COSE213: Data Structure

Lecture 10 review

Minseok Jeon

2024 Fall

남은 수업 일정

- 11/29 (오늘) : 다원 탐색 트리 (Lecture 11)
- 12/06: 그래프 (Lecture 12)
- 12/13: 기말고사 (Final)
- 12/20: 기말고사 review

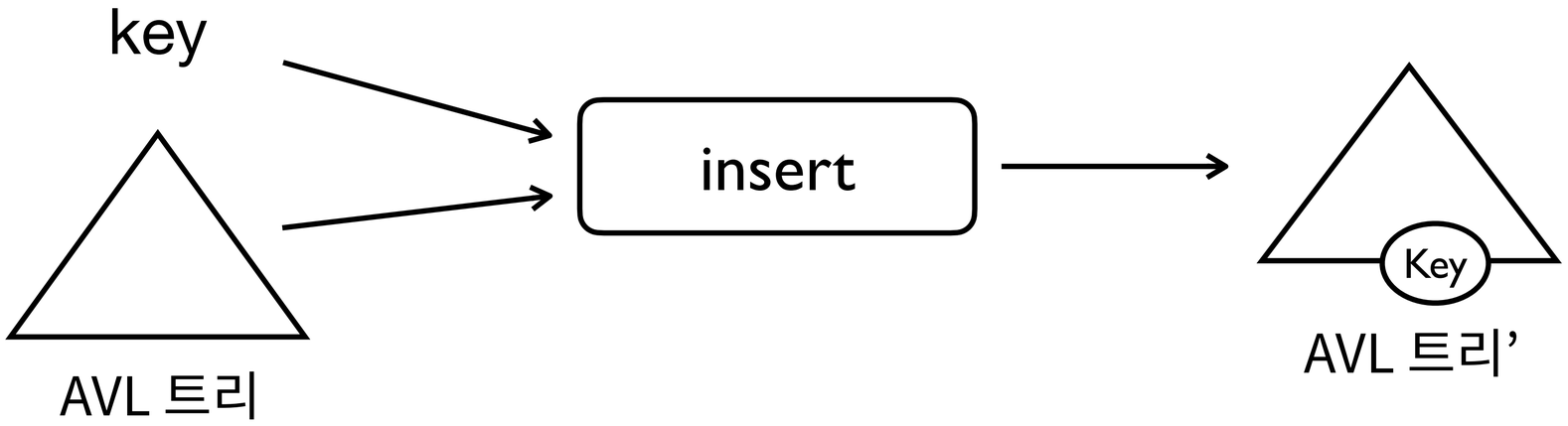
Typo

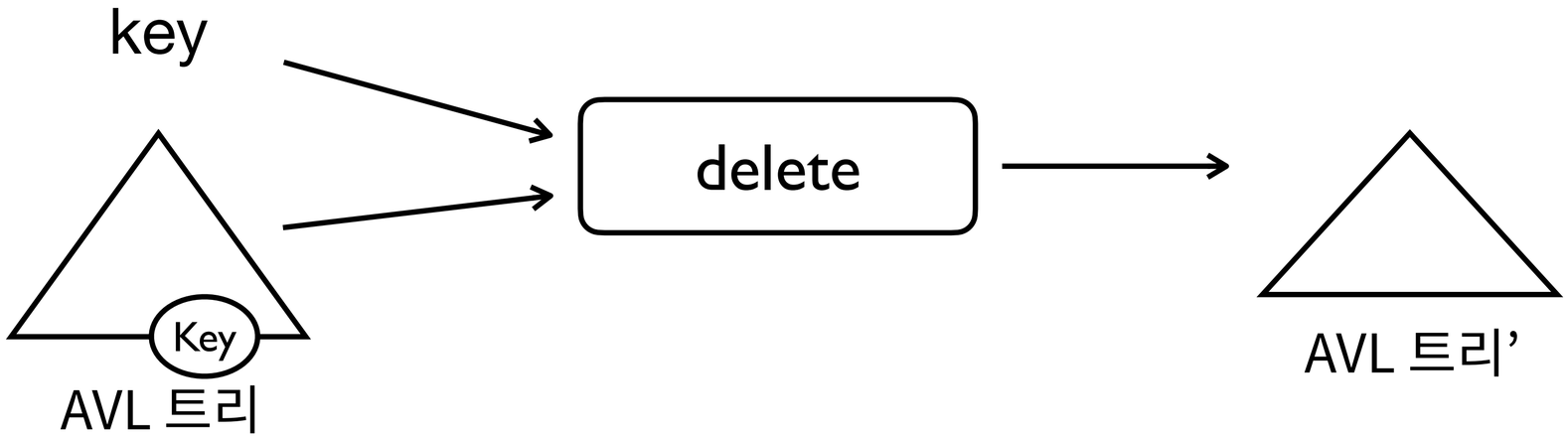
- Lecture 9 (AVL Tree) 슬라이드 중

```
procedure balancing(root)
  if (getBalance(root) > 1) and (getBalance(root->left) ≥ 0) then
    return rotateRight(root)
  elif (getBalance(root) > 1) and (getBalance(root->left) < 0) then
    root.left ← rotateLeft(root.left)
    return rotateRight(root)
  elif (getBalance(root) < -1) and (getBalance(root->right) ≤ 0) then
    return rotateLeft(root)
  elif (getBalance(root) < -1) and (getBalance(root->right) > 0) then
    root.right ← rotateRight(root.right)
    return rotateLeft(root)
  return root
```

Quiz

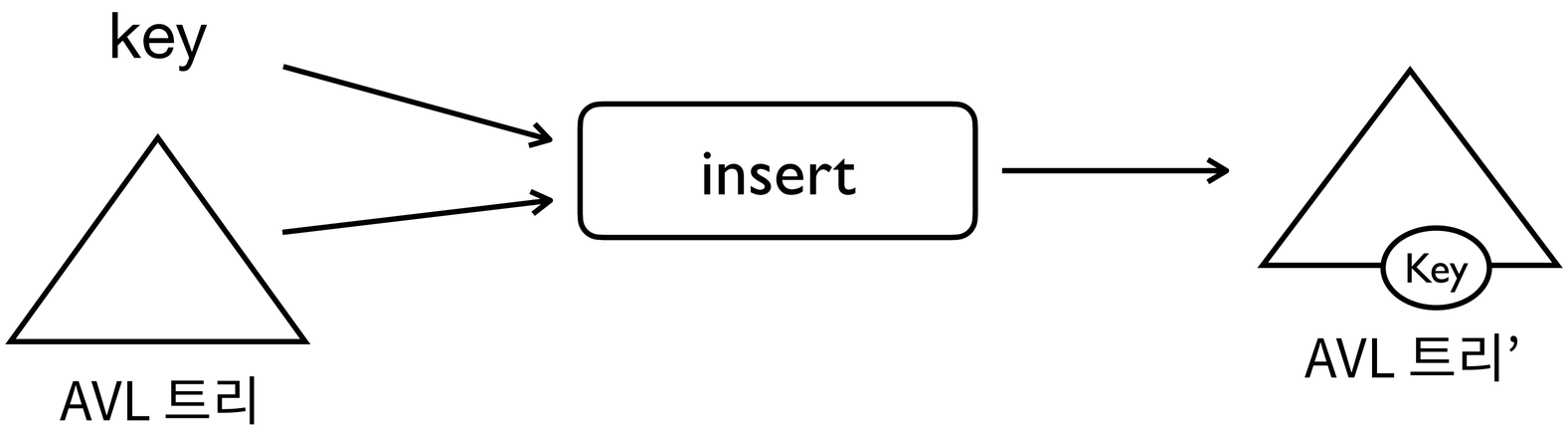
- AVL트리에서 삽입(insert) 또는 삭제(delete)에서 밸런싱(또는 회전)은 최대 몇 회 발생할까?





Quiz

Theorem AVL 트리에서 삽입을 했을 때, 최대 한번의 balancing만 일어난다.

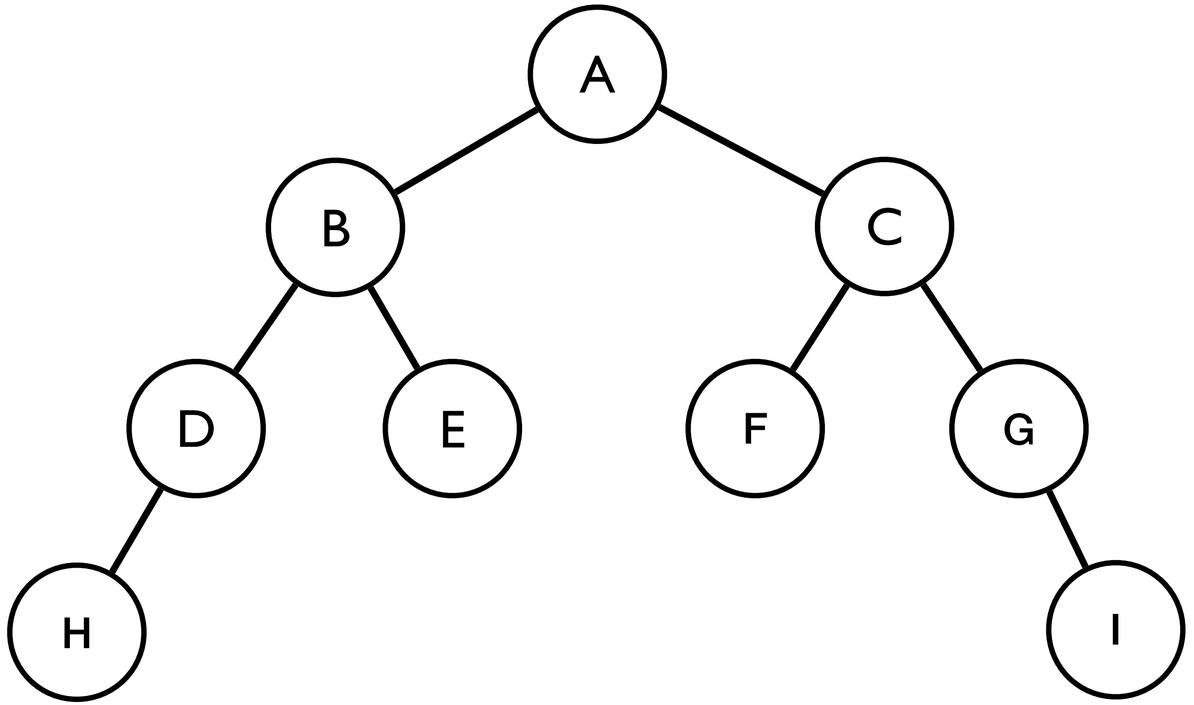


Quiz

Theorem AVL 트리에서 삽입을 했을 때, 최대 한번의 balancing만 일어난다.

Proof sketch.

(1) 삽입하기 전에는 AVL트리임



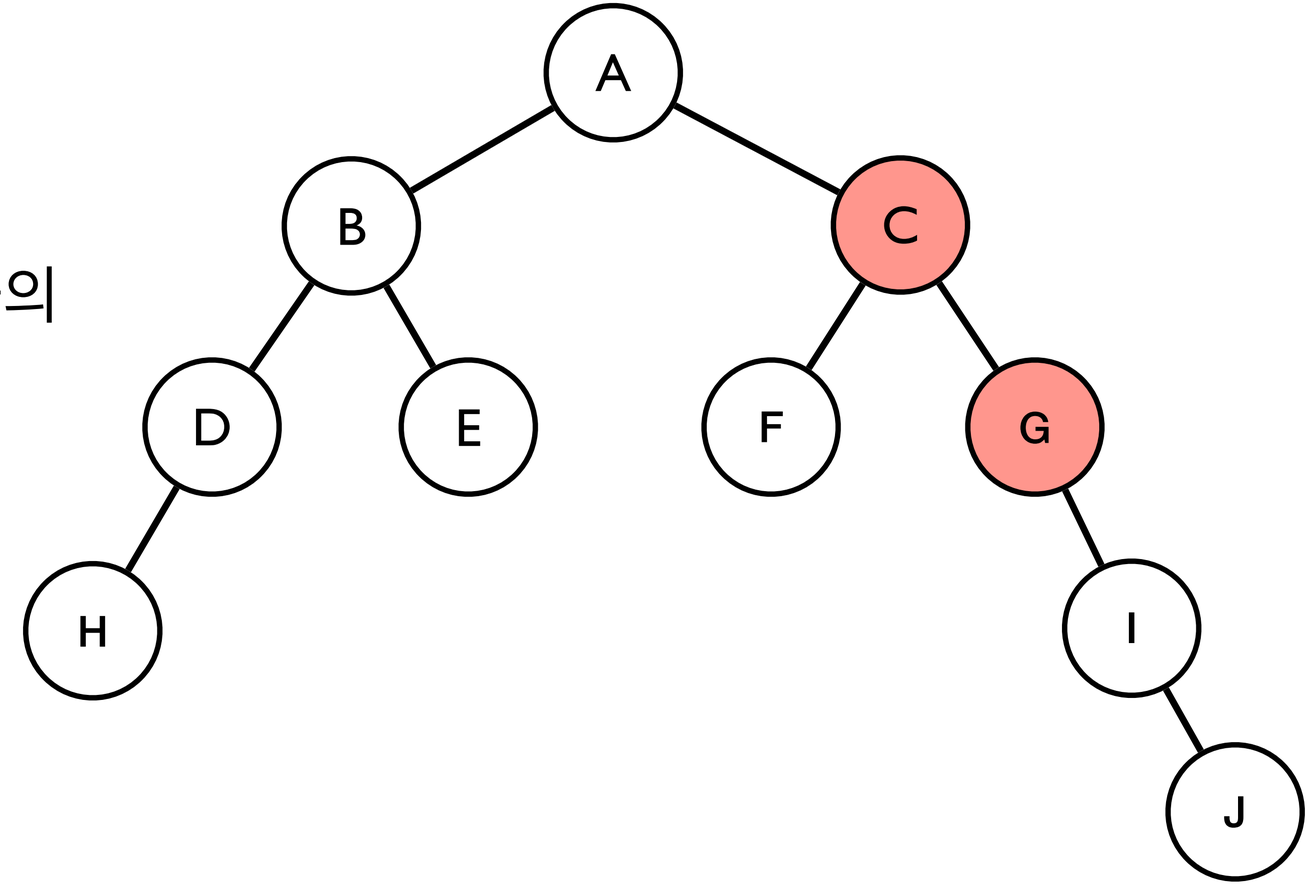
Quiz

Theorem AVL 트리에서 삽입을 했을 때, 최대 한번의 balancing만 일어난다.

Proof sketch.

(1) 삽입하기 전에는 AVL트리임

(2) 삽입 후 불균형이 발생 시 그 위로 조상노드들의 균형인수가 1 변화함



Quiz

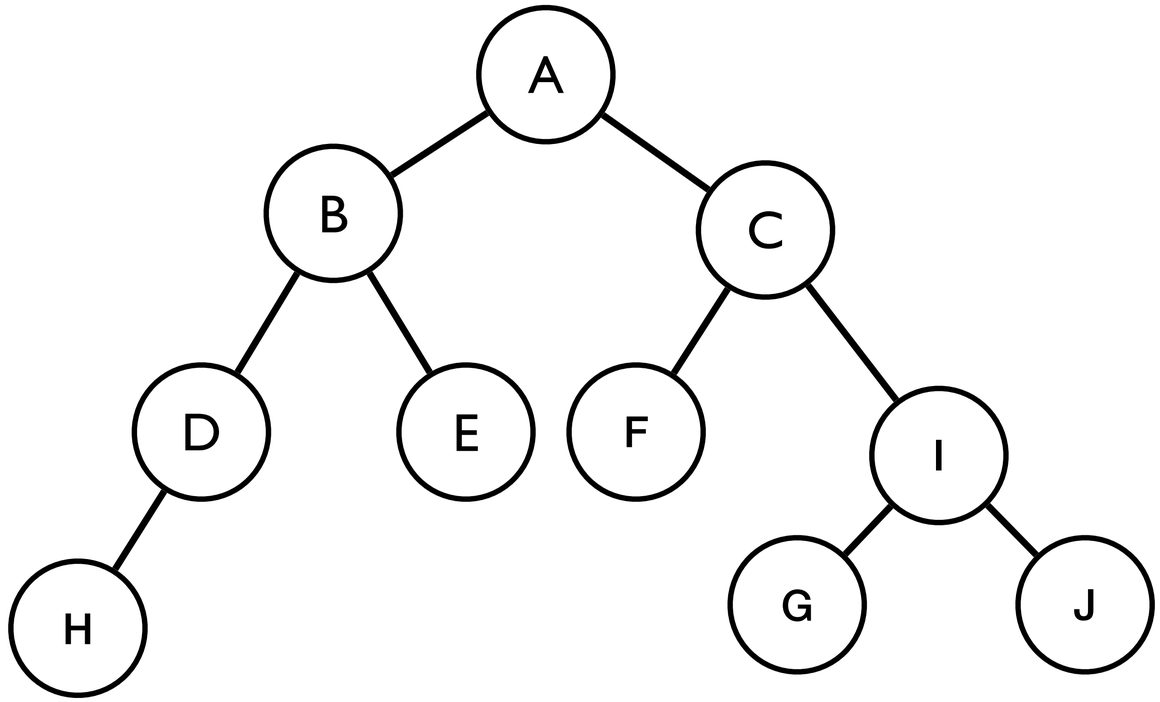
Theorem AVL 트리에서 삽입을 했을 때, 최대 한번의 balancing만 일어난다.

Proof sketch.

(1) 삽입하기 전에는 AVL트리임

(2) 삽입 후 불균형이 발생 시 그 위로 조상노드들의 균형인수가 1 변화함

(3) 회전 후에는 (2)에서 조상노드들의 변화한 균형인수가 원래대로 돌아옴



Quiz

Theorem AVL 트리에서 삽입을 했을 때, 최대 한번의 balancing만 일어난다.

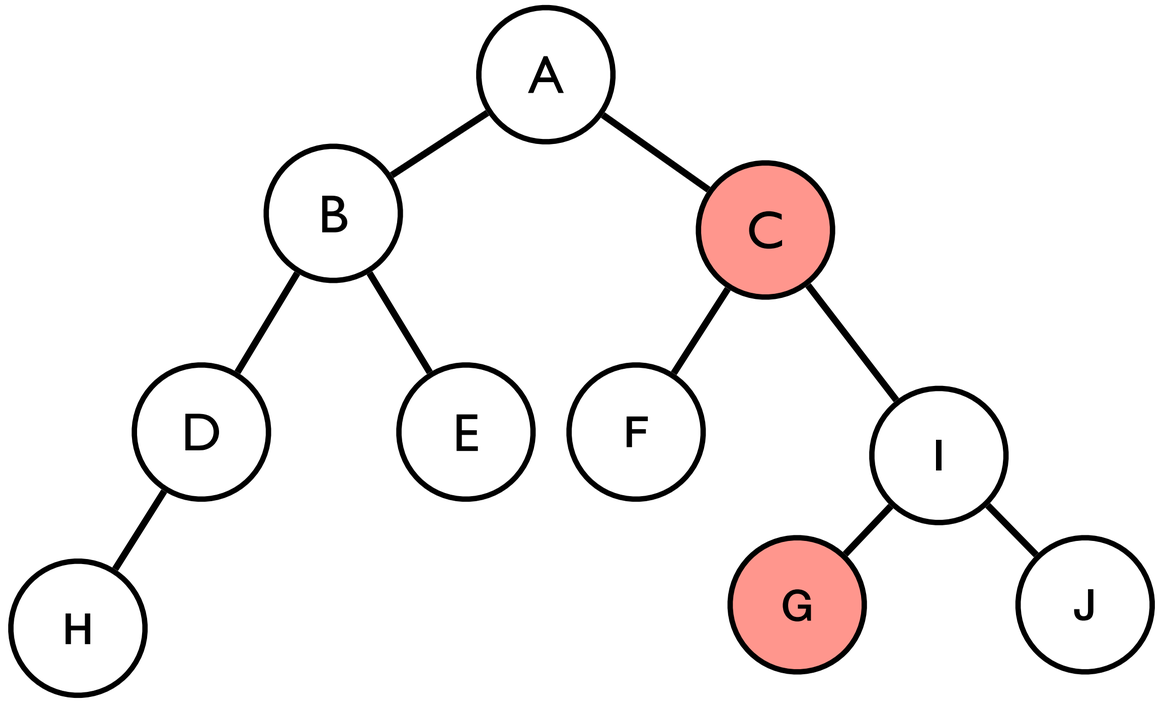
Proof sketch.

(1) 삽입하기 전에는 AVL트리임

(2) 삽입 후 불균형이 발생 시 그 위로 조상노드들의
균형인수가 1 변화함

(3) 회전 후에는 (2)에서 조상노드들의 변화한
균형인수가 원래대로 돌아옴

(4) 삽입 전 조상노드들의 균형인수들은 모두
-1, 0, 1이므로 더이상의 balancing은 일어나지 않음



Quiz

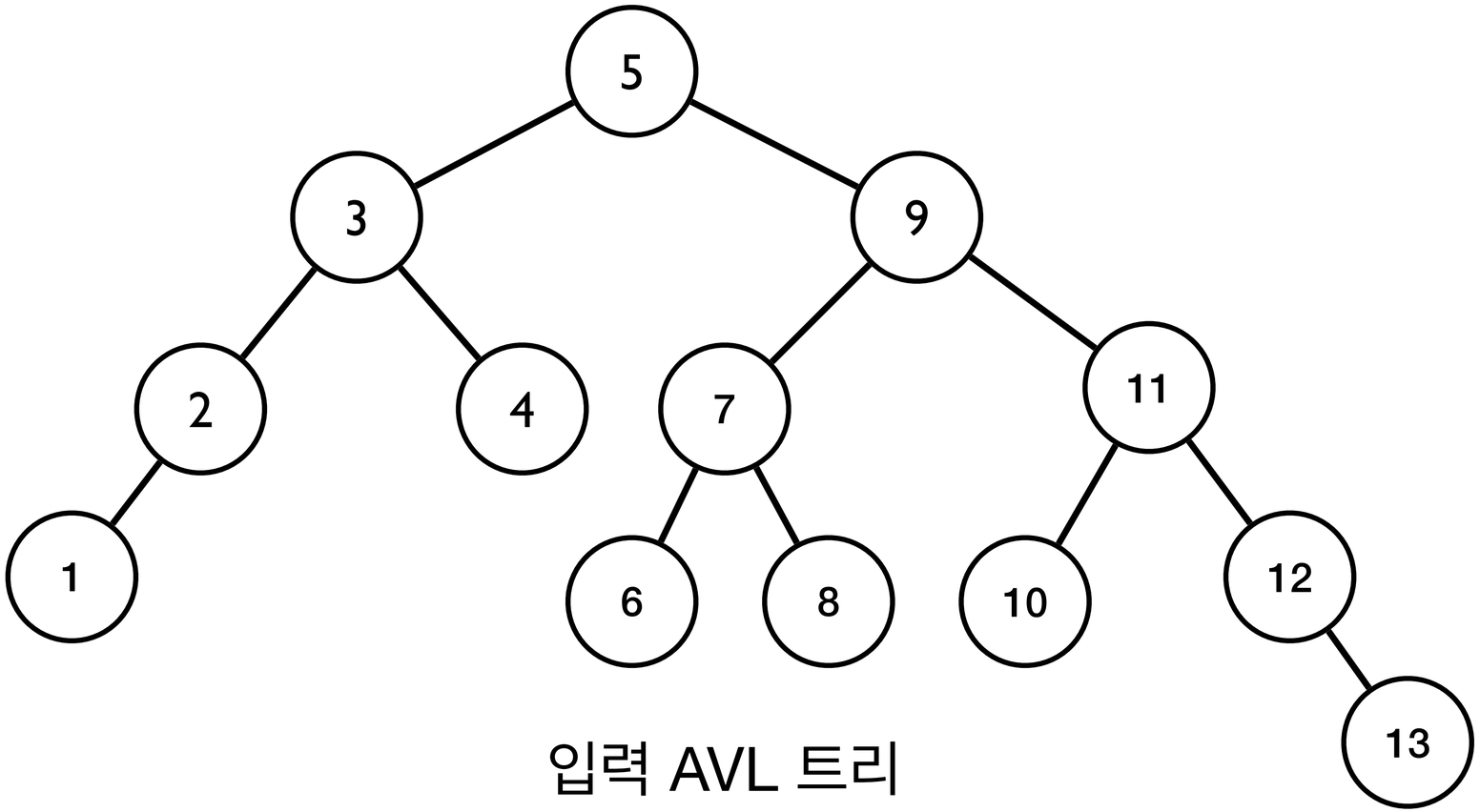
- 반면, AVL 트리에서 삭제을 했을 때, balancing은 여러번(두번 이상) 일어날 수 있다.

  반례:

Quiz

- 반면, AVL 트리에서 삭제을 했을 때, balancing은 여러번(두번 이상) 일어날 수 있다.

  반례:



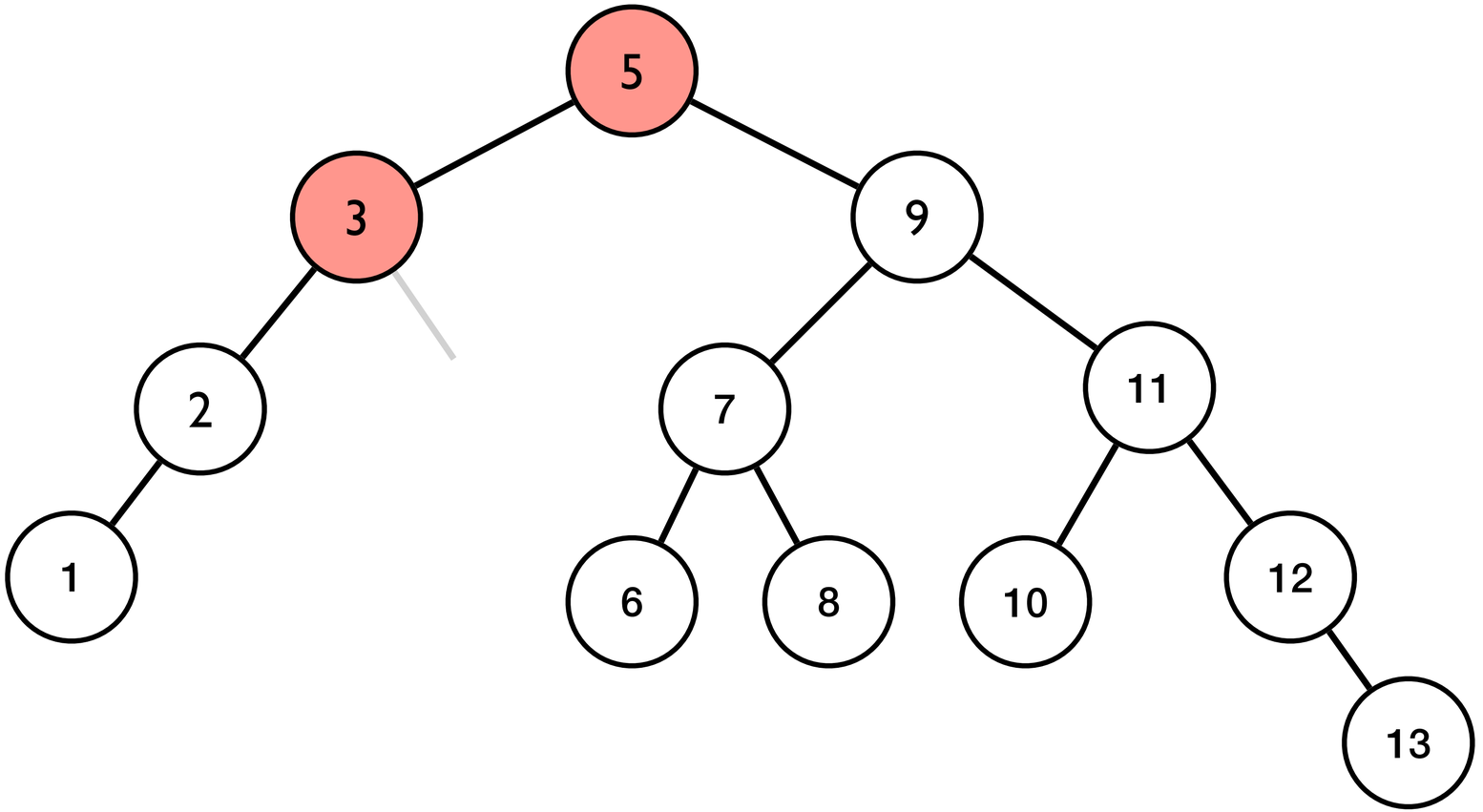
입력 AVL 트리

지우고자 하는 키 : 4

Quiz

- 반면, AVL 트리에서 삭제을 했을 때, balancing은 여러번(두번 이상) 일어날 수 있다.

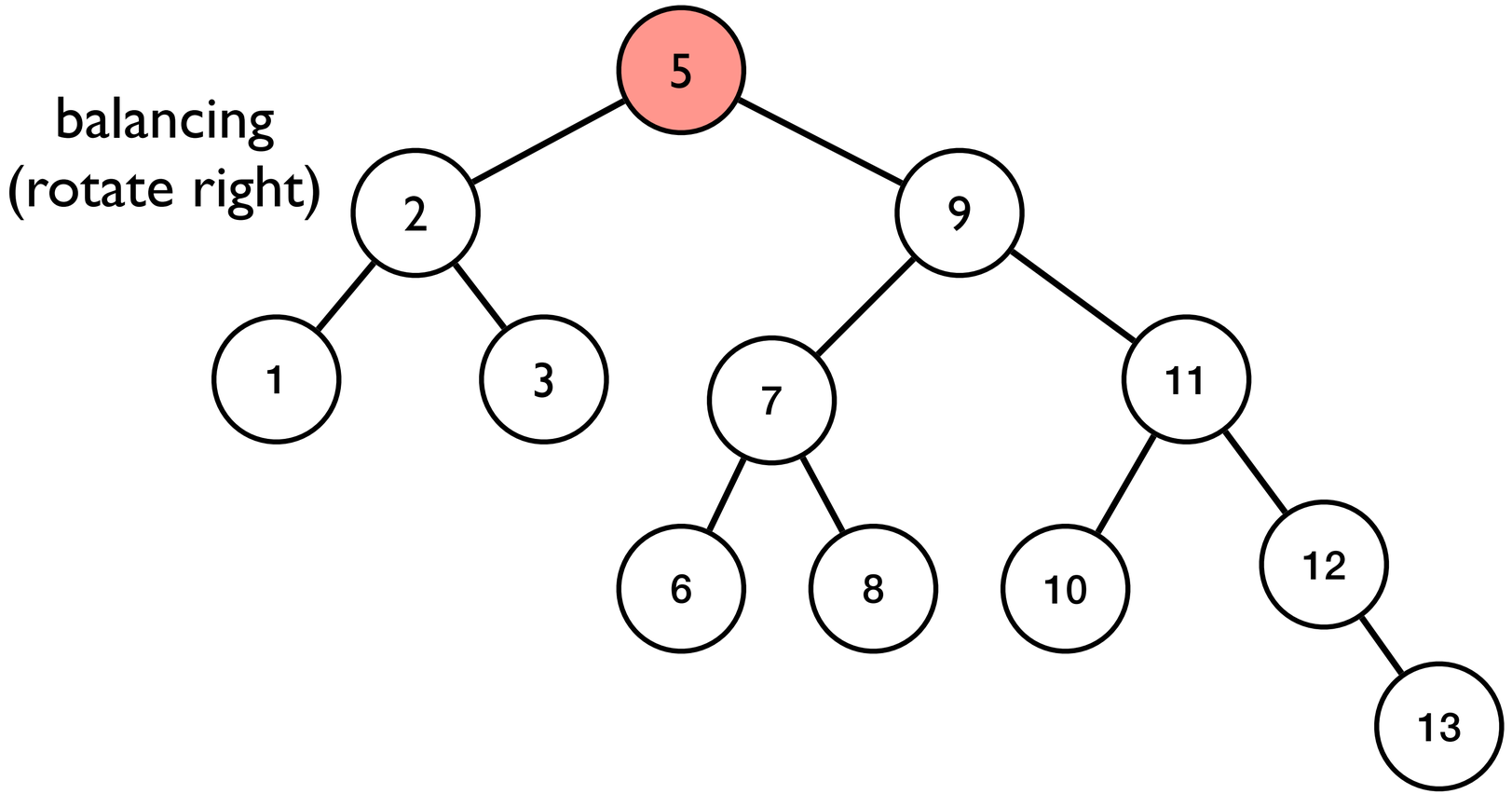
  반례:



Quiz

- 반면, AVL 트리에서 삭제을 했을 때, balancing은 여러번(두번 이상) 일어날 수 있다.

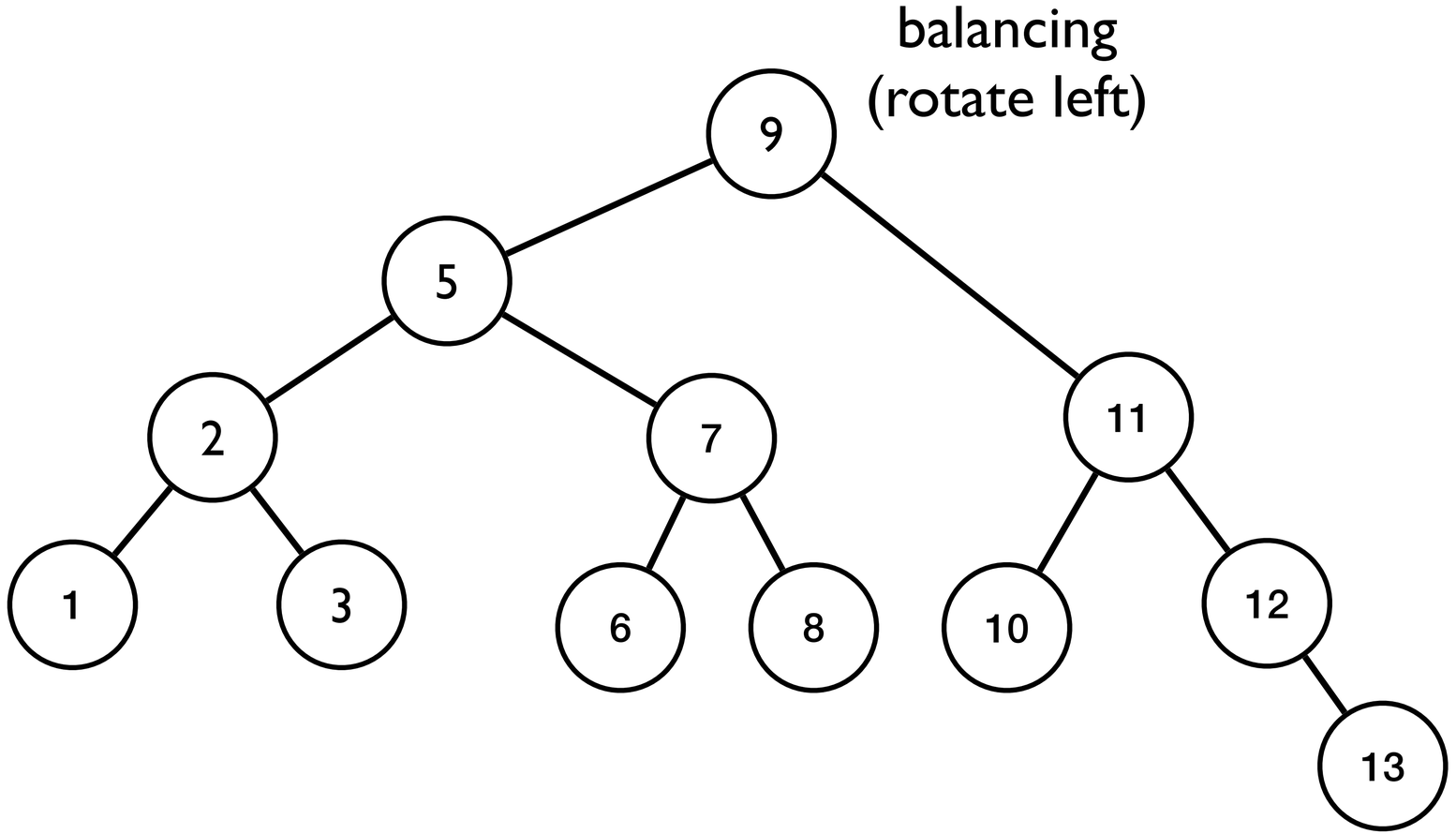
  반례:



Quiz

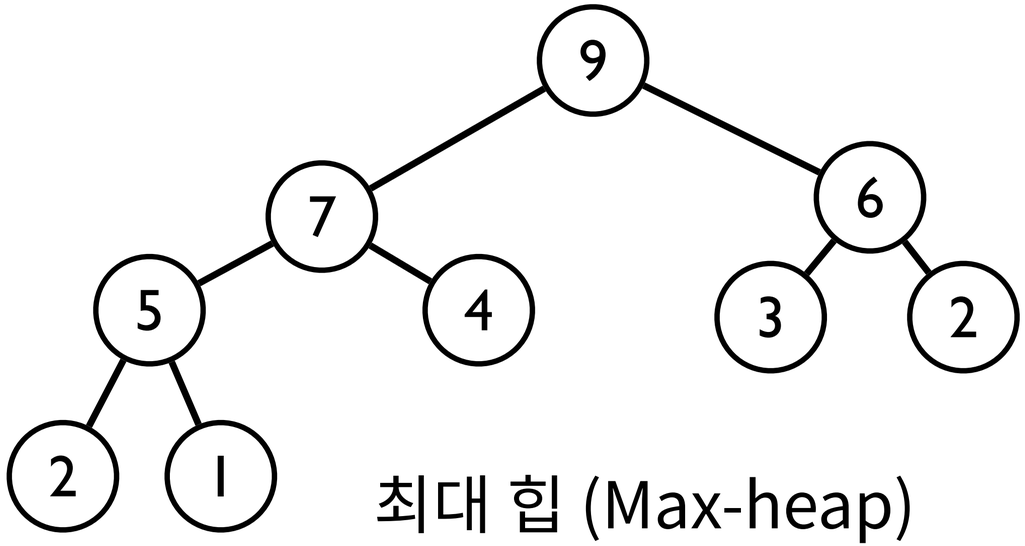
- 반면, AVL 트리에서 삭제을 했을 때, balancing은 여러번(두번 이상) 일어날 수 있다.

  반례:

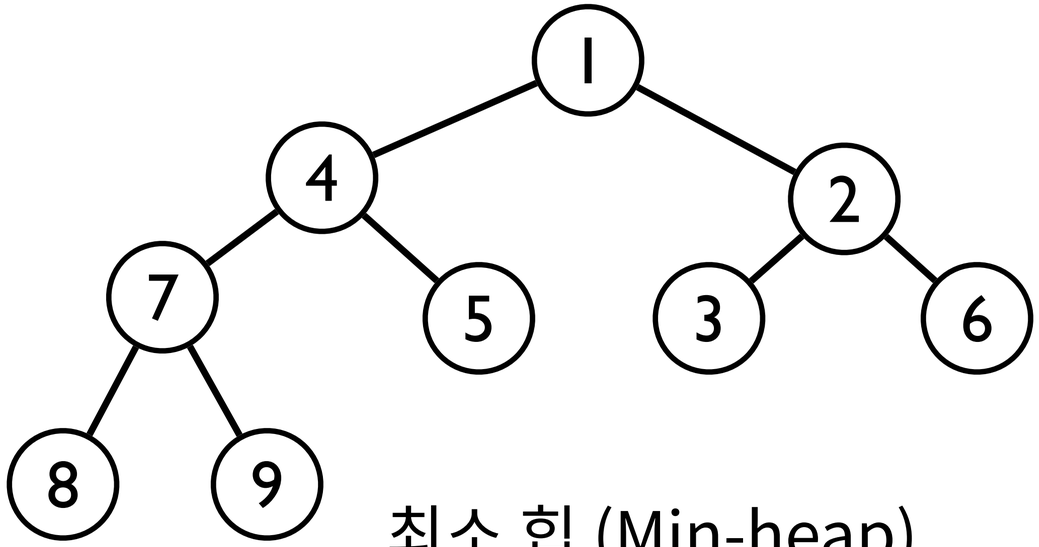


힙(Heap) 자료구조

- 최대 힙 (Max-heap): 부모 노드의 키 값이 자식노드들의 키값보다 항상 크거나 같음
  - 루트 노드가 가장 큰 키값을 가짐
- 최소 힙 (Min-heap): 부모 노드의 키 값이 자식노드들의 키값보다 항상 작거나 같음
  - 루트 노드가 가장 작음 키값을 가짐
- 힙 자료구조는 완전 이진트리임

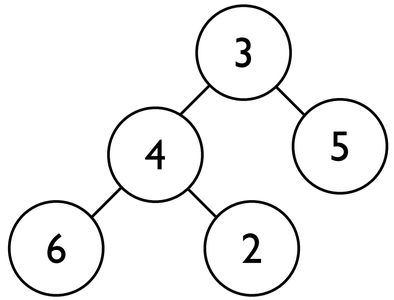


최대 힙 (Max-heap)



최소 힙 (Min-heap)

완전이진트리를 힙 자료구조로 바꾸기

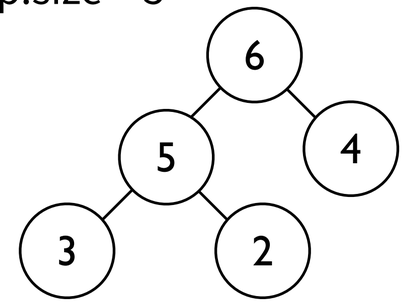


	3	4	5	6	2	

완전이진 트리 (arr)

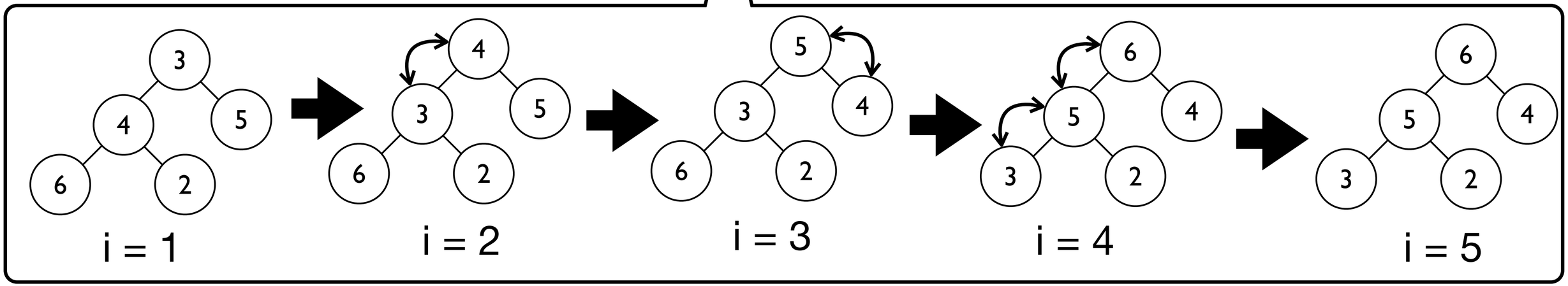
```
procedure buildHeap1(arr)
  heap ← allocateHeap()
  heap.arr ← arr
  heap.size ← length(arr)
  heap.capacity ← maxCapacity()
  for i = 1 to heap.size do
    heapifyUp(heap, i)
  end for
  return heap
```

heap.capacity =maxCapacity()
heap.size = 5

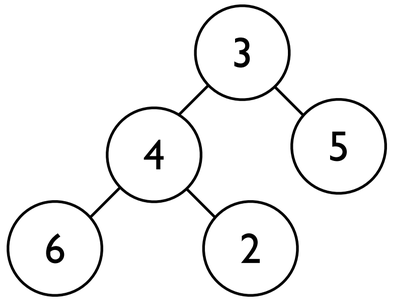


	6	5	4	3	2	

heap.arr



완전이진트리를 힙 자료구조로 바꾸기

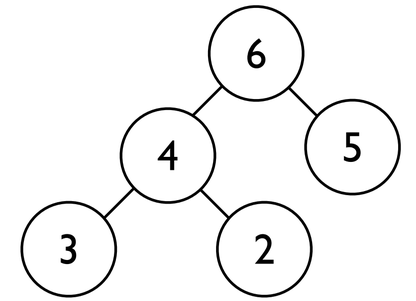


	3	4	5	6	2	

완전이진 트리 (arr)

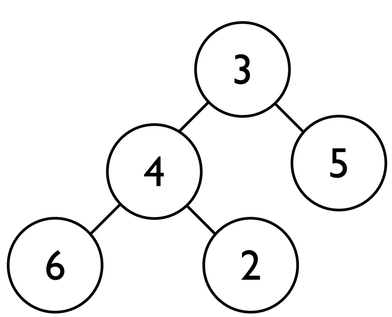
```
procedure buildHeap2(arr)
  heap ← allocateHeap()
  heap.arr ← arr
  heap.size ← length(arr)
  heap.capacity ← maxCapacity()
  for i = heap.size to 1 do
    heapifyDown(heap, i)
  end for
  return heap
```

heap.capacity =maxCapacity()
heap.size = 5

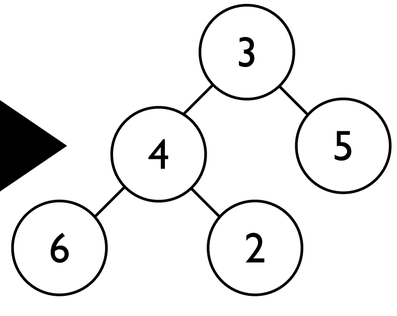


	6	5	4	3	2	

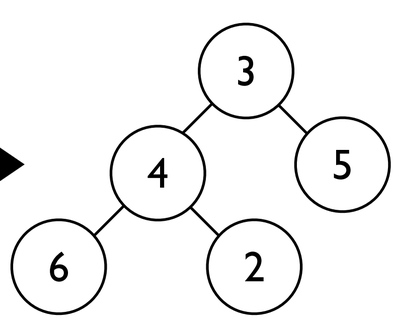
heap.arr



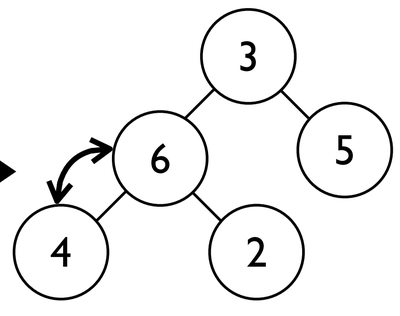
i = 5



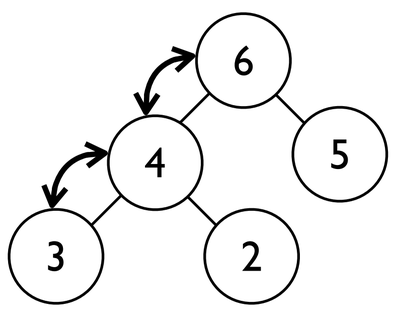
i = 4



i = 3



i = 2



i = 1

Quiz

- 아래 (잘못된) 알고리즘들을 사용하였을 때 힙(heap)이 되지 않는 입력(완전 이진 트리)을 각각 보이시오.

```
procedure buildHeap1(arr)
  heap ← allocateHeap()
  heap.arr ← arr
  heap.size ← length(arr)
  heap.capacity ← maxCapacity()
  for i = heap.size to 1 do
    heapifyUp(heap, i)
  end for
  return heap
```

반례:

```
procedure buildHeap2(arr)
  heap ← allocateHeap()
  heap.arr ← arr
  heap.size ← length(arr)
  heap.capacity ← maxCapacity()
  for i = 1 to heap.size do
    heapifyDown(heap, i)
  end for
  return heap
```

반례:

힙 자료구조의 응용 1 : k번째 큰 수 찾기(select k)

```c
#include "Heap.h"
#include <stdio.h>

int selectK(int arr[], int arr_size, int k) {
    Heap* heap = create();
    for (int i = 0; i < arr_size; i++) {
        insert(heap, arr[i]);
    }
    for (int i = 1; i < k; i++) {
        deleteRoot(heap);
    }
    int result = deleteRoot(heap);
    destroy(heap);
    return result;
}

int main() {
    int arr[] = {10, 20, 15, 30, 40};
    int k = 3;
    int result = selectK(arr, 5, k);
    printf("The %d-th largest element is %d\n", k, result);
    return 0;
}
```

힙 자료구조의 응용 3 : 힙 정렬(heap sort)

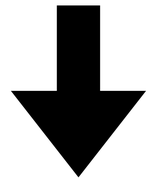
```c
#include "Heap.h"
#include <stdio.h>
#include <stdlib.h>

void myheapsort(int arr[], int size) {
    Heap* heap = create();
    for (int i = 0; i < size; i++) {
        insert(heap, arr[i]);
    }
    for (int i = 0; i < size; i++) {
        arr[i] = deleteRoot(heap);
    }
    destroy(heap);
}

int main() {
    int arr[] = {4, 6, 1, 2, 5, 3, 8};
    int size = sizeof(arr) / sizeof(arr[0]);
    myheapsort(arr, size);
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

arr

4	6	1	2	5	3	8



myheapsort(arr,7)

arr

8	6	5	4	3	2	1

힙 자료구조의 응용 3 : 힙 정렬(heap sort)

- 힙 정렬은 시간 복잡도가 $O(n \log n)$인 정렬 알고리즘

```
procedure heapsort(arr)
  heap ← create()
  for i = 0 to length(arr) -1 do
    insert(heap, arr[i])
  end for
  for i = length(arr) -1 to 0 do
    arr[i] = delete(heap)
  end for
 return arr
```

VS

```
procedure bubblesort(arr)
  for i = 0 to length(arr) -1 do
    for j = 0 to length(arr) - i -1 do
      if (arr[j] > arr[j+1]) then
        swap(arr[j], arr[j+1])
    end for
  end for
return arr
```

시간 복잡도 : $O(n \log n)$

시간 복잡도 : $O(n^2)$