# Priority Queue
# 우선순위 큐

# Priority Queue 란?



## 1 Priority Queue ( 우선순위 큐 )

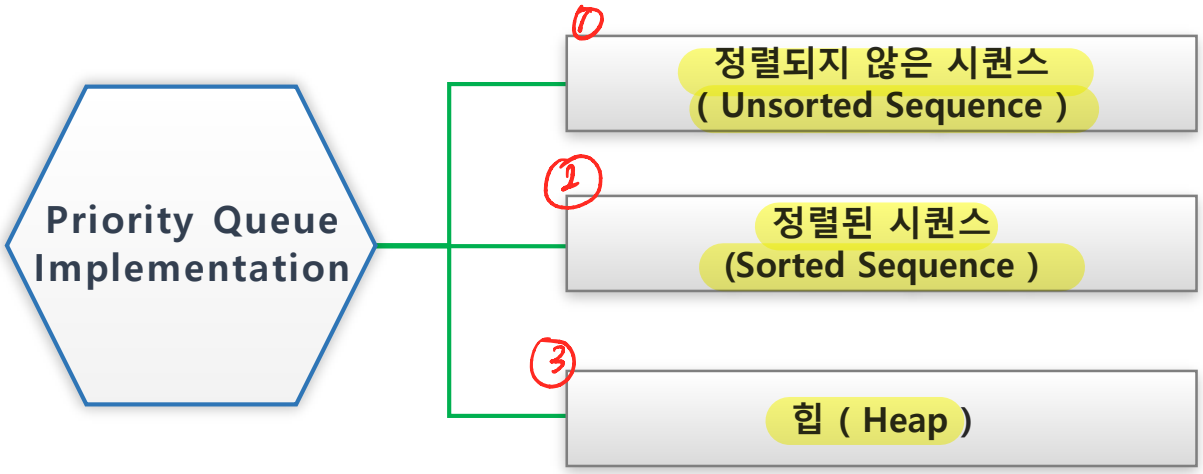
- entry 자료를 저장하는 자료구조 중 하나
- 기존 Queue의 FIFO(First In First Out)를 사용하지 않는다 → 삽입된 시간 기준 X!
- 들어간 순서에 상관 없이, 우선순위(Priority)에 따라 결과가 달라진다
- Priority(Key)는 사용자가 결정한다

— Ex) Min Priority Queue : Key 값이 작을수록, 우선순위 ↑

Max Priority Queue : Key 값이 클수록, 우선순위 ↑

# Priority Queue 구현 방법



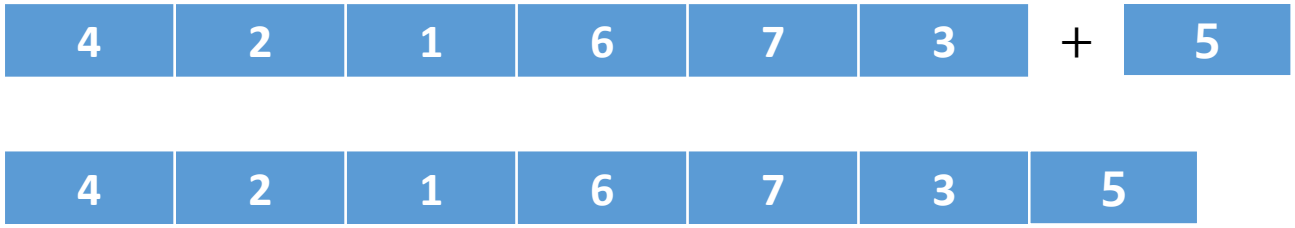



- 핵심 ADT : insert, removeMin(removeMax)
  - Insert : 원소 추가
  - removeMin(removeMax) : 가장 우선순위가 높은 원소 제거

## 2 Unsorted Array 기반 Priority Queue

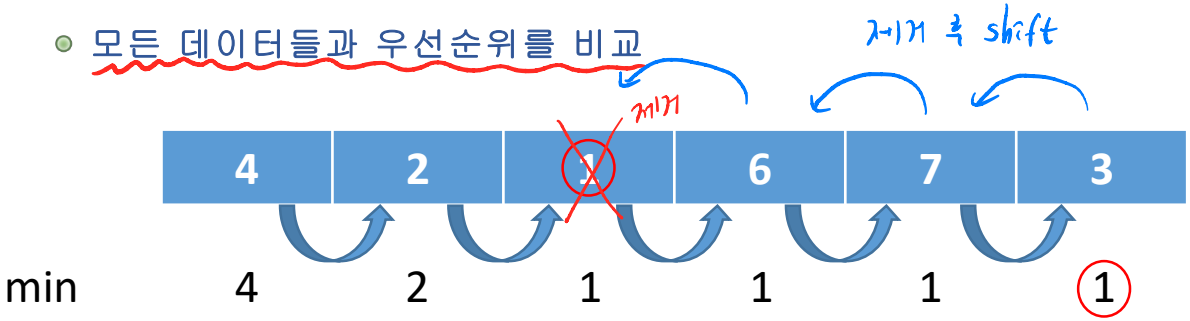
### Insert : $O(1)$

- 기존의 Queue 처럼, 하나의 원소를 배열에 추가




### removeMin : $O(n)$

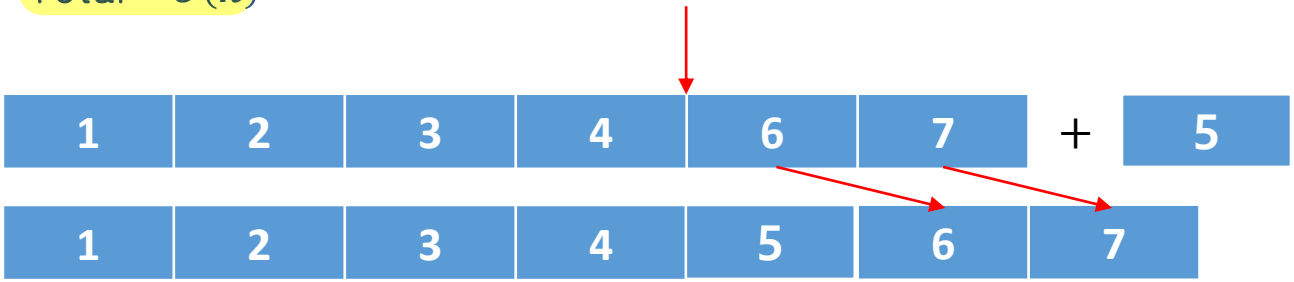
- 모든 데이터들과 우선순위를 비교



## 3 Sorted Array 기반 Priority Queue

### Insert : $O(n)$

- 모든 데이터들과 우선순위를 비교하고, 삽입될 위치를 찾는다 : $O(log\ n)$
- 삽입될 위치를 확보하기 위해 배열의 값을 옮긴다 : $O(n)$
- Total : $O(n)$




### removeMin : $O(1)$

- 우선순위로 정렬 되어있기 때문에, 바로 꺼내는 것이 가능

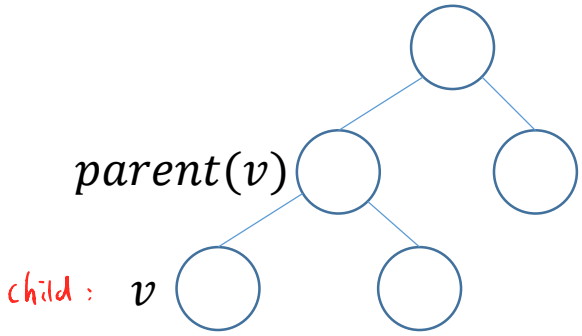
# 힙 ( Heap ) 기반 Priority Queue



2 Heap : 아래의 2가지 Property를 만족하는 Binary Tree

- Complete Binary Tree : 항상 왼쪽부터 순서대로 채워져야 한다

구조적




- Heap-Order : 트리의 내부노드(Internal Node)는 우선순위를 만족해야한다

순서적

child parent

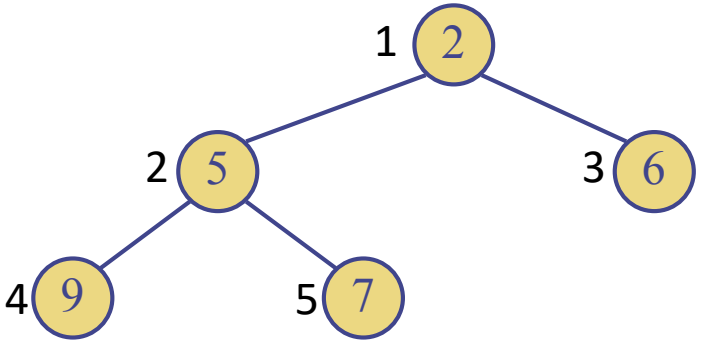
ex) Min-Priority Queue ⇒ $key(v) \geq key(parent(v))$

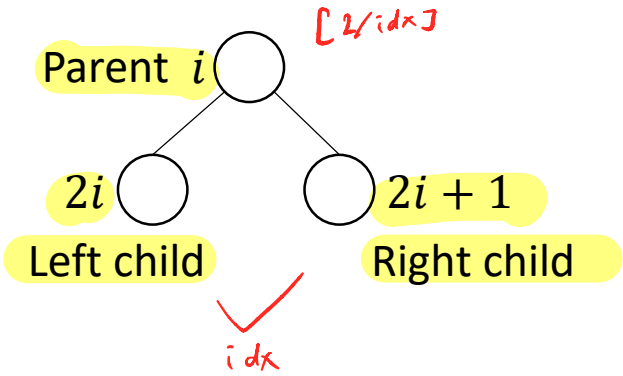
Max-Priority Queue ⇒ $key(v) \leq key(parent(v))$

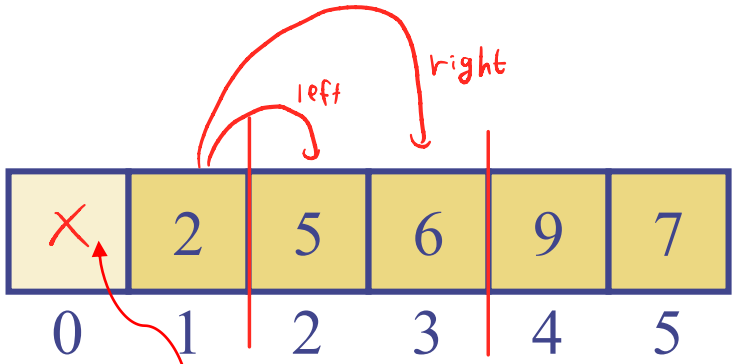
∴ 우선순위가 가장 높은 Entry는 Root에 존재!

# Heap 구현

- Heap은 Complete Binaray Tree 이기 때문에 Array를 사용하여서 쉽게 구현 가능




규칙성을 갖기 위해 0번 index를 비워 둔다! (dummy variable)

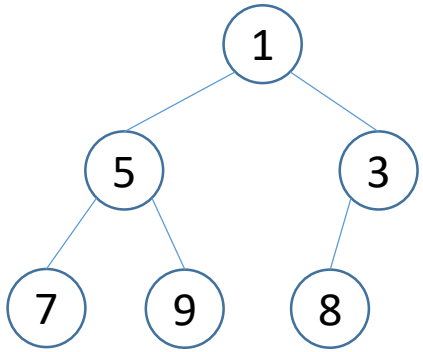
# 힙 ( Heap ) 기반 Priority Queue



힙에서의 삽입

- Insert 2 to Min-Heap

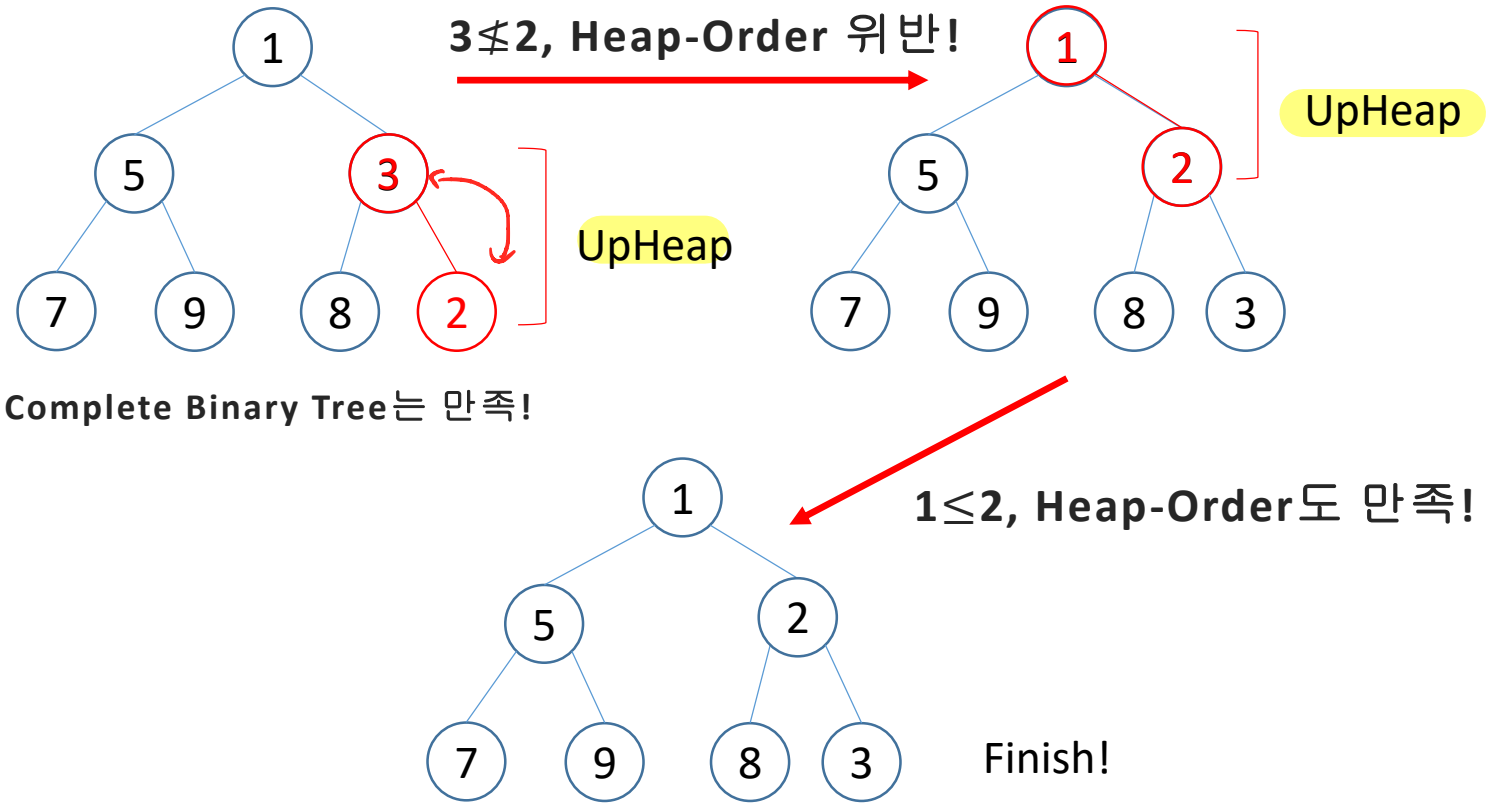
유념할 것! Heap을 유지해야 한다.
조건 1) Complete Binary Tree
조건 2) Heap-Order



# 힙 ( Heap ) 기반 Priority Queue

- Insert 2 to Min-Heap

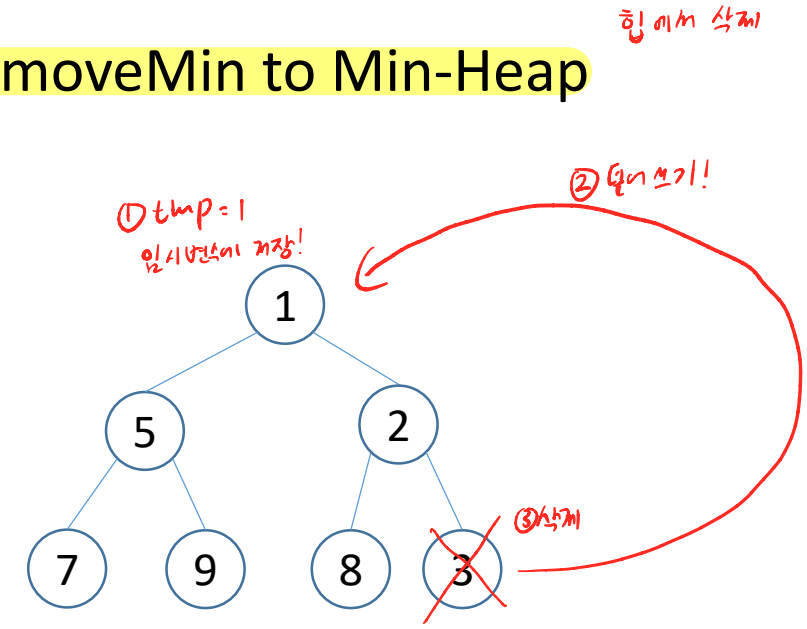
유념할 것! Heap을 유지해야 한다.
조건 1) Complete Binary Tree
조건 2) Heap-Order



# 힙 ( Heap ) 기반 Priority Queue



- removeMin to Min-Heap



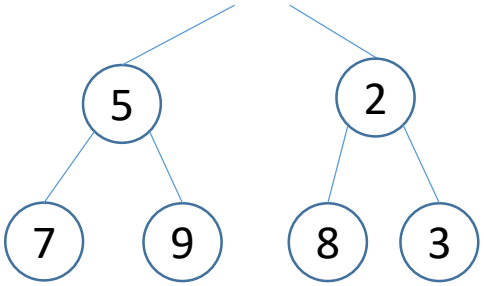

유념할 것! Heap을 유지해야 한다.
조건 1) Complete Binary Tree
조건 2) Heap-Order

# 힙 ( Heap ) 기반 Priority Queue



- removeMin to Min-Heap

유념할 것! Heap을 유지해야 한다.
조건 1) Complete Binary Tree
조건 2) Heap-Order



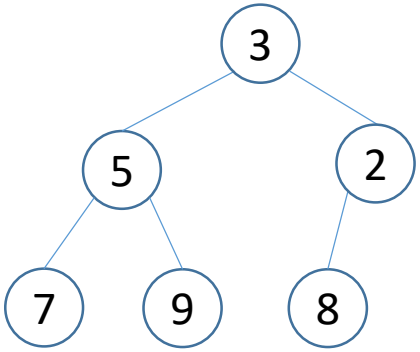

조건 1 위배!

# 힙 ( Heap ) 기반 Priority Queue



- removeMin to Min-Heap

유념할 것! Heap을 유지해야 한다.
조건 1) Complete Binary Tree
조건 2) Heap-Order




Down heap 수행!

**Complete Binary Tree는 만족!**

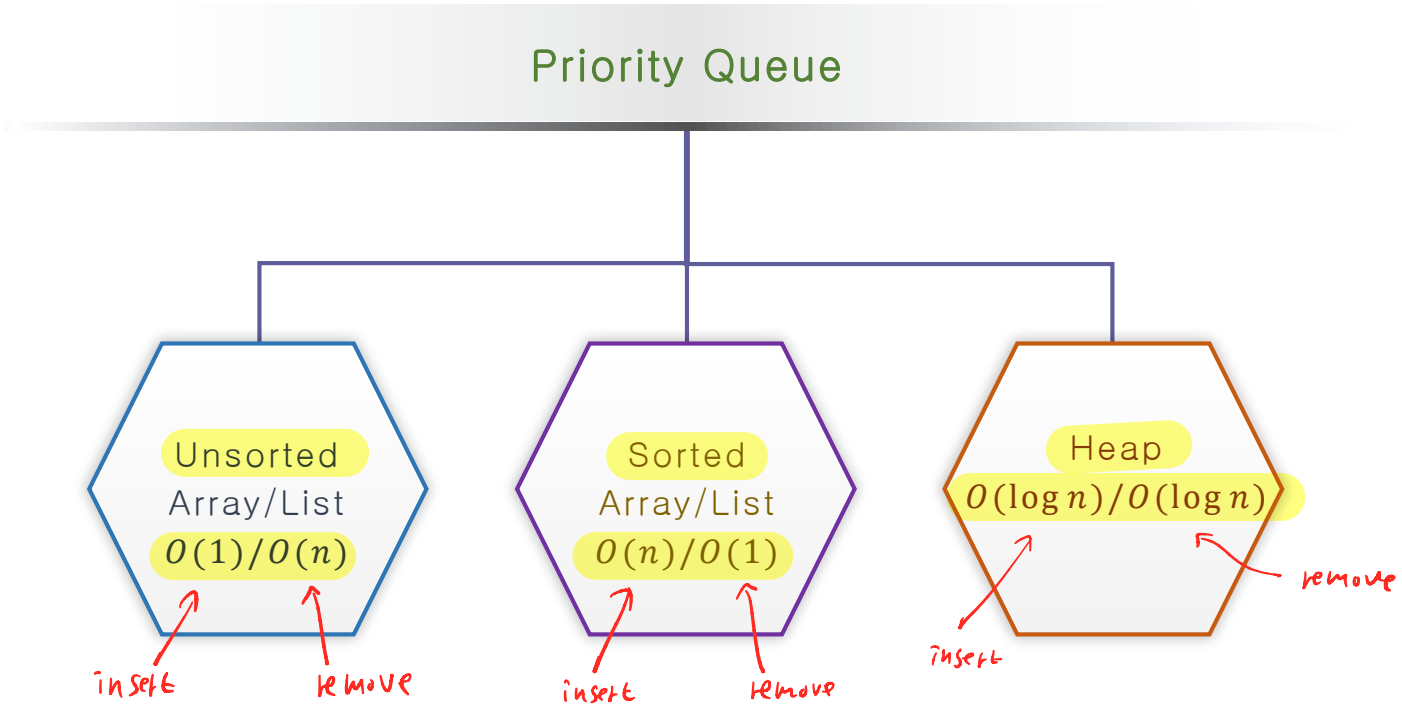
# 힙 ( Heap ) 기반 Priority Queue



- removeMin to Min-Heap

유념할 것! Heap을 유지해야 한다.
조건 1) Complete Binary Tree
조건 2) Heap-Order

- Insert / removeMin의 시간복잡도



# 힙 생성 (Heap Construction)



Button-up heap construction

- 힙 생성(Construction) 방법
  - 데이터가 입력될 때 마다, Property 확인(on line방식)
    - $O(n) * O(\log n) \Rightarrow O(n \log n)$의 시간복잡도
  - 일단 배열에 전부 입력 후, Property를 확인하는 방법(off line방식)
    - $O(n)$의 시간복잡도

https://www.growingwiththeweb.com/data-structures/binary-heap/build-heap-proof/

# 힙 ( Heap ) 기반 Priority Queue



```cpp
#include <iostream>
#include <string>
#include <vector>

using namespace std;
enum direction { MIN = 1, MAX = -1 };          // min heap or max heap을 결정할 값. cf) a > b와 -a < -b는 결과가 같다.

class Heap {
private:
    vector<int> v;
    int heap_size;
    int root_index;
    int direction;                             // min, max에 따라 Heap의 종류가 결정됨.
public:
    Heap(int direction) {
        v.push_back(-1);                       // dummy variable
        this->heap_size = 0;
        this->root_index = 1;
        this->direction = direction;
    }
    void swap(int idx1, int idx2) {            // 두 index를 교환 하는 함수
    }
    void upHeap(int idx) {                     // for insertion
    }
    void downHeap(int idx) {                   // for removal
    }
    void insert(int e) {
    }
    int pop() {
    }
    int top() {
    }
    int size() {
    }
    bool isEmpty() {
    }
    void print() {                             // heap의 원소를 위에서 아래로, 왼쪽에서 오른쪽으로 출력할 함수
                                               // 단, 비어있는 경우 -1을 출력하도록 함.
    }
};
```

# 힙 ( Heap ) 기반 Priority Queue



```
int main( ) {
    Heap PQ(MIN);
    PQ.insert(1);
    cout << "Heap construction : "; PQ.print( );
    PQ.insert(10);
    cout << "Heap construction : "; PQ.print( );
    PQ.insert(3);
    cout << "Heap construction : "; PQ.print( );
    PQ.insert(7);
    cout << "Heap construction : "; PQ.print( );
    PQ.insert(6);
    cout << "Heap construction : "; PQ.print( );
    for (int i = 0; i < 5; i++) {
        PQ.pop( );
        cout << "Heap construction : "; PQ.print( );
    }
    system("pause");
    return 0;
}
```

```
Heap construction : 1
Heap construction : 1 10
Heap construction : 1 10 3
Heap construction : 1 7 3 10
Heap construction : 1 6 3 10 7
Heap construction : 3 6 7 10
Heap construction : 6 10 7
Heap construction : 7 10
Heap construction : 10
Heap construction : -1
계속하려면 아무 키나 누르십시오 . . .
```


Thank you