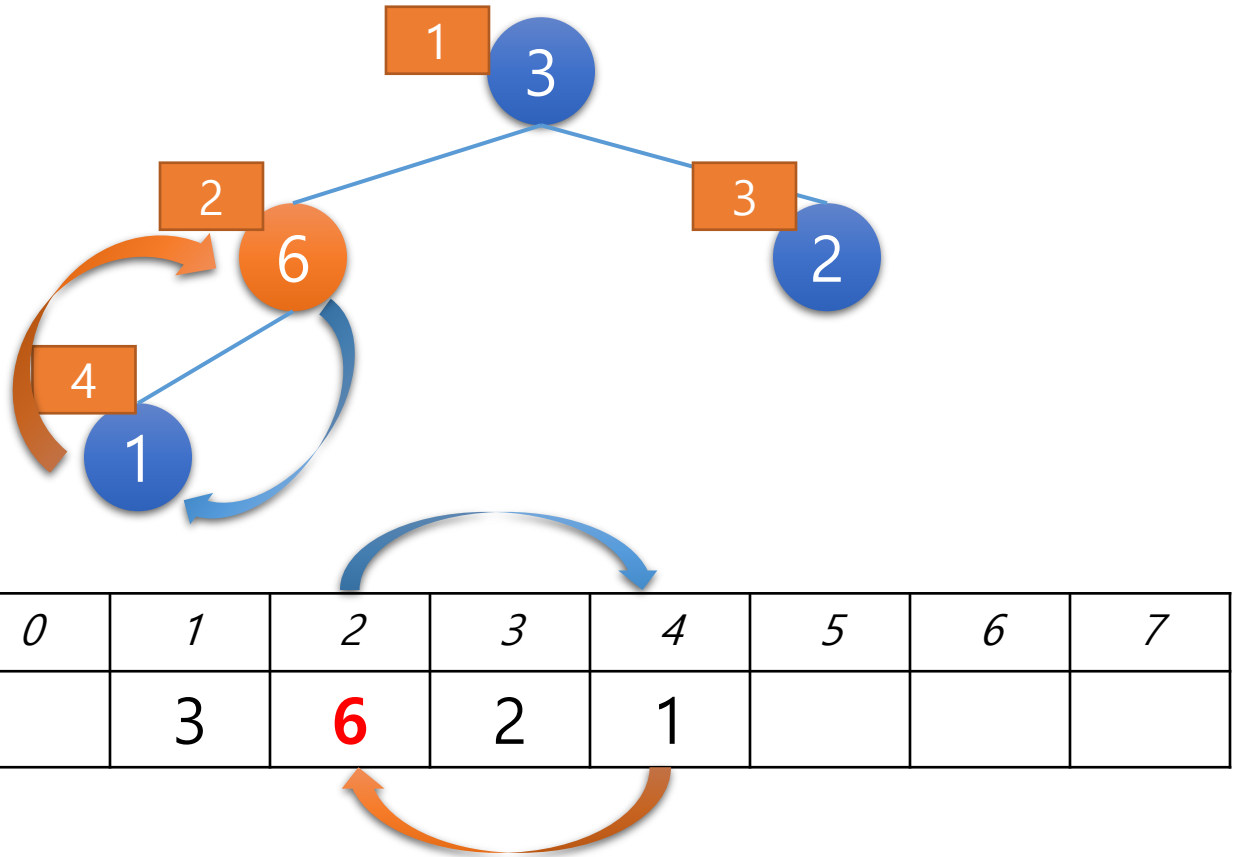


0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
	3	1	2	**6**			

2,1,3,6,4,5,7 을 차례로 추가하기

힙에 저장된 3, 1, 2, 6 에 대하여 힙 특성 유지시키기

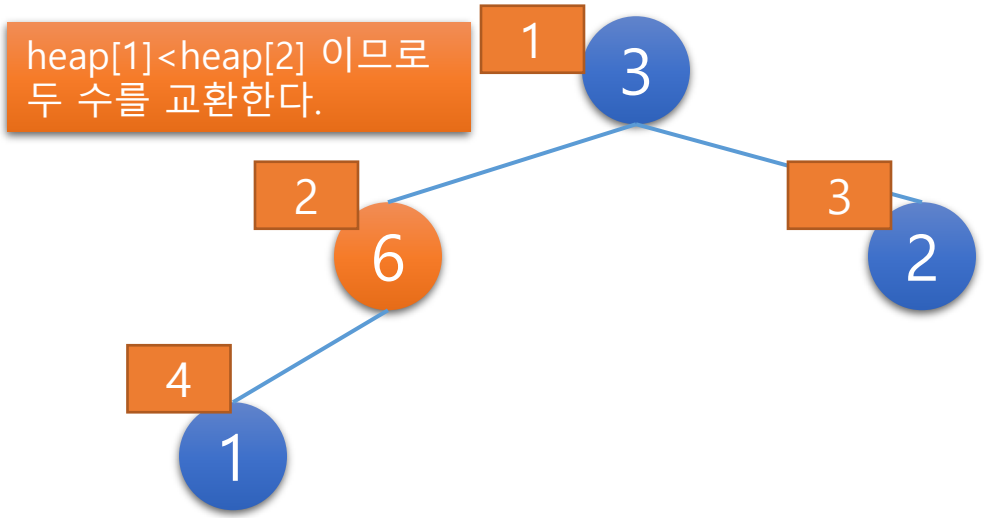
push_heap(); 진행중
swap(heap[2], heap[4]);



2,1,3,6,4,5,7 을 차례로 추가하기

힙에 저장된 3, 6, 2, 1 에 대하여 힙 특성 유지시키기

push_heap(); 진행중

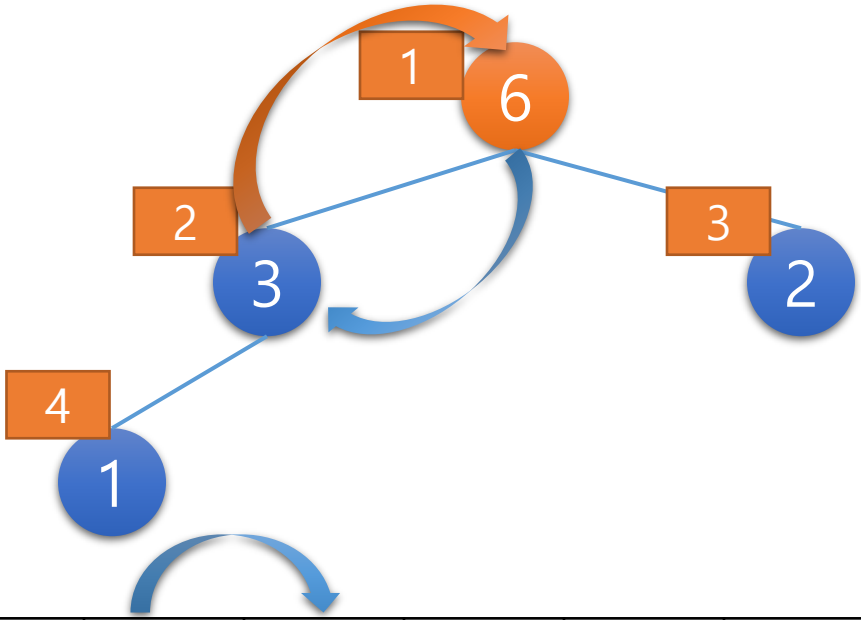


0	1	2	3	4	5	6	7
	3	**6**	2	1			

2,1,3,6,4,5,7 을 차례로 추가하기

힙에 저장된 3, 6, 2, 1 에 대하여 힙 특성 유지시키기

push_heap(); 진행중
swap(heap[1], heap[2]);

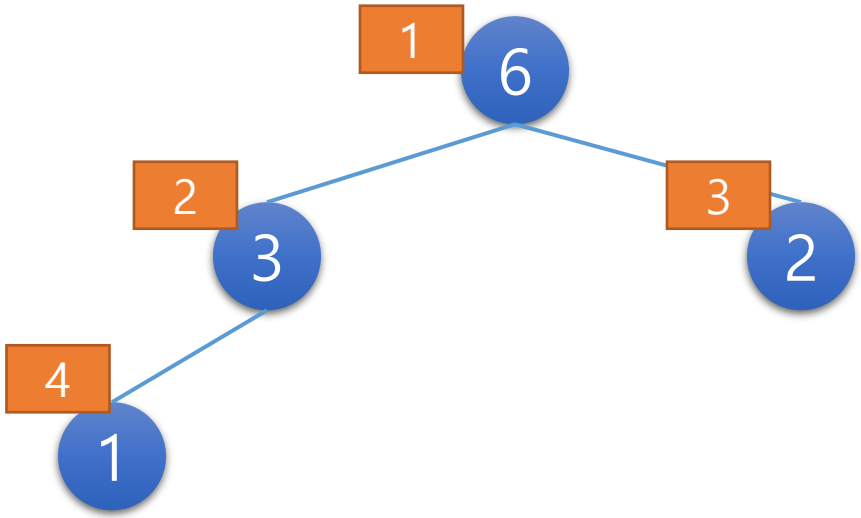


0	1	2	3	4	5	6	7
	6	3	2	1			

2,1,3,6,4,5,7 을 차례로 추가하기

힙이 6, 3, 2, 1 로 힙 특성이 유지되었다.

push_heap(); 종료

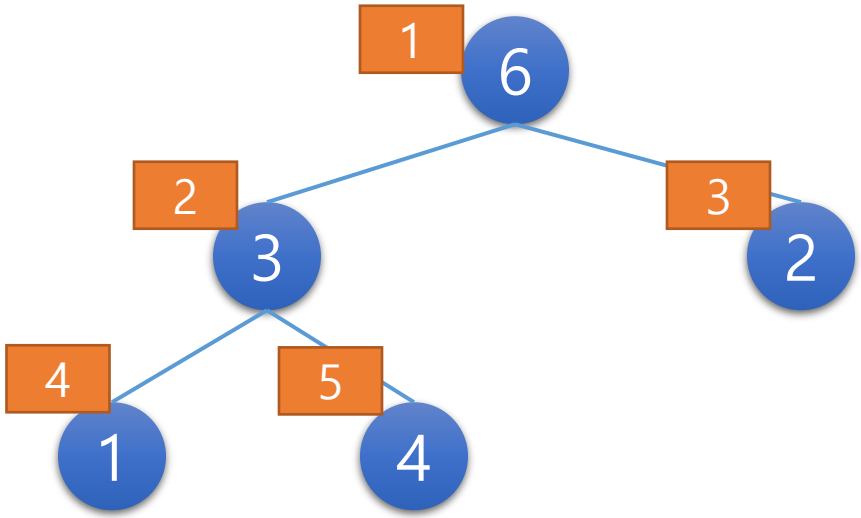


0	1	2	3	4	5	6	7
	6	3	2	1			

2,1,3,6,4,5,7 을 차례로 추가하기

4를 heap에 추가한다.

먼저 heap[]의 5번 index에 4을 대입한다.
heap[++hn] = 4; /// heap[5] = 4;

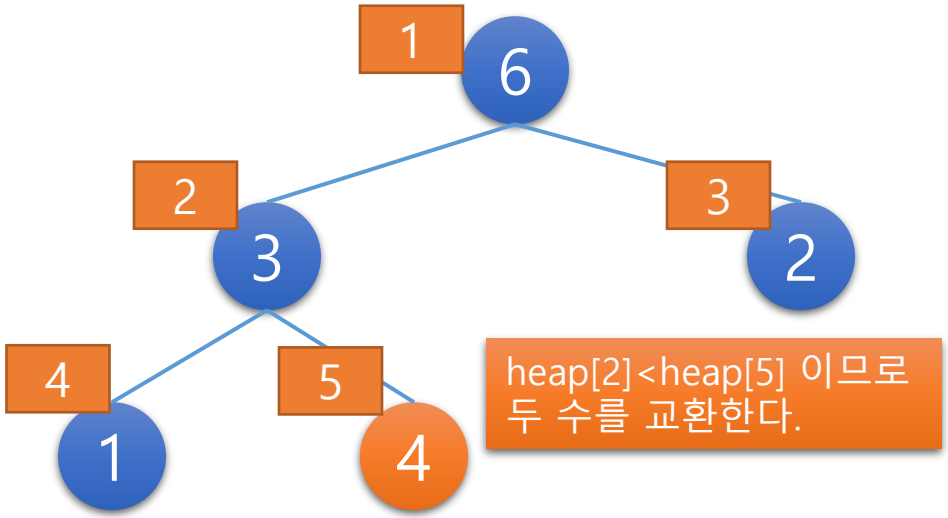


0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
	6	3	2	1	**4**		

2,1,3,6,4,5,7 을 차례로 추가하기

힙에 저장된 6, 3, 2, 1, 4 에 대하여 힙 특성 유지시키기

push_heap()함수를 호출하여 heap 특성을 유지시킨다.
push_heap(); 시작

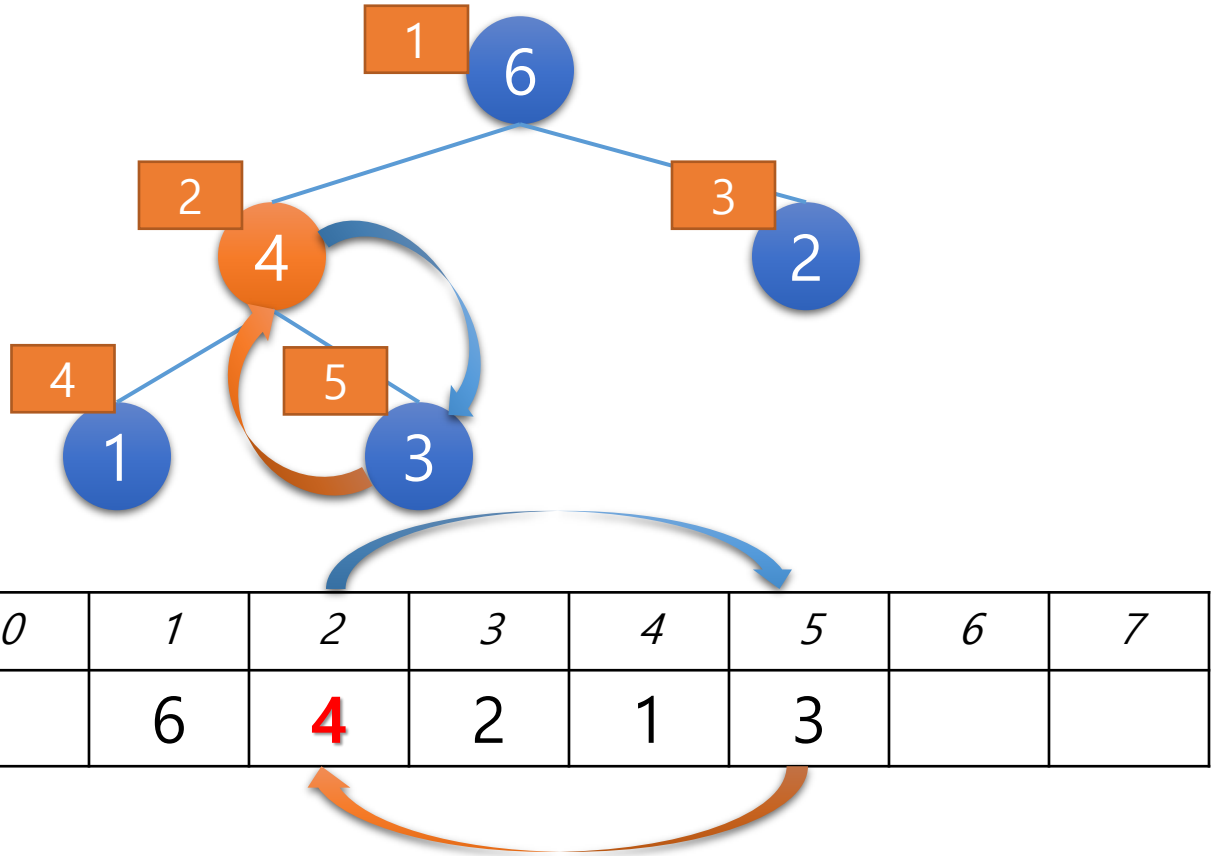


0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
	6	3	2	1	**4**		

2,1,3,6,4,5,7 을 차례로 추가하기

힙에 저장된 6, 3, 2, 1, 4 에 대하여 힙 특성 유지시키기

push_heap(); 진행중
swap(heap[2], heap[5]);

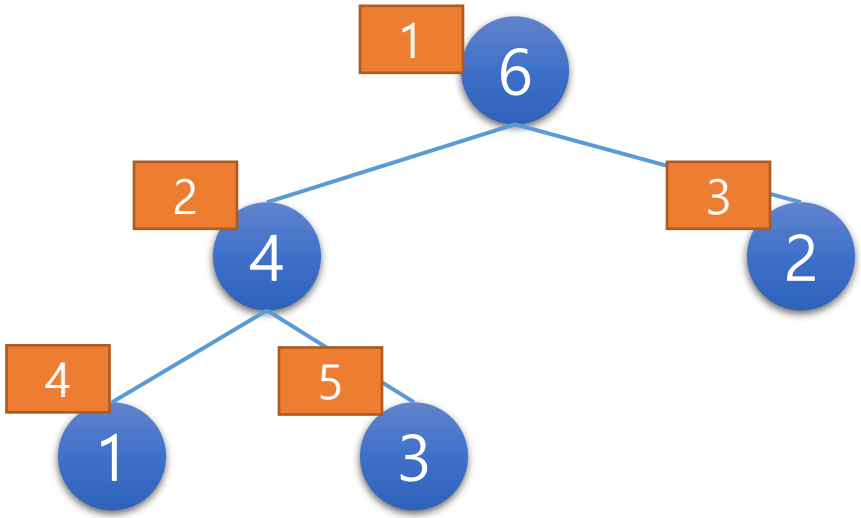


0	1	2	3	4	5	6	7
	6	4	2	1	3		

2,1,3,6,4,5,7 을 차례로 추가하기

힙이 6, 4, 2, 1, 3 으로 힙 특성이 유지되었다.

push_heap(); 완료

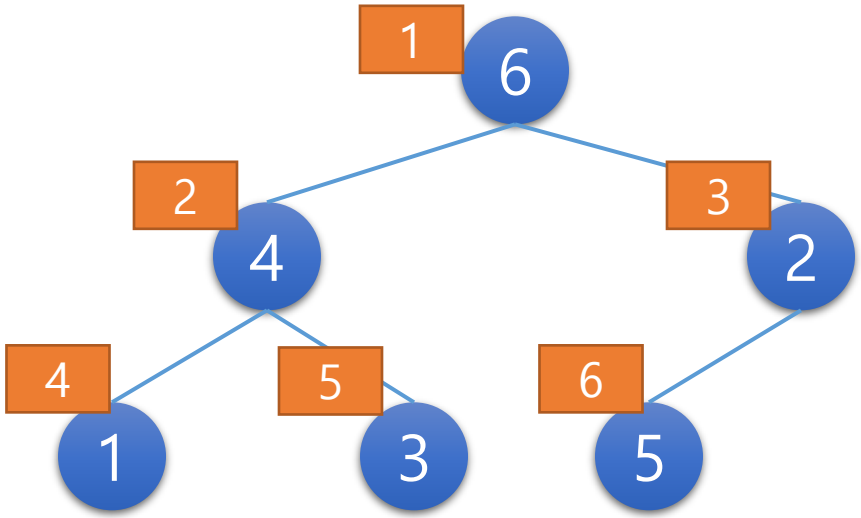


0	1	2	3	4	5	6	7
	6	4	2	1	3		

2,1,3,6,4,5,7 을 차례로 추가하기

5를 heap에 추가한다.

먼저 heap[]의 6번 index에 5을 대입한다.
heap[++hn] = 5; /// heap[6] = 5;

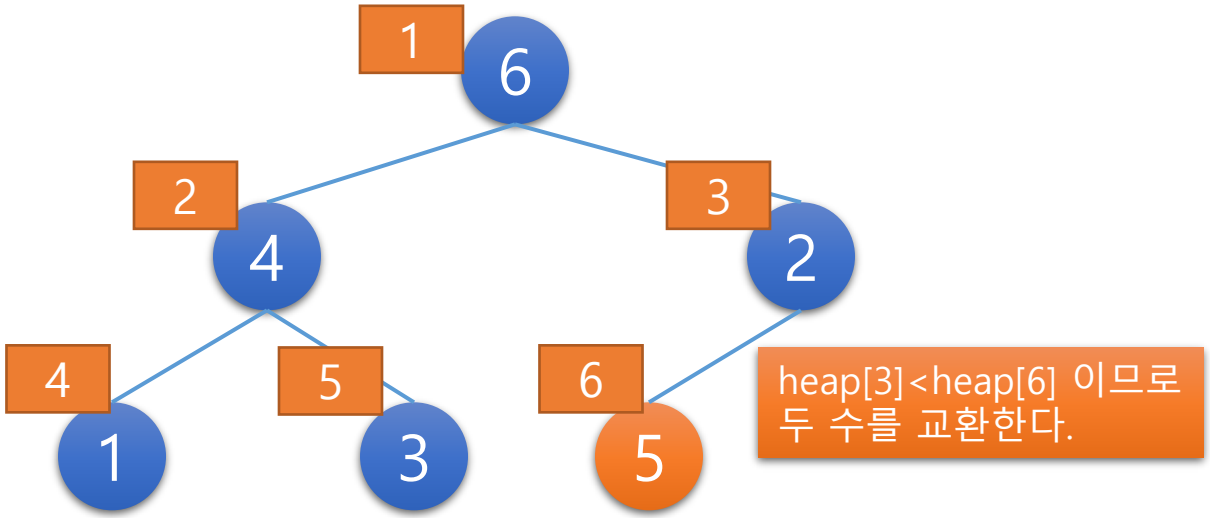


0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
	6	4	2	1	3	**5**	

2,1,3,6,4,5,7 을 차례로 추가하기

힙에 저장된 6, 4, 2, 1, 3, 5 에 대하여 힙 특성 유지시키기

push_heap()함수를 호출하여 heap 특성을 유지시킨다.
push_heap(); 시작

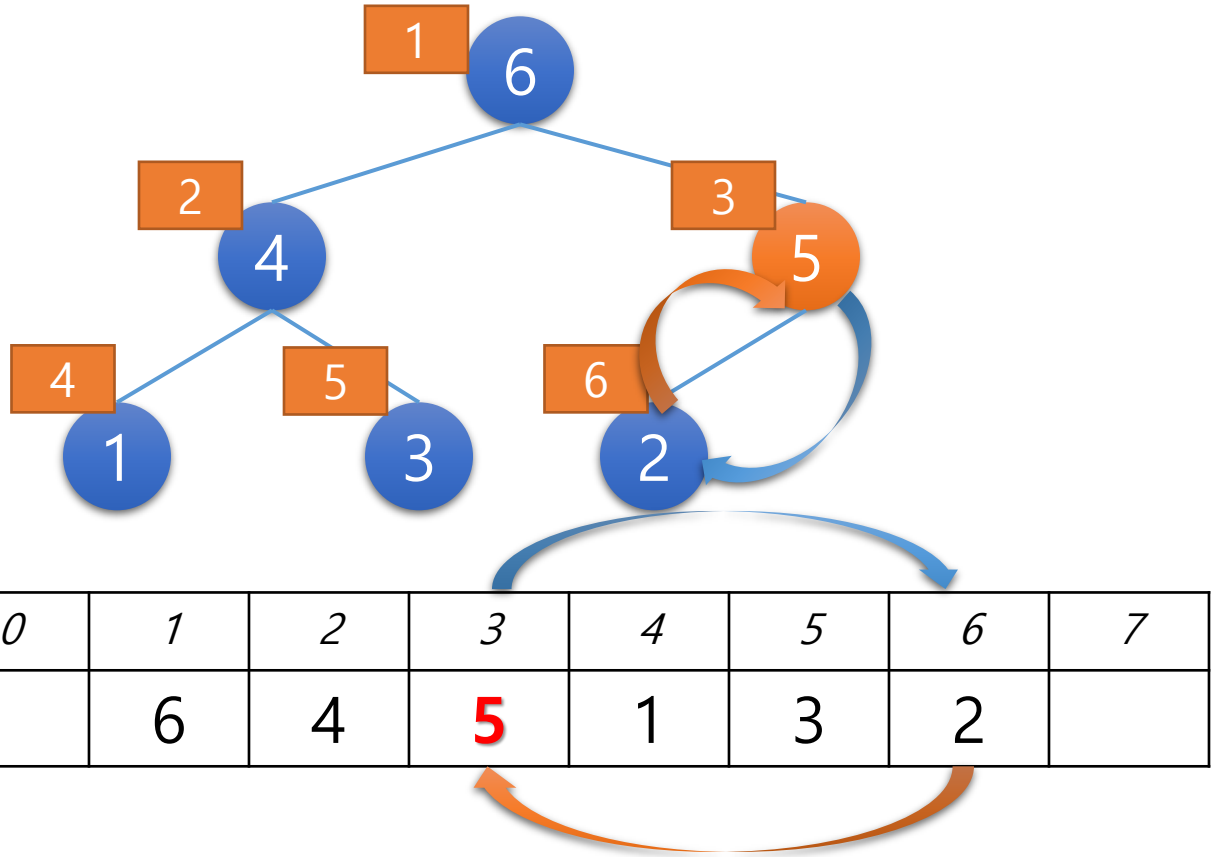


0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
	6	4	2	1	3	**5**	

2,1,3,6,4,5,7 을 차례로 추가하기

힙에 저장된 6, 4, 2, 1, 3, 5 에 대하여 힙 특성 유지시키기

push_heap(); 진행중
swap(heap[3], heap[6]);

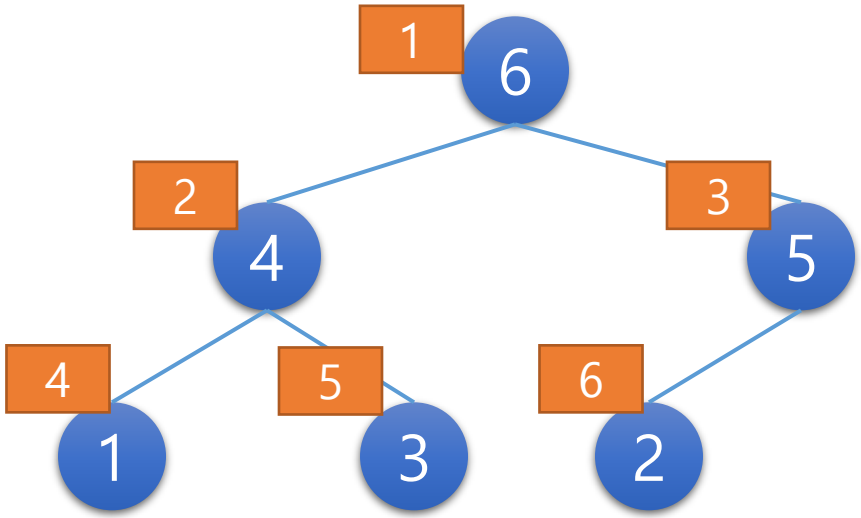


0	1	2	3	4	5	6	7
	6	4	**5**	1	3	2	

2,1,3,6,4,5,7 을 차례로 추가하기

힙이 6, 4, 5, 1, 3, 2 으로 힙 특성이 유지되었다.

push_heap(**5**); 완료

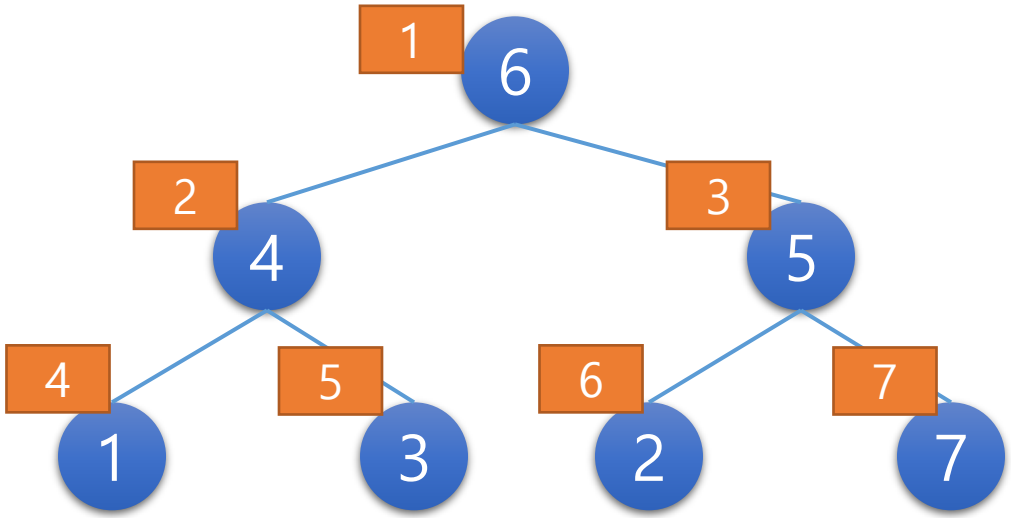


0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
	6	4	5	1	3	2	

2,1,3,6,4,5,7 을 차례로 추가하기

7을 heap에 추가한다.

먼저 heap[]의 7번 index에 7을 대입한다.
heap[++hn] = 7; /// heap[7] = 7;

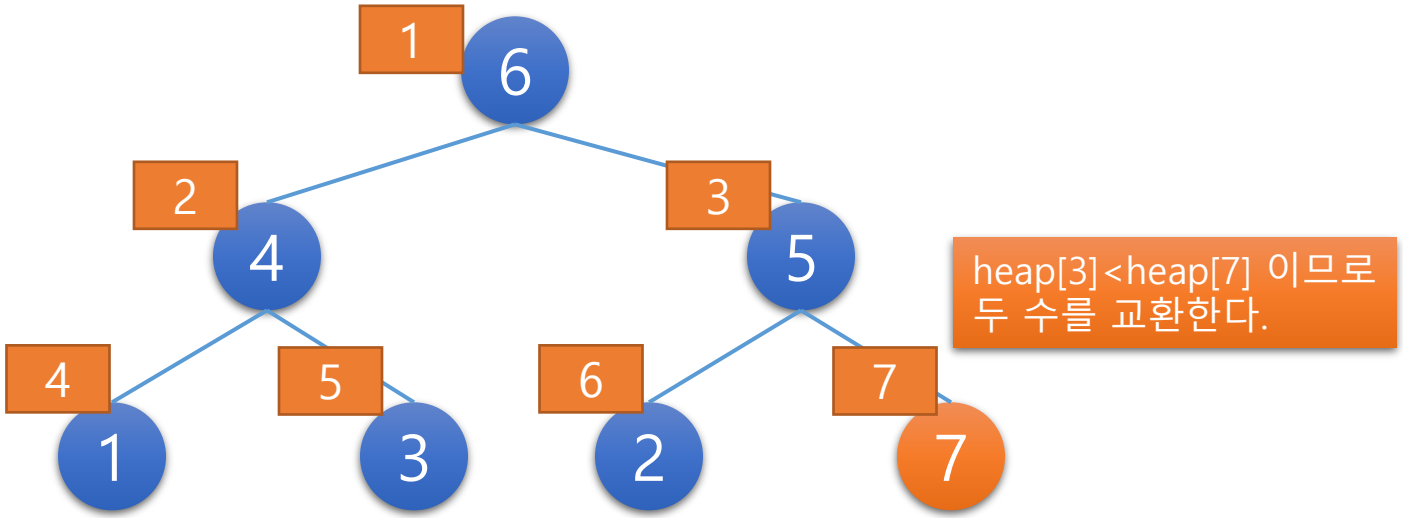


0	1	2	3	4	5	6	7
	6	4	5	1	3	2	7

2,1,3,6,4,5,7 을 차례로 추가하기

힙에 저장된 6, 4, 5, 1, 3, 2, 7 에 대하여 힙 특성 유지시키기

push_heap()함수를 호출하여 heap 특성을 유지시킨다.
push_heap(7); 시작

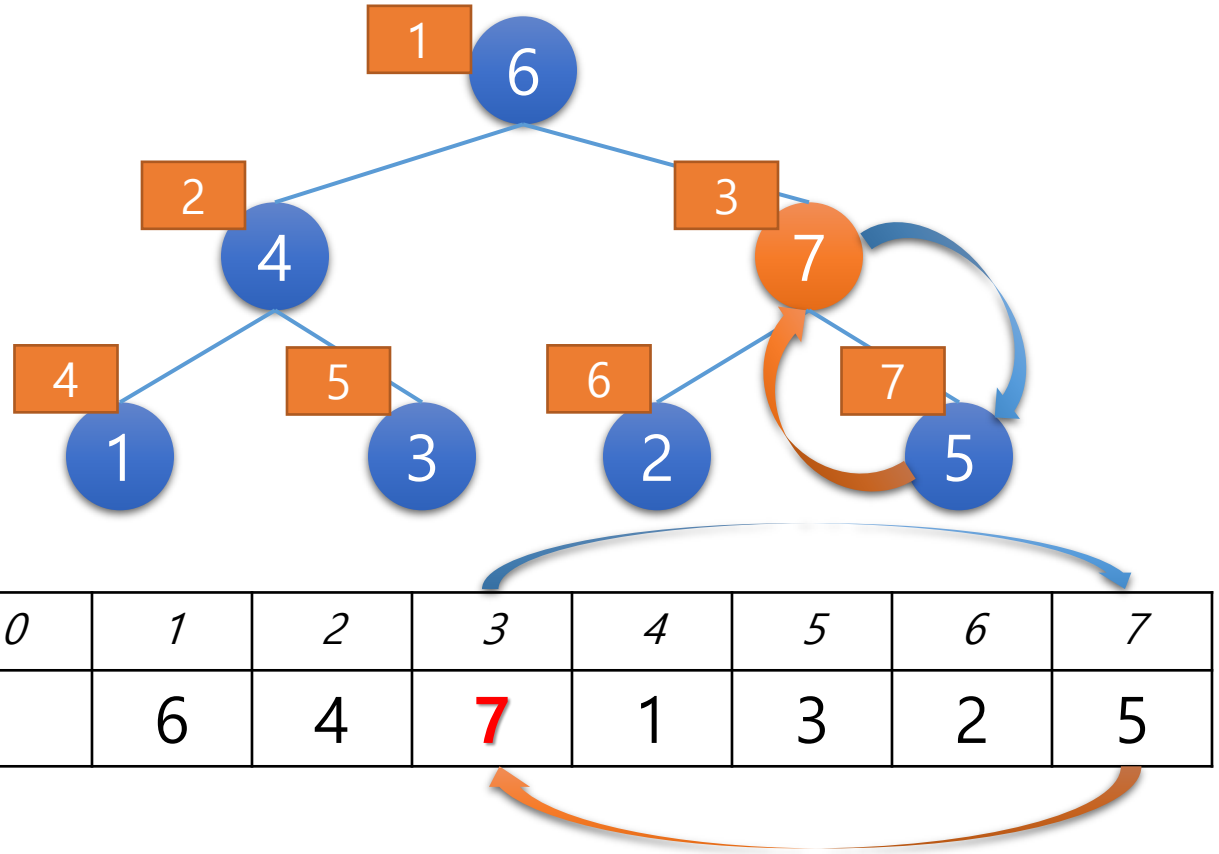


0	1	2	3	4	5	6	7
	6	4	5	1	3	2	7

2,1,3,6,4,5,7 을 차례로 추가하기

힙에 저장된 6, 4, 5, 1, 3, 2, 7 에 대하여 힙 특성 유지시키기

push_heap(7); 진행중
swap(heap[3], heap[7]);

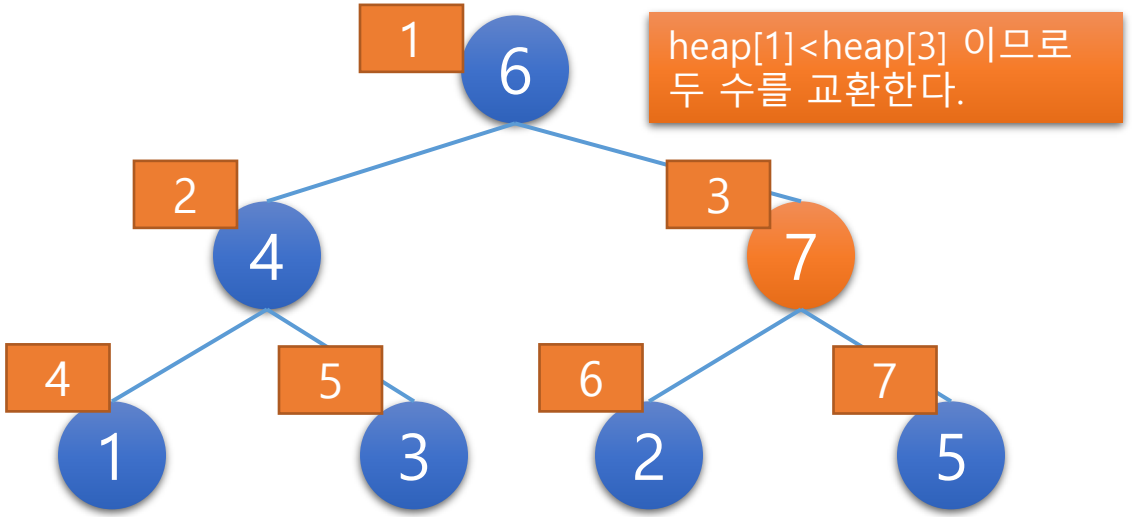


0	1	2	3	4	5	6	7
	6	4	7	1	3	2	5

2,1,3,6,4,5,7 을 차례로 추가하기

힙에 저장된 6, 4, 7, 1, 3, 2, 5 에 대하여 힙 특성 유지시키기

push_heap(7); 진행중

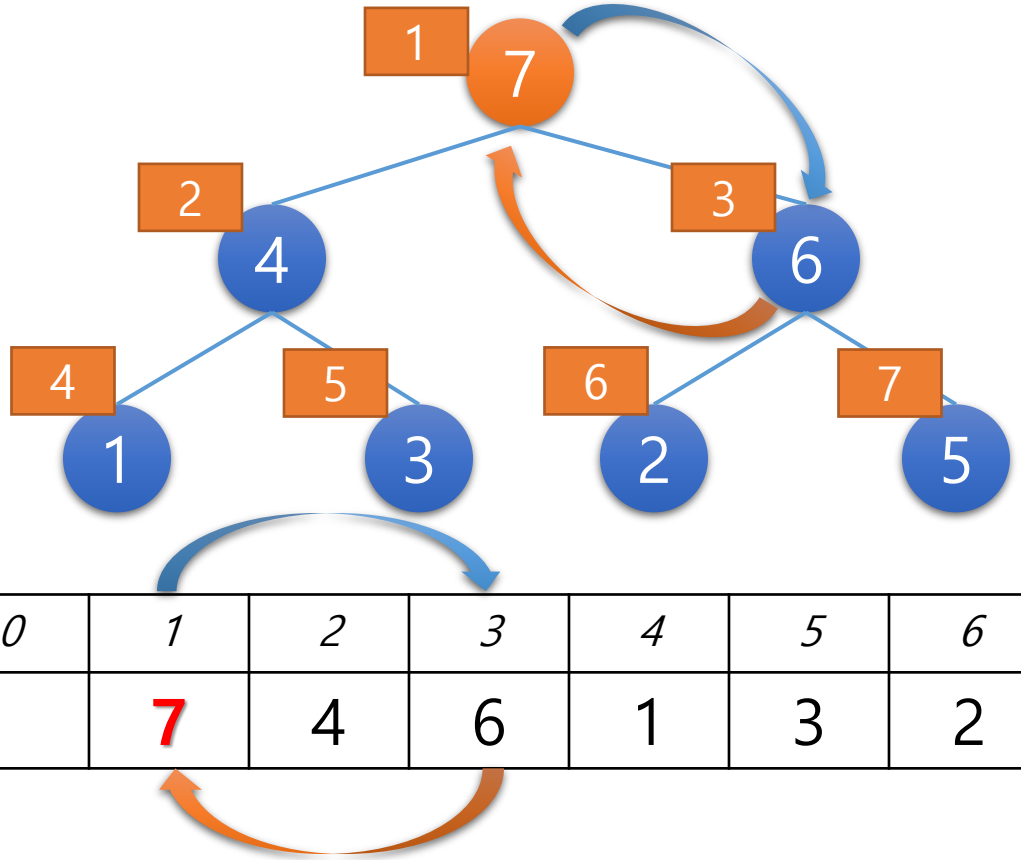


0	1	2	3	4	5	6	7
	6	4	7	1	3	2	5

2,1,3,6,4,5,7 을 차례로 추가하기

힙에 저장된 6, 4, 7, 1, 3, 2, 5 에 대하여 힙 특성 유지시키기

push_heap(7); 진행중
swap(heap[1], heap[3]);

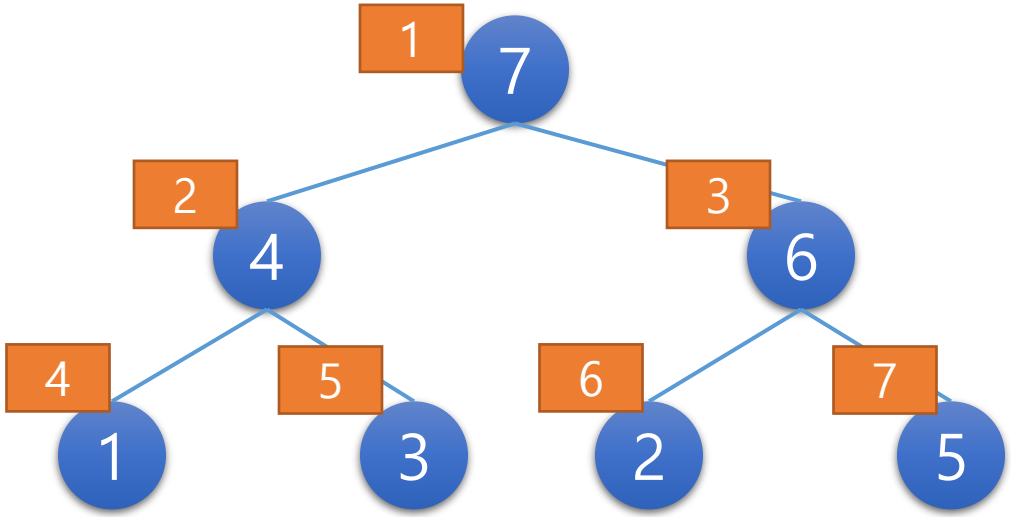


0	1	2	3	4	5	6	7
	7	4	6	1	3	2	5

2,1,3,6,4,5,7 을 차례로 추가하기

힙이 7, 4, 6, 1, 3, 2, 5 로 힙 특성이 유지되었다.

push_heap(7); 완료



0	1	2	3	4	5	6	7
	7	4	6	1	3	2	5

Push_heap 프로세스 정리

- 배열을 이용한 구현
  1. max heap을 구성할 배열을 heap[]라고 하자.
  2. heap[]배열의 마지막 원소의 인덱스를 hn 이라고 하자.
  3. 따라서 hn 는 heap[]의 원소의 개수가 되므로
     원소가 하나도 없는 경우 hn 는 0이다.
- push_heap 프로세스 :
  1. hn 을 1 증가한 후 hn 위치에 원소를 추가한다.
     (heap[++ hn] 와 같은 방법을 사용할 수 있다.)
  2. c = hn으로 시작한다. c가 1보다 크고 자식 노드의 값(heap[c])이
     부모 노드의 값(heap[c/2])보다 큰 동안 다음을 반복한다.
     - Heap[c]와 heap[c/2]를 교환한다.
     - c = c / 2;
  3. 하나의 원소가 추가될 때 max heap을 유지하기 위한 시간은 O(log(hn))

최대힙 push_heap()함수 구현 예 : user loop ver01

```
// max_heap의 push_heap() 함수

void swap( int &a, int &b) {
    int t=a; a=b; b=t;
}

void push( int nd){
    // heap의 맨 뒤에 New Data nd 추가
    heap[++hn] = nd;
    // c는 마지막 위치부터 루트 인덱스가 아닌 동안
    for ( int c = hn; c > 1; c /= 2){
        // 부모노드의 값보다 크다면 교환
        if (heap[c] > heap[c/2])
            swap(heap[c], heap[c/2]);
        else break ;
    }
}
```

최대힙 push_heap()함수 구현 예 : user loop ver02

```
// max_heap의 push_heap() 함수
// swap을 사용하지 않는 version


void push(const int&nd){
    int c = ++hn;
    // c는 마지막 위치부터 루트 인덱스가 아닌 동안
    for (; c > 1 && nd > heap[c >> 1]; c >>= 1){
        // 부모노드의 값보다 md가 크다면 부모의 값을 child로 내려 보내기
        heap[c] = heap[c >> 1];
    heap[c] = nd;
}
```

최대힙 push_heap()함수 구현 예 : user recursion

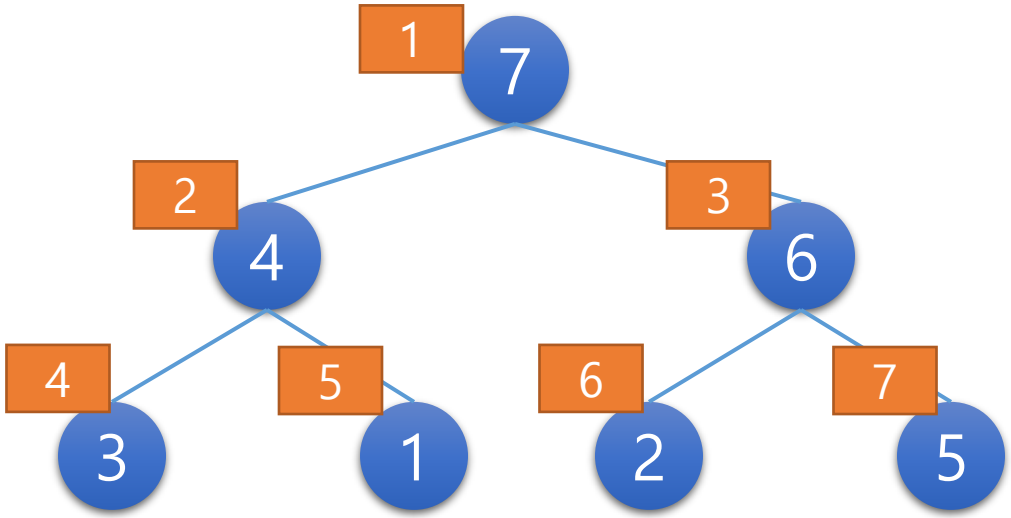
```
// max_heap의 push_heap() 함수

void swap( int &a, int &b) {
    int t=a; a=b; b=t;
}

void push( int c) {
    // c가 루트가 아니고 c노드의 값이 부모 노드의 값보다 크다면 교환
    if (c > 1 && heap[c] > heap[c/2]) {
        swap(heap[c], heap[c/2]);
        push(c / 2); // 부모의 위치로 이동
    }
}

void push_heap( int nd){
    heap[ ++hn ] = nd;
    push(hn);
}
```

Max Heap
: pop heap

Heap의 원소를
삭제하기

Heap(max heap)에서 원소를 삭제하기

힙에 저장된 7, 4, 6, 3, 1, 2, 5 을 차례로 삭제하는 예를 살펴보자.

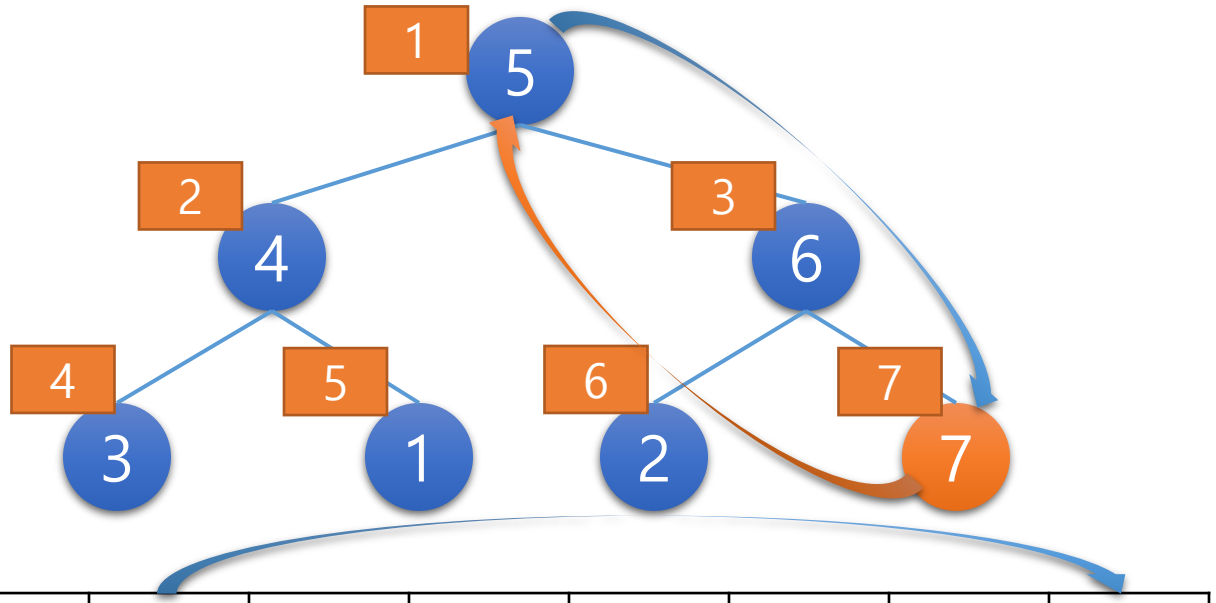


0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
	7	4	6	3	1	2	5

Heap(max heap)에서 원소를 삭제하기

힙에 저장된 **7**, 4, 6, 3, 1, 2, 5 에서 7을 삭제

먼저 heap[1]과 heap[7]의 값을 바꾼다.
swap(heap[1], heap[7]);

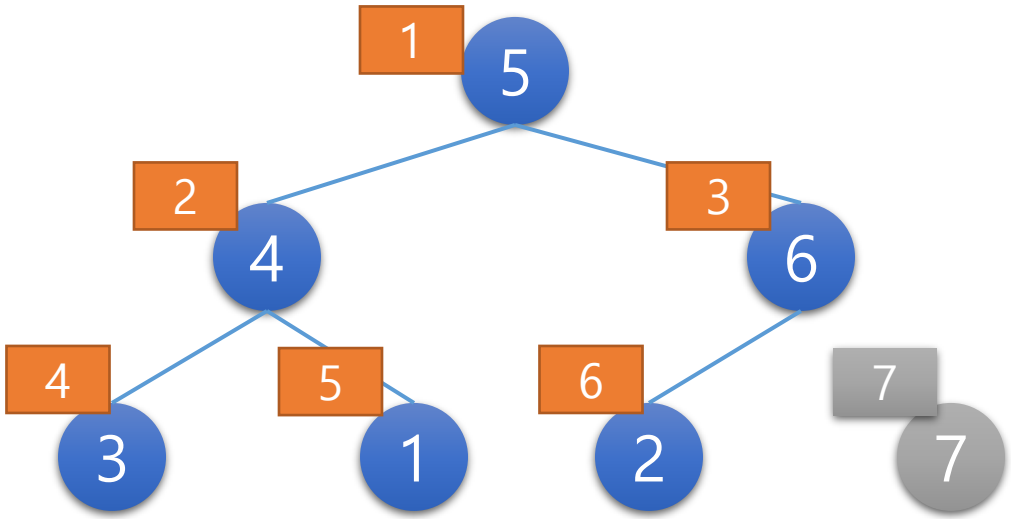


0	1	2	3	4	5	6	7
	5	4	6	3	1	2	**7**

Heap(max heap)에서 원소를 삭제하기

힙에 5, 4, 6, 3, 1, 2가 남고 7이 삭제되었다.

원소의 개수를 1개 줄인다. hn = hn – 1;

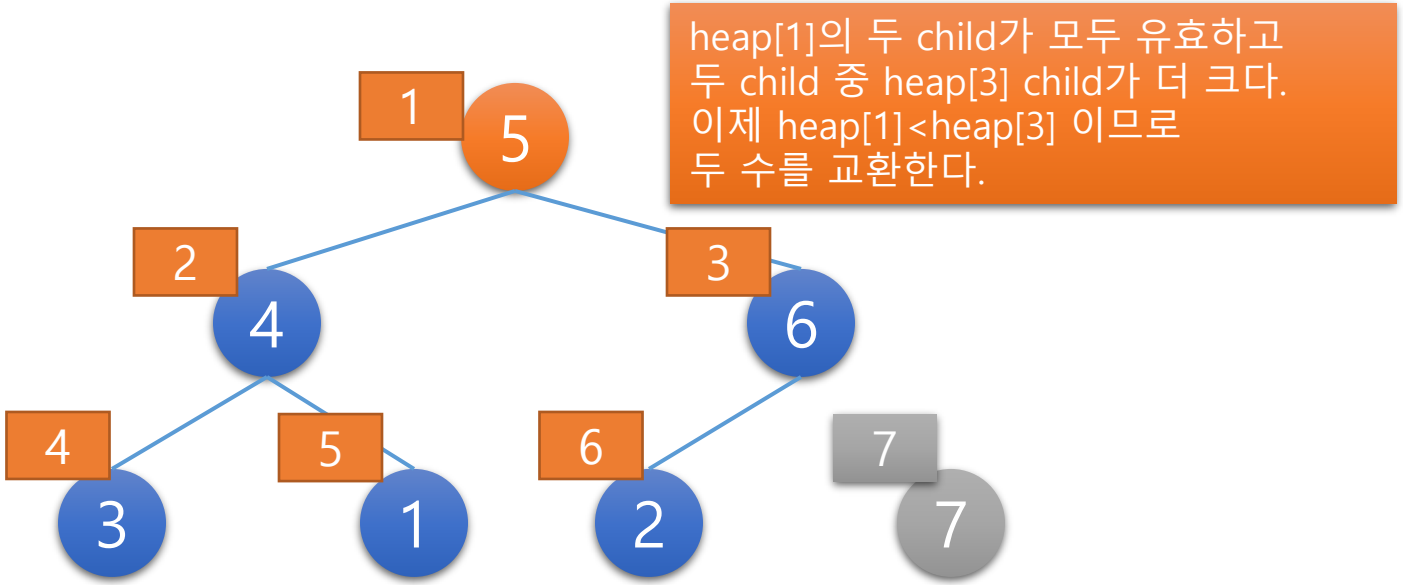


0	1	2	3	4	5	6	7
	5	4	6	3	1	2	7

Heap(max heap)에서 원소를 삭제하기

힙에 저장된 5, 4, 6, 3, 1, 2 에 대하여 힙 특성 유지시키기

pop_heap()함수를 호출하여 heap 특성을 유지시킨다.
pop_heap(); 시작

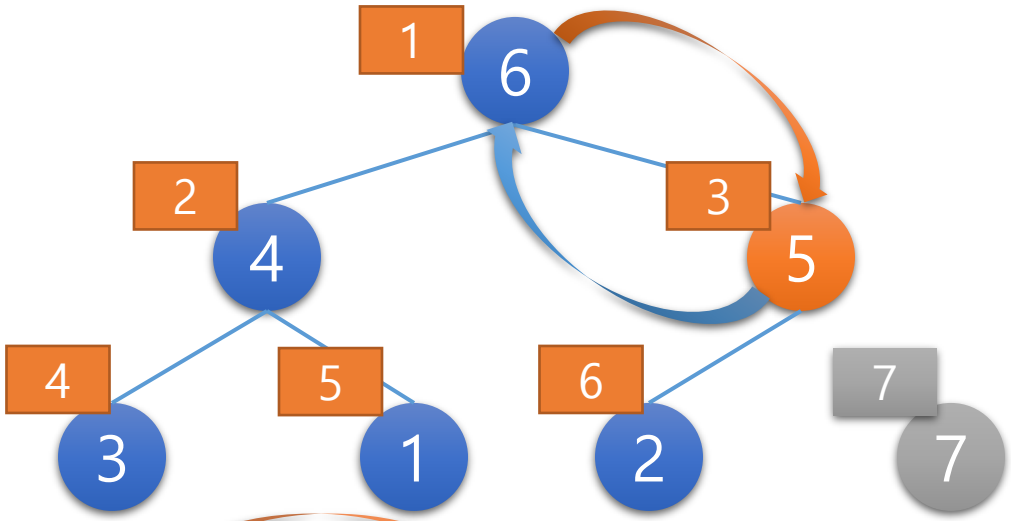


0	1	2	3	4	5	6	7
	5	4	6	3	1	2	7

Heap(max heap)에서 원소를 삭제하기

힙에 저장된 6, 4, 5, 3, 1, 2 에 대하여 힙 특성 유지시키기

pop_heap(); 진행중
swap(heap[1], heap[3]);

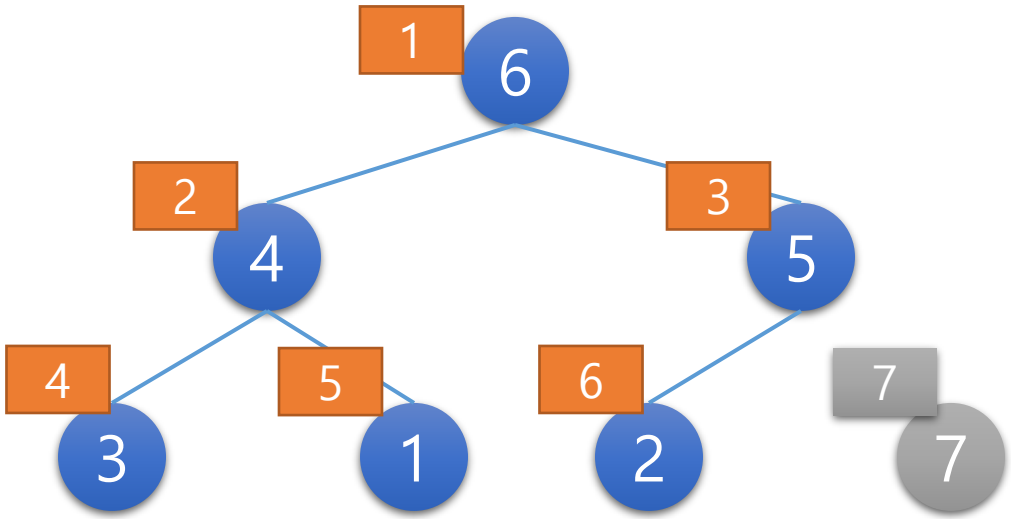


0	1	2	3	4	5	6	7
	6	4	**5**	3	1	2	7

Heap(max heap)에서 원소를 삭제하기

힙이 6, 4, 5, 3, 1, 2 로 힙 특성이 유지되었다.

pop_heap(); 완료

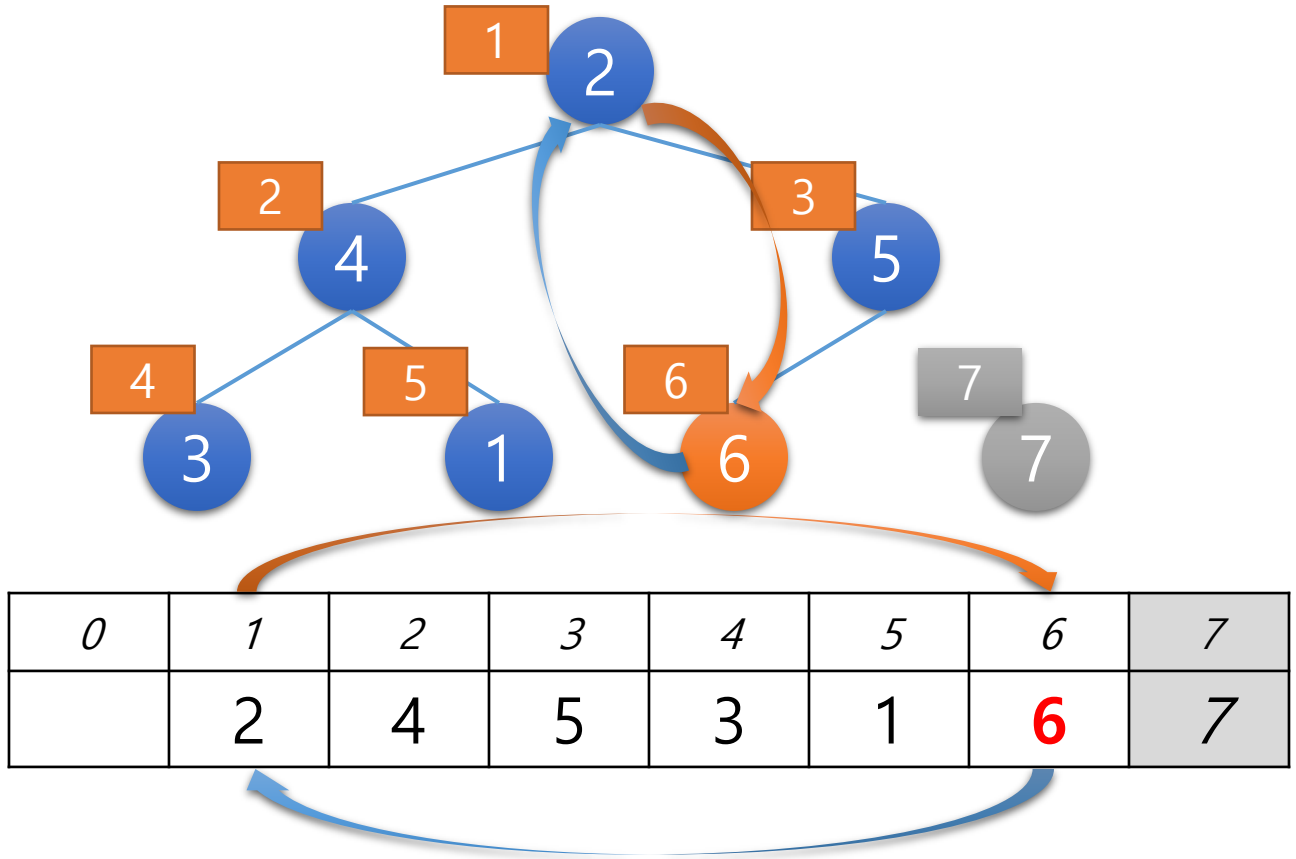


0	1	2	3	4	5	6	7
	6	4	5	3	1	2	7

Heap(max heap)에서 원소를 삭제하기

힙에 저장된 **6**, 4, 5, 3, 1, 2 에서 **6**을 삭제

먼저 heap[1]과 heap[6]의 값을 바꾼다.
swap(heap[1], heap[6]);

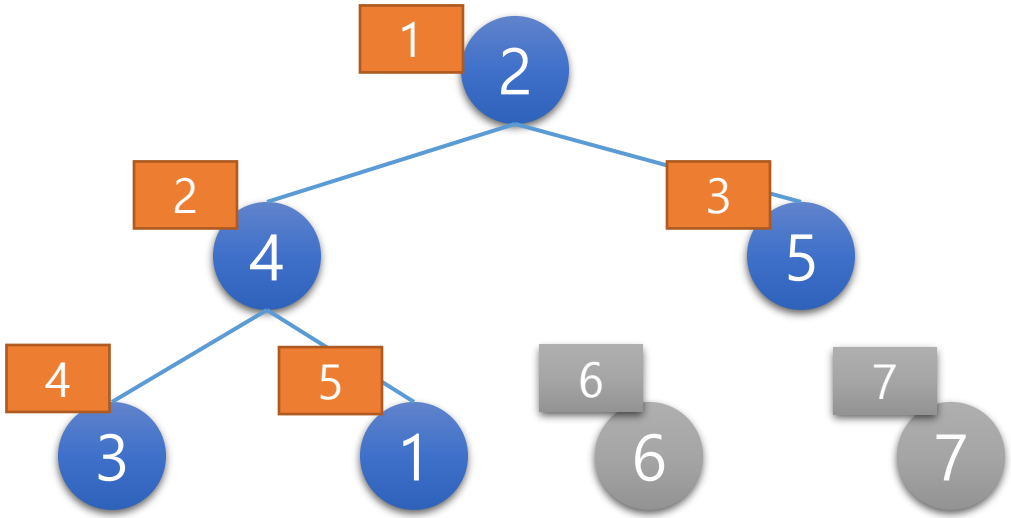


0	1	2	3	4	5	6	7
	2	4	5	3	1	**6**	7

Heap(max heap)에서 원소를 삭제하기

힙에 2, 4, 5, 3, 1 이 남고 6이 삭제되었다.

원소의 개수를 1개 줄인다. hn = hn – 1;

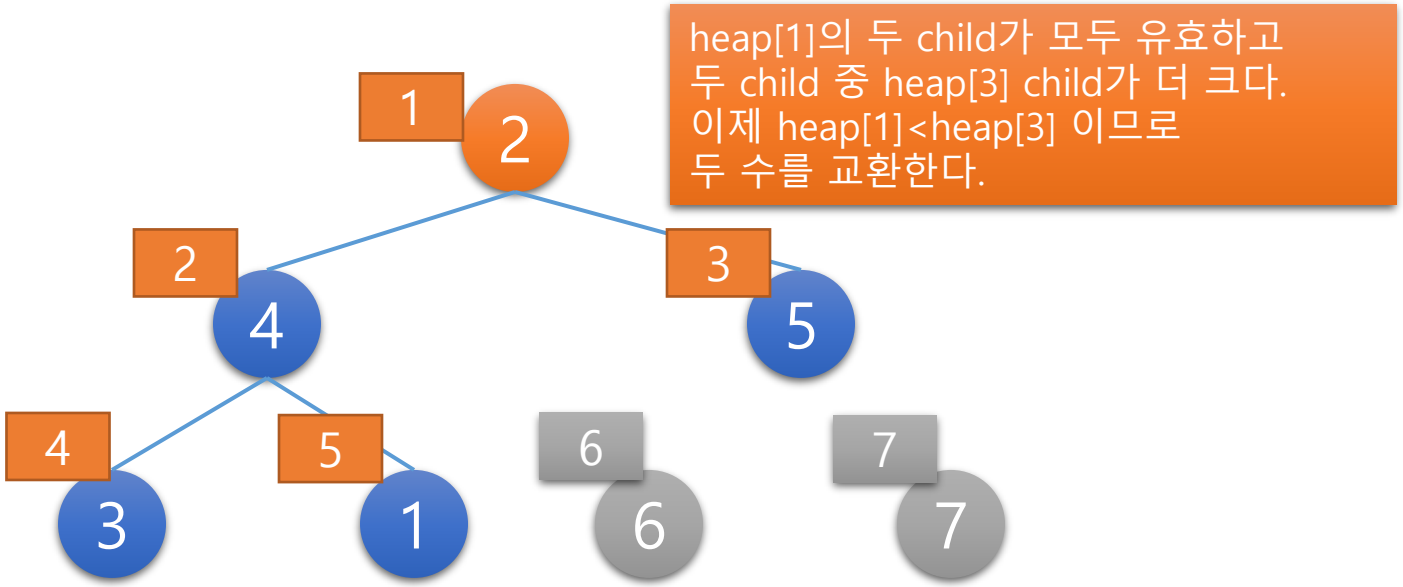


0	1	2	3	4	5	6	7
	2	4	5	3	1	6	7

Heap(max heap)에서 원소를 삭제하기

힙에 저장된 2, 4, 5, 3, 1 에 대하여 힙 특성 유지시키기

pop_heap()함수를 호출하여 heap 특성을 유지시킨다.
pop_heap(); 시작

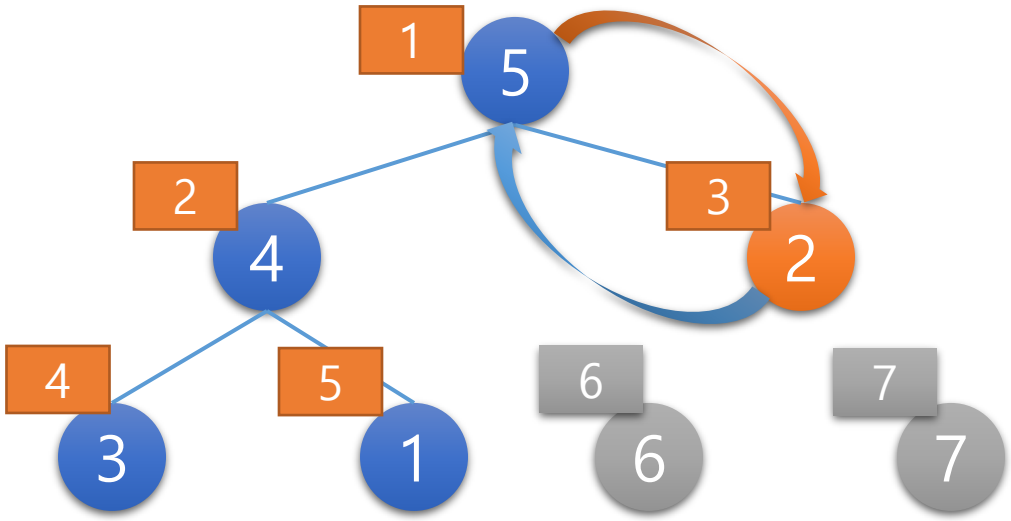


0	1	2	3	4	5	6	7
	2	4	5	3	1	6	7

Heap(max heap)에서 원소를 삭제하기

힙에 저장된 2, 4, 5, 3, 1 에 대하여 힙 특성 유지시키기

pop_heap(); 진행중
swap(heap[1], heap[3]);

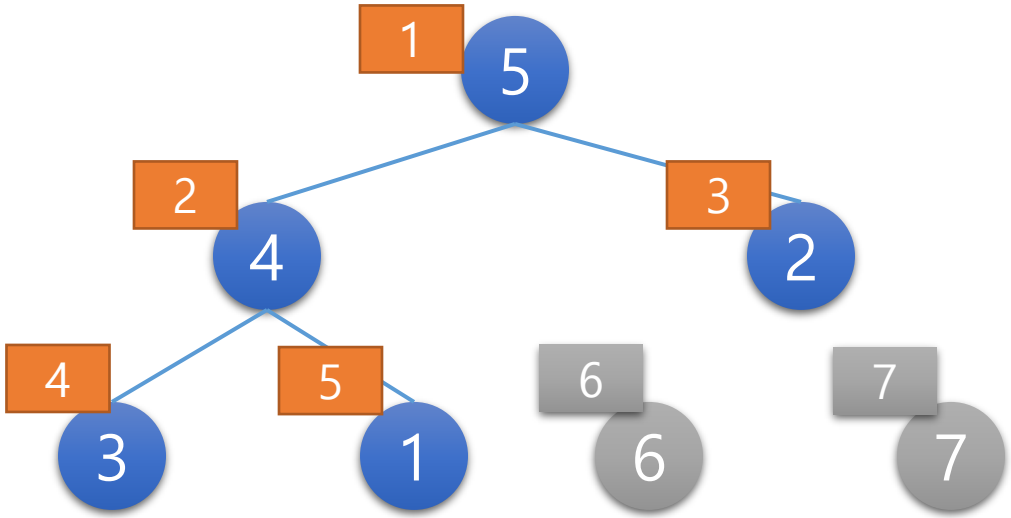


0	1	2	3	4	5	6	7
	5	4	**2**	3	1	6	7

Heap(max heap)에서 원소를 삭제하기

힙이 5, 4, 2, 3, 1 로 힙 특성이 유지되었다.

pop_heap(); 종료

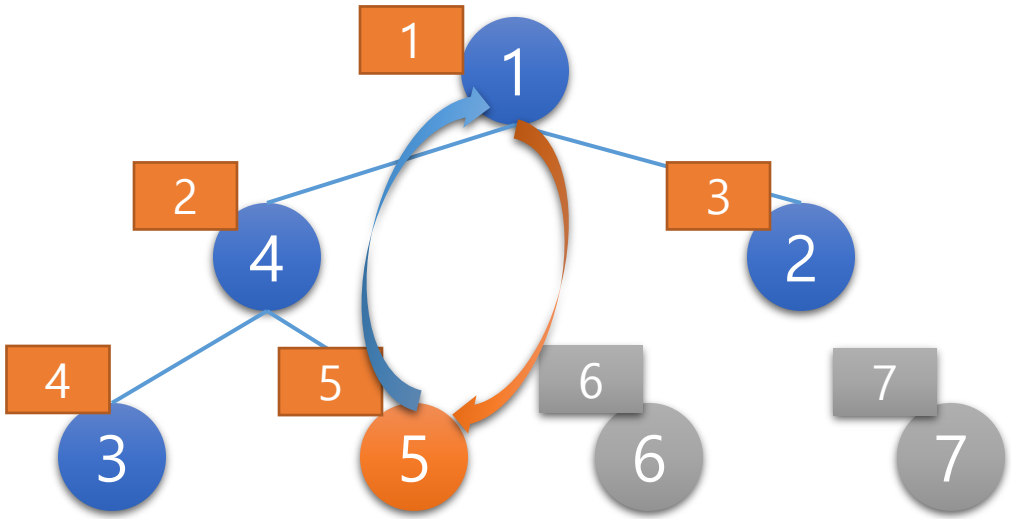


0	1	2	3	4	5	6	7
	5	4	2	3	1	6	7

Heap(max heap)에서 원소를 삭제하기

힙에 저장된 **5**, 4, 2, 3, 1 에서 **5**를 삭제

먼저 heap[1]과 heap[5]의 값을 바꾼다.
swap(heap[1], heap[5]);

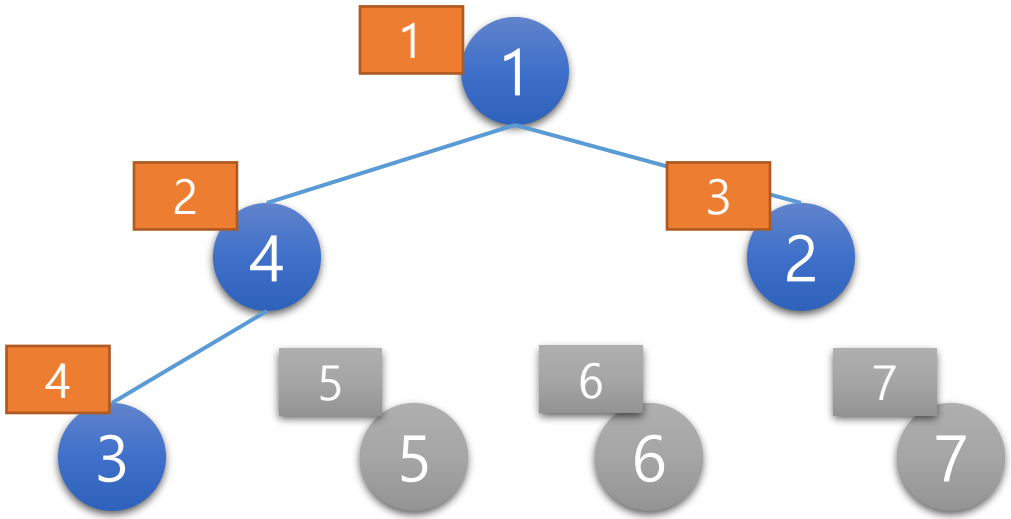


0	1	2	3	4	5	6	7
	1	4	2	3	**5**	6	7

Heap(max heap)에서 원소를 삭제하기

힙에 3, 4, 2, 1 이 남고 5가 삭제되었다.

원소의 개수를 1개 줄인다. hn = hn – 1;

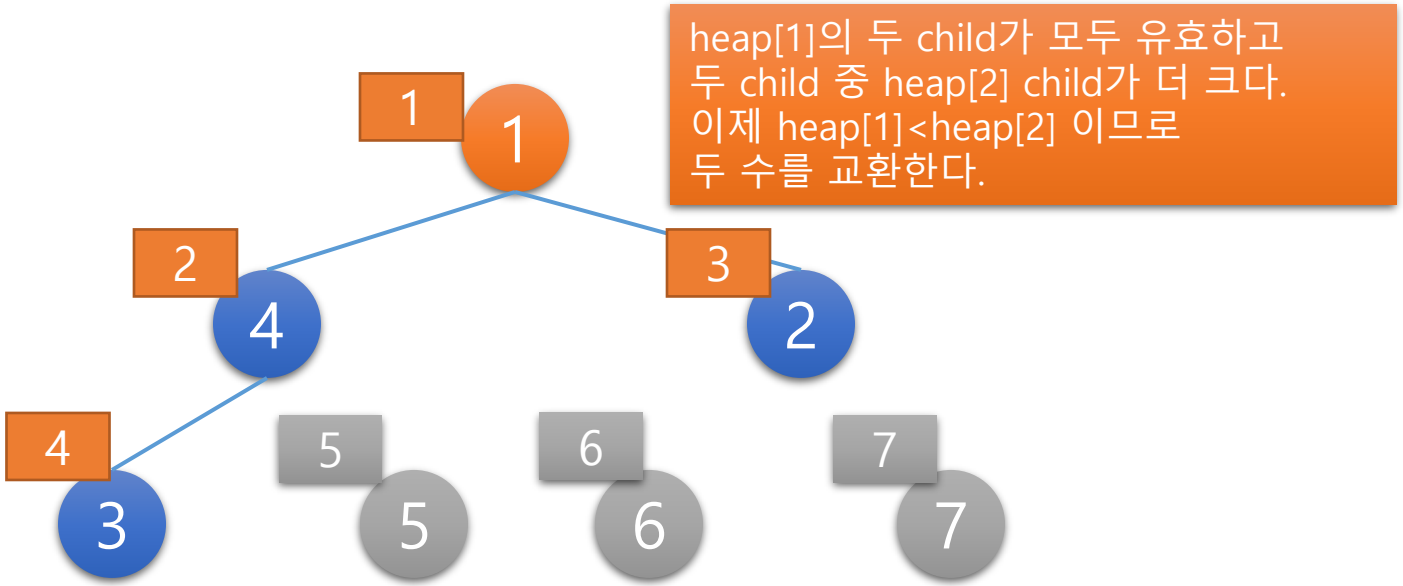


0	1	2	3	4	5	6	7
	1	4	2	3	5	6	7

Heap(max heap)에서 원소를 삭제하기

힙에 저장된 1, 4, 2, 3 에 대하여 힙 특성 유지시키기

pop_heap()함수를 호출하여 heap 특성을 유지시킨다.
pop_heap(); 시작

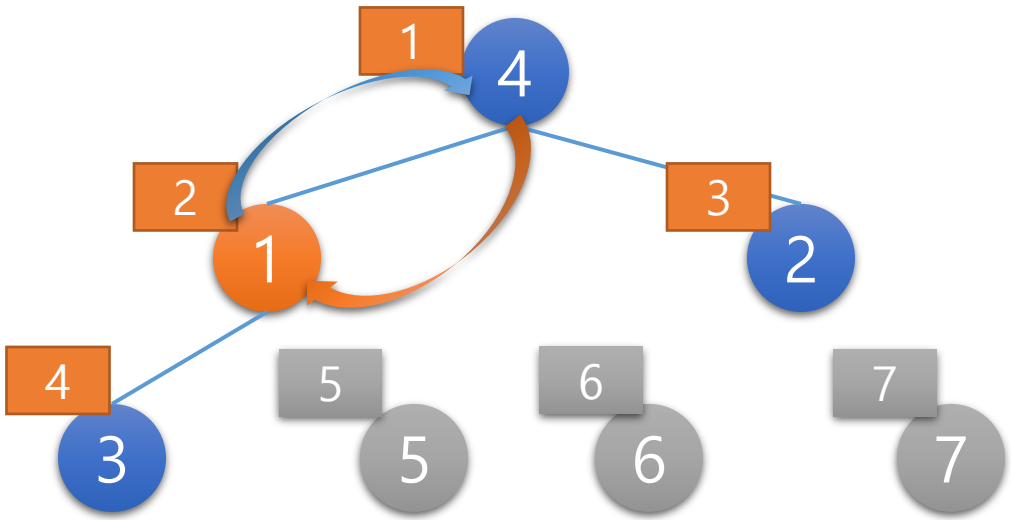


0	1	2	3	4	5	6	7
	1	4	2	3	5	6	7

Heap(max heap)에서 원소를 삭제하기

힙에 저장된 1, 4, 2, 3 에 대하여 힙 특성 유지시키기

pop_heap(); 진행중
swap(heap[1], heap[2]);



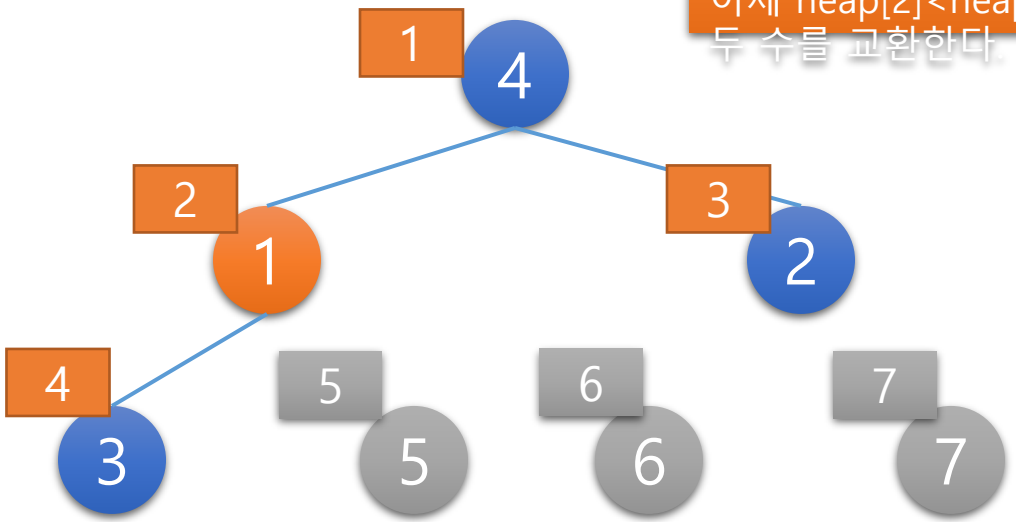
0	1	2	3	4	5	6	7
	4	1	2	3	5	6	7

Heap(max heap)에서 원소를 삭제하기

힙에 저장된 1, 4, 2, 3 에 대하여 힙 특성 유지시키기

pop_heap(); 진행중
swap(heap[1], heap[2]);

Heap[2]에 대하여 재귀적으로 적용시킨다.
heap[2]의 child는 heap[4]뿐이다.
이제 heap[2]<heap[4] 이므로
두 수를 교환한다.

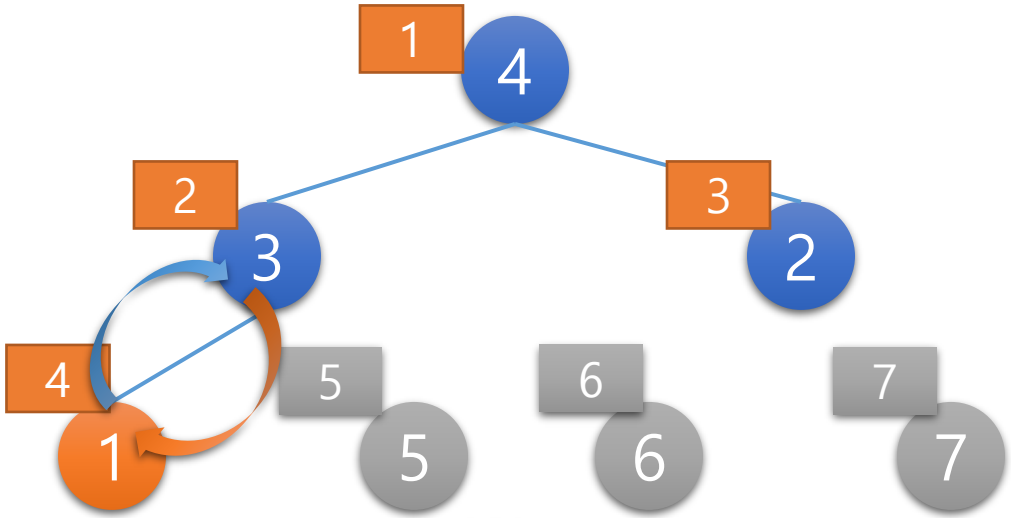


0	1	2	3	4	5	6	7
	4	1	2	3	5	6	7

Heap(max heap)에서 원소를 삭제하기

힙에 저장된 1, 4, 2, 3 에 대하여 힙 특성 유지시키기

pop_heap(); 진행중
swap(heap[2], heap[4]);

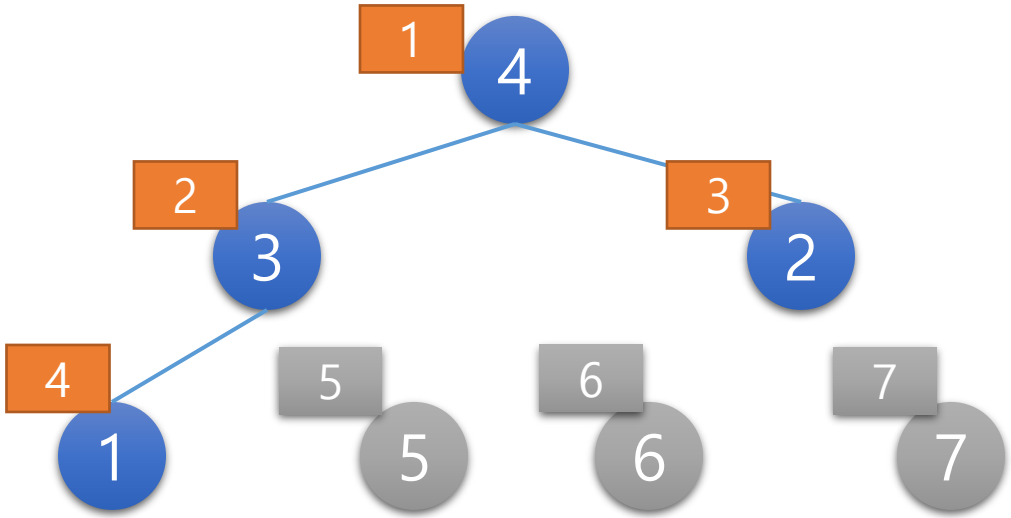


0	1	2	3	4	5	6	7
	4	3	2	1	5	6	7

Heap(max heap)에서 원소를 삭제하기

힙이 4, 3, 2, 1 로 힙 특성이 유지되었다.

pop_heap(); 종료

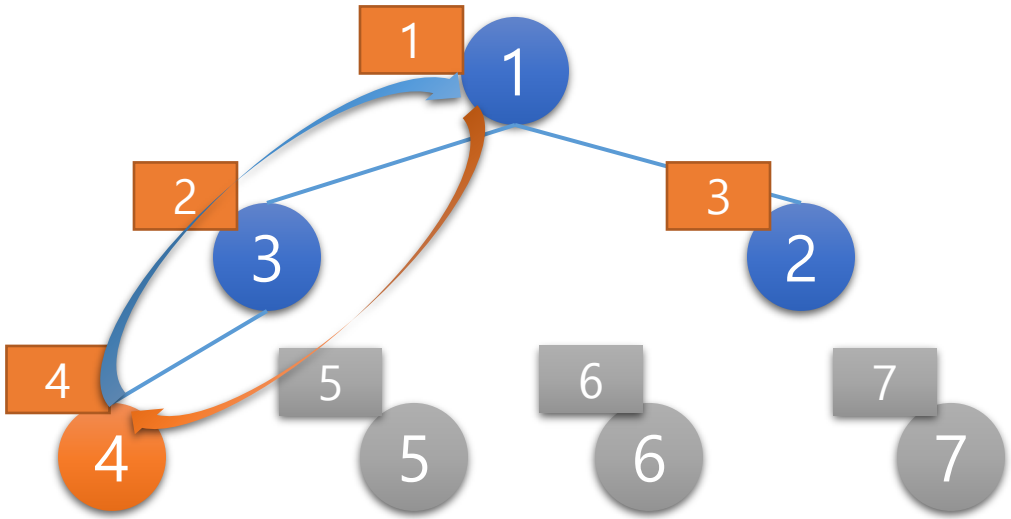


0	1	2	3	4	5	6	7
	4	3	2	1	5	6	7

Heap(max heap)에서 원소를 삭제하기

힙에 저장된 **4**, 3, 2, 1 에서 **4**를 삭제

먼저 heap[1]과 heap[4]의 값을 바꾼다.
swap(heap[1], heap[4]);

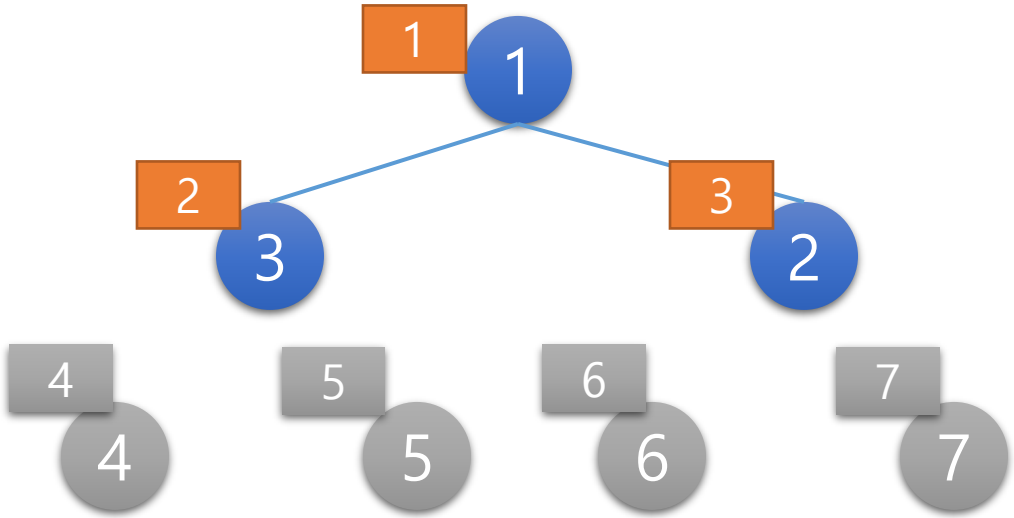


0	1	2	3	4	5	6	7
	1	3	2	**4**	5	6	7

Heap(max heap)에서 원소를 삭제하기

힙에 1, 3, 2 이 남고 4가 삭제되었다.

원소의 개수를 1개 줄인다. hn = hn – 1;

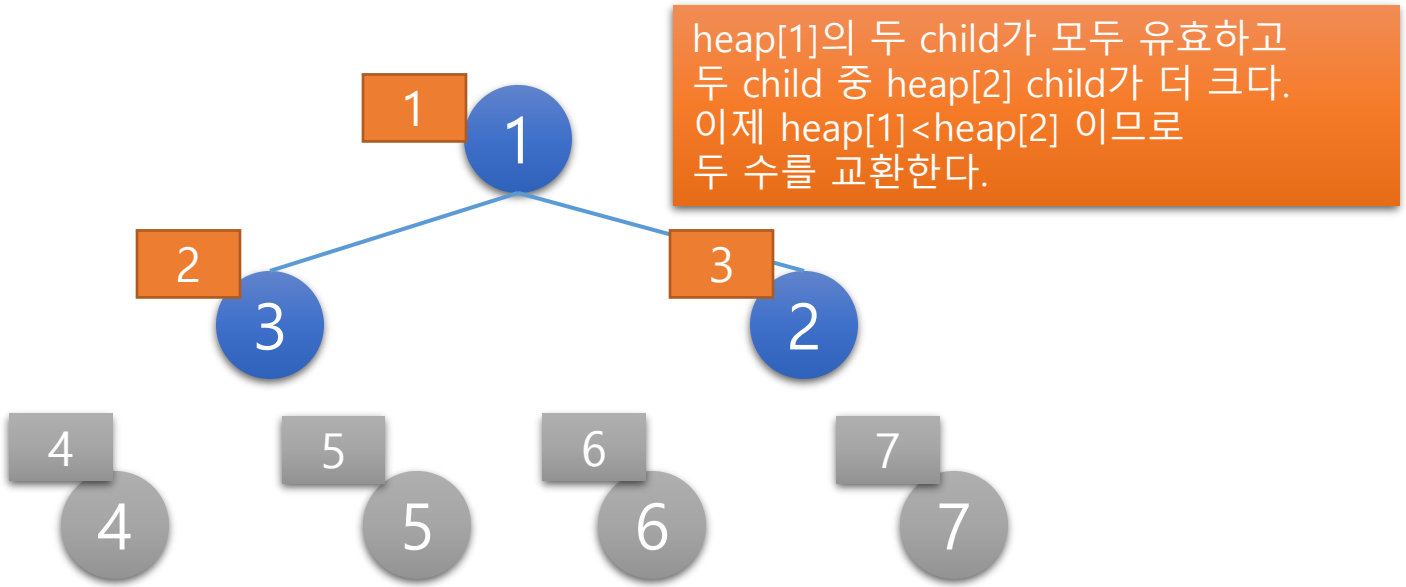


0	1	2	3	4	5	6	7
	1	3	2	4	5	6	7

Heap(max heap)에서 원소를 삭제하기

힙에 저장된 1, 3, 2에 대하여 힙 특성 유지시키기

pop_heap()함수를 호출하여 heap 특성을 유지시킨다.
pop_heap(); 시작

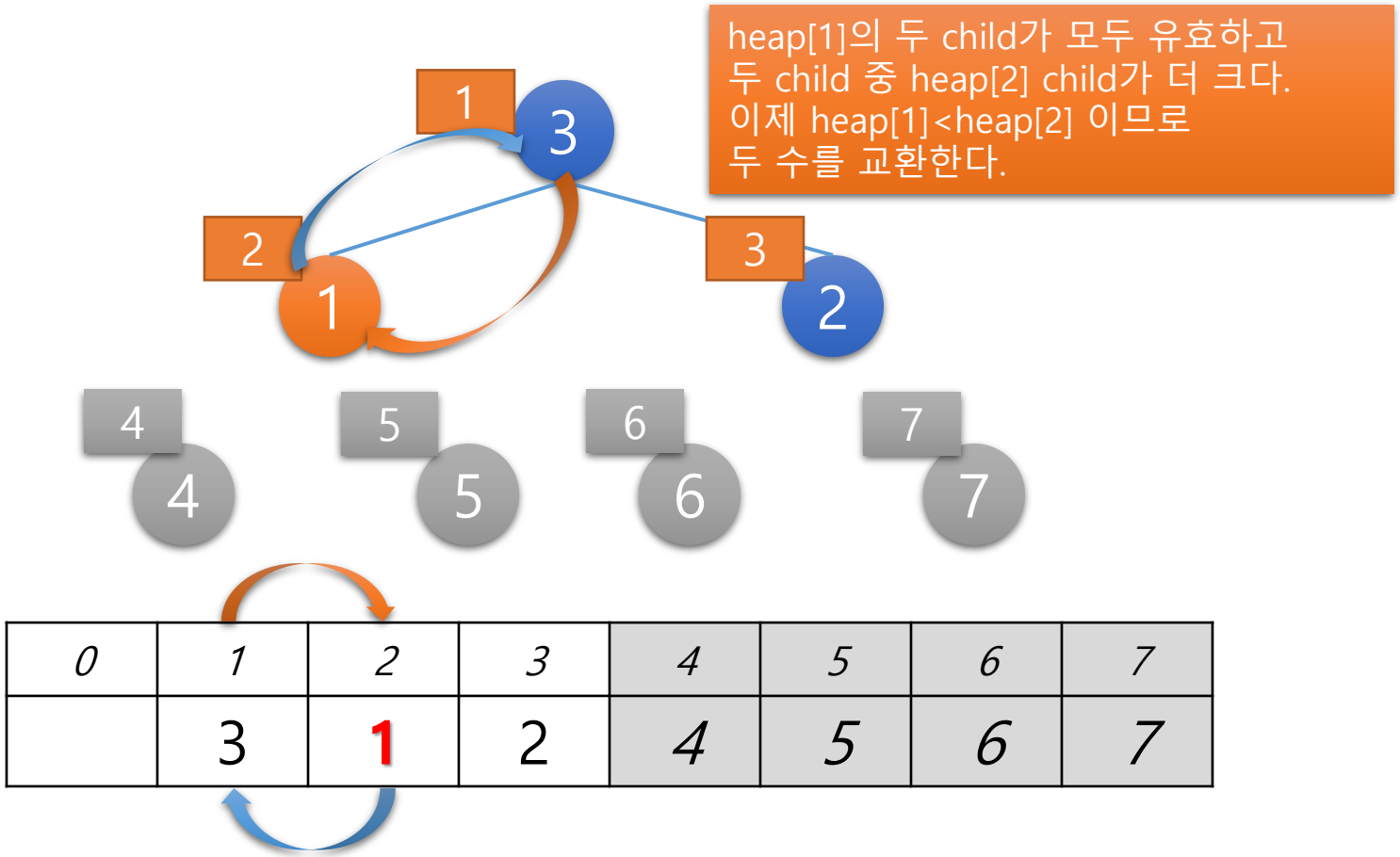


0	1	2	3	4	5	6	7
	1	3	2	4	5	6	7

Heap(max heap)에서 원소를 삭제하기

힙에 저장된 1, 3, 2에 대하여 힙 특성 유지시키기

pop_heap()함수를 호출하여 heap 특성을 유지시킨다.
pop_heap(); 진행중

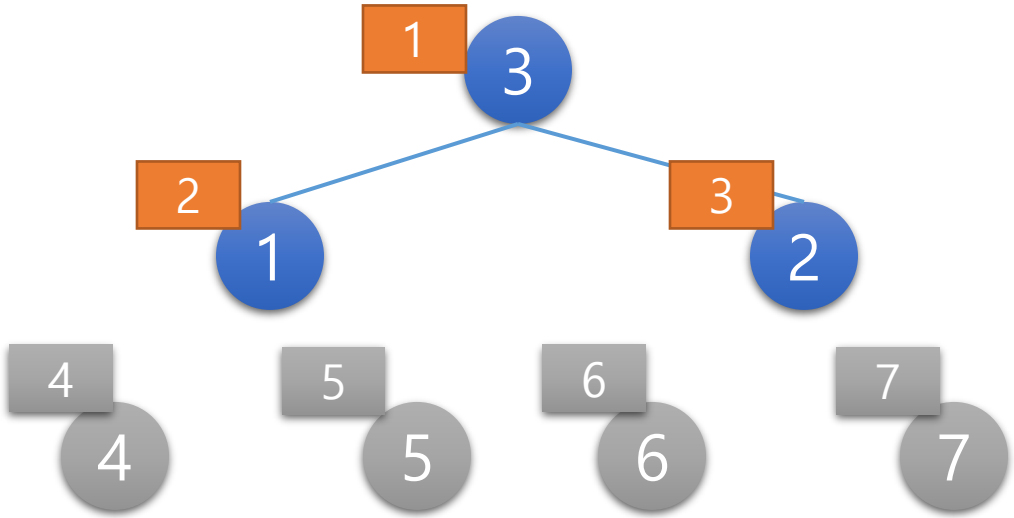


0	1	2	3	4	5	6	7
	3	1	2	4	5	6	7

Heap(max heap)에서 원소를 삭제하기

힙이 3, 1, 2 로 힙 특성이 유지되었다.

pop_heap(); 종료

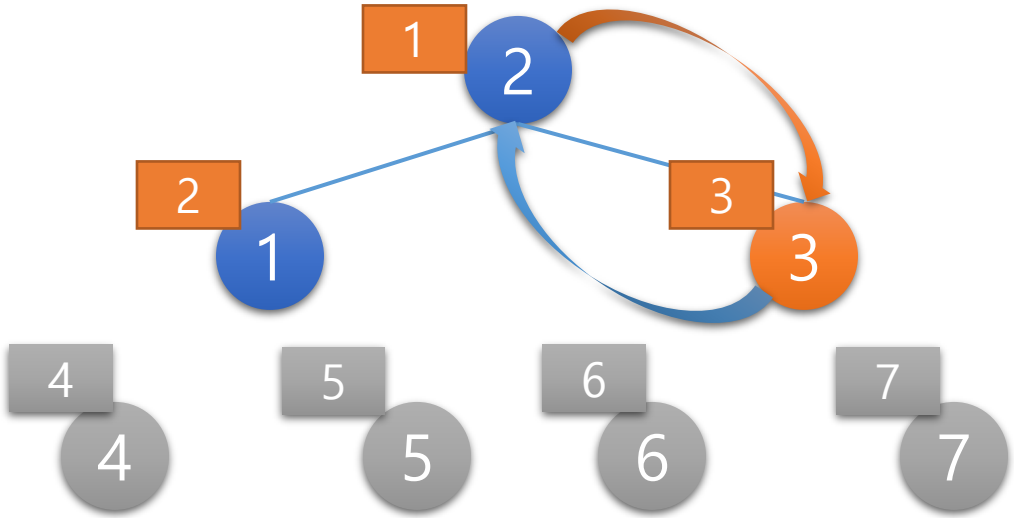


0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
	3	1	2	*4*	*5*	*6*	*7*

Heap(max heap)에서 원소를 삭제하기

힙에 저장된 **3**, 1, 2 에서 **3**를 삭제

먼저 heap[1]과 heap[3]의 값을 바꾼다.
swap(heap[1], heap[3]);

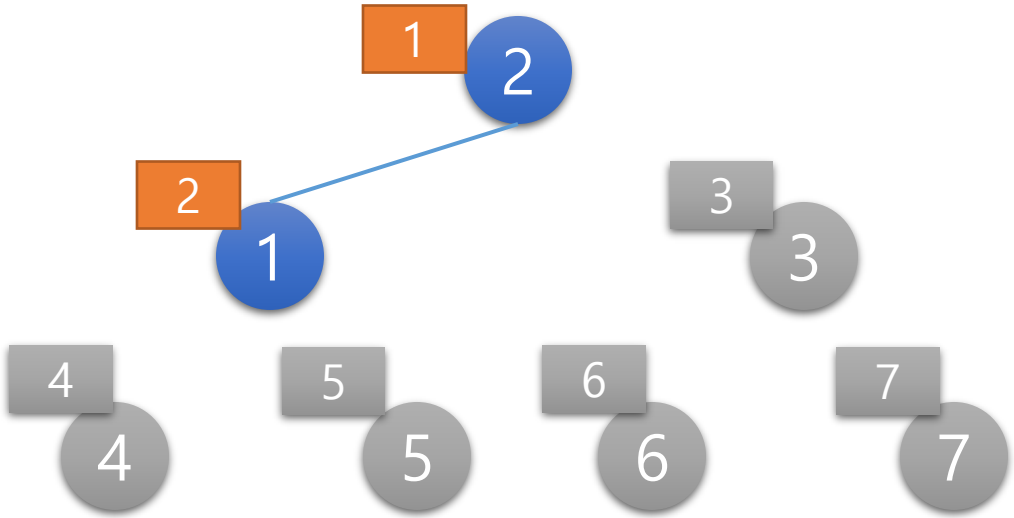


0	1	2	3	4	5	6	7
	2	1	**3**	4	5	6	7

Heap(max heap)에서 원소를 삭제하기

힙에 2, 1 이 남고 3이 삭제되었다.

원소의 개수를 1개 줄인다. hn = hn – 1;

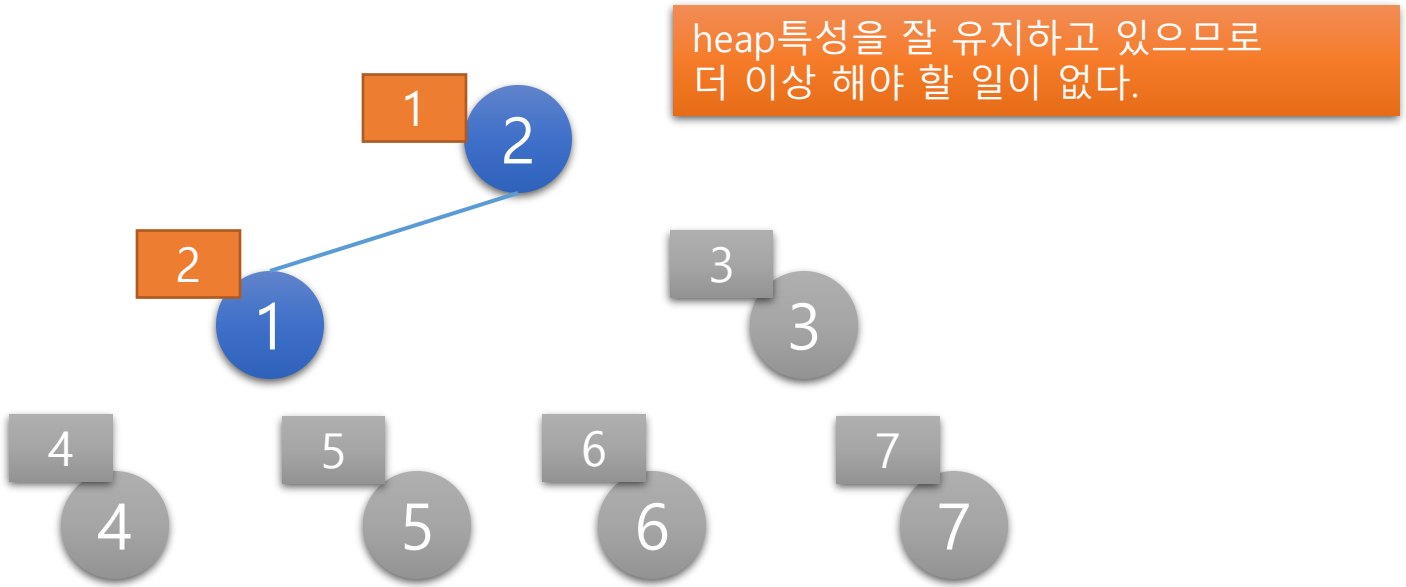


0	1	2	3	4	5	6	7
	2	1	3	4	5	6	7

Heap(max heap)에서 원소를 삭제하기

힙에 저장된 2, 1에 대하여 힙 특성 유지시키기

pop_heap()함수를 호출하여 heap 특성을 유지시킨다.
pop_heap(); 시작,종료

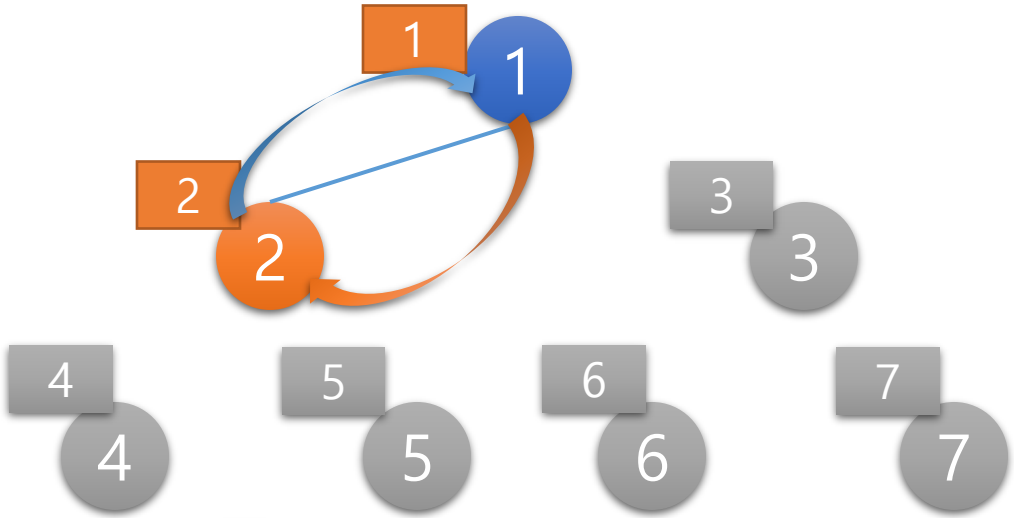


0	1	2	3	4	5	6	7
	2	1	3	4	5	6	7

Heap(max heap)에서 원소를 삭제하기

힙에 저장된 **2**, 1 에서 **2**를 삭제

먼저 heap[1]과 heap[2]의 값을 바꾼다.
swap(heap[1], heap[2]);

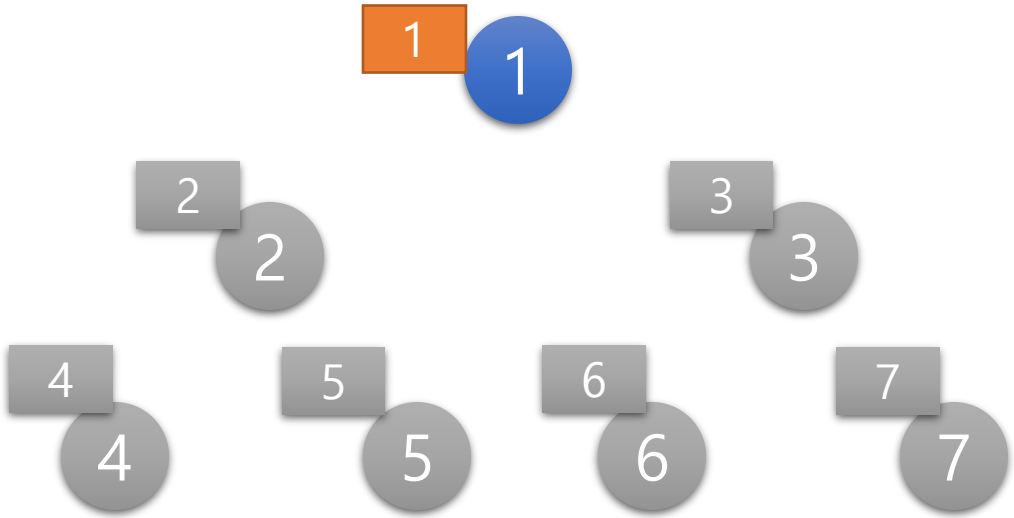


0	1	2	3	4	5	6	7
	1	**2**	3	4	5	6	7

Heap(max heap)에서 원소를 삭제하기

힙에 1 이 남고 2가 삭제되었다.

원소의 개수를 1개 줄인다. hn = hn – 1;

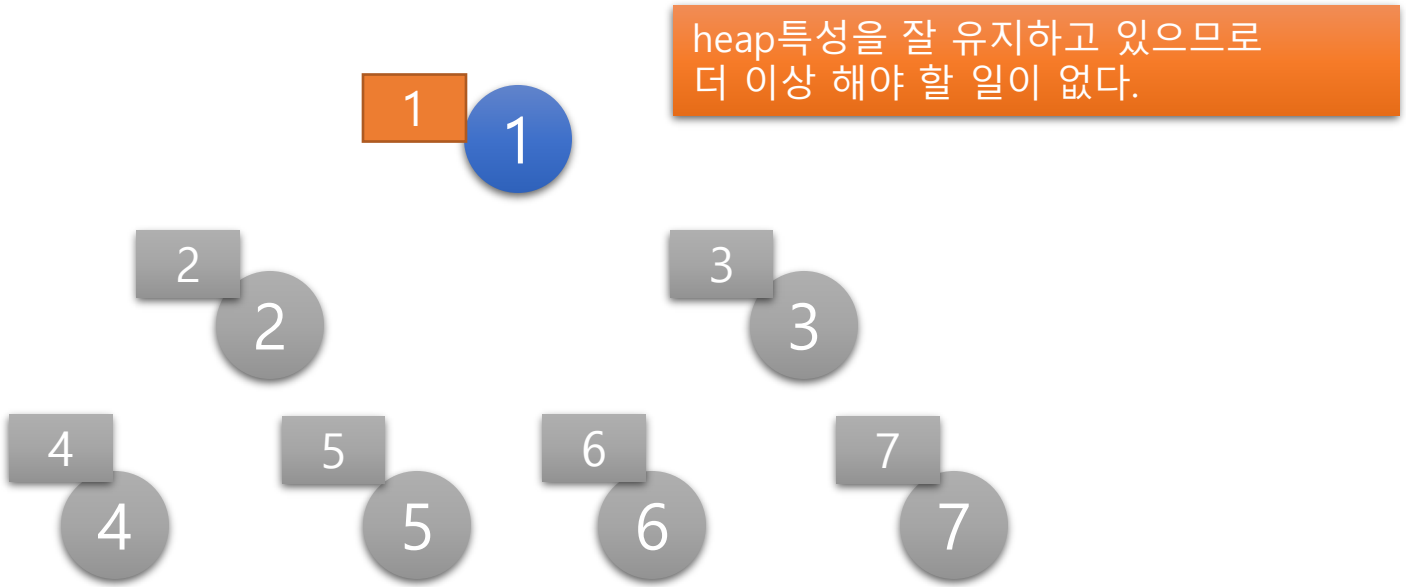


0	1	2	3	4	5	6	7
	1	2	3	4	5	6	7

Heap(max heap)에서 원소를 삭제하기

힙에 저장된 2, 1에 대하여 힙 특성 유지시키기

pop_heap()함수를 호출하여 heap 특성을 유지시킨다.
pop_heap(); 시작,종료

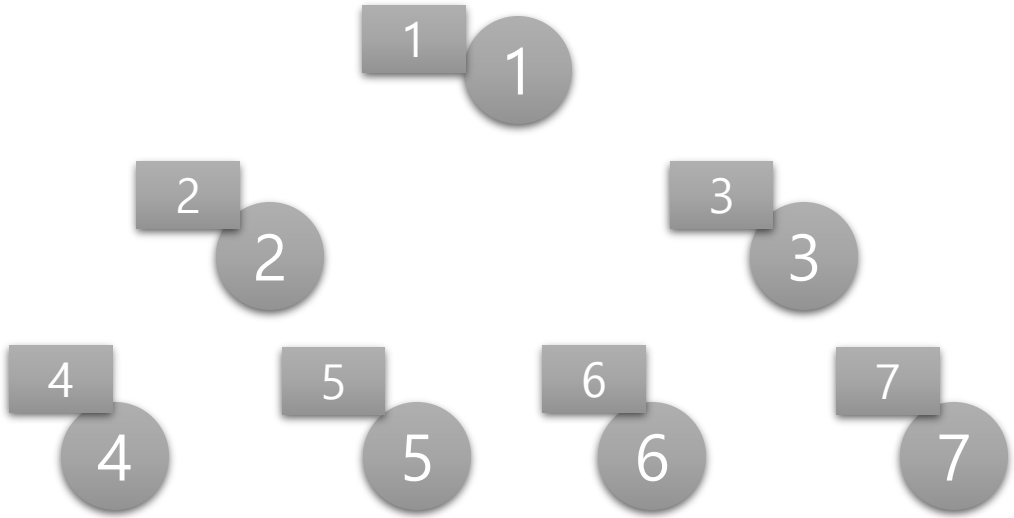


0	1	2	3	4	5	6	7
	1	2	3	4	5	6	7

Heap(max heap)에서 원소를 삭제하기

힙에 저장된 **1** 에서 **1**을 삭제

이제 모든 원소가 삭제되었다.
그리고 heap[]에는 원소가 오름차순으로 정렬되어 저장되어 있다.



0	1	2	3	4	5	6	7
	1	2	3	4	5	6	7

Pop_heap 프로세스 정리

- heap에서의 삭제는 heap[1]을 삭제하는 것이다.
  1. 삭제 후에도 max_heap의 특성(완전이진트리, 부모노드의 값>=자식노드의 값)이 유지되어야 한다.
  2. [완전이진트리] 특성을 유지하기 위하여 heap[1]과 heap[hn]을 교환한 후 hn--
  3. 이제 루트부터 단말까지 가면서 [부모노드의 값>=자식노드의 값] 특성이 트리 전체에 유지되도록 재귀적으로 적용시킨다.
- pop_heap 프로세스 :
  1. Heap[1]과 heap[hn]값을 교환한다. => 완전이진트리 속성 유지를 위하여
  2. 원소의 개수를 1개 줄인다.
  3. 인덱스 c = 2라고 초기값을 정하고 c가 hn보다 작거나 같은 동안 다음 프로세스를 반복한다.
     - c가 hn보다 작고(오른쪽 자식이 있고) heap[c+1] > heap[c]라면 c = c + 1;
     - heap[c] > heap[c/2] 라면 두 수를 교환한다. swap(heap[c/2], heap[c])
     - 그렇지 않으면 프로세스 종료.
     - c = c*2;로 바꾸고 재귀적으로 계속 진행
  4. 하나의 원소가 삭제될 때 max_heap을 유지하기 위한 시간은 O(log(hn))

최대힙 pop_heap()함수 구현 예 : user loop ver01

```
/// max_heap의 pop_heap() 함수
/// pop_heap(); 를 호출한다.

void swap( int &a, int &b) { int t=a; a=b; b=t;}
void pop_heap(){
    swap(heap[1], heap[hn--]);
   /// int c = 2  : 루트의 번호가 1 이므로 루트의 왼쪽 child의 번호는 2가 된다.
   /// c <=hn    : c 가 힙의 범위 안에 있다면
   /// c *= 2     : 다음 child의 위치로 이동
    for ( int c = 2;c <= hn; c *= 2){
        /// 오른쪽 child가 있고 그 값이 왼쪽 child값보다 크다면
        if ( c < hn && heap[c + 1] > heap[c]) c++;

        /// child값 (heap[c]) 이  paren값(heap[c/2])보다 크다면
        if (heap[c] > heap[c / 2]) swap(heap[c], heap[c / 2]);
        else break ;
    }
}
```

최대힙 pop_heap()함수 구현 예 : user loop ver02_01

```
/// max_heap의 pop_heap() 함수
/// swap을 사용하지 않는 version


void pop_heap(){
   int c = 2;
   int nd = heap[hn];     // 적절한 위치를 찾아 삽입될 data
   heap[hn--] = heap[1]; // 삭제된 data 즉 정렬된 data
   /// int c = 2  : 루트의 번호가 1 이므로 루트의 왼쪽 child의 번호는 2가 된다.
   /// c <=hn    : c 가 힙의 범위 안에 있다면
   for (;c<=hn && heap[(c+=(c<hn && heap[c+1]>heap[c]))] > nd; c <<= 1)
       heap[c >> 1] = heap[c];
   heap[c>>1] = nd;
}
```

최대힙 pop_heap()함수 구현 예 : user loop ver02_02

```
/// max_heap의 pop_heap() 함수
/// swap을 사용하지 않는 version


void pop_heap(){
   int c = 2;
   int nd = heap[hn];     // 적절한 위치를 찾아 삽입될 data
   heap[hn--] = heap[1]; // 삭제된 data 즉 정렬된 data
   /// int c = 2  : 루트의 번호가 1 이므로 루트의 왼쪽 child의 번호는 2가 된다.
   /// c <=hn    : c 가 힙의 범위 안에 있다면
   for (;c <= hn; c <<= 1){
      if (c < hn && heap[c + 1] > heap[c]) ++c;
      if (heap[c] > nd)
          heap[c >> 1] = heap[c];
   }
   heap[c>>1] = nd;
}
```

최대힙 push_heap()함수 구현 예 : user recursion

```
/// max_heap의 pop_heap() 함수

void pop ( int c){
    /// 오른쪽 child가 있고 그 값이 왼쪽 child값보다 크다면
    if (c < hn && heap[c+1] > heap[c]) c++;

    /// child값 (heap[c]) 이  paren값(heap[c/2])보다 크다면
    if (c <= hn && heap[c] > heap[c/2]) {
        swap(heap[c], heap[c/2]);
        pop (c*2); /// 다음 child의 위치로 이동
    }
    /// else return;
}

void pop_heap(){
    swap(heap[1], heap[hn--]);
    pop(2);
}
```

Min Heap

최소힙

Min heap : 최소힙

다음 2가지 특성을 만족하는 경우 **최소 힙(min heap)**이라 한다.

1. **완전이진트리 (complete binary tree)**
   : 이진 트리의 원소가 왼쪽부터 채워지는 트리를 말한다.
   : 왼쪽부터 채우는데 더 이상 채울 수 없는 경우
   새로운 레벨의 왼쪽부터 채운다.
2. **부모 노드의 값 <= 자식 노드의 값**
   min heap visualization
3. **최소 힙은 최대 힙과 비교하여 부등호가 반대인 점만 다르다.**
   ✓ 자세한 설명은 생략한다.
   ✓ 구현의 예는 다음과 같다. :
   부등호의 방향만 달라지는 점에 유의한다.

최소힙 push_heap()함수 구현 예 : user loop

```
// max_heap의 push_heap() 함수

void swap( int &a, int &b) {
    int t=a; a=b; b=t;
}

void push( int nd){
    // heap의 맨 뒤에 New Data nd 추가
    heap[++hn] = nd;
    // c는 마지막 위치부터 루트 인덱스가 아닌 동안
    for ( int c = hn; c > 1; c /= 2){
        // 부모노드의 값보다 크다면 교환
        if (heap[c] < heap[c/2])
            swap(heap[c], heap[c/2]);
        else break ;
    }
}
```

최소힙 push_heap()함수 구현 예 : user recursion

```
// max_heap의 push_heap() 함수

void swap( int &a, int &b) {
    int t=a; a=b; b=t;
}

void push( int c) {
    // c가 루트가 아니고 c노드의 값이 부모 노드의 값보다 크다면 교환
    if (c > 1 && heap[c] < heap[c/2]) {
        swap(heap[c], heap[c/2]);
        push(c / 2); // 부모의 위치로 이동
    }
}

void push_heap( int nd){
    heap[ ++hn ] = nd;
    push(hn);
}
```

최소힙 pop_heap()함수 구현 예 : user loop

```
/// max_heap의 pop_heap() 함수
/// pop_heap(); 를 호출한다.

void swap( int &a, int &b) { int t=a; a=b; b=t;}
void pop_heap(){
    swap(heap[1], heap[hn--]);
    /// int c = 2  : 루트의 번호가 1 이므로 루트의 왼쪽 child의 번호는 2가 된다.
    /// c <=hn    : c 가 힙의 범위 안에 있다면
    /// c *= 2     : 다음 child의 위치로 이동
    for ( int c=2;c <= hn; c *= 2){
        /// 오른쪽 child가 있고 그 값이 왼쪽 child값보다 크다면
        if ( c < hn && heap[c + 1] < heap[c]) c++;

        /// child값 (heap[c]) 이  paren값(heap[c/2])보다 크다면
        if (heap[c] < heap[p]) swap(heap[c], heap[c / 2]);
        else break ;
    }
}
```

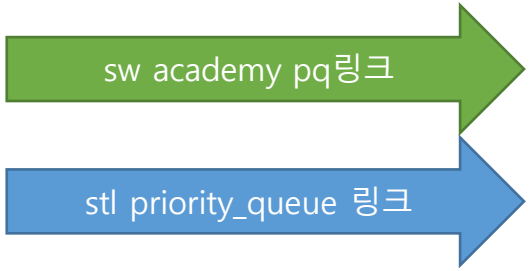
최소힙 push_heap()함수 구현 예 : user recursion

```
/// max_heap의 pop_heap() 함수

void pop ( int c){
    /// 오른쪽 child가 있고 그 값이 왼쪽 child값보다 크다면
    if (c < hn && heap[c+1] < heap[c]) c++;

    /// child값 (heap[c]) 이  paren값(heap[c/2])보다 크다면
    if (c <= hn && heap[c] < heap[c/2]) {
        swap(heap[c], heap[c/2]);
        pop (c*2); /// 다음 child의 위치로 이동
    }
    /// else return;
}

void pop_heap(){
    swap(heap[1], heap[hn--]);
    pop_heap(2);
}
```

Priority Queue
우선순위 큐

Priority Queue : 우선순위 큐

- 큐는 선입선출(First In First Out)자료구조이다.
- 우선순위 큐는 들어온 순서가 아닌 정의된 우선순위에 따라 먼저 처리된다.
- 우선순위 큐는 여러 가지 자료들로 구현할 수 있으나
  위와 같은 특성이 heap과 많이 닮아 있어 heap으로 구현하는 경우가 많다.
- 우선순위 큐를 구현 하는 경우는 다음과 같은 경우이다.
  - ✓ 한 문제에서 여러 개의 PQ(heap)을 필요로 하는 경우
  - ✓ 한 문제에서 서로 다른 성질의 PQ(또는 heap)를 필요로 하는 경우
    예를 들어 max Heap과 min Heap을 모두 사용할 필요가 있는 경우



PriorityQueue 클래스 구현 예 1 : swap()

```
const int SIZE = 1 << 20;

bool Max( int a, int b) { return a>b; }
bool Min( int a, int b) { return a<b; }
void swap( int &a, int &b) {
    int t = a; a=b; b = t;
}

struct PQ {
    int heap[SIZE], hn;

    // init함수를 통하여 결정
    bool (*cmp)( int, int);

    // order==1 : max heap
    // order==0 : min heap
    void init (int flag = 0) {
        hn = 0;
        cmp = flag? Max:Min;
    }

    int top() {
        return heap[1];
    }
    void push( int data) {
        heap[++hn] = data;
        for ( int c=hn; c>1; c>>=1) {
            if (cmp(heap[c], heap[c>>1]))
                swap(heap[c], heap[c>>1]);
            else break;
        }
    }
    void pop() {
        swap(heap[1], heap[hn--]);
        for ( int c=2; c<=hn; c<<=1) {
            if (c<hn && cmp(heap[c+1], heap[c])) c++;
            if (cmp(heap[c], heap[c>>1]))
                swap(heap[c], heap[c>>1]);
            else break;
        }
    }
} maxpq, minpq;

int main() {
    maxpq.init(1);
    minpq.init(0);
    //. . .
    return 0;
}
```

PriorityQueue 클래스 구현 예 2 : no_swap()

```
const int SIZE = 1 << 20;

bool Max( int a, int b) { return a>b; }
bool Min( int a, int b) { return a<b; }
// void swap( int &a, int &b) {
//    int t = a; a=b; b = t;
// }

struct PQ{
    int heap[LM], hn;

    // init함수를 통하여 결정
    bool (*cmp)( int, int);

    // order==1 : max heap
    // order==0 : min heap
    void init(int flag = 0){
        hn = 0;
        cmp = flag? Max:Min;
    }

    int top() {
        return heap[1];
    }
    void push(int nd){
        int c = ++hn;
        for(;c>1 && cmp(nd, heap[c>>1]);c>>=1)
            heap[c] = heap[c>>1];
        heap[c] = nd;
    }

    void pop(){
        int nd = heap[hn], p = 1;
        heap[hn--] = heap[1];
        for(int c=p<<1; c<=hn; p=c, c<<=1){
            if(c<hn && cmp(heap[c+1], heap[c])) ++c;
            if(!cmp(heap[c], nd)) break;
            heap[p] = heap[c];
        }
        heap[p] = nd;
    }
} maxpq, minpq;

int main() {
    maxpq.init(1);
    minpq.init(0);
    //. . .
    return 0;
}
```

PriorityQueue 클래스 구현 예 : 템플릿 version

```
const int LM = 20010;

template<class T>
bool Max(T&a, T&b) { return a > b;}

template<class T>
bool Min(T&a, T&b) { return a < b;}

template<class T>
void swap(T&a, T&b){
    T t = a; a = b; b = t;
}

template<class T>
struct PriorityQueue{
    T heap[LM];
    int hn;
    bool (*comp)(T&a, T&b);
    void init( int sign){
        hn = 0;
        comp = sign? Max<T>:Min<T>;
    }
    T top(){ return heap[1];}
    int size(){ return hn;}
    bool empty(){ return hn==0;}
    void push_heap( int num){
        heap[++hn] = num;
        int c = hn;
        for (;c>1 ; c >>= 1){
            if (comp(heap[c], heap[c>>1]))
                swap(heap[c], heap[c>>1]);
            else break;
        }
    }
    T pop_heap(){
        T ret=heap[1];
        swap(heap[1], heap[hn--]);
        for (int c = 2;c <= hn; c <<= 1){
            if (c < hn && comp(heap[c+1], heap[c])) c++;
            if (comp(heap[c], heap[c>>1]))
                swap(heap[c], heap[c>>1]);
            else break;
        }
        return ret;
    }
};
PriorityQueue< int > minpq, maxpq;

int main() {
    maxpq.init(1);
    minpq.init(0);
    //. . .
    return 0;
}
```

Priority Queue
우선순위 큐

real time update version

Sample source – real time update PQ 1

```
const int LM = (int)1e5 + 5;
int Q, value[LM];

bool Max(int a, int b) {
    if (value[a] != value[b]) return value[a] > value[b];
    return a > b;
}
bool Min(int a, int b) {
    if (value[a] != value[b]) return value[a] < value[b];
    return a < b;
}

struct PQ {
    int heap[LM];  // id: value[]'s index
    int index[LM]; // for rtu
    int hn;
    bool(*cmp)(int, int);
    void init(int flag) {
        hn = 0;
        cmp = flag ? Max : Min;
    }
    void swap(int&a, int&b) {
        int t = a; a = b; b = t;
        t = index[a]; index[a] = index[b]; index[b] = t;
    }
    int top() {
        if (hn == 0) return -1;
        return heap[1];
    }
    void push(int nd) {
        heap[++hn] = nd;
        index[nd] = hn;
        upheap(hn);
    }
    void pop() {
        if (hn == 0) return;
        swap(heap[1], heap[hn]);
        index[heap[hn--]] = 0;
        downheap(1);
    }
    void upheap(int c) {
        for (; c > 1; c >>= 1) {
            if (cmp(heap[c], heap[c >> 1]))
                swap(heap[c], heap[c >> 1]);
            else break;
        }
    }
    void downheap(int c) {
        for (c <<= 1 ; c <= hn; c <<= 1) {
            if (c < hn && cmp(heap[c + 1], heap[c])) c++;
            if (cmp(heap[c], heap[c >> 1]))
                swap(heap[c], heap[c >> 1]);
            else break;
        }
    }
    void update(int nd) {
        int idx = index[nd];
        if (idx == 0) return;
        upheap(idx);
        downheap(idx);
    }
    void erase(int nd) {
        int idx = index[nd];
        if (idx == 0) return;
        swap(heap[idx], heap[hn]);
        index[heap[hn--]] = 0;
        if(hn){ upheap(idx); downheap(idx);
        }
    }
}minpq, maxpq;
```

Sample source – real time update PQ 2

```
bool Max(int a, int b) { return point[a] != point[b]? point[a] > point[b]:a < b;}
bool Min(int a, int b) { return point[a] != point[b]? point[a] < point[b]:a > b;}
struct PQ {
    int hp[LM + 1], index[LM], hn;
    bool (*cmp)(int, int);
    void init(bool(*f)(int, int)) { cmp = f, hn = 0; }
    int top() { return hp[1]; }
    void push(int nid) { up(nid, ++hn); }
    void pop() { erase(top()); }
    void update(int nid) {
        if (index[nid] == 0) return;
        up(nid, index[nid]), down(nid, index[nid]);
    }
    void erase(int tg) {
        if (index[tg] == 0) return;                 // 이미 삭제된 경우
        int nid = hp[hn--];
        index[nid] = index[tg], index[tg] = 0;      // 삭제 표시
        if (tg == nid) return;                      // 마지막 데이터 삭제인 경우
        up(nid, index[nid]), down(nid, index[nid]);// 중간 id삭제의 경우 up이 될 수도 있다.
    }
    void up(int id, int c) {
        for (; c > 1 && cmp(id, hp[c / 2]); c /= 2)
            hp[c] = hp[c / 2], index[hp[c]] = c;
        hp[c] = id, index[id] = c;
    }
    void down(int id, int c) {
        for (c *= 2; c <= hn && cmp(hp[c += (c < hn && cmp(hp[c + 1], hp[c]))], id); c *= 2)
            hp[c / 2] = hp[c], index[hp[c]] = c / 2;
        hp[c / 2] = id, index[id] = c / 2;
    }
}maxpq[10], minpq[10];
```

Sample source – real time update PQ 3

```
struct PQ {
    int hn, hp[LM], idx[LM];
    bool cmp(int a, int b) { return prior[a] > prior[b]; }
    void clear() { hn = 0; }
    int size() { return hn; }
    int top() { return hp[1]; }
    void push(int nd) { up(nd, ++hn);}
    void pop() { down(hp[hn--], 1); }
    void update(int nd) {
        if (idx[nd] > hn) return;
        up(nd, idx[nd]), down(nd, idx[nd]);
    }
    void erase(int tg) {
        idx[hp[hn]] = idx[tg];
        update(hp[hn--]);
    }
    void up(int nd, int c) {
        for (; c > 1 && cmp(nd, hp[c >> 1]); c >>= 1)
            hp[c] = hp[c >> 1], idx[hp[c]] = c;
        hp[c] = nd, idx[nd] = c;
    }
    void down(int nd, int c) {
        for (c <<= 1; c <= hn && cmp(hp[c += (c < hn && cmp(hp[c + 1], hp[c]))], nd); c <<= 1)
            hp[c >> 1] = hp[c], idx[hp[c]] = c >> 1;
        hp[c >> 1] = nd, idx[nd] = c >> 1;
    }
}pq;
```

Priority Queue
우선순위 큐

Lazy update version

Sample source – lazy update PQ

```
consvt int LM = (int)1e5 + 5;

int value[LM];
struct Data {
    int id, val;
};

bool Max(Data&a, Data&b) {
    if (a.val != b.val) return a.val > b.val;
    return a.id > b.id;
}
bool Min(Data&a, Data&b) {
    if (a.val != b.val) return a.val < b.val;
    return a.id < b.id;
}
struct PQ {
    Data heap[LM * 2];
    int hn;
    bool(*cmp)(Data&, Data&);
    void init(int flag) {
        hn = 0;
        cmp = flag ? Max : Min;
    }
    void swap(Data&a, Data&b) {
        Data t = a; a = b; b = t;
    }
    int top() {
        while (hn && value[heap[1].id] != heap[1].val)
            pop();
        if (hn == 0) return -1;
        return heap[1].id;
    }
    void push(Data t) {
        heap[++hn] = t;
        for (int c = hn; c > 1; c >>= 1) {
            if (cmp(heap[c], heap[c >> 1]))
                swap(heap[c], heap[c >> 1]);
            else break;
        }
    }
    void pop() {
        if (hn == 0) return;
        swap(heap[1], heap[hn]);
        hn--;
        for (int c = 2; c <= hn; c <<= 1) {
            if (c < hn && cmp(heap[c + 1], heap[c])) c++;
            if (cmp(heap[c], heap[c >> 1]))
                swap(heap[c], heap[c >> 1]);
            else break;
        }
    }
};
```

관련 문제

- jungol_2082 힙정렬2
- jungol_1318 못생긴 수
- jungol_1190 모두 더하기
- jungol_1929 책꽂이 만들기
- jungol_1570 중앙값
- jungol_3337 쇼핑몰

감사합니다.