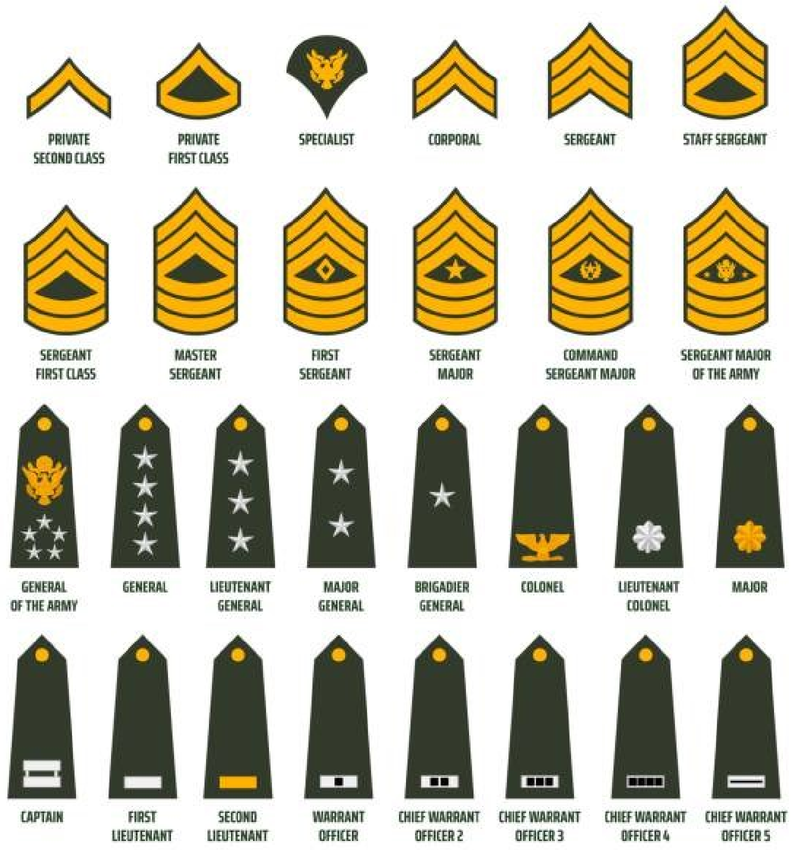
Data Structure

Heap

Shin Hong

28 Apr 2023



DS&A. Chapter 8.3 Heap

Priority Queue

- A priority queue is a collection of elements, that provides insertion and removal of elements in order of priority
  - operations
    - `insert (elem, key)`
    - `min()`
    - `removeMin ()`

- A priority queue manages elements according to their priorities, not their positions or the order of their arrivals
  - e.g., Suppose a certain flight is fully booked an hour prior to departure. Because of the possibility of cancellations, the airline maintains a priority queue of standby passengers hoping to get a seat. The priority of each passenger is determined by the fare paid and the frequent-flyer status.

Key and Comparator

- Each element is assigned with a key which defines the ranking (or ordering)
  - The ordering of keys must be totally defined without any contradiction (i.e., total ordering)
    - Reflexive property: $k \leq k$
    - Antisymmetric property: if $k_1 \leq k_2$ and $k_2 \leq k_1$, then $k_1 = k_2$
    - Transitive property: if $k_1 \leq k_2$ and $k_2 \leq k_3$, then $k_1 \leq k_3$

- A comparator is a function that receives two key objects and determines the ordering in them
  - e.g., a geometric algorithm may compare points p and q in 2D space, by their x-coordinate (that is, p ≤ q if p.x ≤ q.x), to sort them from left to right,
  - e.g., another algorithm may compare them by their y-coordinate (that is, p ≤ q if p.y ≤ q.y), to sort them from bottom to top.

Priority Queue with Lists

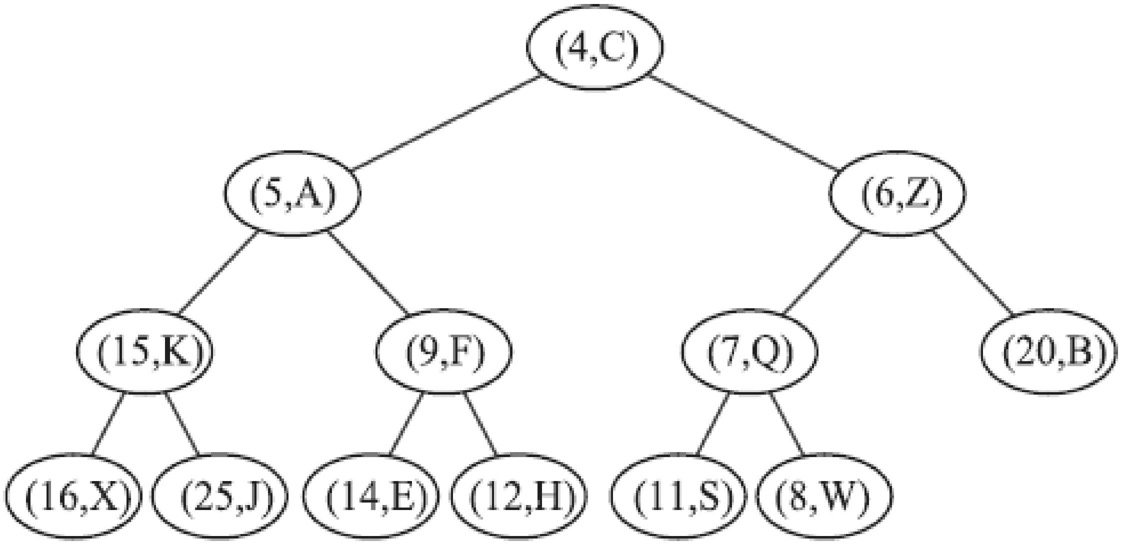
- Implementation with an unsorted list
  - insertion: O(1)
  - removeMin: O(n)

- Implementation with a sorted list
  - insertion: O(n)
  - removeMin: O(1)

Heap

- A heap is a complete binary tree such that the element at a node always precedes those of the children

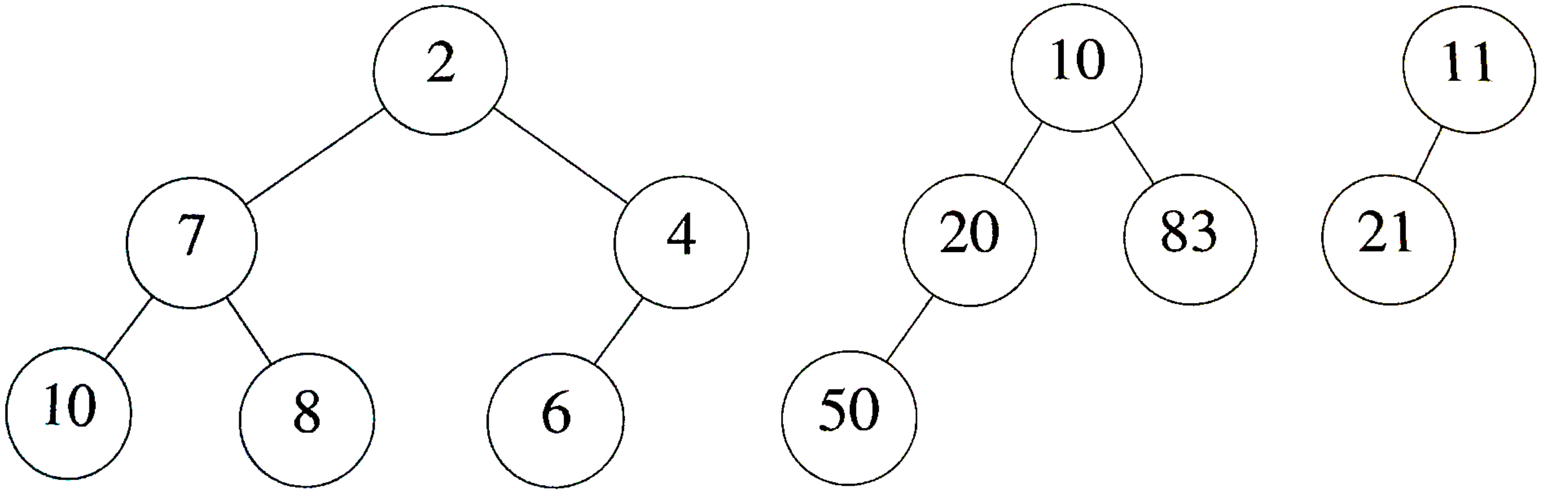
- A heap provides both insertion and removal in O(log *n*) which significantly improves list-based priority queue
  - the height of a complete binary tree is $\lfloor \log n \rfloor$



Heap-order Property

- In a min-heap T, for every node v other than the root, the key associated with v is greater than or equal to the key associate with v's parent.

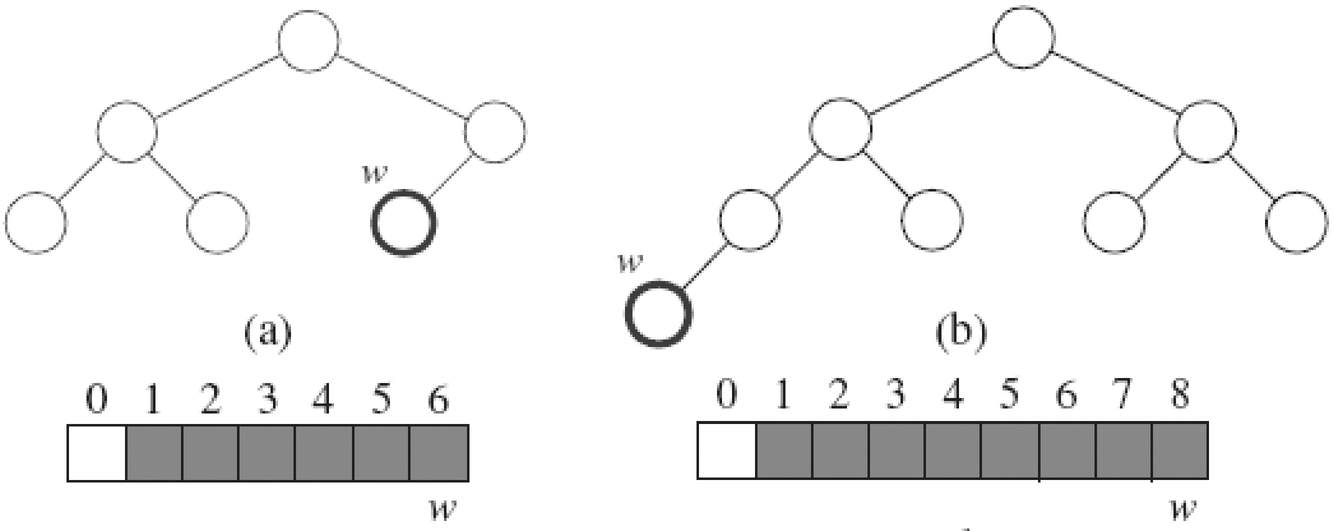
- By the heap-order property:
  - an element with the minimum key is always placed at the root,
  - the key encountered on a path from the root to a leaf node are in non-decreasing order



Complete Binary Tree

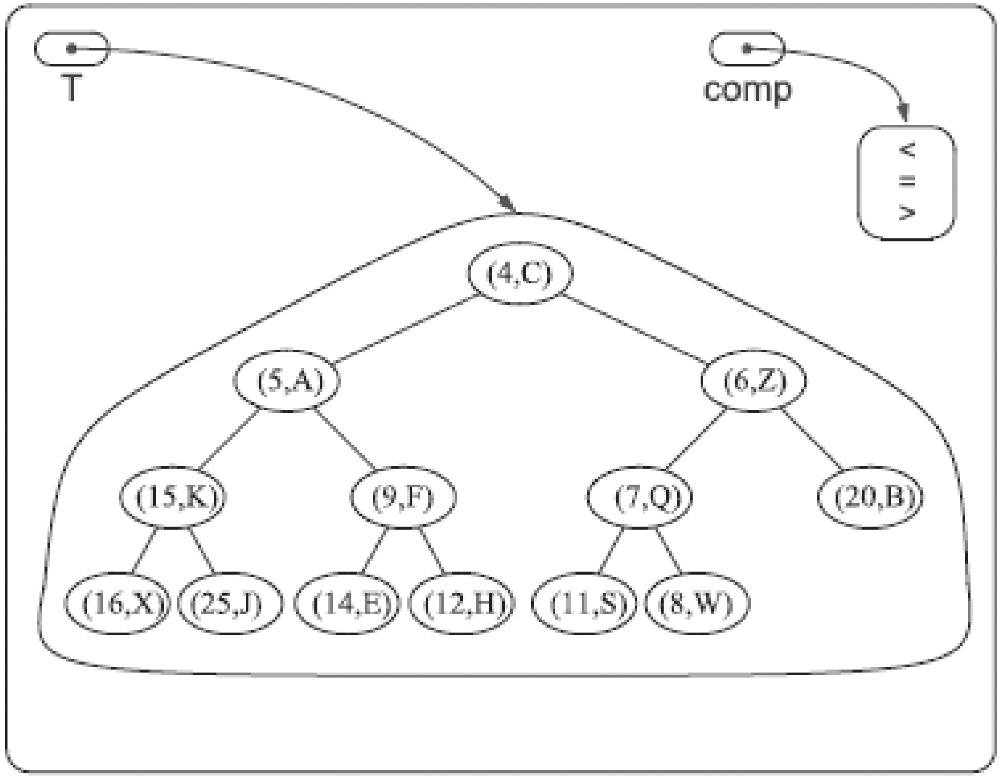
- an array (or vector) is especially suitable for representing a complete binary tree

- the level number of a node, $f(n)$ in a binary tree is defined as follows:
  - if n is the root, $f(n) = 1$
  - if n is the left child of node u, $f(n) = 2f(u)$
  - if n is the right child of node u, $f(n) = 2f(u) + 1$



Priority Queue with Heap

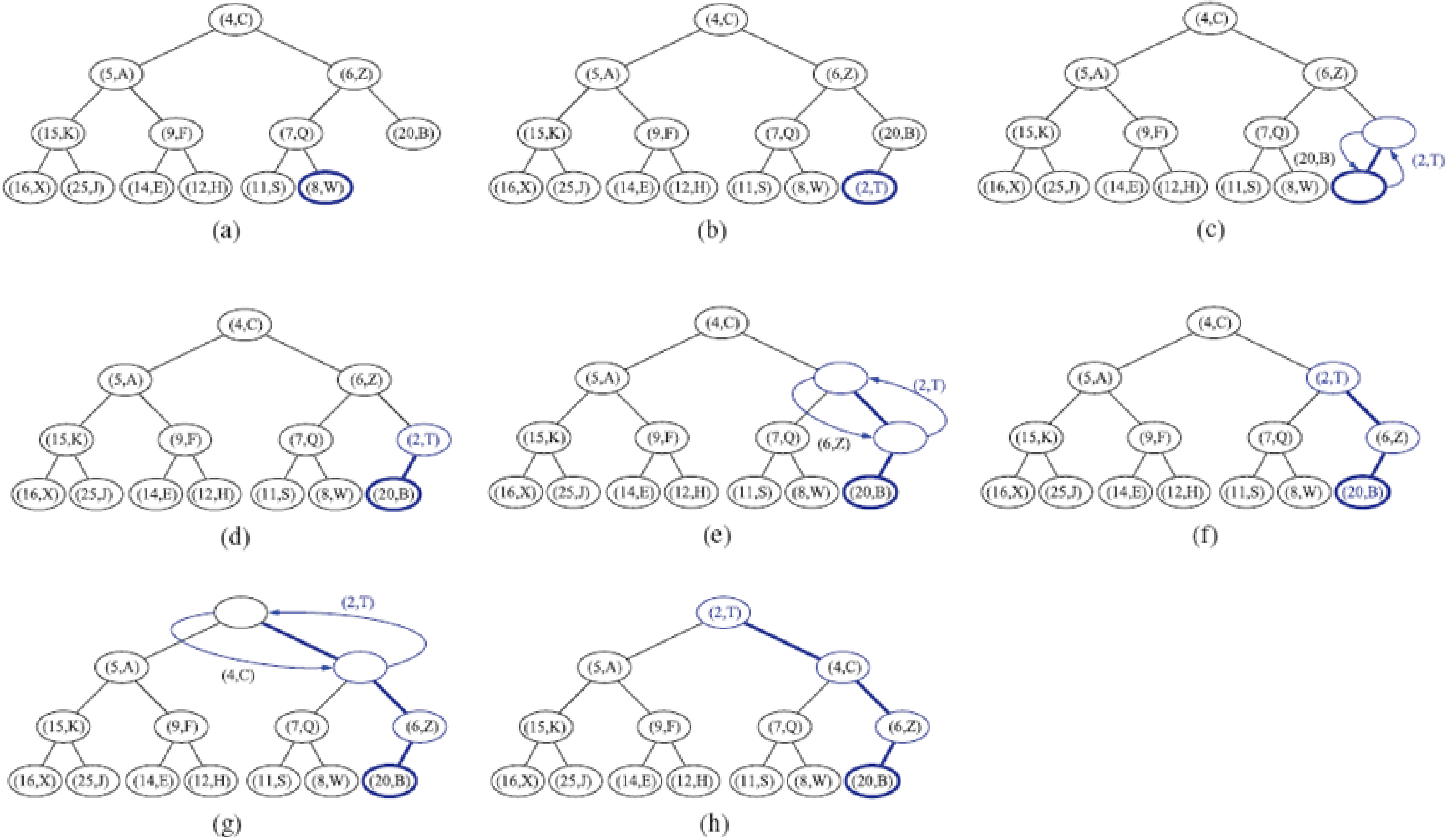
- a heap-based priority queue consists of:
  - heap: a complete binary tree whose nodes store the elements and whose keys satisfy the heap-order property
  - comp: a comparator that defines the total order relation among the keys



Insertion

1. add a new node n s.t. the new node becomes the last node
   - it keeps the tree complete
   - it may violate the heap-order property

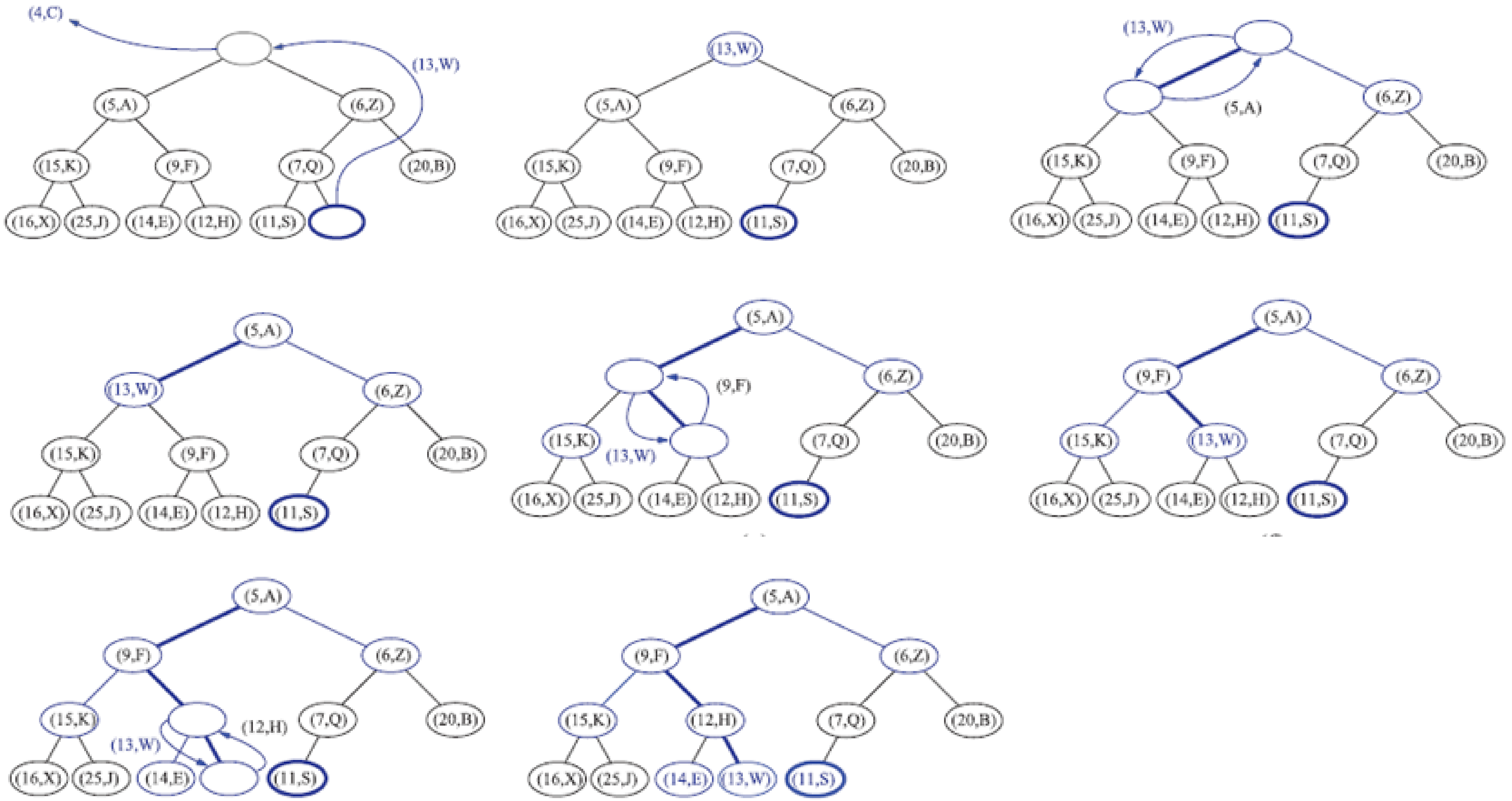
2. if the key of n is less than that of its parent, swap n and the parent
   - repeat this step until the key of the parent node is less than that of n or n becomes the root
     - the heap-order property will be satisfied again


(a)
(4,C) (5,A) (6,Z) (15,K) (9,F) (7,Q) (20,B) (16,X) (25,J) (14,E) (12,H) (11,S) (8,W)
(b)
(4,C) (5,A) (6,Z) (15,K) (9,F) (7,Q) (20,B) (16,X) (25,J) (14,E) (12,H) (11,S) (8,W) (2,T)
(c)
(4,C) (5,A) (6,Z) (15,K) (9,F) (7,Q) (20,B) (2,T) (16,X) (25,J) (14,E) (12,H) (11,S) (8,W)
(d)
(4,C) (5,A) (6,Z) (15,K) (9,F) (7,Q) (2,T) (16,X) (25,J) (14,E) (12,H) (11,S) (8,W) (20,B)
(e)
(4,C) (5,A) (2,T) (15,K) (9,F) (7,Q) (6,Z) (16,X) (25,J) (14,E) (12,H) (11,S) (8,W) (20,B)
(f)
(4,C) (5,A) (2,T) (15,K) (9,F) (7,Q) (6,Z) (16,X) (25,J) (14,E) (12,H) (11,S) (8,W) (20,B)
(g)
(2,T) (5,A) (4,C) (15,K) (9,F) (7,Q) (6,Z) (16,X) (25,J) (14,E) (12,H) (11,S) (8,W) (20,B)
(h)
(2,T) (5,A) (4,C) (15,K) (9,F) (7,Q) (6,Z) (16,X) (25,J) (14,E) (12,H) (11,S) (8,W) (20,B)

Removal

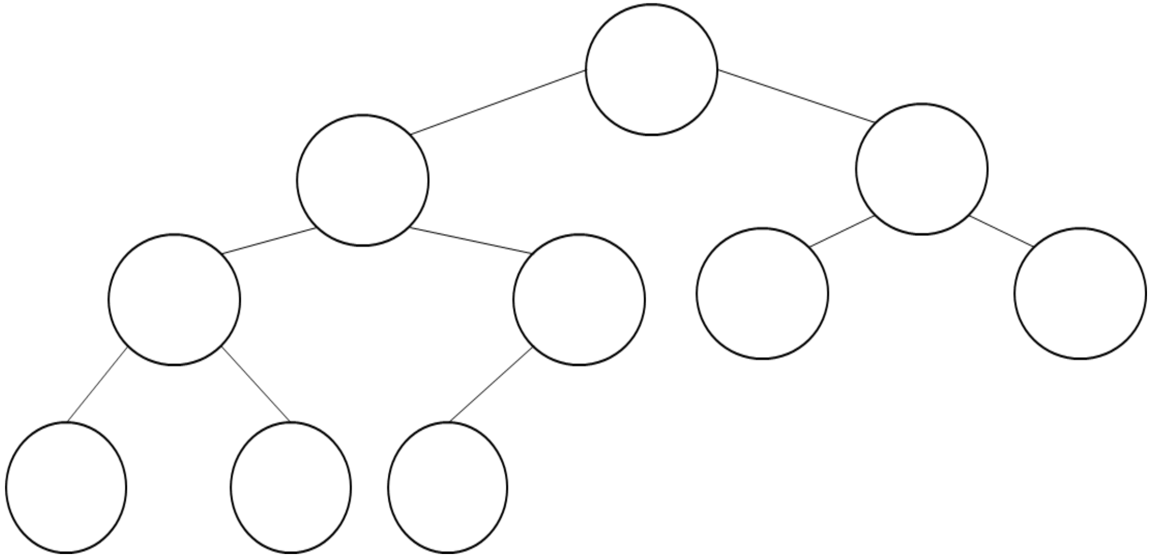
- An element with the minimal key is always found at the root

- After removal, the root must be replaced with another while keeping the tree complete and keeping the heap-order property
  1. move the last node n to the root
     - keep the tree complete
     - the heap-order property may be violated
  2. swap n and a child with the least key if the key of n is greater than that of one or two children
     - repeat this step until the tree restores the heap-order property
     - this process is called as heapify


(4,C)
(13,W)
(5,A)
(6,Z)
(15,K)
(9,F)
(7,Q)
(20,B)
(16,X)
(25,J)
(14,E)
(12,H)
(11,S)
(13,W)
(5,A)
(6,Z)
(15,K)
(9,F)
(7,Q)
(20,B)
(16,X)
(25,J)
(14,E)
(12,H)
(11,S)
(13,W)
(5,A)
(6,Z)
(15,K)
(9,F)
(7,Q)
(20,B)
(16,X)
(25,J)
(14,E)
(12,H)
(11,S)
(5,A)
(13,W)
(6,Z)
(15,K)
(9,F)
(7,Q)
(20,B)
(16,X)
(25,J)
(14,E)
(12,H)
(11,S)
(5,A)
(9,F)
(6,Z)
(15,K)
(13,W)
(7,Q)
(20,B)
(16,X)
(25,J)
(14,E)
(12,H)
(11,S)
(5,A)
(9,F)
(6,Z)
(15,K)
(13,W)
(7,Q)
(20,B)
(16,X)
(25,J)
(14,E)
(12,H)
(11,S)
(5,A)
(9,F)
(6,Z)
(15,K)
(12,H)
(13,W)
(7,Q)
(20,B)
(16,X)
(25,J)
(14,E)
(11,S)
(5,A)
(9,F)
(6,Z)
(15,K)
(12,H)
(7,Q)
(20,B)
(16,X)
(25,J)
(14,E)
(13,W)
(11,S)

Heap Sort

```
heapsort(elem * a, int n) {
   for (i = 2 ; i <= n ; i++)
      insert(a, i) ;
   for (i = n - 1 ; i > 1 ; i--) {
      swap(&(a[1]),&(a[i+1])) ;
      heapify(a, n) ;
   }
}
```



ex. (26, 5, 77, 1, 61, 11, 59, 15, 48, 19)