



7 트리

IT CookBook, C로 배우는 쉬운 자료구조(개정 3판)

❖ 학습목표

- 히프의 자료구조를 이해한다.

❖ 내용

- 트리의 이해
- 이진 트리
- 이진 트리의 구현
- 이진 트리의 순회
- 이진 탐색 트리
- **히프의 개념과 연산 및 구현**



7. 힙의 개념과 연산 및 구현

❖ 힙^{heap}의 개념

- 완전 이진 트리에 있는 노드 중에서 키값이 가장 큰 노드나 키값이 가장 작은 노드를 찾기 위해서 만든 자료구조
- 최대 힙^{max heap}
 - 키값이 가장 큰 노드를 찾기 위한 완전 이진 트리
 - {부모노드의 키값 \geq 자식노드의 키값}
 - 루트 노드 : 키값이 가장 큰 노드
- 최소 힙^{min heap}
 - 키값이 가장 작은 노드를 찾기 위한 완전 이진 트리
 - {부모노드의 키값 \leq 자식노드의 키값}
 - 루트 노드 : 키값이 가장 작은 노드



7. 힙의 개념과 연산 및 구현

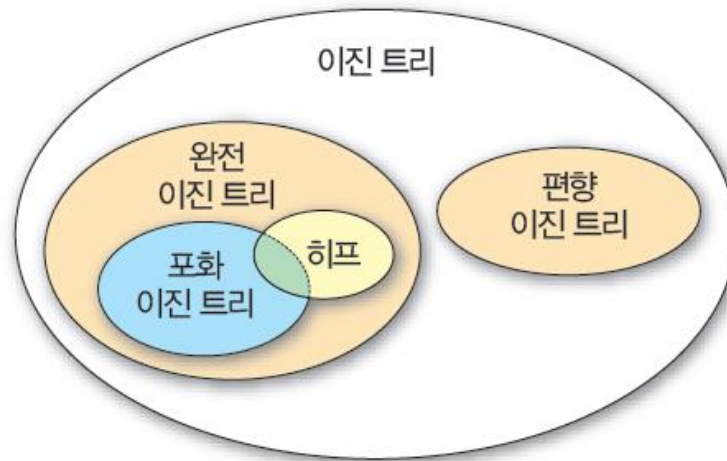
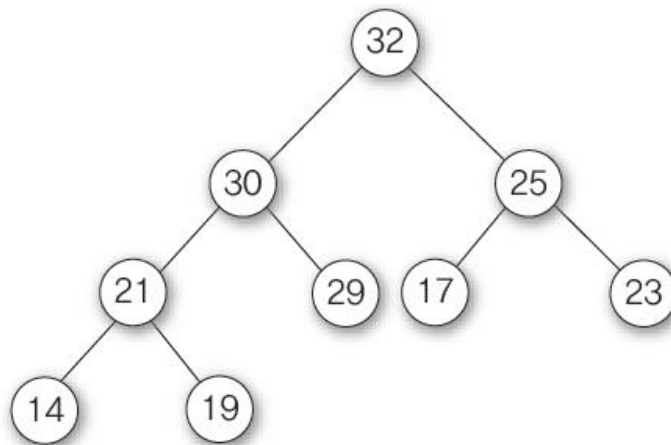
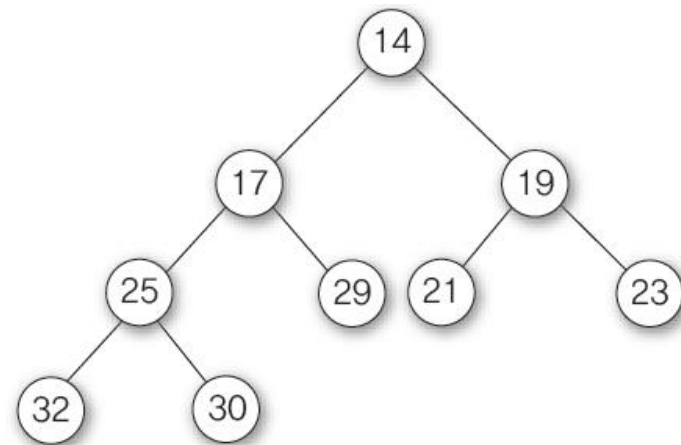


그림 7-53 이진 트리와 힙의 관계

■ 힙의 예



(a) 최대 힙



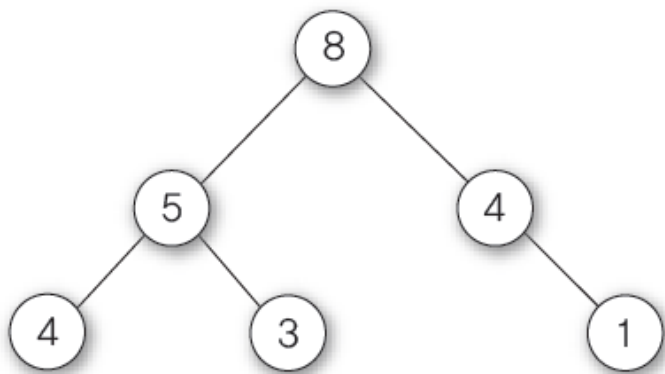
(b) 최소 힙

그림 7-54 힙의 예

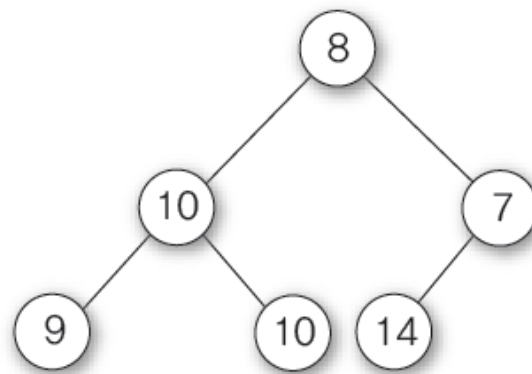


7. 힙의 개념과 연산 및 구현

■ 힙이 아닌 이진 트리의 예



(a) 완전 이진 트리가 아님



(b) 부모 노드의 키값과 자식 노드의 키값 사이의 크기 관계 미성립

그림 7-55 힙이 아닌 이진 트리의 예



7. 힙의 개념과 연산 및 구현 : 추상 자료형

❖ 힙의 추상 자료형

ADT 7-2

힙의 추상 자료형

ADT Heap

데이터 : 원소 n 개로 구성된 완전 이진 트리로서 각 노드의 키값은 자식 노드의 키값보다 크거나 같다
(부모 노드의 키값 \geq 자식 노드의 키값).

연산 :

$\text{heap} \in \text{Heap}; \text{item} \in \text{Element};$

// 공백 힙을 생성하는 연산

$\text{createHeap}() ::= \text{create an empty heap};$

// 힙이 공백인지 검사하는 연산

$\text{isEmpty}(\text{heap}) ::= \text{if (heap is empty) then return true;}$
 $\text{else return false;}$

// 힙의 적당한 위치에 원소(item)를 삽입하는 연산

$\text{insertHeap}(\text{heap}, \text{item}) ::= \text{insert item into heap};$



7. 힙의 개념과 연산 및 구현 : 추상 자료형

// 힙에서 키값이 가장 큰 원소를 삭제하고 반환하는 연산

```
deleteHeap(heap) ::= if (isEmpty(heap)) then return error;  
                    else {  
                        item  $\leftarrow$  힙에서 가장 큰 원소;  
                        remove {힙에서 가장 큰 원소};  
                        return item;  
                    }
```

End Heap()



7. 힙의 개념과 연산 및 구현 : 삽입 연산

❖ 힙의 삽입 연산

1단계 : 완전 이진 트리의 조건이 만족하도록 다음 자리를 확장

- 노드가 n 개인 완전 이진 트리에서 다음 노드의 확장 자리는 $n+1$ 번의 노드
- $n+1$ 번 자리에 노드를 확장하고, 그 자리에 삽입할 원소를 임시 저장

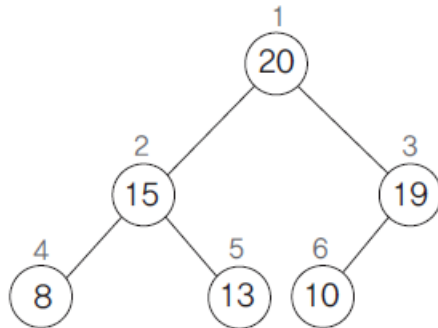
2단계 : 부모 노드와 크기 조건이 만족하도록 삽입 원소의 위치를 찾음

- 현재 위치에서 부모노드와 비교하여 크기 관계를 확인
- {현재 부모노드의 키값 \geq 삽입 원소의 키값}의 관계가 성립하지 않으면,
현재 부모노드의 원소와 삽입 원소의 자리를 서로 바꿈

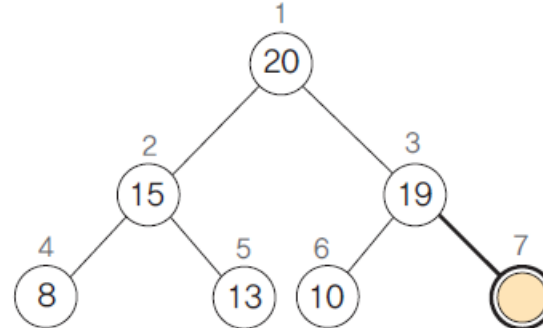


7. 힙의 개념과 연산 및 구현 : 삽입 연산

- 힙에서의 삽입 연산 예1) 17을 삽입하는 경우
 - 노드를 확장하여 임시로 저장한 위치에서의 부모노드와 크기를 비교하여 힙의 크기관계가 성립하므로, 현재 위치를 삽입 원소의 자리로 확정

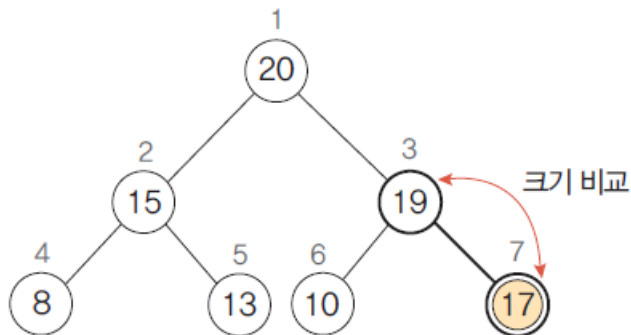


(a) 삽입 전의 힙



(b) 1단계 : 완전 이진 트리의 다음 자리인 7번 노드 확장

① 확장한 자리에 삽입 원소 17을 임시 저장



(c) 2단계 : 삽입 원소 17의 자리 확정

② 현재 위치에서 부모 노드와 크기 비교 후 자리 확정

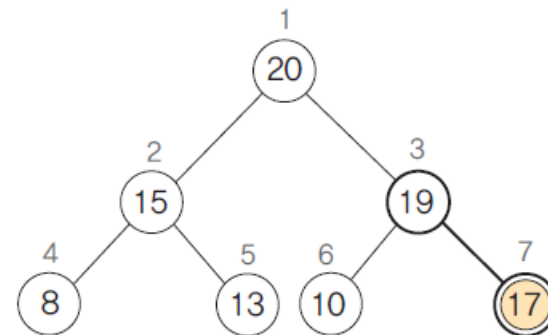


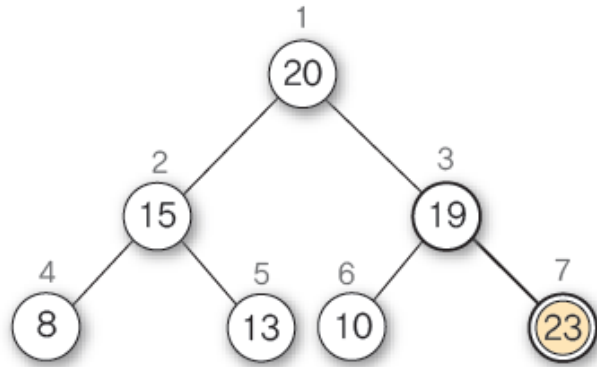
그림 7-56 힙에서의 삽입 연산 예 1 : 원소 17 삽입하기



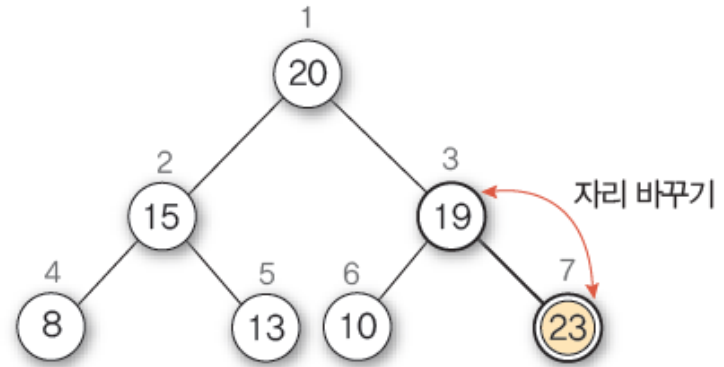
7. 힙의 개념과 연산 및 구현 : 삽입 연산

■ 힙에서의 삽입 연산 예2) 23을 삽입하는 경우

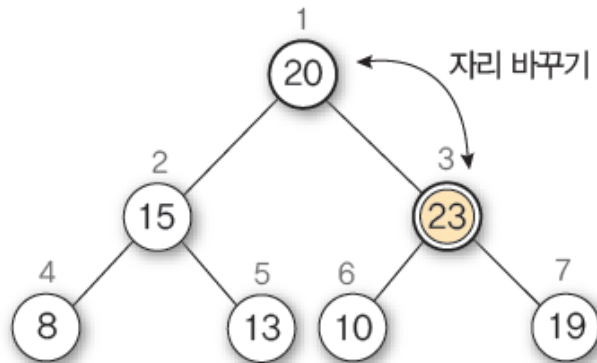
① 확장한 자리에 삽입 원소 23을 임시 저장



② (부모 노드 19 < 삽입 노드 23)이므로 자리 바꾸기



③ (부모 노드 20 < 삽입 노드 23)이므로 자리 바꾸기



④ 더 이상 비교할 부모 노드가 없으므로 자리 확정

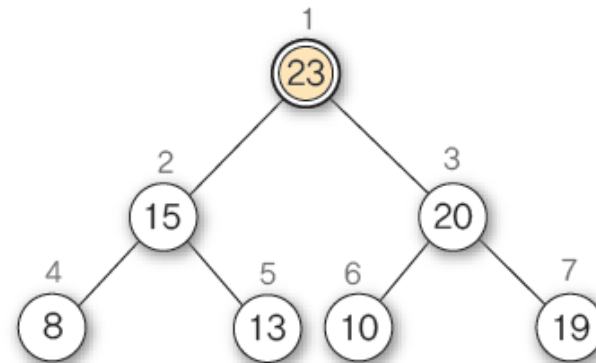


그림 7-57 힙에서의 삽입 연산 예 2 : 원소 23 삽입하기



7. 힙의 개념과 연산 및 구현 : 삽입 연산

■ 힙에서의 삽입 연산 알고리즘

알고리즘 7-11 최대 힙의 노드 삽입

```
insertHeap(heap, item)
  if (n = heapSize) then heapFull();
  ① n ← n + 1;
  ② for (i ← n; ;) do {
    if (i = 1) then exit;
    ③ if (item ≤ heap[⌊i/2⌋]) then exit;
    ④ heap[i] ← heap[⌊i/2⌋];
    ⑤ i ← ⌊i/2⌋;
  }
  ⑥ heap[i] ← item;
end insertHeap()
```



7. 힙의 개념과 연산 및 구현 : 삽입 연산

- ❶ 현재 힙의 크기를 하나 증가시켜서 노드 위치를 확장
- ❷ 확장한 노드 번호가 임시 삽입 위치 i 가 됨
- ❸ 삽입할 원소 $item$ 과 부모 노드 $heap[\lfloor i/2 \rfloor]$ 를 비교하여 부모 노드보다 작거나 같으면 임시 삽입 위치 i 를 삽입 원소의 위치로 확정, ❹을 수행
- ❹ 삽입할 원소 $item$ 이 부모 노드보다 크면, 부모 노드와 자식 노드의 자리를 맞바꿔 최대 힙의 관계를 만들어야 하므로 부모 노드 $heap[\lfloor i/2 \rfloor]$ 의 원소를 현재의 임시 삽입 위치 $heap[i]$ 에 저장
- ❺ $\lfloor i/2 \rfloor$ 를 임시 삽입 위치 i 로 하여 ❷~❹를 반복하면서 $item$ 을 삽입할 위치를 찾음
- ❻ 찾은 위치에 삽입할 원소 $item$ 을 저장하면 최대 힙의 재구성 작업이 완성되므로 삽입 연산을 종료



7. 힙의 개념과 연산 및 구현 : 삭제 연산

❖ 힙의 삭제 연산

- 힙에서는 루트 노드의 원소만을 삭제 할 수 있음

1단계 : 루트 노드의 원소를 삭제하여 반환

2단계 : 원소의 개수가 $n-1$ 개로 줄었으므로, 노드의 수가 $n-1$ 인 완전 이진 트리로 조정

- 노드가 n 개인 완전 이진 트리에서 노드 수 $n-1$ 개의 완전 이진 트리가 되기 위해서 마지막 노드, 즉 n 번 노드를 삭제
- 삭제된 n 번 노드에 있던 원소는 비어있는 루트노드에 임시 저장

3단계 : 완전 이진 트리 내에서 루트에 임시 저장된 원소의 제자리를 찾음

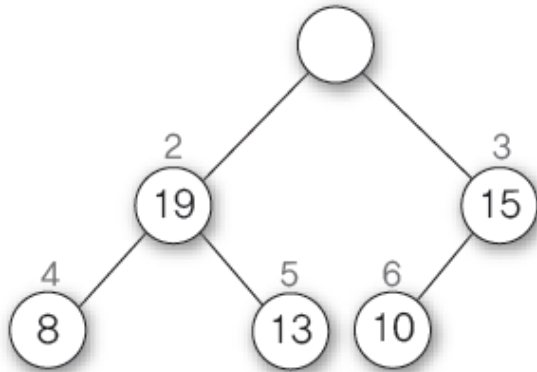
- 현재 위치에서 자식노드와 비교하여 크기 관계를 확인
- {임시 저장 원소의 키값 \geq 현재 자식노드의 키값}의 관계가 성립하지 않으면, 현재 자식노드의 원소와 임시 저장 원소의 자리를 서로 바꿈



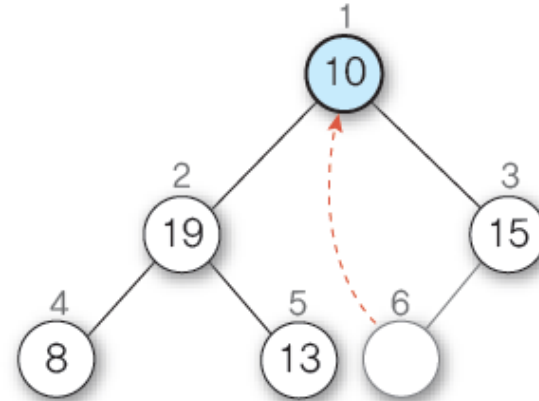
7. 힙의 개념과 연산 및 구현 : 삭제 연산

■ 힙에서의 삭제 연산 예

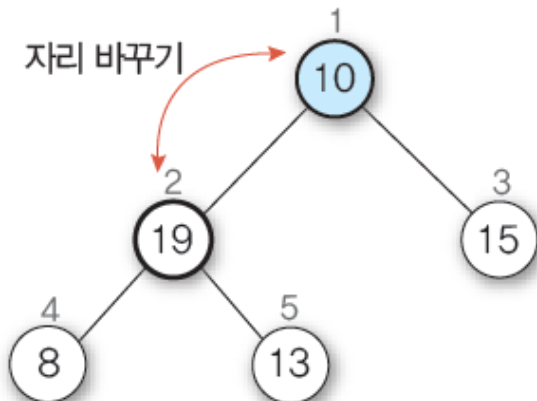
① 루트 노드의 원소 삭제



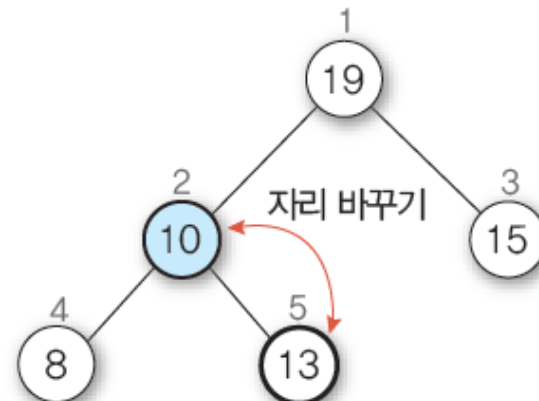
② 마지막 노드 삭제 후 원소를 루트로 이동



③ (삽입 노드 10 < 자식 노드 19)이므로 자리 바꾸기



④ (삽입 노드 10 < 자식 노드 13)이므로 자리 바꾸기



7. 힙의 개념과 연산 및 구현 : 삭제 연산

⑤ 자리 확정

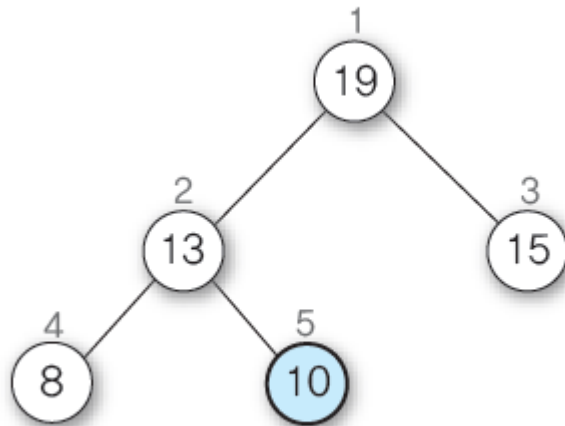


그림 7-58 힙에서 루트의 원소를 삭제하는 예



7. 힙의 개념과 연산 및 구현 : 삭제 연산

알고리즘 7-12 최대 힙의 노드 삭제

```
deleteHeap(heap)
  if (n = 0) then return error;
  ① item ← heap[1];
  ② temp ← heap[n];
  ③ n ← n - 1;
  ④ i ← 1;
    j ← 2;
  ⑤ { while (j ≤ n) do {
      if (j < n) then
        if (heap[j] < heap[j + 1]) then j ← j + 1;
      if (temp ≥ heap[j]) then exit;
      heap[i] ← heap[j];
  ⑥ { i ← j;
      j ← j * 2;
    }
  ⑦ heap[i] ← temp;
  ⑧ return item;
end deleteHeap()
```


7. 힙의 개념과 연산 및 구현 : 삭제 연산

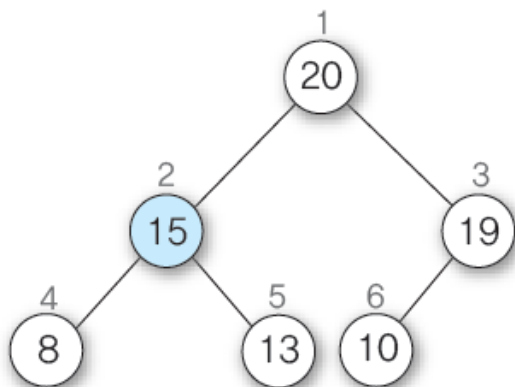
- ❶ 루트 노드 $\text{heap}[1]$ 을 변수 item 에 저장
- ❷ 마지막 노드의 원소 $\text{heap}[n]$ 을 변수 temp 에 임시로 저장
- ❸ 마지막 노드를 삭제하였으므로 힙 배열의 원소 개수가 하나 감소
- ❹ 마지막 노드의 원소였던 temp 의 임시 저장 위치 i 는 루트 노드의 자리인 1번이 됨
- ❺ 현재 저장 위치에서 왼쪽 자식 노드 $\text{heap}[j]$ 와 오른쪽 자식 노드 $\text{heap}[j+1]$ 이 있을 때, 키값이 큰 자식 노드의 키값과 temp 를 비교하여 temp 가 크거나 같으면 현재의 임시 저장 위치를 temp 자리로 확정하고,
❻을 수행
- ❻ temp 가 자식 노드 $\text{heap}[j]$ 보다 작으면 자식 노드와 자리를 바꾸고 ❺~❻을 반복하면서 temp 의 자리를 찾음
- ❼ 찾은 위치에 temp 를 저장하여 최대 힙의 재구성 작업을 완성
- ❽ 처음에 삭제된 루트 노드를 저장한 변수 item 을 반환, 삭제 연산을 종료



7. 힙의 개념과 연산 및 구현

❖ 힙의 구현

- 1차원 배열의 순차 자료구조를 이용하면 인덱스 관계를 이용하여 부모 노드를 찾기가 쉬움
- 힙을 순차 자료구조로 표현한 예



[0]		
[1]	20	부모 노드의 인덱스 = $\lfloor i/2 \rfloor = 2/2 = 1$
[2]	15	왼쪽 자식 노드의 인덱스 = $i \times 2 = 2 \times 2 = 4$
[3]	19	
[4]	8	
[5]	13	오른쪽 자식 노드의 인덱스 = $i \times 2 + 1 = 2 \times 2 + 1 = 5$
[6]	10	

그림 7-59 힙을 순차 자료구조로 표현한 예

7. 힙의 개념과 연산 및 구현

- 최대 힙의 알고리즘을 구현한 프로그램
 - 공백 힙에 원소 다섯 개(10, 45, 19, 11, 96)를 삽입하여 최대 힙을 구성

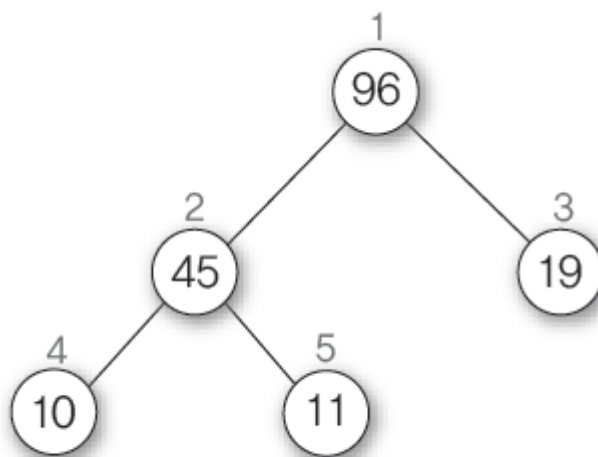
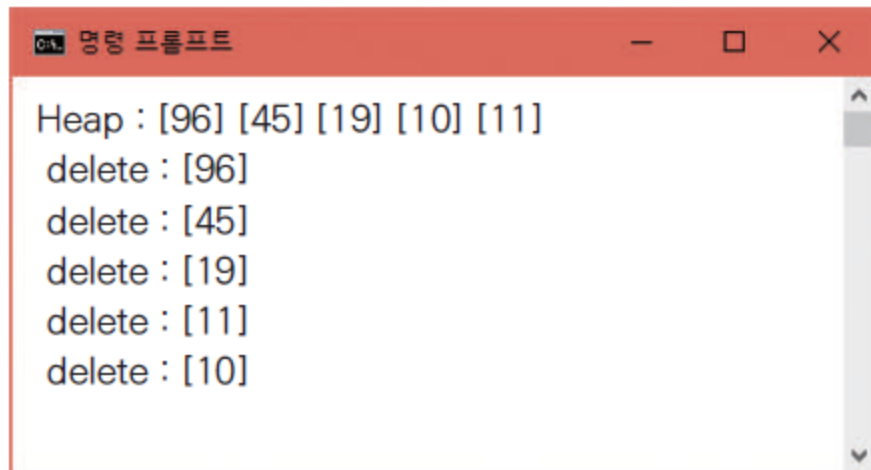


그림 7-60 원소 10, 45, 19, 11, 96을 삽입하여 완성한 최대 힙



7. 힙의 개념과 연산 및 구현

- 순차 자료구조를 이용해 최대 힙 구현하기 프로그램 : [교재 384p](#)
- 실행 결과



```
명령 프롬프트
Heap : [96] [45] [19] [10] [11]
delete : [96]
delete : [45]
delete : [19]
delete : [11]
delete : [10]
```



7. 힙의 개념과 연산 및 구현

- 18~28행의 insertHeap()은 힙에 item값을 삽입하는 연산을 수행
- 30~51행의 deleteHeap()은 힙의 루트 노드를 삭제하고 나머지 노드들을 힙으로 재구성한 후에 삭제한 루트노드를 반환하는 연산을 수행
- 57~58행은 1차원 배열에 저장된 힙을 인덱스 1번인 루트부터 순서대로 출력
- 63~68행은 공백 힙을 생성한 후 원소를 하나씩 삽입
 - 64행 : 공백 힙에 원소 10을 삽입. 원소 10은 힙의 루트 노드가 됨

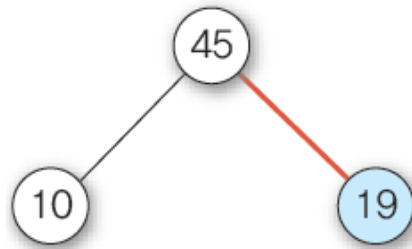


- 65행 : 힙에 원소 45를 삽입. 2번 노드를 확장하고 삽입. 삽입 노드 45가 부모 노드 10보다 크기가 크므로 부모 노드와 자리를 맞바꿈



7. 힙의 개념과 연산 및 구현

- 66행 : 힙에 원소 19를 삽입. 3번 노드를 확장하고 삽입한 후에 부모 노드와 크기를 비교. 삽입 노드 19는 부모 노드 45보다 작으므로 현재 위치를 확정.

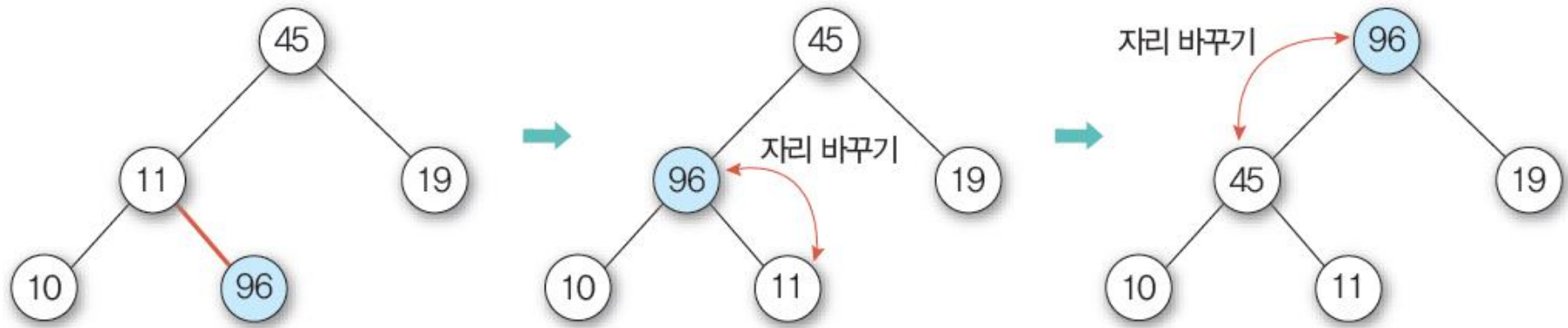


- 67행 : 힙에 원소 11을 삽입. 4번 노드를 확장하고 삽입한 후에 부모 노드와 크기를 비교. 삽입노드 11은 부모 노드 10보다 크므로 부모 노드와 자리를 맞바꿈. 다시 현재 위치에서 부모 노드와 크기를 비교. 삽입 노드 11은 부모 노드 45보다 작으므로 현재 위치를 확정



7. 힙의 개념과 연산 및 구현

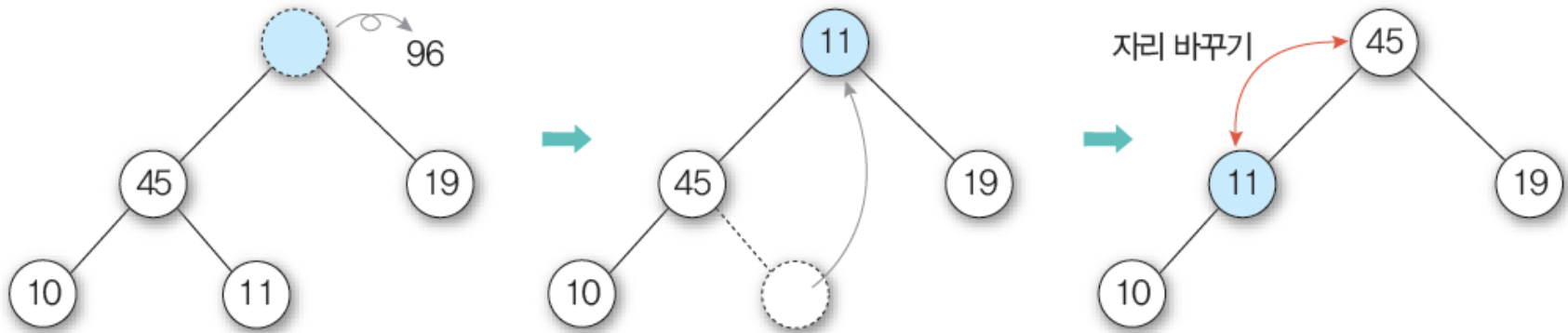
- 68행 : 힙에 원소 96을 삽입. 5번 노드를 확장하고 삽입한 후에 부모 노드와 크기를 비교. 삽입 노드 96은 부모 노드 11보다 크므로 부모 노드와 자리를 맞바꿈. 새로운 현재 위치에서 다시 부모노드와 크기를 비교. 삽입 노드 96은 현재의 부모 노드 45보다 크므로 부모 노드와 자리를 맞바꿈. 새로운 현재 위치가 루트여서 더 이상 비교할 부모 노드가 없으므로 현재 위치를 확정



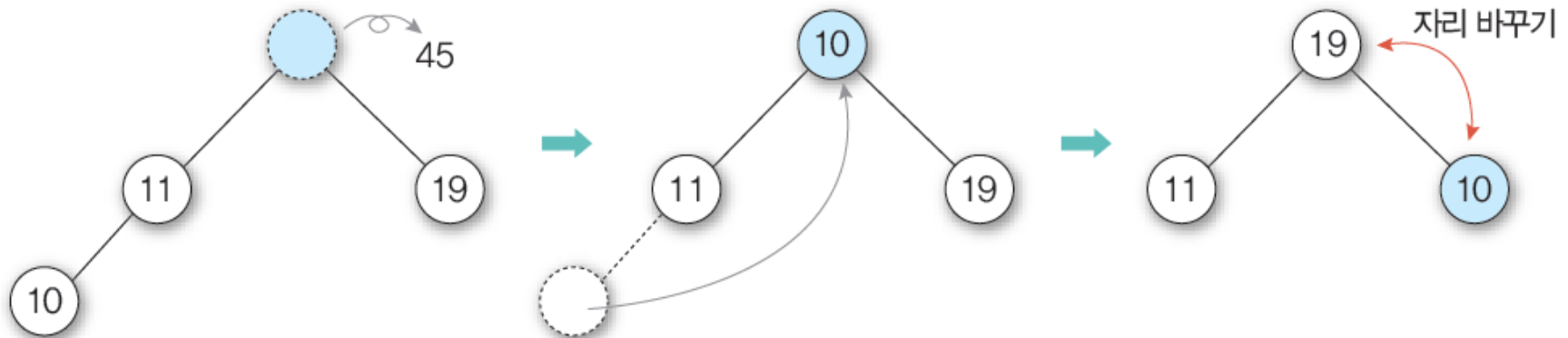
- 73~76행은 힙의 원소 개수만큼 삭제 연산을 반복 수행하면서 삭제된 원소를 출력
 - $i = 1$ 일 때 : 루트 노드를 삭제하고 나머지 원소들에 대해 힙을 재구성한 후에, 삭제된 루트 노드의 원소 96을 출력. 루트 노드의 원소를 삭제하여 원소가 네 개로 줄었으므로, 5번 노드의 자리를 삭제하고 5번 자리에 있던 원소 11은 루트 노드 자리에 임시로 저장



7. 히프의 개념과 연산 및 구현

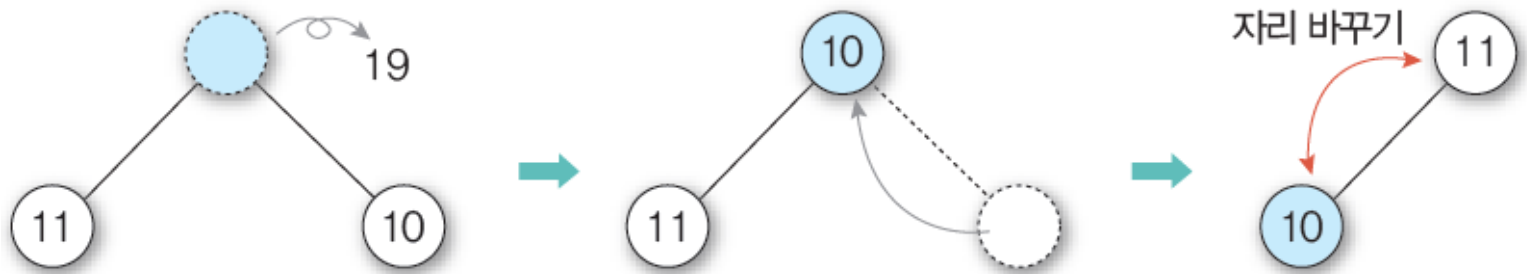


- i = 2일 때 : 루트 노드를 삭제하고 나머지 원소들에 대해 힙트를 재구성한 후에, 삭제된 루트 노드의 원소 45를 출력. 루트 노드의 원소를 삭제하여 원소가 세 개로 줄었으므로, 4번 노드의 자리를 삭제하고 4번 자리에 있던 원소 10은 루트 노드 자리에 임시로 저장



7. 힙의 개념과 연산 및 구현

- $i = 3$ 일 때 : 루트 노드를 삭제하고 나머지 원소들에 대해 힙을 재구성한 후에, 삭제된 루트 노드의 원소 19를 출력. 루트 노드의 원소를 삭제하여 원소가 두 개로 줄었으므로, 3번 노드의 자리를 삭제하고 3번 자리에 있던 원소 10은 루트 노드 자리에 임시로 저장



- $i = 4$ 일 때 : 루트 노드를 삭제하고 나머지 원소들에 대해 힙을 재구성한 후에, 삭제된 루트 노드의 원소 11을 출력. 루트 노드의 원소를 삭제하여 원소가 한 개로 줄었으므로, 2번 노드의 자리를 삭제하고 2번 자리에 있던 원소 10은 루트 노드 자리에 임시로 저장. 원소 10의 현재 위치가 단말 노드로 비교할 자식 노드가 없으므로 현재 위치를 확정



7. 힙의 개념과 연산 및 구현

- $i = 5$ 일 때 : 루트 노드를 삭제하고 삭제된 루트 노드의 원소 10을 출력. 원소가 다섯 개인 힙에서 삭제 연산을 다섯 번 수행하였으므로 힙이 공백이 됨.
for 문의 반복 연산을 종료



Thank You

