

알튜비튜

우선순위 큐

오늘은 STL에서 제공하는 container adaptor인 priority queue에 대해 알아봅니다.
가장 최근의 데이터를 뽑는 스택, 제일 먼저 들어갔던 데이터를 뽑는 큐와 달리 우선순위가 가장 높은 데이터를 뽑는 자료구조입니다.

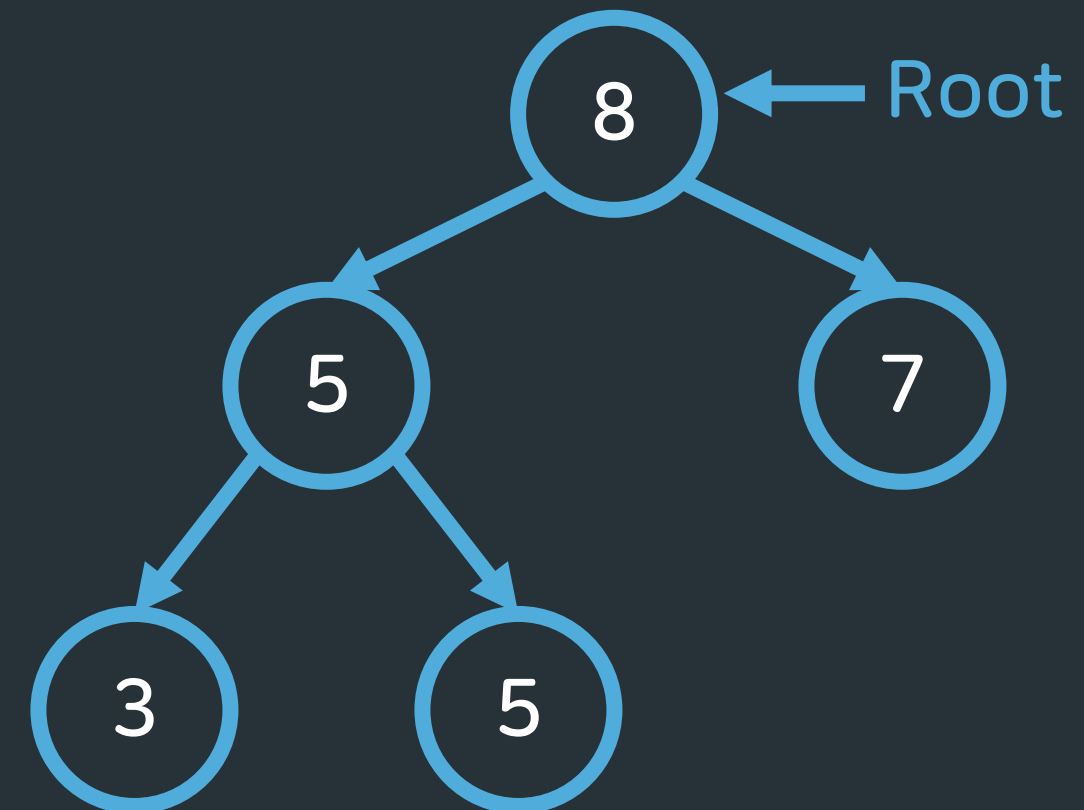


— FAMILY —
EMERGENCY ROOM



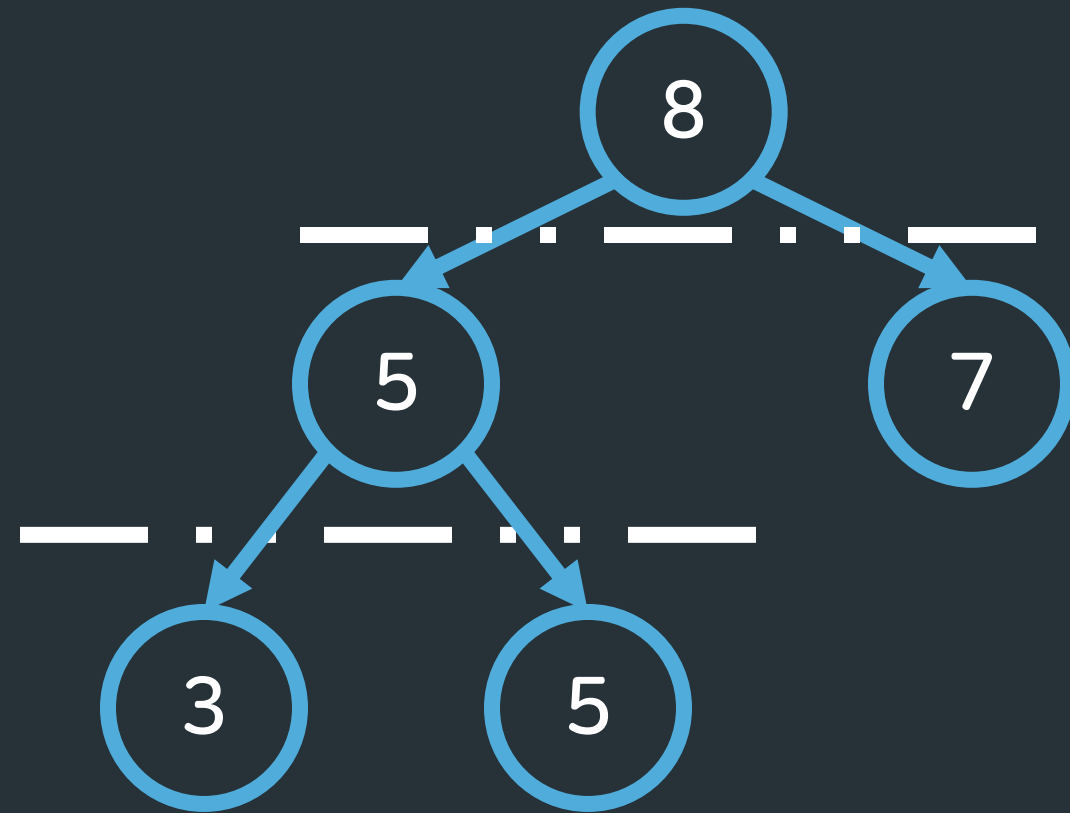
Priority Queue

- 우선순위가 높은 데이터가 먼저 나옴
- 자료의 Root 노드에서만 모든 연산이 이루어짐
- 모든 연산에 대한 시간 복잡도는 $O(\log n)$
- Heap으로 구현
- Heap의 조건
 1. 완전 이진 트리
 2. 상위 노드의 값은 모든 하위 노드의 값보다 우선순위가 크거나 같다



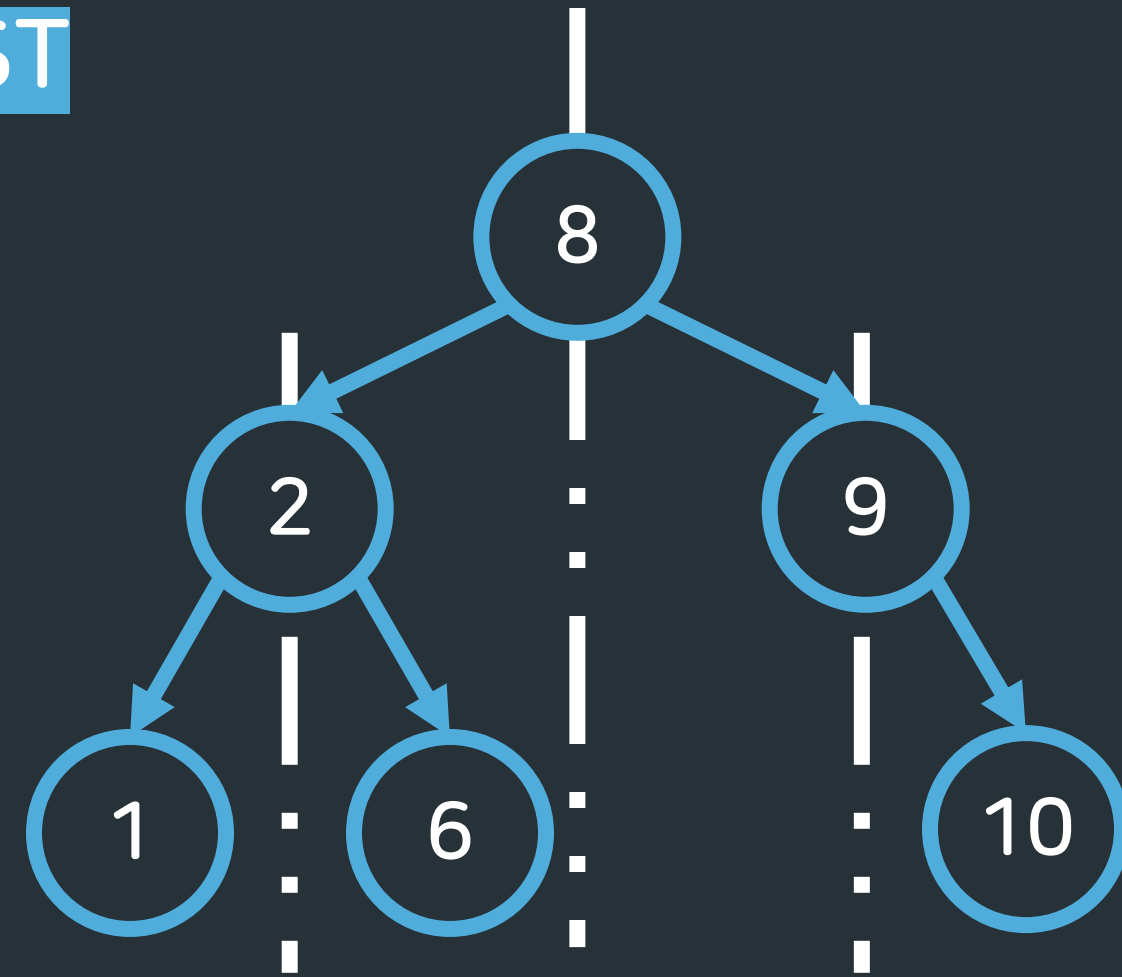
Heap과 BST의 차이

Heap



(상위) \geq (하위)
상하관계

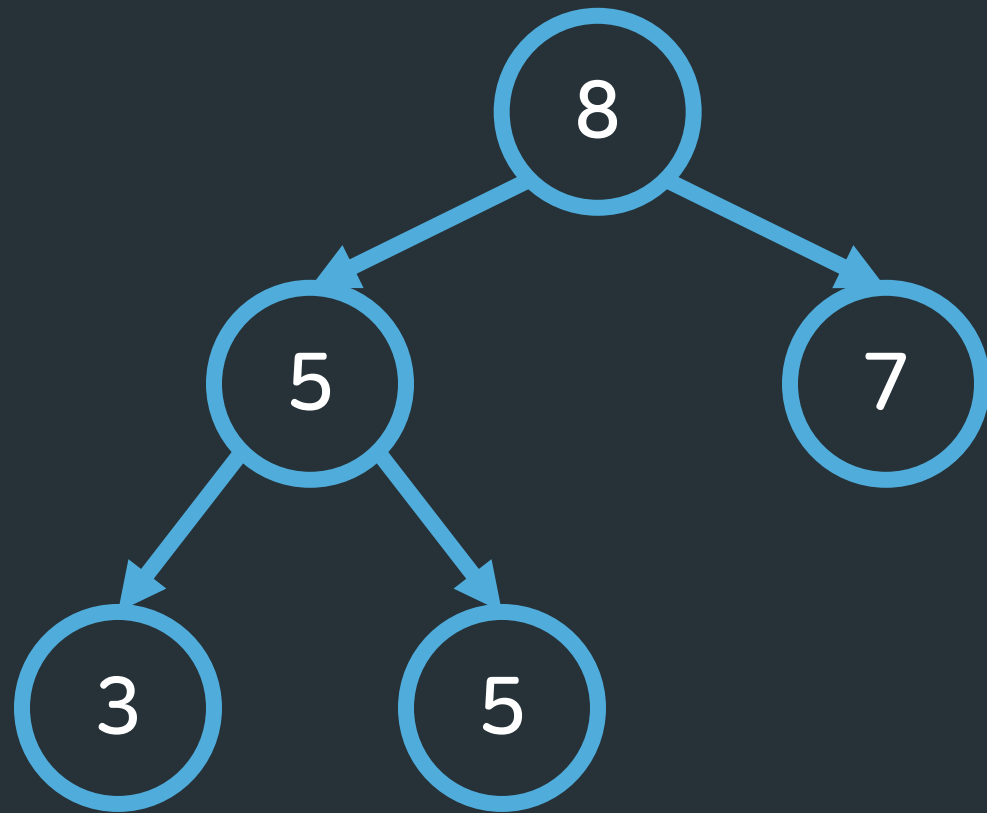
BST



(왼쪽) $<$ (루트) $<$ (오른쪽)
좌우관계

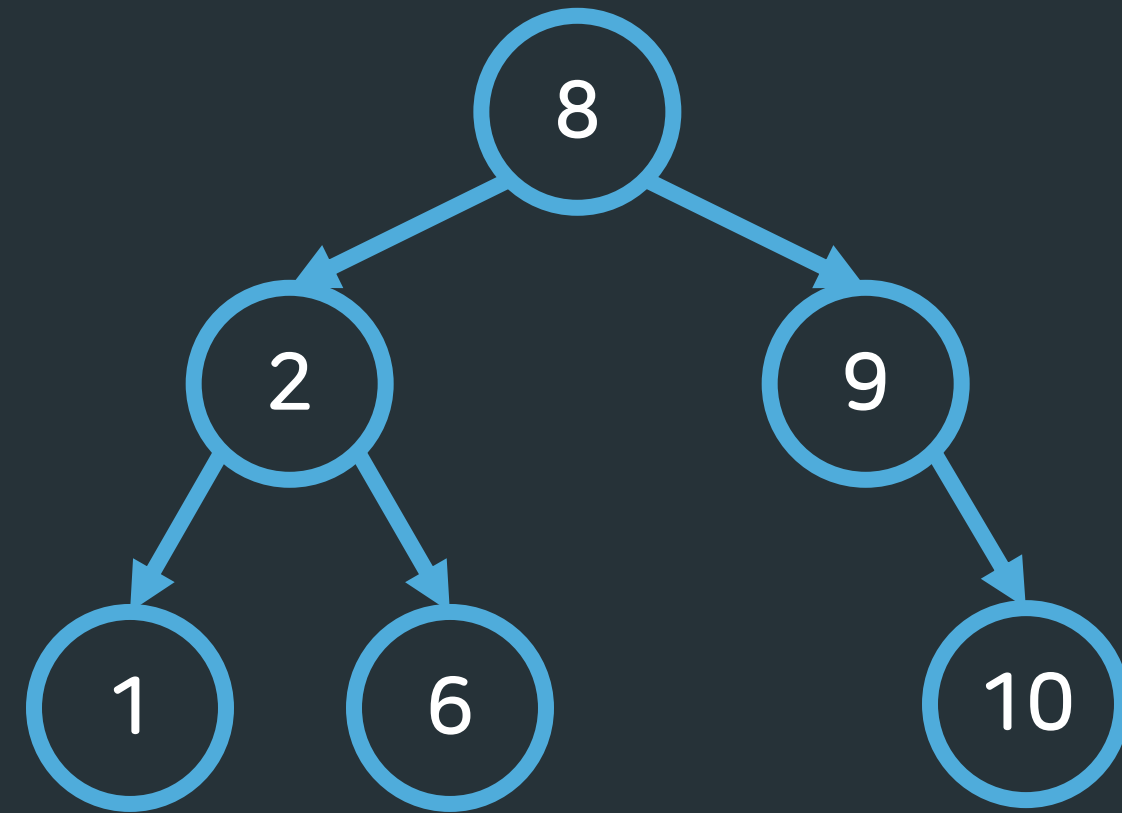
Heap과 BST의 차이

Heap

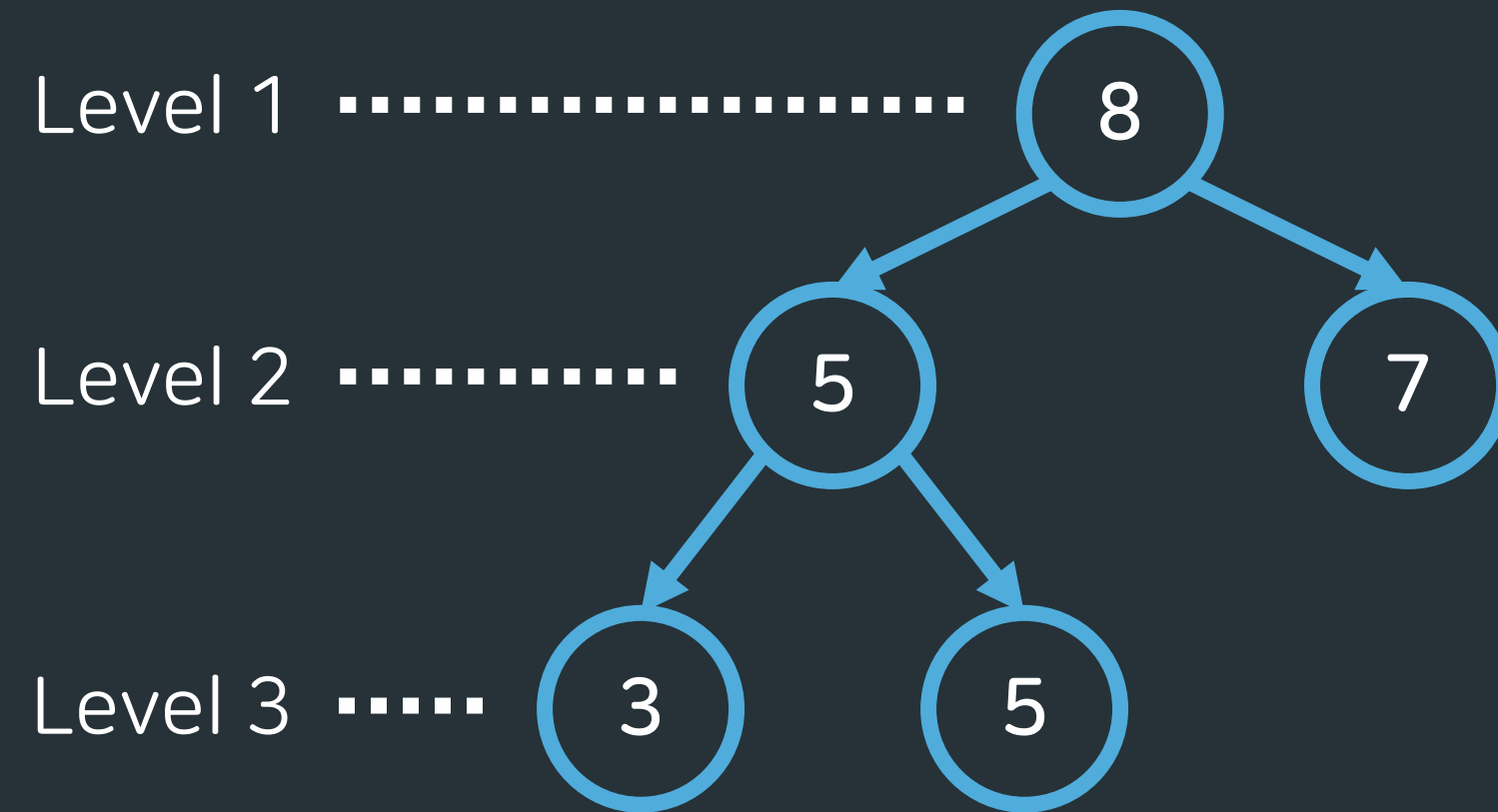


중복 0
완전 이진 트리

BST



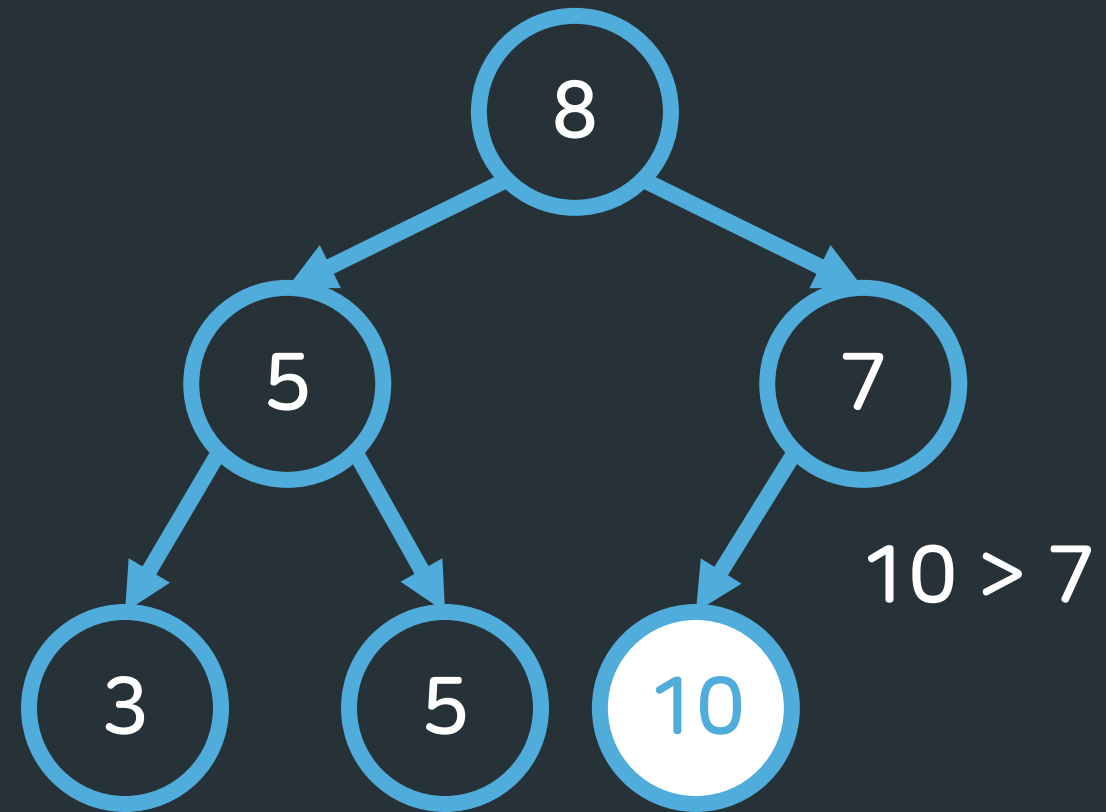
중복 X
완전 이진 트리일 필요 없음



Complete Binary Tree

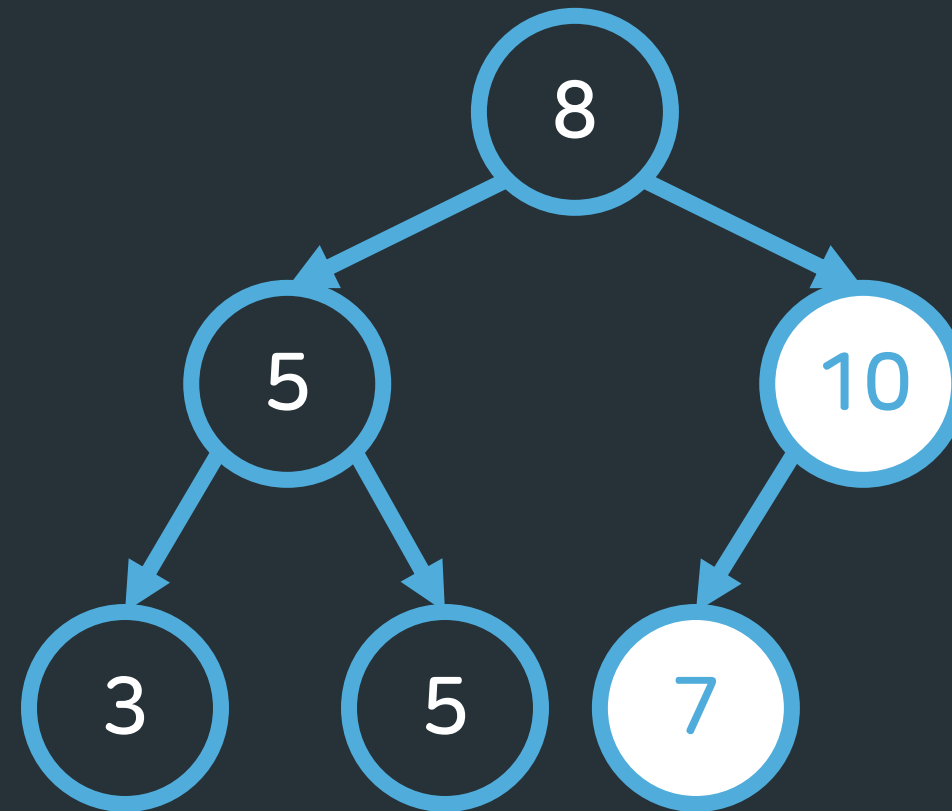
- 마지막 레벨을 제외하고 모든 레벨을 다 채움
- 마지막 레벨의 모든 노드는 왼쪽부터 빈 공간 없이 채움

최대 힙에 데이터 삽입



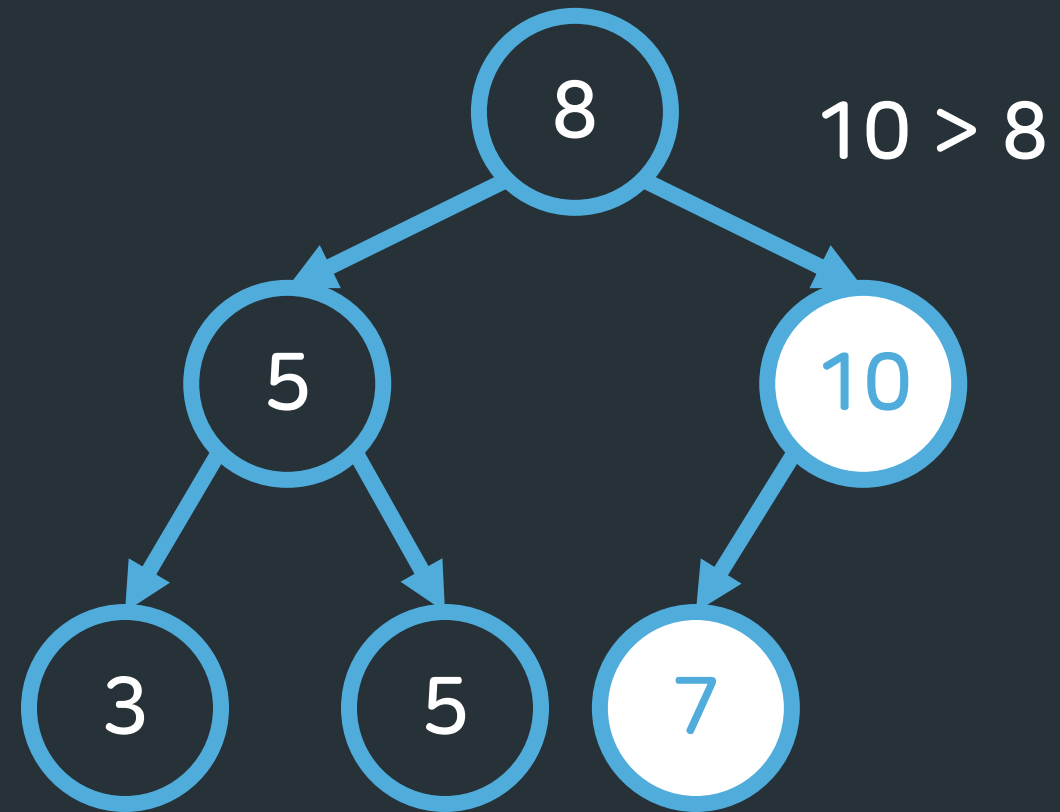
key = 10

최대 힙에 데이터 삽입



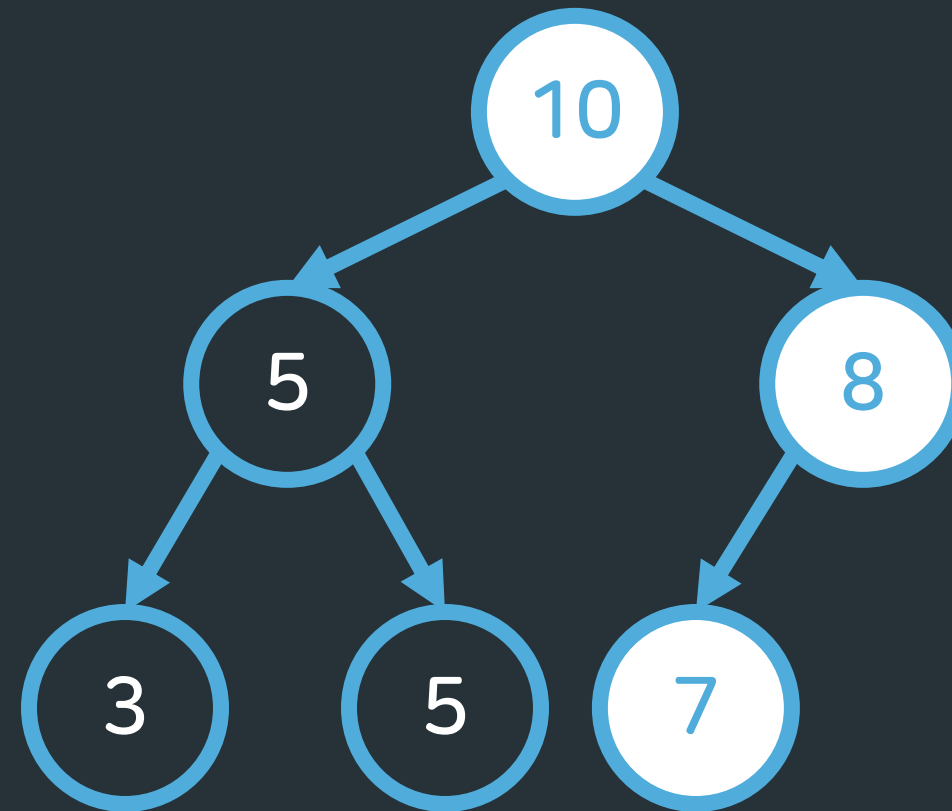
key = 10

최대 힙에 데이터 삽입



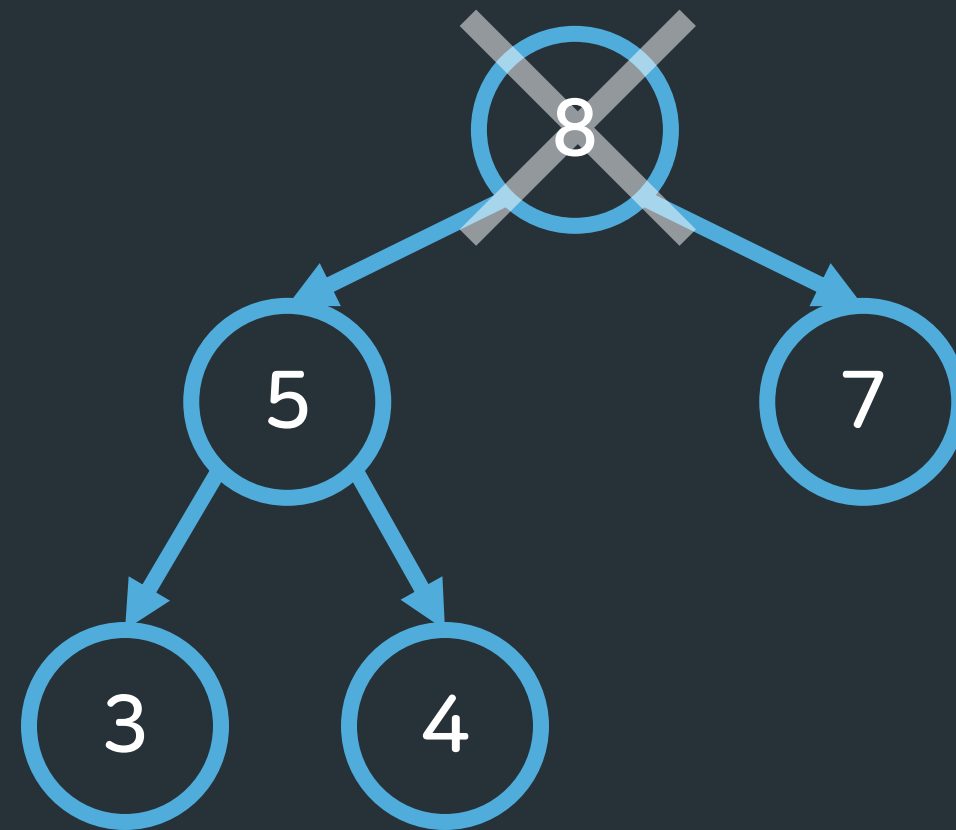
key = 10

최대 힙에 데이터 삽입

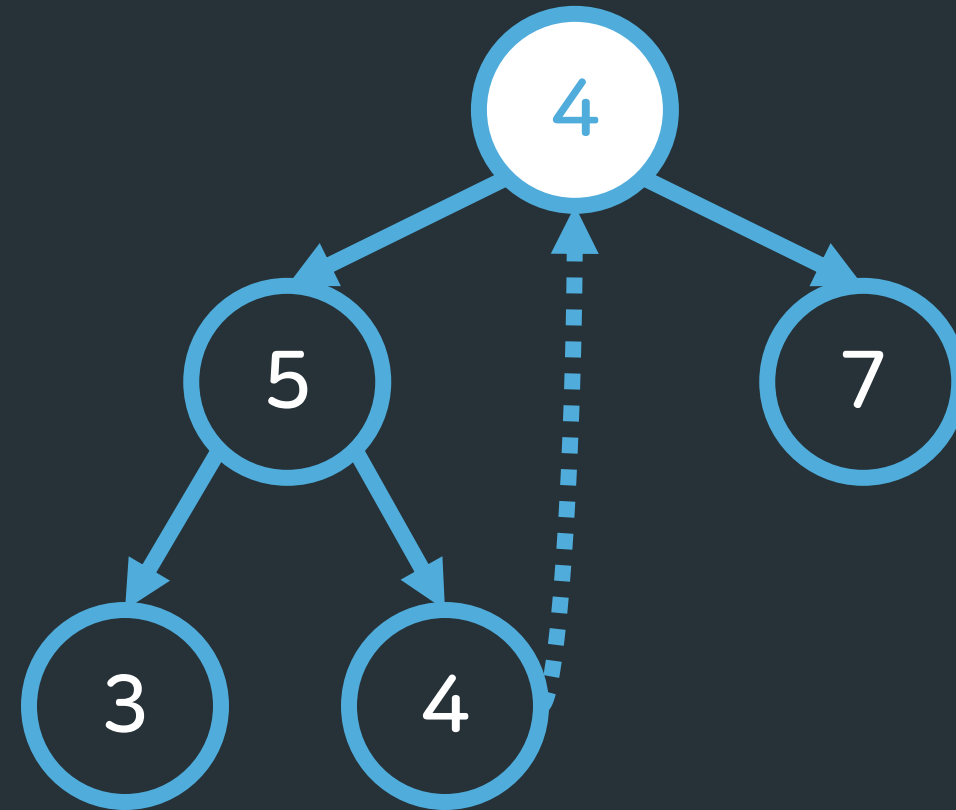


key = 10

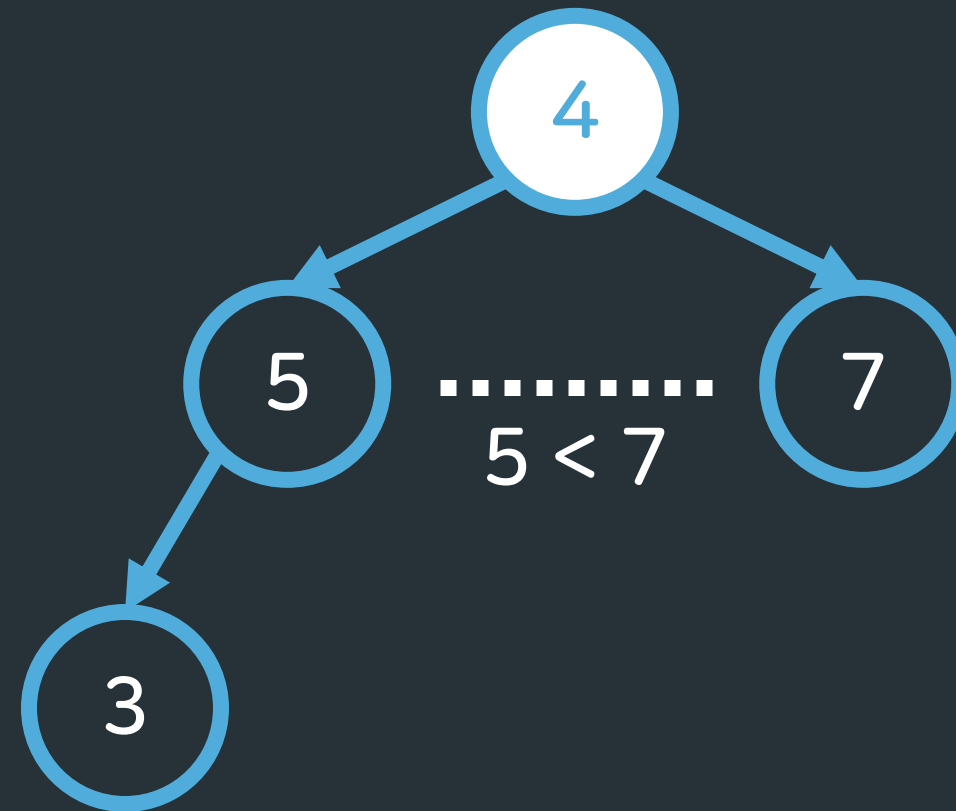
최대 힙에서 데이터 삭제



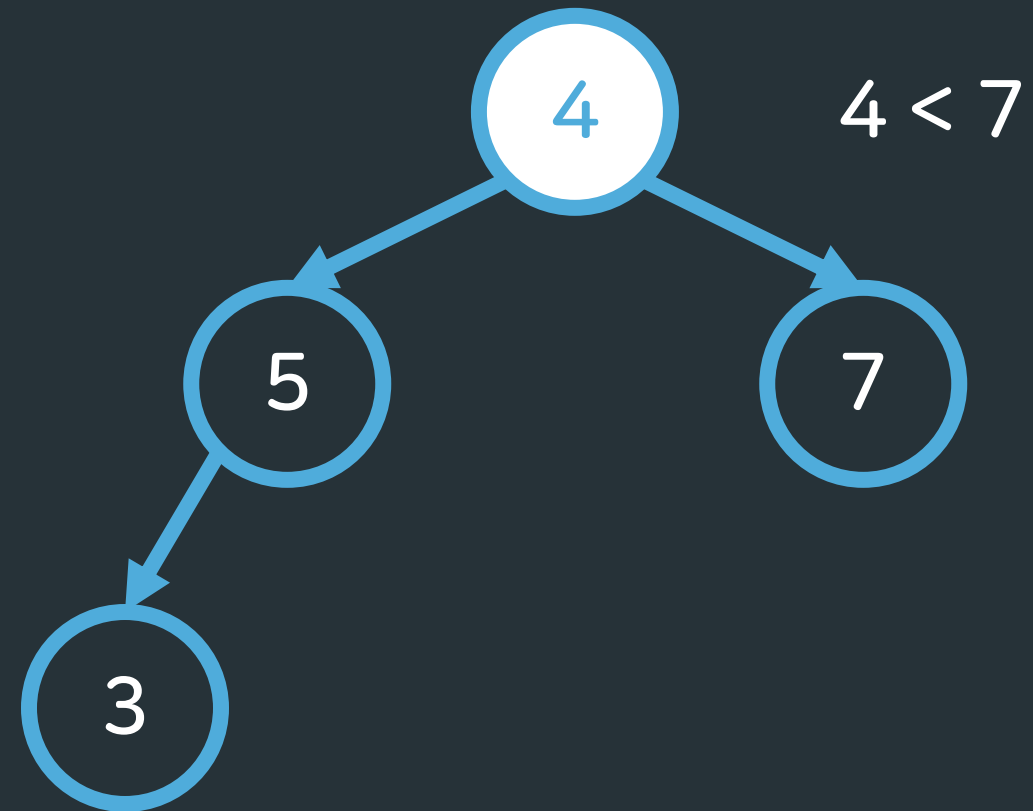
최대 힙에서 데이터 삭제



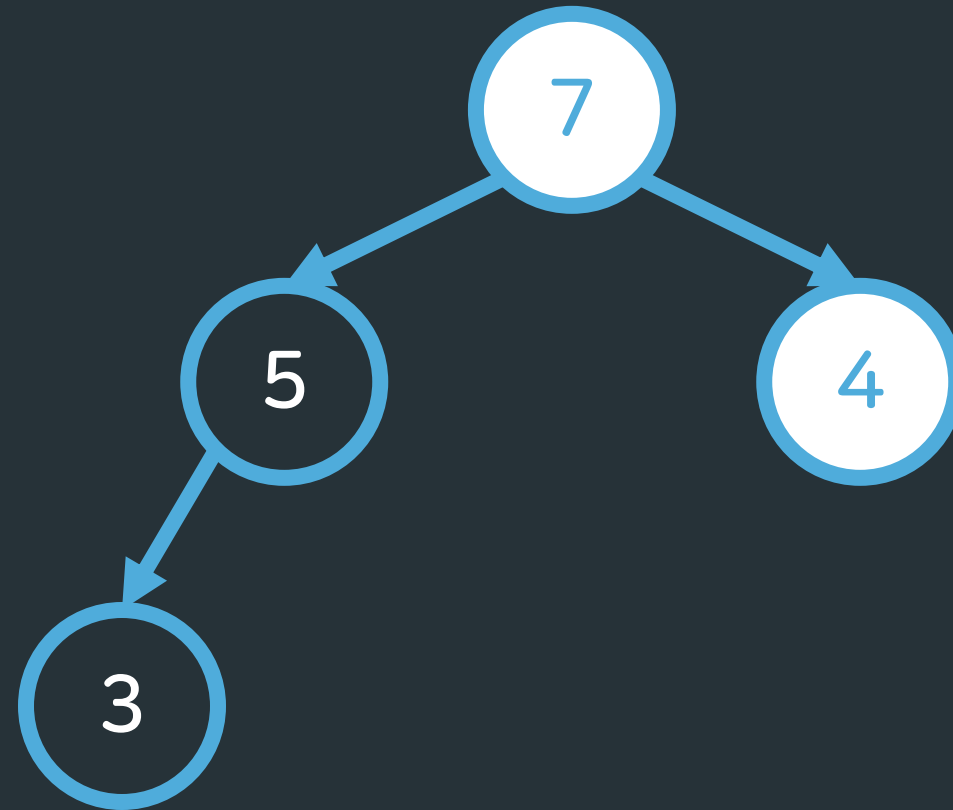
최대 힙에서 데이터 삭제



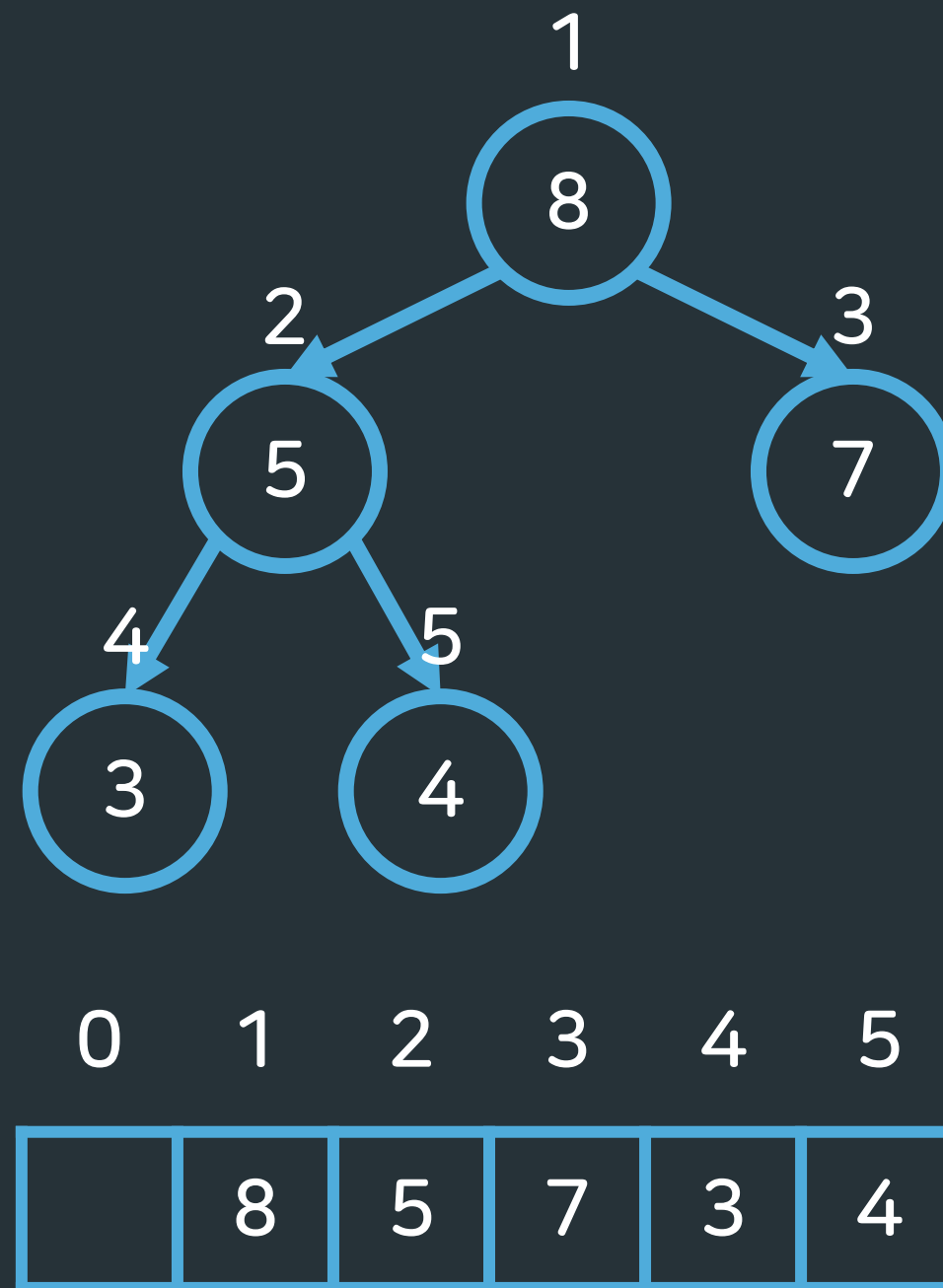
최대 힙에서 데이터 삭제



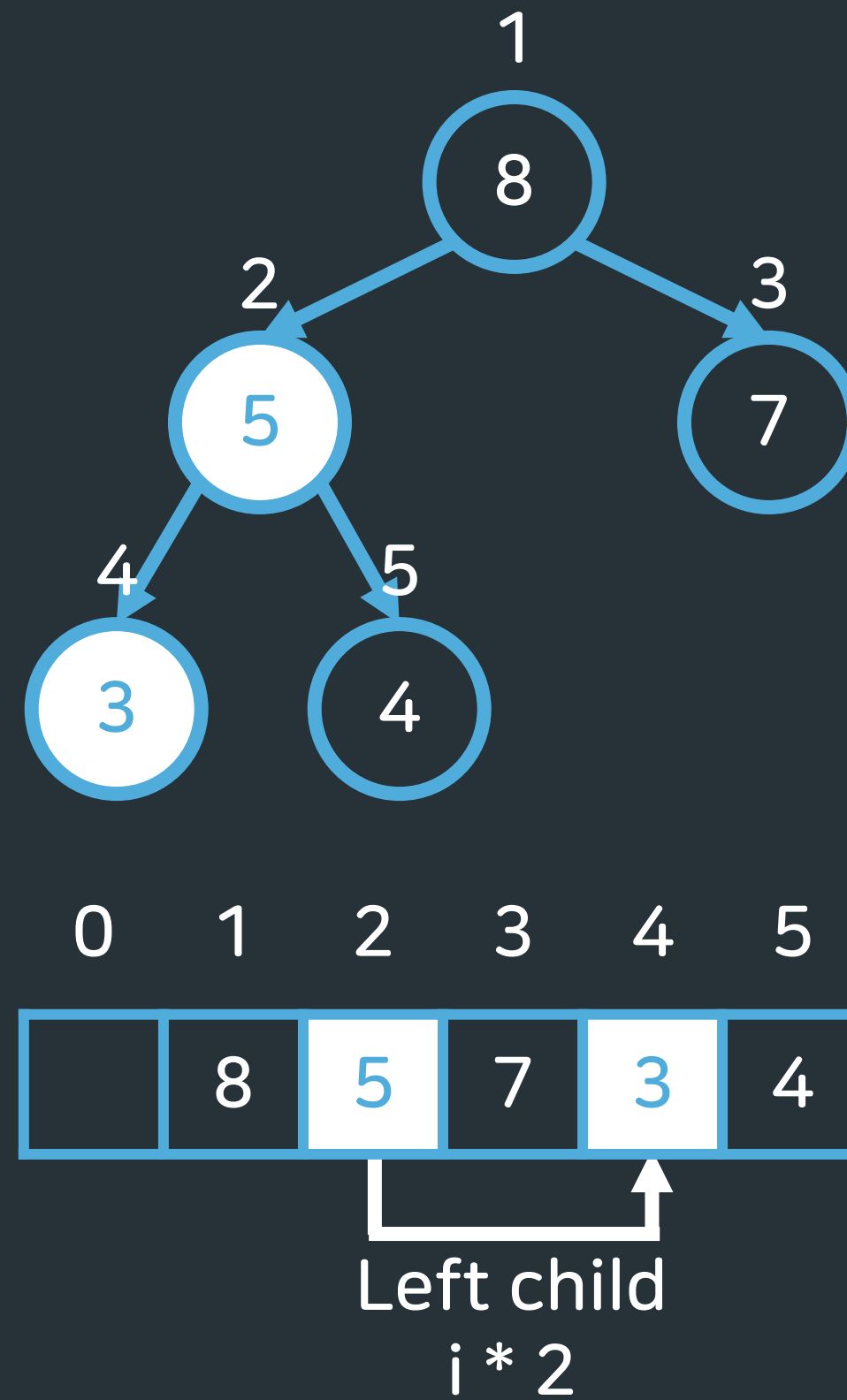
최대 힙에서 데이터 삭제



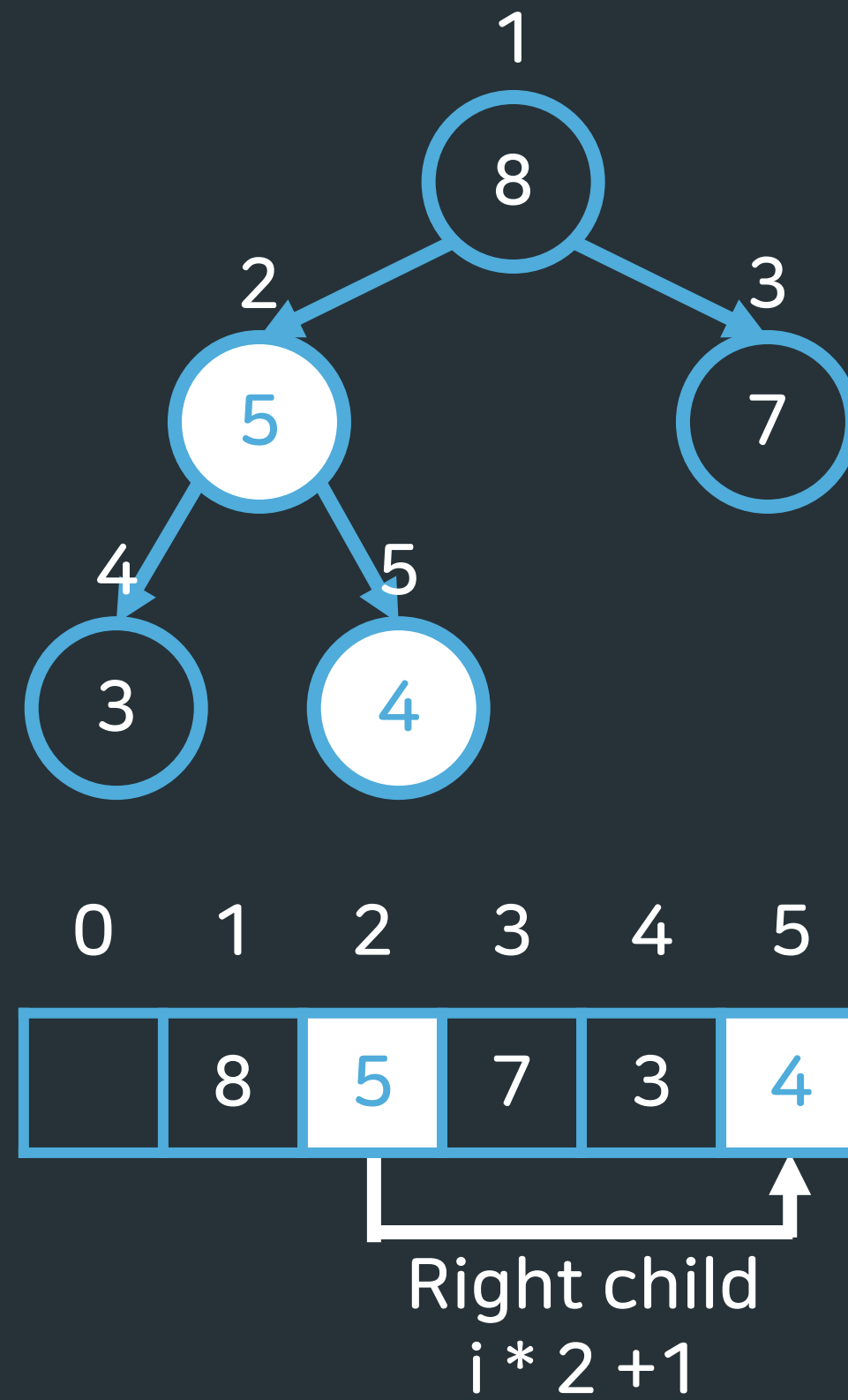
배열로 힙 구현하기



배열로 힙 구현하기



배열로 힙 구현하기



/<> 11279번 : 최대 힙 - Silver 2

문제

- 다음의 명령을 처리하는 최대 힙 프로그램 만들기
 1. 정수 x 가 주어진다.
 2. x 가 자연수라면 최대 힙에 x 추가
 3. x 가 0이라면 최대 힙에서 가장 큰 값을 출력하고 제거. 최대 힙이 비었다면 0 출력

제한 사항


- 명령의 수 N 의 범위는 $1 \leq N \leq 100,000$
- 명령과 함께 주어지는 정수 x 의 범위는 $0 \leq x \leq 2^{31}$

예제 입력

```
13
0
1
2
0
0
3
2
1
0
0
0
0
```

예제 출력

```
0
2
1
3
2
1
0
0
```



Search:

[Reference](#)
[<queue>](#)
[priority_queue](#)

[register](#)
[log in](#)

C++

[Information](#)
[Tutorials](#)
[Reference](#)
[Articles](#)
[Forum](#)

Reference

C library:

Containers:

<array>

<deque>

<forward_list>

<list>

<map>

<queue>

<set>

<stack>

<unordered_map>

<unordered_set>

<vector>

Input/Output:

Multi-threading:

Other:

<queue>

priority_queue

queue

priority_queue

priority_queue::priority_queue

member functions:

priority_queue::emplace

priority_queue::empty

priority_queue::pop

You were redirected to [cplusplus.com/priority_queue](#) || See search results for: "**priority_queue**"

class template

std::priority_queue

<queue>

```
template <class T, class Container = vector<T>,
          class Compare = less<typename Container::value_type> > class priority_queue;
```

Priority queue

Priority queues are a type of container adaptors, specifically designed such that its first element is always the greatest of the elements it contains, according to some *strict weak ordering* criterion.

This context is similar to a *heap*, where elements can be inserted at any moment, and only the *max heap* element can be retrieved (the one at the top in the *priority queue*).

Priority queues are implemented as *container adaptors*, which are classes that use an encapsulated object of a specific container class as its *underlying container*, providing a specific set of member functions to access its elements. Elements are *popped* from the "back" of the specific container, which is known as the *top* of the priority queue.

The underlying container may be any of the standard container class templates or some other specifically designed container class. The container shall be accessible through *random access iterators* and support the following operations:

- empty()
- size()
- front()
- push_back()
- pop_back()

The standard container classes `vector` and `deque` fulfill these requirements. By default, if no container class is specified for a particular `priority_queue` class instantiation, the standard container `vector` is used.

Support of *random access iterators* is required to keep a heap structure internally at all times. This is done automatically by the container adaptor by automatically calling the algorithm functions `make_heap`, `push_heap` and `pop_heap` when needed.

heapq — Heap queue algorithm

Source code: [Lib/heapq.py](#)

This module provides an implementation of the heap queue algorithm, also known as the priority queue algorithm.

Heaps are binary trees for which every parent node has a value less than or equal to any of its children. This implementation uses arrays for which `heap[k] <= heap[2*k+1]` and `heap[k] <= heap[2*k+2]` for all *k*, counting elements from zero. For the sake of comparison, non-existing elements are considered to be infinite. The interesting property of a heap is that its smallest element is always the root, `heap[0]`.

The API below differs from textbook heap algorithms in two aspects: (a) We use zero-based indexing. This makes the relationship between the index for a node and the indexes for its children slightly less obvious, but is more suitable since Python uses zero-based indexing. (b) Our pop method returns the smallest item, not the largest (called a “min heap” in textbooks; a “max heap” is more common in texts because of its suitability for in-place sorting).

These two make it possible to view the heap as a regular Python list without surprises: `heap[0]` is the smallest item, and `heap.sort()` maintains the heap invariant!

To create a heap, use a list initialized to `[]`, or you can transform a populated list into a heap via function `heapify()`.

heapq

- Python의 heapq 모듈은 우선순위 큐의 알고리즘을 제공 (자료구조 X)
- Python에서 우선순위 큐는 리스트를 사용
- append()와 pop() 대신 heapq에서 제공하는 함수를 이용해 삽입, 삭제

C++과의 차이점

- 자료구조가 아닌, 함수 제공
- 리스트를 이용하기 때문에, zero-based index
 - > top을 조회하기 위해서는 0번 인덱스 참조
- heapq의 함수들은 기본적으로 min heap 알고리즘을 제공
- max heap을 사용하기 위해서는 값을 변경해서 사용

```
import heapq as hq # heapq 모듈을 hq라는 이름으로 import

heap_list = [] # 힙으로 사용할 리스트 생성

hq.heappush(heap_list, new_value) # heap_list에 heap구조를 유지하며 x 삽입

popped_value = hq.heappop(heap_list) # heap_list를 heap구조로 유지하며 top삭제 후 반환

top = heap_list[0] # top은 0번 index에

popped_value = hq.heappushpop(heap_list, new_value) # 새 값을 push하고 pop한 값을 리턴

hq.heapify(existing_list) # 기존의 리스트를 heap 구조로 변환
```


/<> 11286번 : 절댓값 힙 - Silver 1

문제

- 절댓값 힙은 다음 두 가지 연산을 지원
 1. 배열에 정수 $x (x \neq 0)$ 를 삽입
 2. 배열에서 절댓값이 가장 작은 값을 출력하고, 그 값을 배열에서 제거. 절댓값이 가장 작은 값이 여러 개인 경우, 가장 작은 수를 출력하고 그 값을 배열에서 제거.

제한 사항

- 연산의 개수 N 의 범위 $1 \leq N \leq 100,000$
- 입력되는 정수 x 의 범위 $-2^{31} < x < 2^{31}$
- 시간 제한 1초 (추가 시간 없음)

예제

예제 입력

18
1
-1
0
0
0
1
1
-1
-1
2
-2
0
0
0
0
0
0
0

예제 출력

-1
1
0
-1
-1
1
1
-2
2
0

우선 순위를 변경해야 하는데...

C++

1. 정렬의 비교함수... 기억하시나요?
2. 왜 `priority_queue`는 기본이 `Max heap`일까요?

```
class template  
std::priority_queue  
template <class T, class Container = vector<T>,  
         class Compare = less<typename Container::value_type> > class priority_queue;
```

Python3

1. 정렬할 때 정렬 기준을 어떻게 바꿨었죠?
2. 그런데, 인자에 `key`가 없는데...?

```
heapq.heappush(heap, item)
```

Push the value *item* onto the *heap*, maintaining the heap invariant.

예시 - less<>

```
C++98 C++11 ?
1 template <class T> struct less {
2     bool operator()(const T& x, const T& y) const {return x<y;}
3     typedef T first_argument_type;
4     typedef T second_argument_type;
5     typedef bool result_type;
6 };
```

- 정렬 → 오름차순
- 우선순위 큐 → 최대 힙

직접 작성한 비교 구조체

```
struct cmp {
    bool operator()(const 자료형 &x1, const 자료형 &x2) {
        ...
    }
};
```

정렬의 비교함수와는 반대!
두번째로 오는 인자(x2)가 우선 순위라고 생각하면 쉽다!

클래스 직접 작성

```
class data:
    def __init__(self, value):    ← 생성자
        self.value = value

    def __lt__(self, nxt):
        return self.value < nxt.value    ← lessthan: 비교 연산자
```

데이터를 튜플로 입력

```
heap = []
hq.heappush(heap, ((key1, key2, ...), data))    ← (key, value) 형태
```

정리

- 우선순위 큐는 힙으로 구현하고, 시간 복잡도가 $O(\log n)$ 인 자료구조
- 효율성을 보는 문제에 사용되는 경우가 많음
- 그리디, 최단 경로 알고리즘 풀이에 활용되기도 함
- comp 정의할 때는 헛갈리지 말기! priority queue는 comp가 true를 반환해야 swap됨! (sort와 반대)
- 정렬은 comp 정의 시 첫 번째 인자 입장으로, priority_queue는 두 번째 인자 입장으로 생각하면 쉬움
- 무한 루프 (pop을 하지 않음), 런타임 에러 (empty 체크 안하고 조회 or 삭제 시도) 조심!!

- default는 C++의 경우 Max heap, Python은 Min heap
- Python의 경우, index는 0부터 사용함을 주의!

필수

- /<> 7662번 : 이중 우선순위 큐 - Gold 5
- /<> 5397번 : 키로거 - Silver 3

3문제 이상 선택

- /<> 2075번 : N번째 큰 수 - Gold 5
- /<> 13975번 : 파일 합치기 3 - Gold 4
- /<> 14235번 : 크리스마스 선물 - Silver 3
- /<> 12018번 : Yonsei TOTO - Silver 3
- /<> 19640번 : 화장실의 규칙 - Gold 5

코드리뷰 O 마감 ~ 4월 7일 목요일 낮 12시

코드리뷰 X 마감 ~ 4월 7일 목요일 밤 12시 (7에서 8일로 넘어가는 자정)

추가제출 마감 ~ 4월 8일 금요일 밤 12시 (8일에서 9일로 넘어가는 자정)