뉴·진·스

힙

김준성 2023-11-01(화)

목차

1.힙

2.힙 개선

3.Live Coding

히

우선순위 큐란?

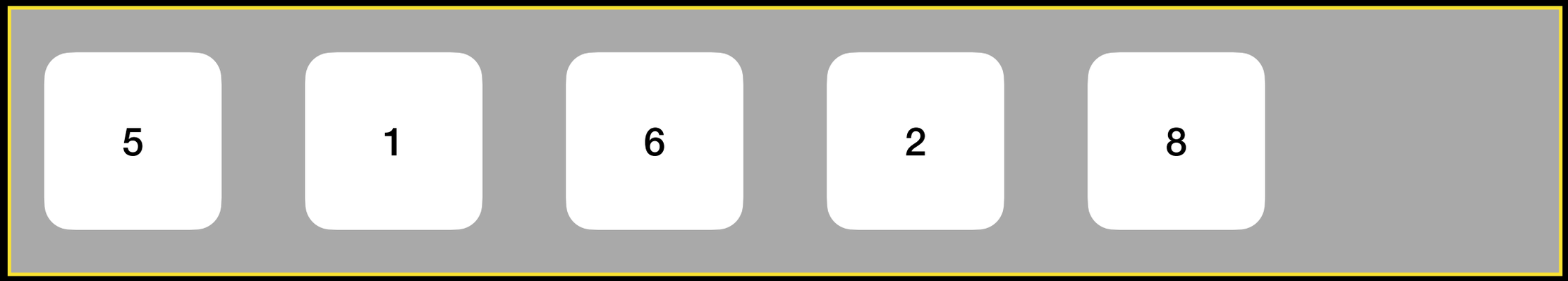
- 스택은 배열의 제일 뒤에 있는(가장 늦게 들어온 데이터)를 뽑아내고, 큐는 배열의 제일 앞에 있는(가장 먼저 들어온 데이터)를 뽑아내는 특성이 있는 자료구조이다.

- 그렇다면 우선순위 큐는???

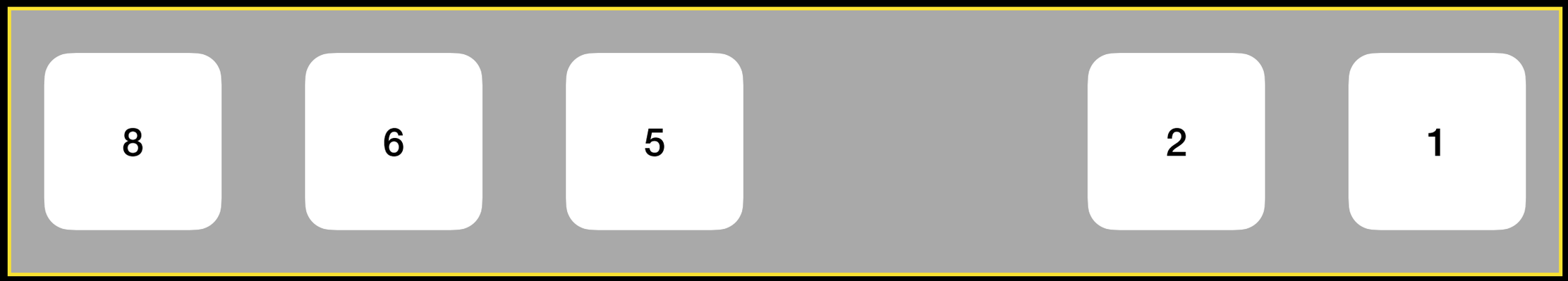
배열의 제일 앞에 있는(+ 가장 우선 순위가 높은) 데이터
를 뽑아내는 특성이 있는 자료구조

최대 값 우선순위 큐 삽입

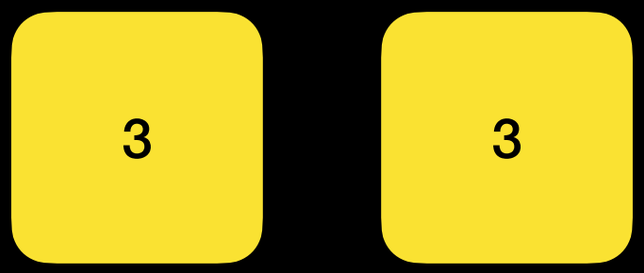
정렬X 배열 O(1)



정렬O 배열 O(n)

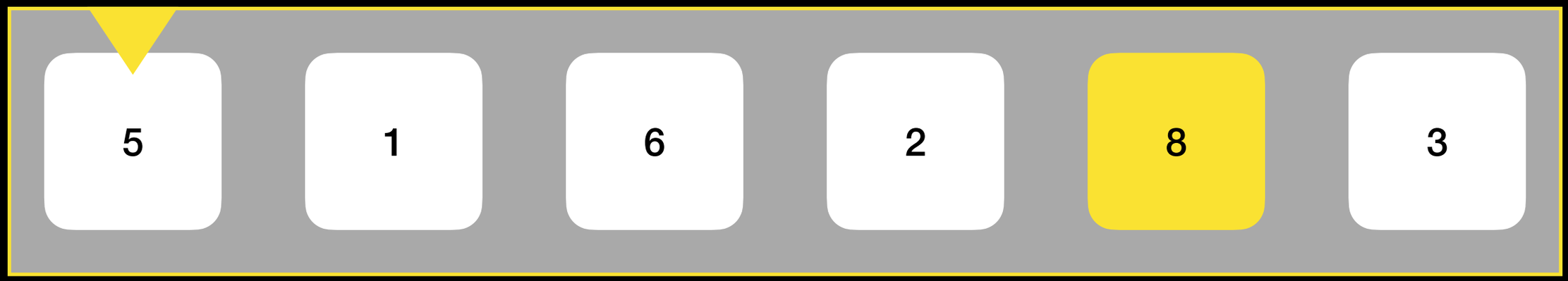


3을 삽입해보자

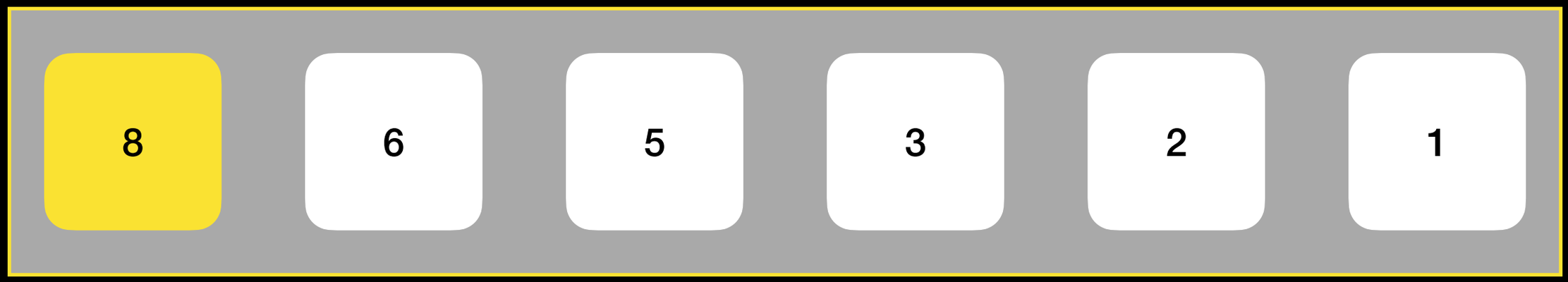


최대 값 우선순위 큐 삭제

정렬X 배열 O(n)



정렬O 배열 O(1)



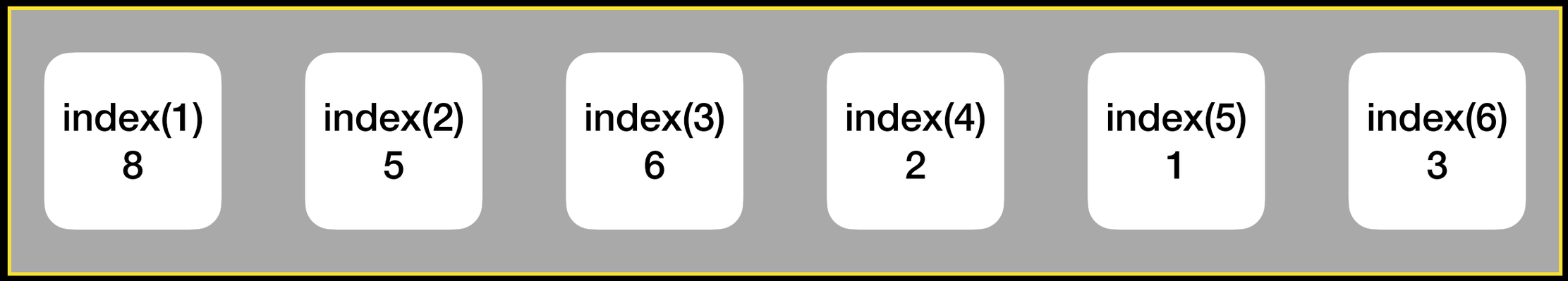
8을 삭제해보자

힙이란?

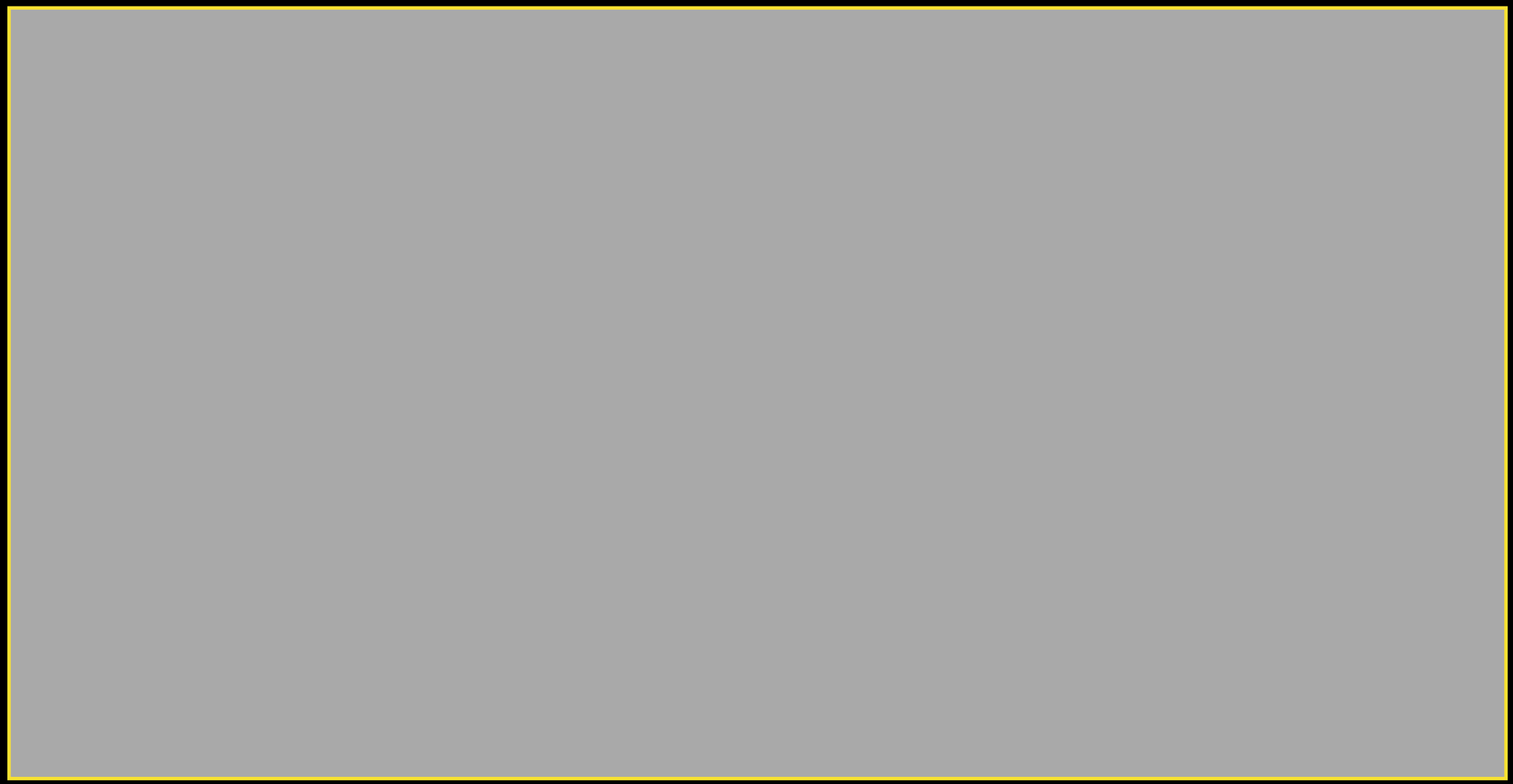
- 우선순위 큐를 구현하기 위한 방법 중 하나
- 사용자가 정의한 우선순위를 바탕으로 데이터를 빠르게 찾거나 뽑도록 만든 자료구조
- 넣을 때마다, 약간의 정렬을 하면서 트리에 값을 추가함.
  -> 상위노드(index: N)은 하위노드(N * 2, N * 2 + 1)보다 큰 값을 가지고 있다.
- 삽입, 삭제 시의 시간 복잡도: O(log n)

```
struct Heap {
    var array = [Int]()
    // ...
}
```

배열



완전 이진 트리



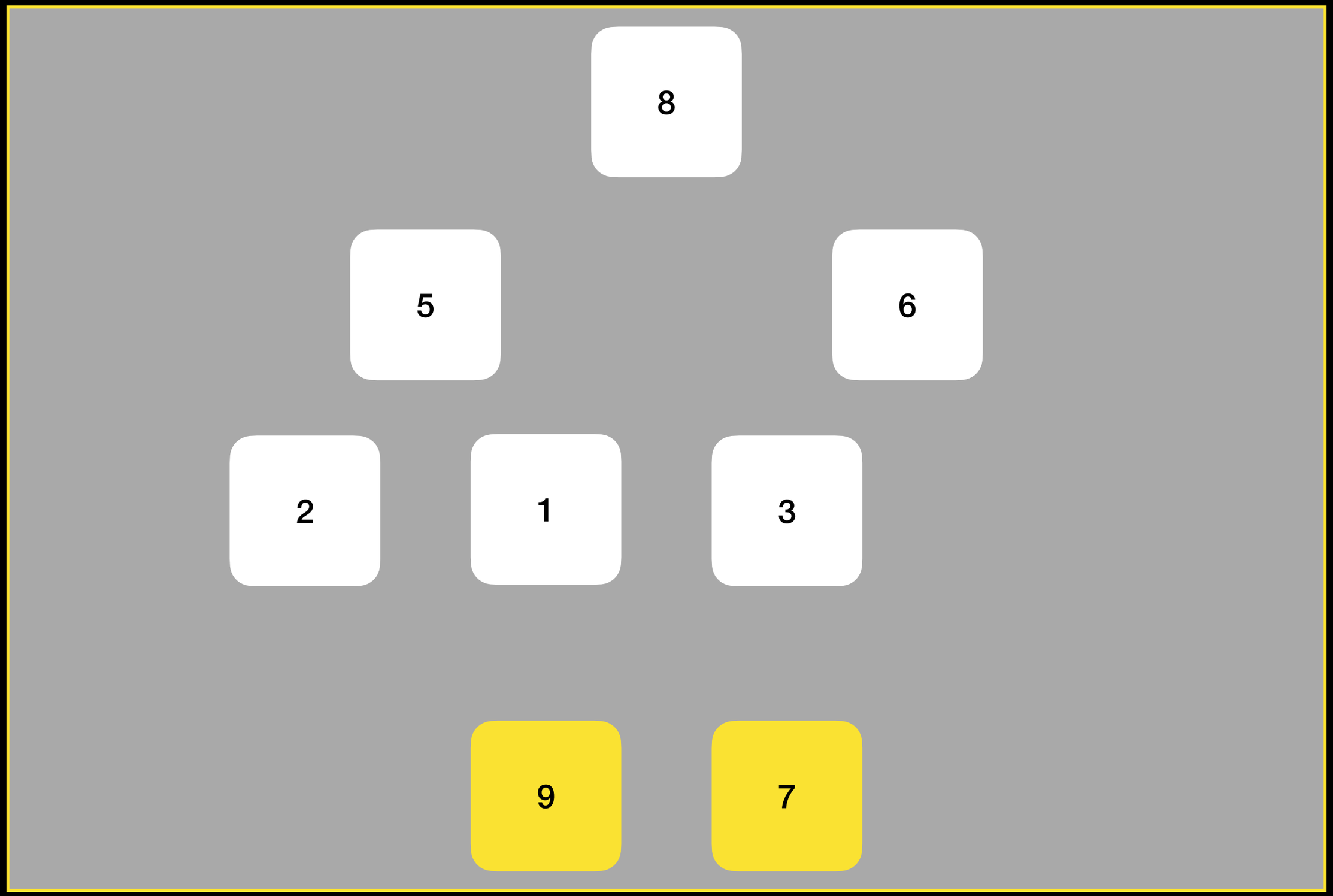
최대 값 힙 삽입

- 배열의 마지막에 값을 배치한 뒤, 부모 인덱스를 타고 올라가며 한계에 도달할 때까지 반복.

```
mutating func enheap(_ newElement: Int) {
    array.append(newElement)

    var childIndex = array.count - 1
    var parentIndex = (childIndex - 1) / 2
    while childIndex > 0 && array[parentIndex] > array[childIndex] {
        array.swapAt(childIndex, parentIndex)
        childIndex = parentIndex
        parentIndex = (childIndex - 1) / 2
    }
}
```

최대 값 힙 삽입



최대 값 힙 삭제

- 배열의 첫 값을 리턴하고, 배열의 가장 마지막 값을 0번째 인덱스에 배치

```
mutating func deheap() -> Int {
    if array.isEmpty {
        return 0
    }

    if array.count == 1 {
        return array.removeLast()
    }

    let value = array[0]
    array[0] = array.removeLast()
    moveDown(from: 0)
    return value
}
```

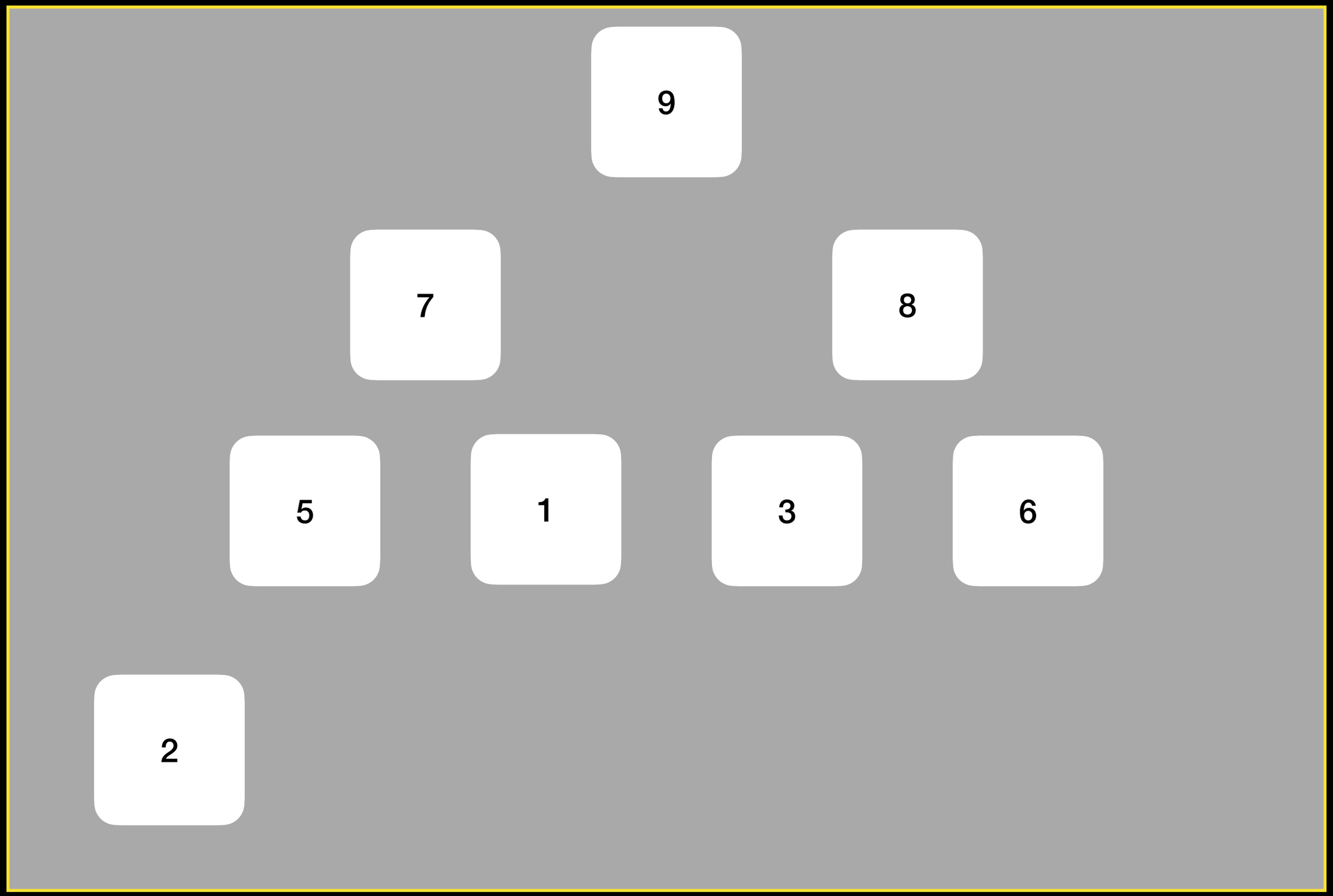
최대 값 힙 삭제

- 왼쪽·오른쪽 자식 인덱스를 타고 내려가며 한계에 도달할 때까지 반복.

```
mutating func moveDown(from index : Int) {
    let leftChildIndex = index * 2 + 1
    let rightChildIndex = leftChildIndex + 1
    var target = index

    if leftChildIndex < array.endIndex && array[target] > array[leftChildIndex] {
        target = leftChildIndex
    }
    if rightChildIndex < array.endIndex && array[target] > array[rightChildIndex] {
        target = rightChildIndex
    }
    if target != index {
        array.swapAt(index, target)
        moveDown(from: target)
    }
}
```

최대 값 힙 삭제



한 개선

클로저 개선

- 클로저를 통해 Heap의 비교연산자를 바꾸면 보다 쉽게 큐를 만들 수 있습니다.

```
struct Heap {
    private var array = [Int]()
    private let compare: (Int, Int) -> Bool

    init(array: [Int] = [Int](), compare: @escaping (Int, Int) -> Bool) {
        self.array = array
        self.compare = compare
    }
    // ...
}
```

```
var heap = Heap(compare: <)
```

뒷 페이지에서 계속...

클로저 개선

- 클로저를 통해 Heap의 비교연산자를 바꾸면 보다 쉽게 큐를 만들 수 있습니다.

```
mutating func enheap(_ newElement: Int) {
    array.append(newElement)

    var childIndex = array.count - 1
    var parentIndex = (childIndex - 1) / 2
    while childIndex > 0 && compare(array[parentIndex], array[childIndex]) {
        array.swapAt(childIndex, parentIndex)
        childIndex = parentIndex
        parentIndex = (childIndex - 1) / 2
    }
}
```

뒷 페이지에서 계속...

클로저 개선

- 클로저를 통해 Heap의 비교연산자를 바꾸면 보다 쉽게 큐를 만들 수 있습니다.

```
mutating func moveDown(from index : Int) {
    let leftChildIndex = index * 2 + 1
    let rightChildIndex = leftChildIndex + 1
    var target = index

    if leftChildIndex < array.count && compare(array[target], array[leftChildIndex]) {
        target = leftChildIndex
    }
    if rightChildIndex < array.count && compare(array[target], array[rightChildIndex]) {
        target = rightChildIndex
    }
    if target != index {
        array.swapAt(index, target)
        moveDown(from: target)
    }
}
```

뒷 페이지에서 계속...

제네릭 개선

- Heap에는 Int 뿐만이 아니라, String, Array 등 다양한 자료형이 반드시 들어갈 수 있어야 합니다. (왜냐하면 그래야 최단경로 문제를 풀 수 있음.)

```
struct Heap<T> {
    private var array = [T]()
    private let compare: (T, T) -> Bool

    init(array: [T] = [T](), compare: @escaping (T, T) -> Bool) {
        self.array = array
        self.compare = compare
    }
    // ...
}
```

```
var heap = Heap<Int>(compare: <)
```

뒷 페이지에서 계속...

제네릭 개선

- Heap에는 Int 뿐만이 아니라, String, Array 등 다양한 자료형이 반드시 들어갈 수 있어야 합니다. (왜냐하면 그래야 최단경로 문제를 풀 수 있음.)

```
mutating func enheap(_ newElement: T) {
    array.append(newElement)

    var childIndex = array.count - 1
    var parentIndex = (childIndex - 1) / 2
    while childIndex > 0 && compare(array[parentIndex], array[childIndex]) {
        array.swapAt(childIndex, parentIndex)
        childIndex = parentIndex
        parentIndex = (childIndex - 1) / 2
    }
}
```

뒷 페이지에서 계속...

제네릭 개선

- Heap에는 Int 뿐만이 아니라, String, Array 등 다양한 자료형이 반드시 들어갈 수 있어야 합니다. (왜냐하면 그래야 최단경로 문제를 풀 수 있음.)

```
mutating func deheap() -> T? {
    if array.isEmpty {
        return nil
    }

    if array.count == 1 {
        return array.removeLast()
    }

    let value = array[0]
    array[0] = array.removeLast()
    moveDown(from: 0)
    return value
}
```

뒷 페이지에서 계속...

재귀 개선

- 재귀로 발생하는 오버헤드로 인해 나중에 시간초과나는 경우가 많습니다.
  따라서 반복문으로 바꿔야 합니다.(모든 재귀는 반복문으로 치환이 가능함.)

```
mutating func improvedDeheap() -> T? {
    if array.isEmpty {
        return nil
    }

    if array.count == 1 {
        return array.removeLast()
    }

    let value = array[0]
    array[0] = array.removeLast()

    var index = 0
```

재귀 개선

```
    var index = 0
    while true {
        let leftChildIndex =  index * 2 + 1
        let rightChildIndex = leftChildIndex + 1

        var target = index
        if leftChildIndex < array.count && compare(array[target], array[leftChildIndex]) {
            target = leftChildIndex
        }

        if rightChildIndex < array.count && compare(array[target], array[rightChildIndex]) {
            target = rightChildIndex
        }

        if target == index {
            break
        }

        array.swapAt(target, index)
        index = target
    }

    return value
}
```

실눈 상태

11279 최대 힙

1927 최소 힙

끝