MYONGJI UNIVERSITY

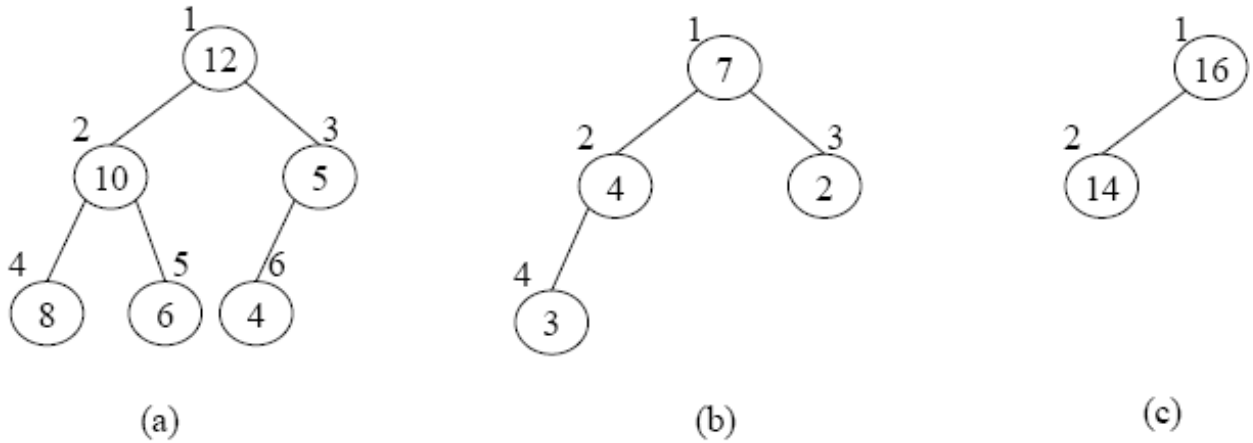
Heap

Dr. Seung Chul Han
Dept. Computer Engineering
Myongji University

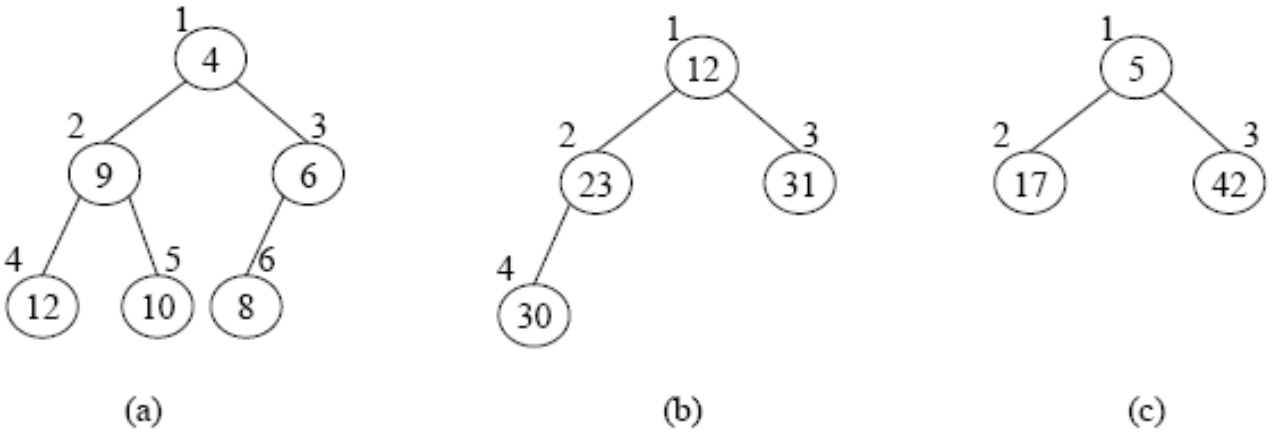
Heap

- 완전 이진트리

  - 특성
    - 최대 히프(max heap): 각 노드의 키값이 그 자식의 키값보다 작지 않은 완전 이진 트리
    - 최소 히프(min heap): 각 노드의 키값이 그 자식의 키값보다 크지 않은 완전 이진 트리



- 최소 히프 예

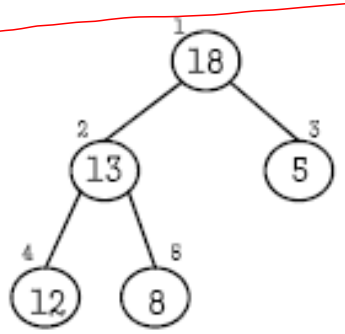


- 최소 히프의 루트는 그 트리에서 가장 작은 키값 가짐
- 최대 히프의 루트는 그 트리에서 가장 큰 키값 가짐

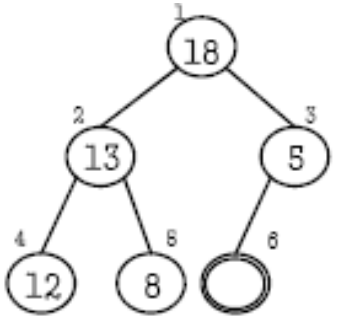
Functions

- ① 생성(create) : 공백 히프의 생성
- ② 삽입(insert) : 새로운 원소를 히프의 적절한 위치에 삽입
- ③ 삭제(delete) : 히프로부터 키값이 가장 큰 원소를 삭제하고 반환

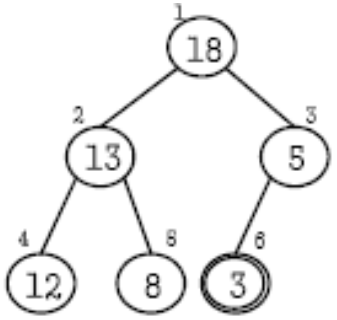
- 히프에서의 삽입
  - 삽입 예
    - 5개의 원소로 된 히프의 예(그림 (a))
    - 이중 원으로 표현한 노드: 새로 확장될 노드의 위치 (그림 (b))
    - 키값 3 삽입시: 바로 완료 (그림 (c))
    - 키값 8 삽입시: 부모 노드와 교환후 완료 (그림 (d), (e))
    - 키값 19 삽입시: 부모 노드와 루트와 연속 교환후 완료 (그림 (f), (g), (h))



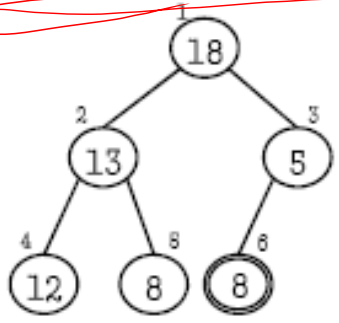
(a) 히프



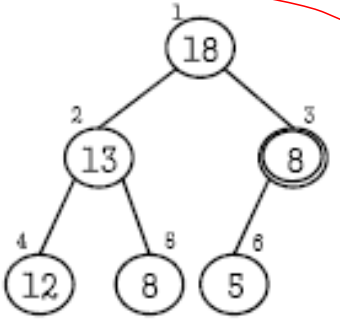
(b) 삽입후의 히프구조



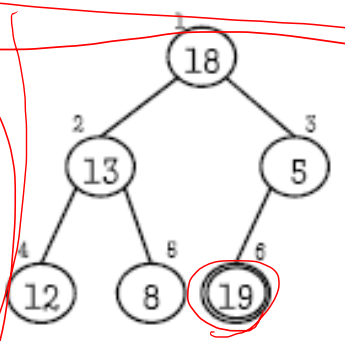
(c) 원소 3을 삽입



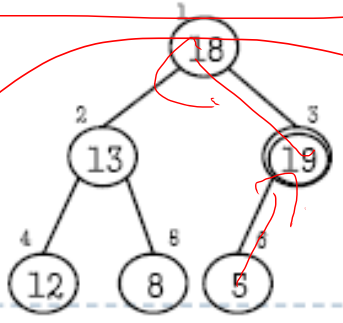
(d) 히프 (a)에 원소 8 삽입



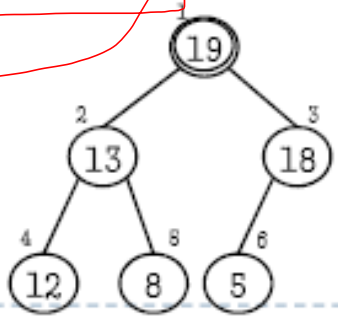
(e) 원소 8을 삽입 후



(f) 히프 (a)에 원소 19삽입



(g)원소의 이동



(h) 원소 19 삽입후

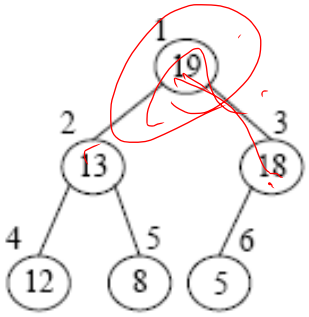
부모노드와 비교

- 부모 노드 위치 결정 가정
  - 연결 표현 사용시: 각 노드에 parent 필드 추가
  - 순차 표현 사용시: 위치 i의 부모노드는 ⌊ i / 2 ⌋

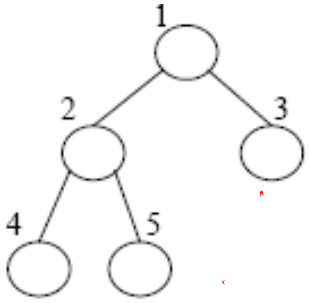
```
insertHeap(Heap,item)
        // 순차 표현으로 구현된 최대 히프
        // 원소 e를 히프 Heap에 삽입, n은 현재 히프의 크기(원소 수)
  if (n = heapSize) then heapFull;  // 히프가 만원이면 히프 크기를
확장
  n ← n+1;  // 새로 첨가될 노드 위치
  for (i←n; ; ) do {
     if (i = 1) then exit;  // 루트에 삽입
     if(item ≤ heap[⌊i/2⌋]) then exit;  // 삽입할 노드의 키값과
                                        // 부모 노드 키값을 비교
     heap[i] ← heap[⌊i/2⌋];  // 부모 노드 키값을 자식노드로 이동
     i ← ⌊i/2⌋;
  }
  heap[i] ← item;
end insertHeap()
```


- 히프에서의 삭제
  - 루트 원소를 삭제
  - 나머지 트리가 다시 히프가 되도록 재조정

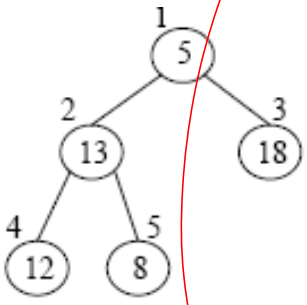
- 첫 번째 삭제 예 (루트 키값 19)
  - 히프의 예 (그림 (a))
  - 루트 원소 19의 삭제후 구조 (그림 (b))
  - 위치 6의 노드를 삭제하고 키값 5를 루트로 이동 (그림 (c))
  - 이 이동 원소값을 두 자식 노드 중에서 큰 원소값과 비교
    - 비교 결과 자식 노드 원소값이 크면 그 값을 부모 노드로 옮긴 다음 이동 원소가 그 자식 노드로 내려가서 다시 히프 검사
    - 만일 자식 노드 원소보다 크면 삭제 연산 종료
    - 이 예의 경우, 5와 18 교환후 종료 (그림 (d))
- 두번째 삭제 예 (루트 키값 18)
  - 삭제(루트 18)후 구조 (그림 (e))
  - 위치 5의 노드를 삭제하고 키값 8을 루트로 이동 (그림 (f))
  - 이 이동 원소값을 두 자식 노드 중에서 큰 원소값과 비교
    - 이 예의 경우, 8과 13, 다시 8과 12 교환후 종료 (그림 (g), (h))



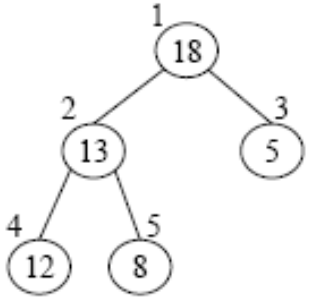
(a) 히프



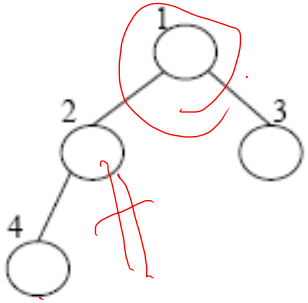
(b) 삭제 후의 히프 구조



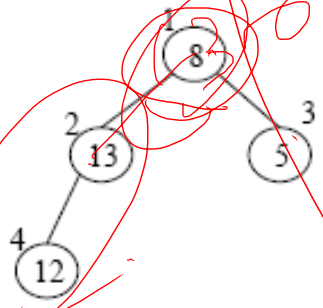
(c) 삭제 중간 단계



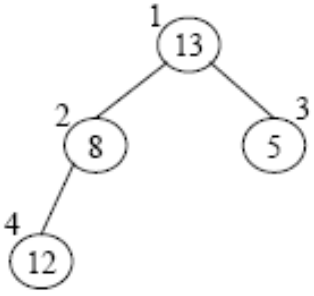
(d) 첫번째 삭제 뒤의 히프



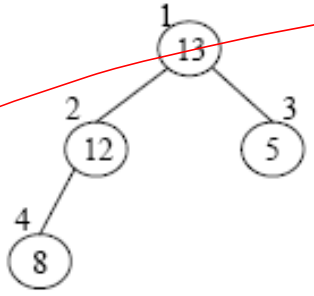
(e) 두번째 삭제 뒤의 히프 구조



(f) 삭제 중간 단계



(g) 삭제 중간 단계



(h) 두번째 삭제가 완료된 히프

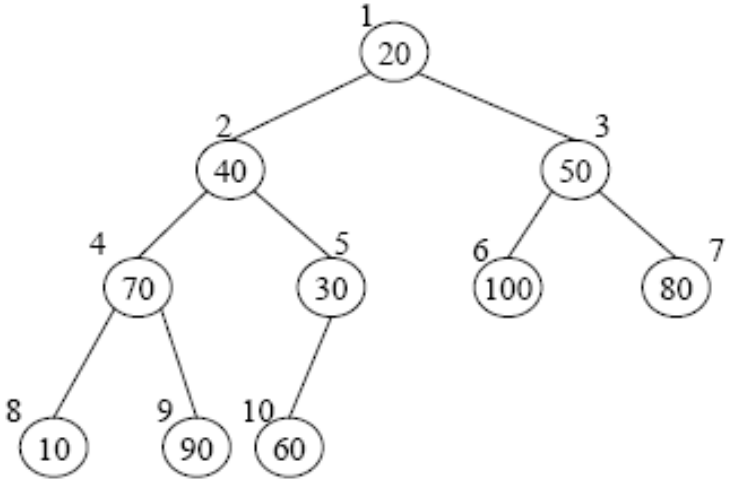
```
deleteHeap(heap)
        // 히프로부터 원소 삭제, n은 현재의 히프 크기(원소 수)
    if (n=0) then return error;  // 공백 히프
    item ← heap[1];    // 삭제할 원소
    temp ← heap[n];    // 이동시킬 원소
    n ← n-1;  // 히프 크기(원소 수)를 하나 감소
    i ← 1;
    j ← 2;      // j는 i의 왼쪽 자식 노드
    while (j ≤ n) do {
        if (j < n) then if (heap[j] < heap[j+1])
            then j ← j+1;      // j는 값이 큰 자식을 가리킨다.
        if (temp ≥ heap[j]) then exit;
        heap[i] ← heap[j]; // 자식을 한 레벨 위로 이동
        i ← j;
        j ← j*2; // i와 j를 한 레벨 아래로 이동
    }
    heap[i] ← temp;
    return item;
end deleteHeap()
```

Binary Tree -> Heap

- 초기에 히프로 되어 있지 않은 완전이원트리 H를 히프로 변환
  - 역레벨순서로 H의 내부 노드 각각을 루트로 하는 서브트리를 차례로 히프로 만들어 나가면 됨
- 완전 이원트리를 히프로 변환하는 알고리즘

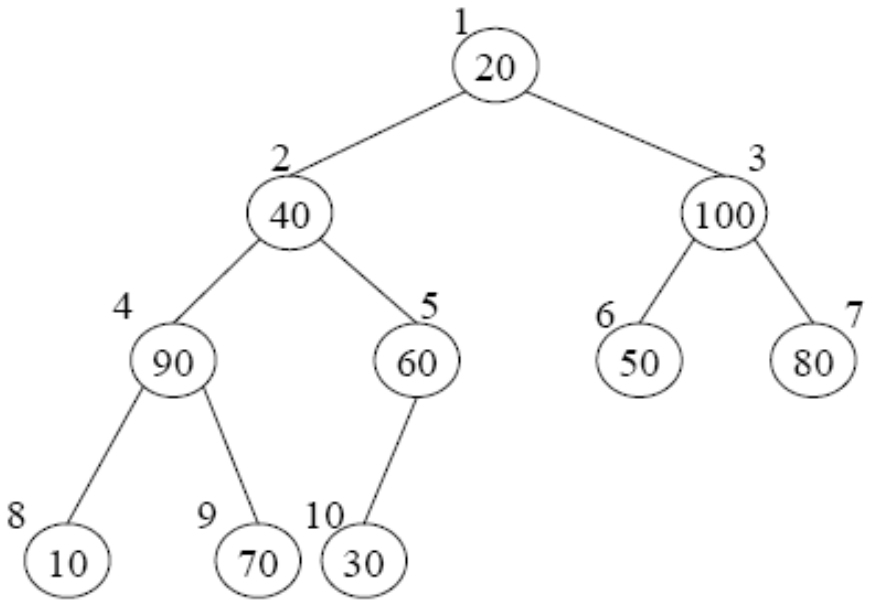
```
makeTreeHeap(H, n)  // H는 히프가 아닌 완전 이진 트리
  for (i ← n/2; i ≥ 1; i ← i-1) do {
  // 각 내부 노드에 대해 레벨 순서의 역으로
     p ← i; t ← H[i]
     for (j ← 2*p; j ≤ n; j ← 2*j) do {
        if (j < n) then
           if (H[j] < H[j+1]) then j ← j+1;
        if (H[p] ≥ H[j]) exit;
        H[p] ← H[j];
        p ← j;  // 부모 노드를 한 레벨 밑으로 이동
     }
     H[p] ← t;
  }
end makeTreeHeap()
```

- 히프 7

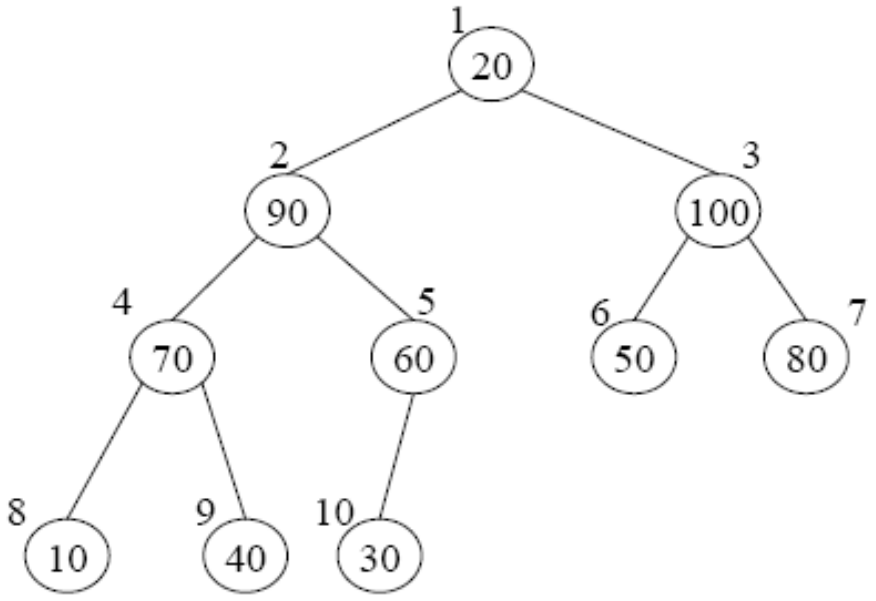


- 내부 노드의 역 레벨 순서: 5, 4, 3, 2, 1
- 5번 노드를 루트로 하는 서브 트리에서 히프 연산 시작
  - 이 노드는 루트보다 키값(60)이 큰 자식을 가지므로 교환(30↔60)
- 다음으로 4번 노드를 루트로 하는 서브 트리 조사
  - 자식 중에 큰 키값(90)을 루트의 키값과 교환(70 ↔ 90)

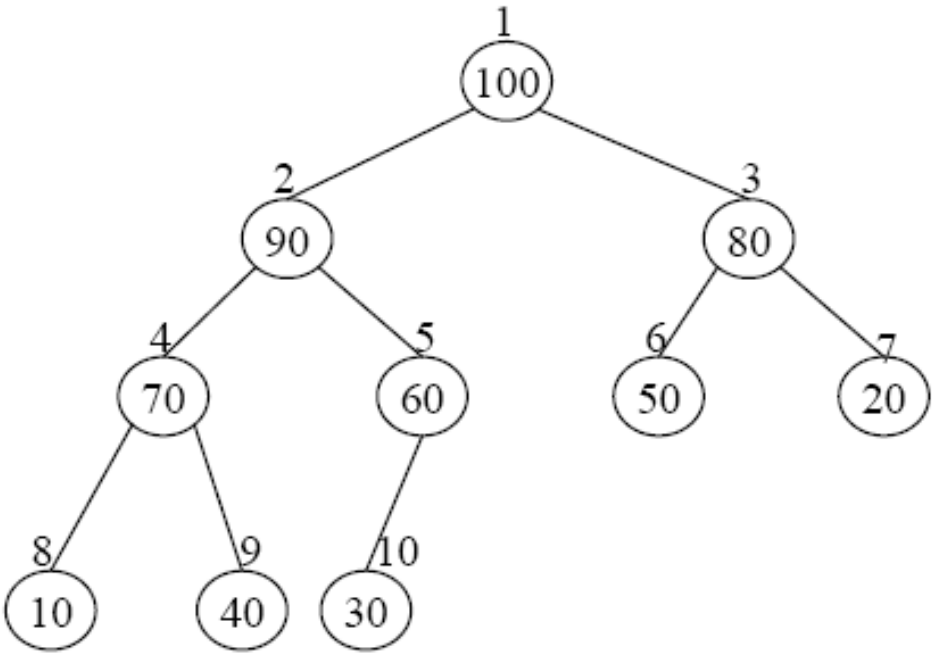
- 다음으로 3번 노드를 루트로 하는 서브 트리 조사
  - 노드6과 노드 3과의 큰 키값 교환(50 ↔ 100)
  - 여기까지의 과정에서 얻어진 트리



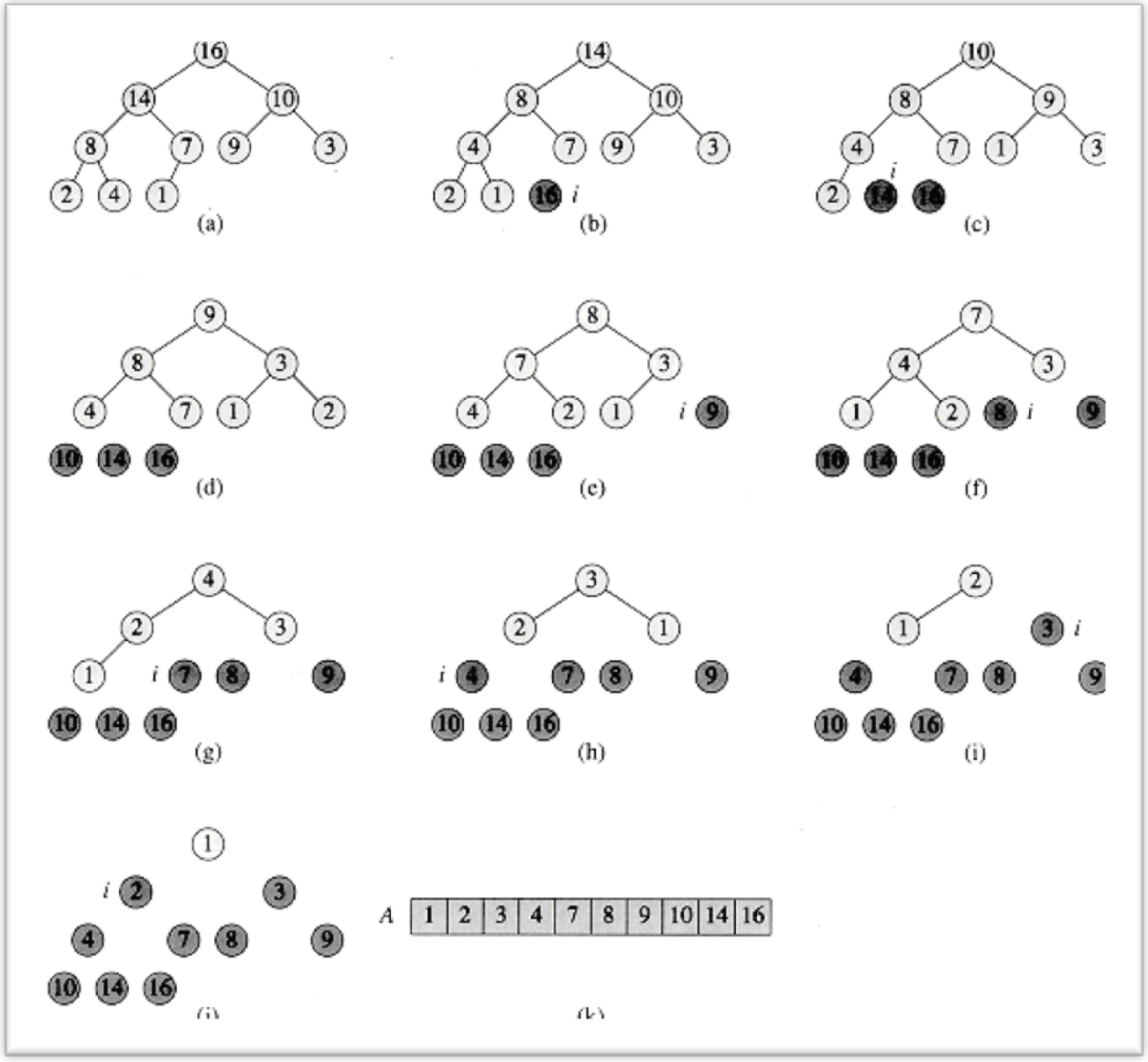
- 다음으로 2번 노드를 루트로 하는 서브트리 조사
  - 키값 40을 왼쪽 자식의 키값(90)과 교환(40 ↔ 90)
  - 다시 계속해서 9번 노드의 키값(70)과 교환(40 ↔ 70)
  - 이 결과로 얻어진 이진 트리



- 끝으로 역레벨순서 마지막 노드인 루트 노드(1번 노드)를 고려
  - 이 루트 노드는 먼저 3번 노드의 키값(100)과 교환(20 ↔ 100)
  - 다시 계속해서 7번 노드의 키값(80)과 교환(20 ↔ 80)
  - 초



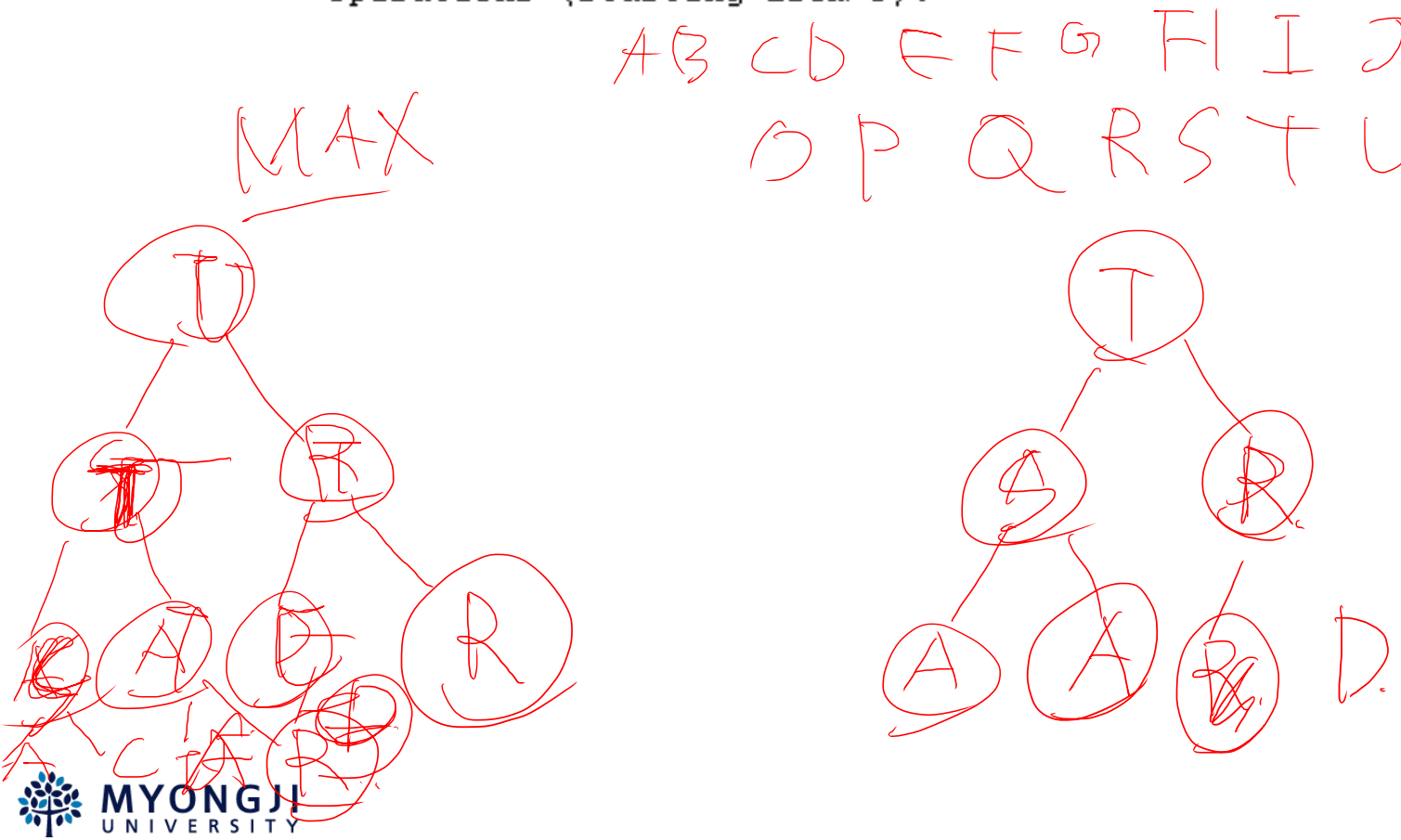
Heap Sort



Quiz. people.eecs.berkeley.edu/~jrs/61b/

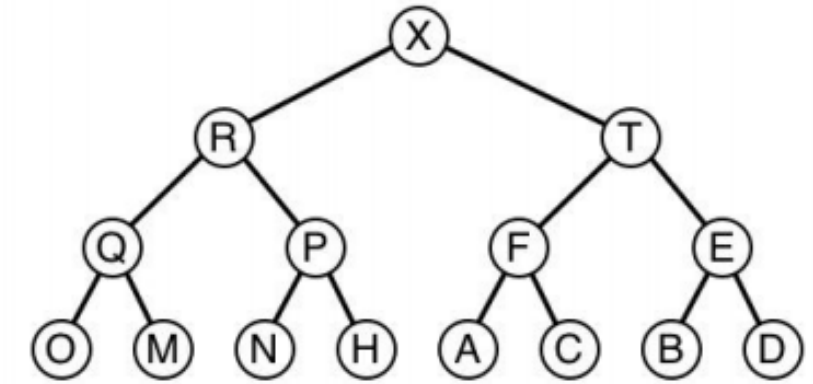
[10] Suppose we wish to create a binary heap containing the keys D A T A S T R U C T U R E. (All comparisons use alphabetical order.)

[a] Show the resulting min-heap if we build it using successive insert() operations (starting from D).



Quiz. www.cs.princeton.edu/courses/archive

6. **Heap operations** (10 points). Consider the following max-heap:



a. (5 points) Draw the result of inserting S.

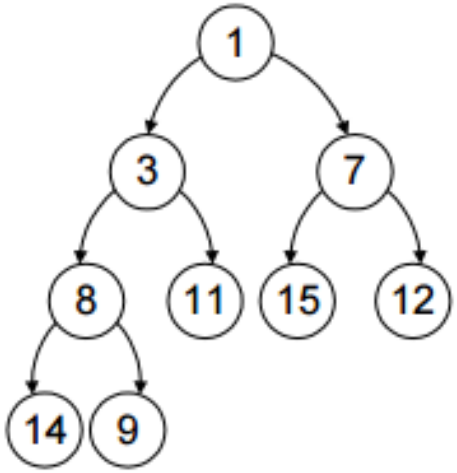
Quiz. www.ics.uci.edu/~eppstein/161

1. (15 points) Let $H = [24, 21, 18, 15, 12, 9, 6, 3]$ be an array of eight numbers, interpreted as a binary heap with the maximum value at its root (the first element of the array). What would be the contents of H after performing a delete-max operation on this heap?

Quiz. courses.cs.washington.edu/courses/cse373/

6. **(20 pts) Heaps**

   a. (8 pts) Draw the binary min heap that results from inserting 11, 9, 12, 14, 3, 15, 7, 8, 1 in that order into an **initially empty binary heap**. You do not need to show the array representation of the heap. You are only required to show the final tree, although if you draw intermediate trees, ***please circle your final result for ANY credit.***



6. (cont.) **Heaps:**

b. (4 pts) Draw the binary heap that results from doing 2 deletemins on the heap you created in part a.. You are only required to show the final tree, although if you draw intermediate trees, ***please circle your final result for ANY credit.***