Heap and priority Queue

박정민

01. 우선순위 큐

우선순위 큐: 우선순위의 개념을 큐에 도입한 자료 구조

- 데이터들이 우선순위를 가지고 있고 우선순위가 높은 데이터가 먼저 나간다.
-

자료 구조	삭제되는 요소
스택(Stack)	가장 최근에 들어온 데이터
큐(Queue)	가장 먼저 들어온 데이터
우선순위 큐(Priority Queue)	가장 우선순위가 높은 데이터

- 우선순위 큐의 이용 사례
  a. 시뮬레이션 시스템
  b. 네트워크 트래픽 제어
  c. 운영 체제에서의 작업 스케줄링
  d. 수치 해석적인 계산

01. 우선순위 큐

우선순위 큐는 배열, 연결리스트, 힙 으로 구현이 가능하다. 이 중에서 힙(heap)으로 구현하는 것이 가장 효율적이다.

-

우선순위 큐를 구현하는 표현 방법	삽입	삭제
순서 없는 배열	O(1)	O(n)
순서 없는 연결 리스트	O(1)	O(n)
정렬된 배열	O(n)	O(1)
정렬된 연결 리스트	O(n)	O(1)
힙(heap)	O(logn)	O(logn)

02. 자료구조 "힙(Heap)" 이란?

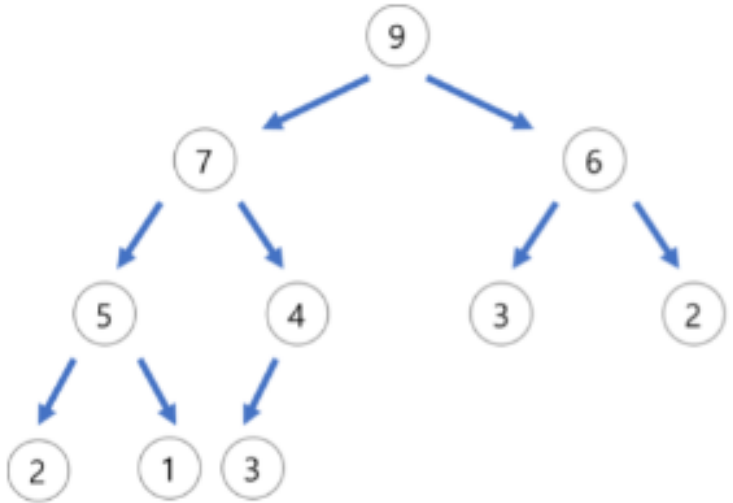
- **완전 이진 트리의 일종**으로 우선순위 큐를 위하여 만들어진 자료구조이다.
- 여러 개의 값들 중에서 최댓값이나 최솟값을 빠르게 찾아내도록 만들어진 자료구조이다.
- 힙은 일종의 **반정렬 상태(느슨한 정렬 상태)** 를 유지한다.
  - 큰 값이 상위 레벨에 있고 작은 값이 하위 레벨에 있다는 정도
  - 간단히 말하면 부모 노드의 키 값이 자식 노드의 키 값보다 항상 큰(작은) 이진 트리를 말한다.
- 힙 트리에서는 중복된 값을 허용한다. (이진 탐색 트리에서는 중복된 값을 허용하지 않는다.)

02. 자료구조 "힙(Heap)" 이란?

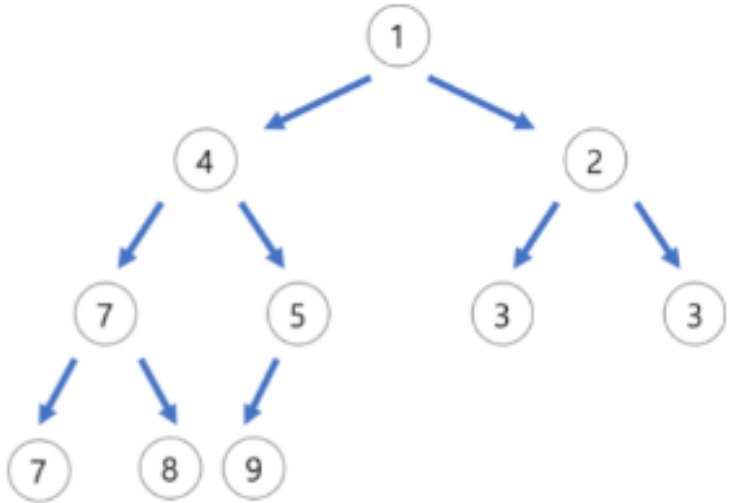
```
{ 9, 8, 7, 6, 5, 4, 3, 2, 1}
```

힙(heap)의 종류

- 최대 힙(max heap)
  - 부모 노드의 키 값이 자식 노드의 키 값보다 크거나 같은 완전 이진 트리
  - key(부모 노드) >= key(자식 노드)
- 최소 힙(min heap)
  - 부모 노드의 키 값이 자식 노드의 키 값보다 작거나 같은 완전 이진 트리
  - key(부모 노드) <= key(자식 노드)
-



-최대 힙(max heap)-



-최소 힙(min heap)-

03. 힙 구현

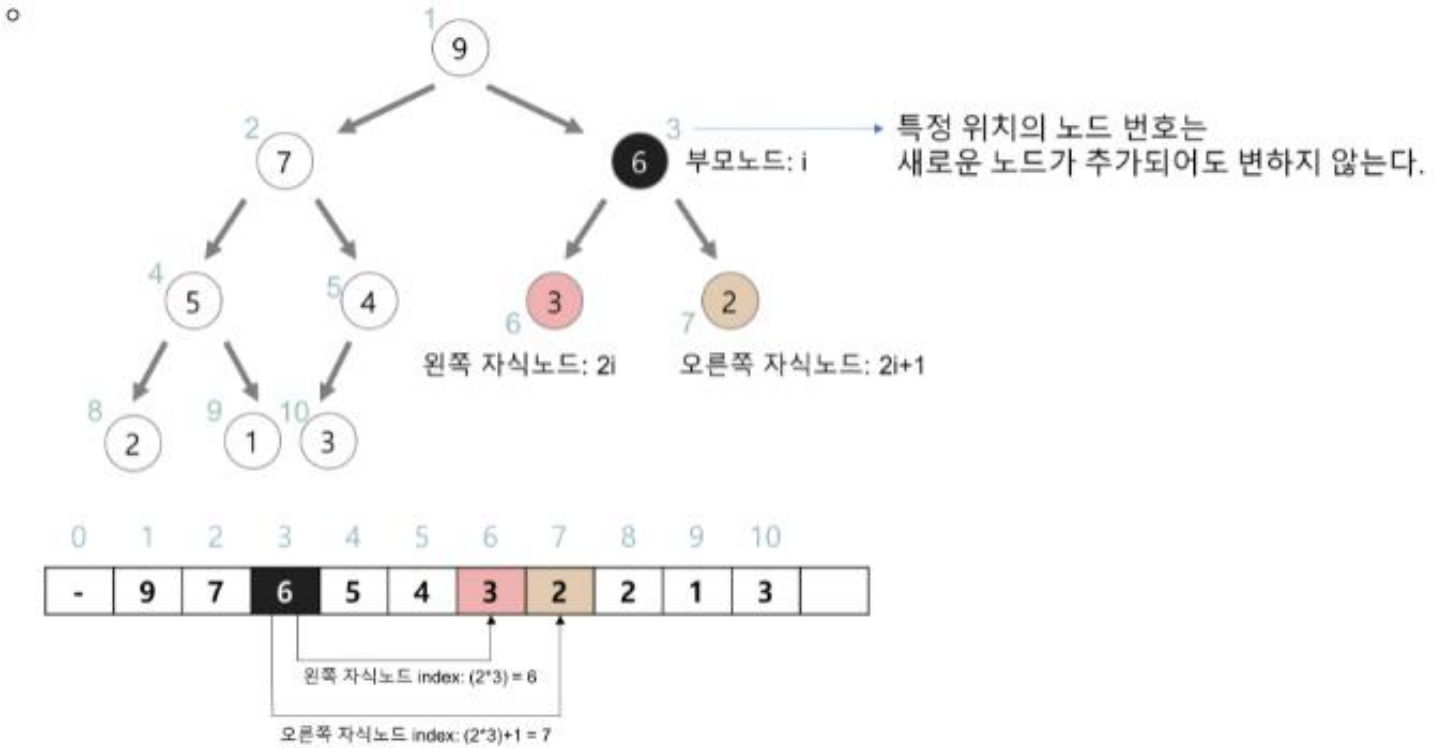
- 힙을 저장하는 표준적인 자료구조는 배열 이다.
- 구현을 쉽게 하기 위하여 배열의 첫 번째 인덱스인 0은 사용되지 않는다.
- 특정 위치의 노드 번호는 새로운 노드가 추가되어도 변하지 않는다.
- (예를 들어 루트 노드의 오른쪽 노드의 번호는 항상 3이다.)

03. 힙 구현

- 힙을 저장하는 표준적인 자료구조는 배열 이다.
- 구현을 쉽게 하기 위하여 배열의 첫 번째 인덱스인 0은 사용되지 않는다.
- 특정 위치의 노드 번호는 새로운 노드가 추가되어도 변하지 않는다.
- (예를 들어 루트 노드의 오른쪽 노드의 번호는 항상 3이다.)

03. 힙에서의 부모 노드와 자식 노드의 관계

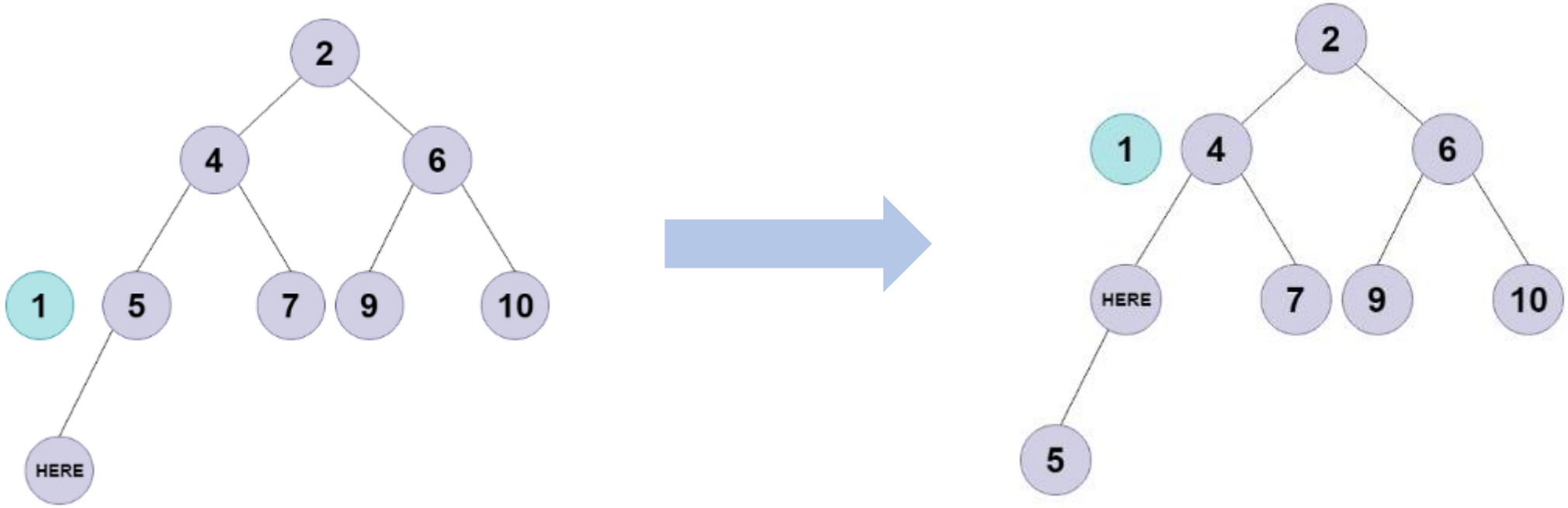
- 왼쪽 자식의 인덱스 = (부모의 인덱스) * 2
- 오른쪽 자식의 인덱스 = (부모의 인덱스) * 2 + 1
- 부모의 인덱스 = (자식의 인덱스) /2



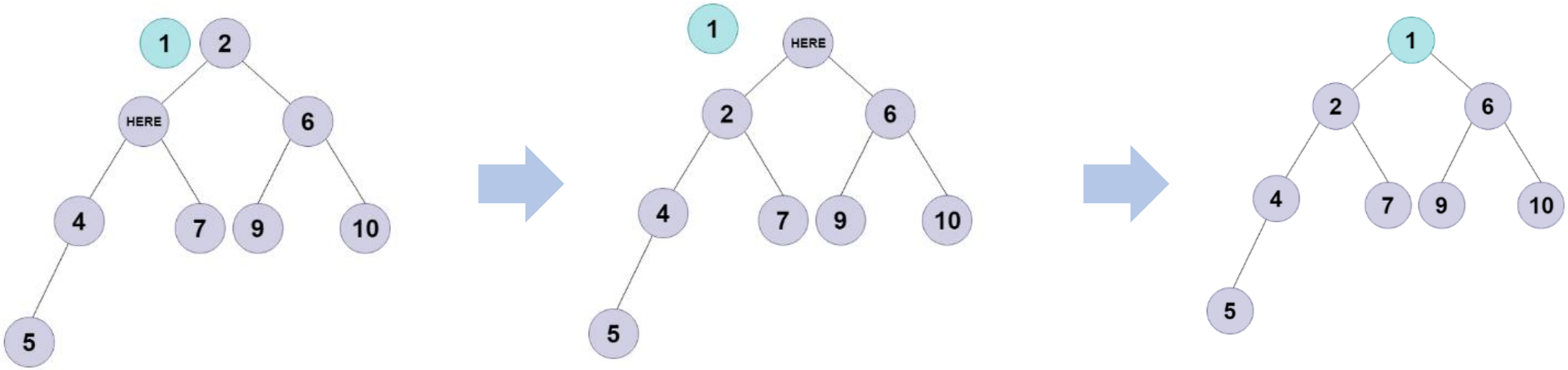
03. 힙 구현

```
typedef struct heap {
    int arr[MAX_N];
    int size;
} heap;
```

03. Heap insert



03. Heap insert

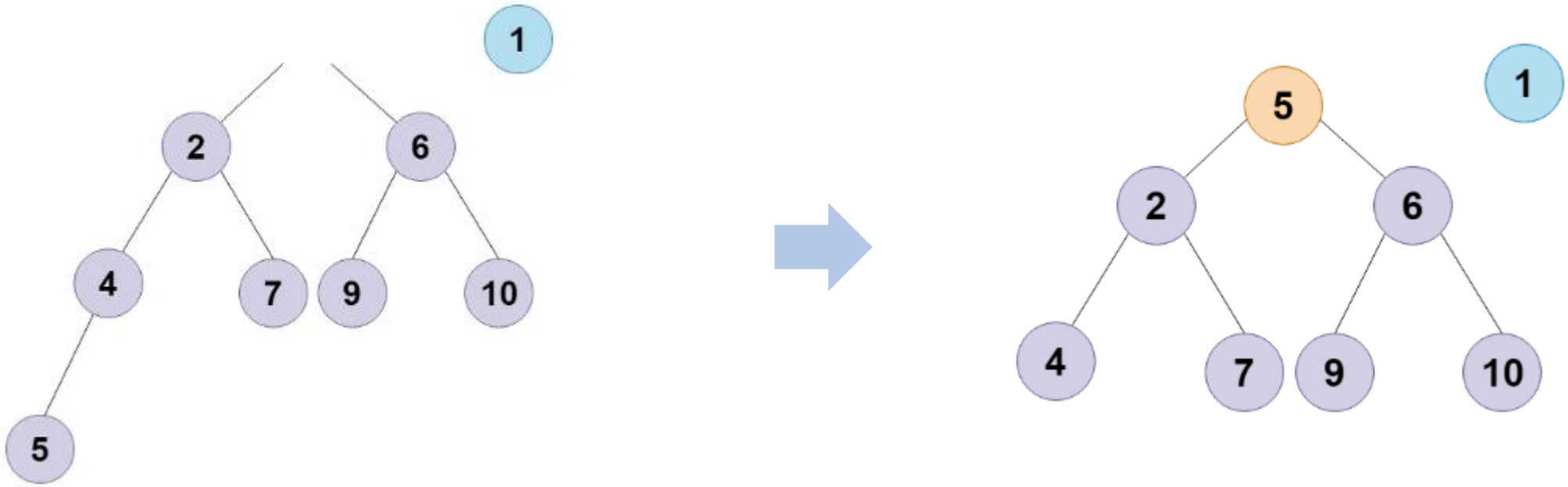


03. Heap Instert code

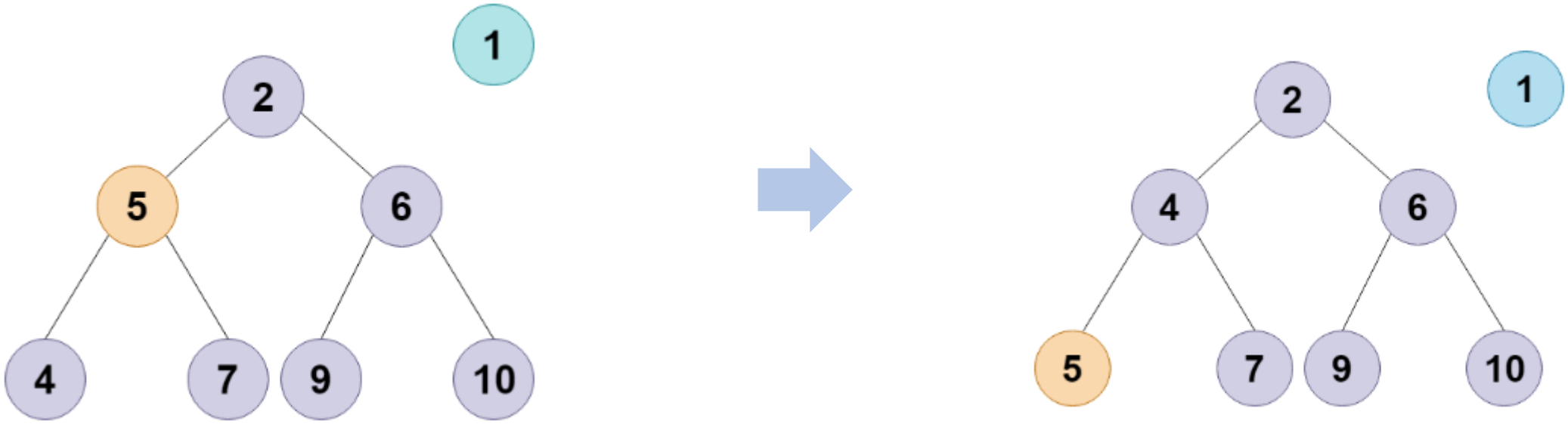
```
void insert(heap* hp, int data) {
    int here = ++hp->size;

    while ((here != 1) && (data < hp->arr[here / 2])) {
        hp->arr[here] = hp->arr[here / 2];
        here /= 2;
    }
    hp->arr[here] = data;
}
```

03. Heap Delete



03. Heap Delete



03. Heap Delete

```
int deleteData(heap* hp) {
    if (hp->size == 0) return -1;
    int ret = hp->arr[1];
    hp->arr[1] = hp->arr[hp->size--];
    int parent = 1;
    int child;

    while (1) {
        child = parent * 2;
        if (child + 1 <= hp->size && hp->arr[child] > hp->arr[child + 1])
            child++;

        if (child > hp->size || hp->arr[child] > hp->arr[parent]) break;

        swap(&hp->arr[parent], &hp->arr[child]);
        parent = child;
    }

    return ret;

}
```

03. Binary Search

03. Binary Search

03. Binary Search 성능

$$n \times \left(\frac{1}{2}\right)^k = 1$$

$$n \times \frac{1}{2^k} = 1$$

$$n = 2^k$$

$$k = \log_2 n$$

03. Binary Search Code

```
int BinarySearch(int dataArr[], int size, int findData) {
    int low = 0, high = size - 1, mid;
    while (low <= high) {
        mid = (low + high) / 2;
        if (dataArr[mid] > findData) high = mid - 1;
        else if (dataArr[mid] < findData) low = mid + 1;
    }
```

Thank you