priority queue & heaps

data structures
fall 2023

Priority Queue

- queue: FIFO
  - a company call center
  - a new call is added to back of queue
  - calls are taken from the front of the queue
- Sometimes, FIFO is not enough.
  - Air-traffic control center
  - Can’t use FIFO as is, you need to add some factors in:
    - plane’s distance, time spent waiting, amount of remaining fuel
- There are other priorities.

Priority Queue

- Priority queue:
  - Collection of prioritized elements that allows arbitrary element insertion
  - Allows the removal of the element that has first priority
- When an element is added to priority queue, user designated its priority by providing a **key**.
  - The element with minimal key will be the next to be removed.
  - (If 1 is more important than 2)

Priority Queue ADT

- insert(k,v)
  - Creates an entry with key **k** and value **v**
- min()
  - Returns (does not remove) a priority queue entry (k,v) having minimal key.
  - If the queue is empty, returns null.
- removeMin()
  - Removes and *returns* an entry (k,v) having minimal key.
  - Returns null if empty.
- size()
  - Returns the number of entries in the queue
- isEmpty()
  - Returns a boolean indicating whether the queue is empty or not.

Priority Queue

- It is possible for a PQ to have multiple entries with the same key.
  - In that case, *min* and *removeMin* will choose arbitrarily among those.

Method	Return Value	Priority Queue Contents
insert(5,A)		{ (5,A) }
insert(9,C)		{ (5,A), (9,C) }
insert(3,B)		{ (3,B), (5,A), (9,C) }
min( )	(3,B)	{ (3,B), (5,A), (9,C) }
removeMin( )	(3,B)	{ (5,A), (9,C) }
insert(7,D)		{ (5,A), (7,D), (9,C) }
removeMin( )	(5,A)	{ (7,D), (9,C) }
removeMin( )	(7,D)	{ (9,C) }
removeMin( )	(9,C)	{ }
removeMin( )	null	{ }
isEmpty( )	true	{ }

Complexities

- For an **unordered** and **ordered** priority queue:

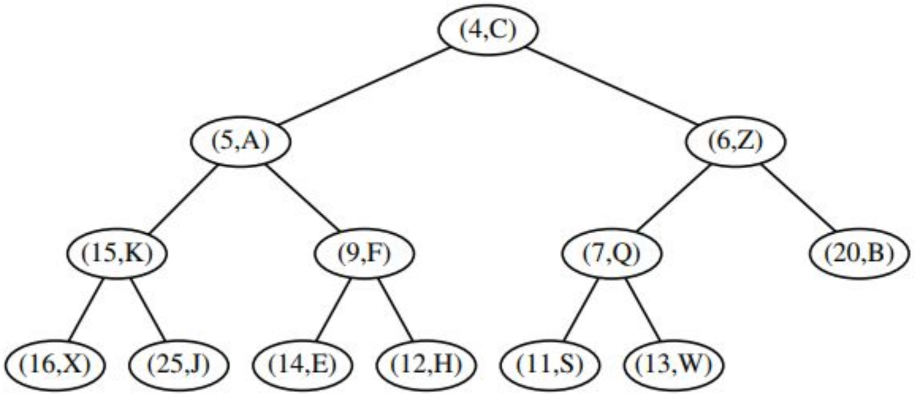
Method	Unsorted List	Sorted List
size	$O(1)$	$O(1)$
isEmpty	$O(1)$	$O(1)$
insert	$O(1)$	$O(n)$
min	$O(n)$	$O(1)$
removeMin	$O(n)$	$O(1)$

Heaps

- A way to implement a priority queue.
  - Binary heap
- Allows to perform insertions and removals in logarithmic time
  - An improvement over list-based implementations of priority queue
- It uses a binary tree structure to find a compromise between elements being entirely unsorted and perfectly sorted.
- Has nothing to do with the **memory heap**
  - A dynamically allocated memory that *processes* use to store data like arrays, strings, objects and dynamically allocated data structures.

Heap

- A binary tree **T** stores entries at its positions and satisfies two additional properties:
    - A relational property in terms of the way keys are stored in T
    - A structural property defined in terms of the shape of T.



Relational property

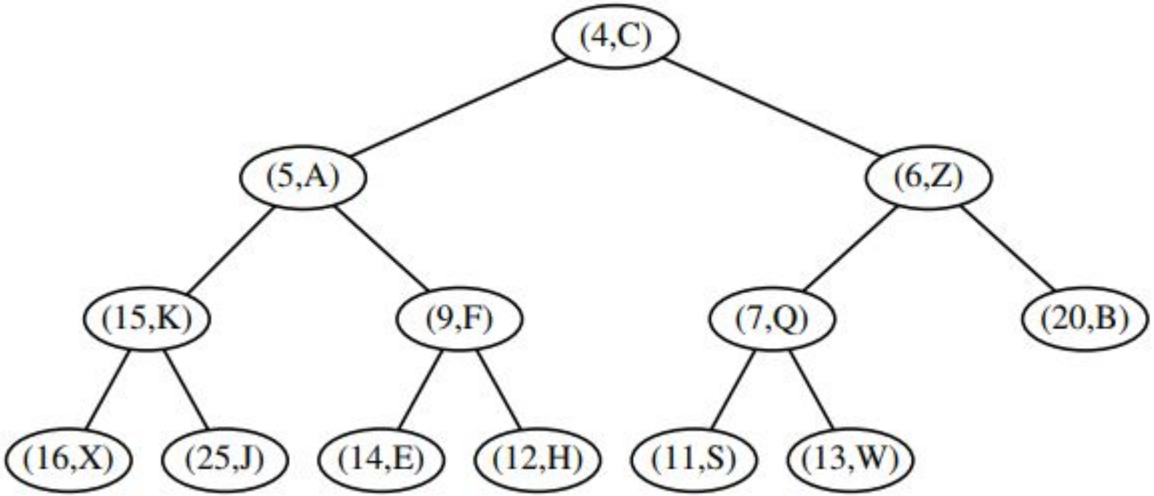
- Heap-order property
  - In a heap T, for every position *p* other than the root, the key stored at *p* is greater than or equal to the key stored at *p*’s parent.
- So, keys encountered on a path from the root to a leaf of T are in **nondecreasing** order.
  - Also, a minimal key is always stored at the root of T.
  - Makes it easy to locate such an entry when *min* or *removeMin* is called.
    - Top of the heap

Structural property

- For the sake of efficiency, we want the heap to have a small height as possible.
- This is enforced by the following requirement:
  - Heap T must be **complete**
- **Complete binary tree property**
  - A heap **T** with height *h* is a **complete** binary tree, if levels 0,1,2,...,h-1 of **T** have the maximal number of nodes possible and the remaining nodes at level *h* reside in the leftmost possible positions at that level.

Completeness

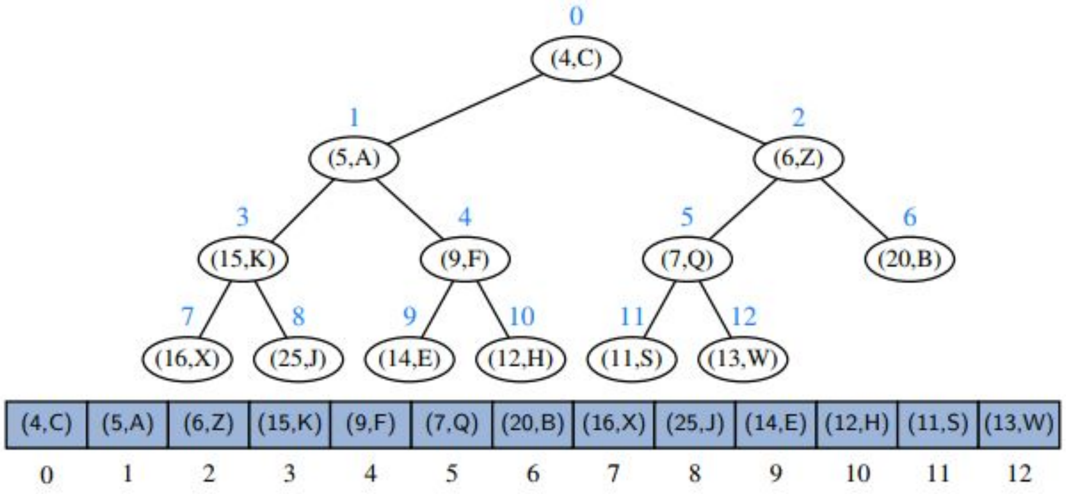
- This below is complete, because levels 0, 1, and 2 are full and six nodes in level 3 are in the **six leftmost possible** positions at that level.



Binary trees

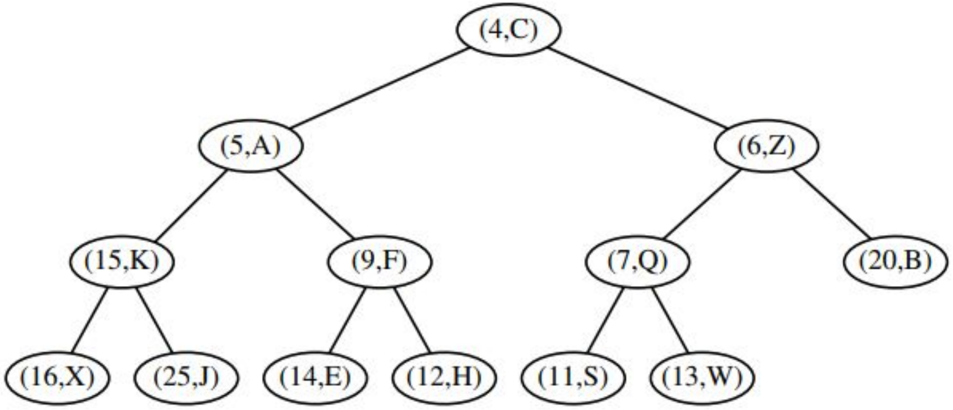
Array-based representation of a complete binary tree

- If *p* is the root, then $f(p)=0$
- If *p* is the left child of pos *q*, then $f(p)=2f(q)+1$
- If *p* is the right child of pos *q*, then $f(p)=2f(q)+2$



min heap & max heap

- The key present at the root node must be less than or equal among the keys present at all of its children.
  - Same property must be recursively true for all sub-trees.
- Inverse is true for **max** heap.



min heap & max heap

- Choosing between these two heaps depend on what you choose as most important and worst
  - For example if you say 0 is the most important (highest priority): use a **min-heap**
  - If you say 0 is the least important, use a **max-heap**.

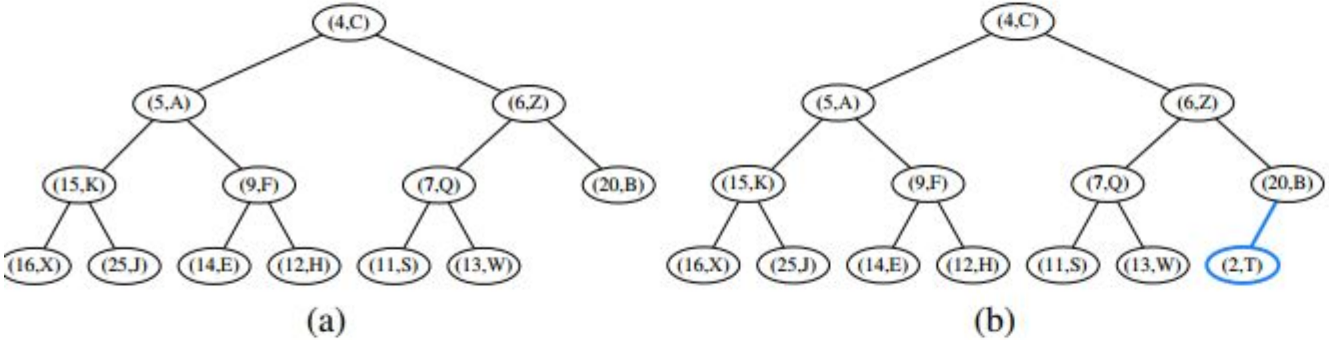
priority queue as a heap

- insert(k,v)
    - We store the pair as an entry at a new node of the tree.
    - We need to maintain the **complete binary tree property**
    - Therefore, new node should be placed at position *p* just beyond the rightmost node at the bottom level of the tree
        - Or as the leftmost position of a new level if the bottom level is already full (or the heap is empty)
    - After adding, the tree may be complete but it is not enough.
        - We also need to satisfy **heap-order property**

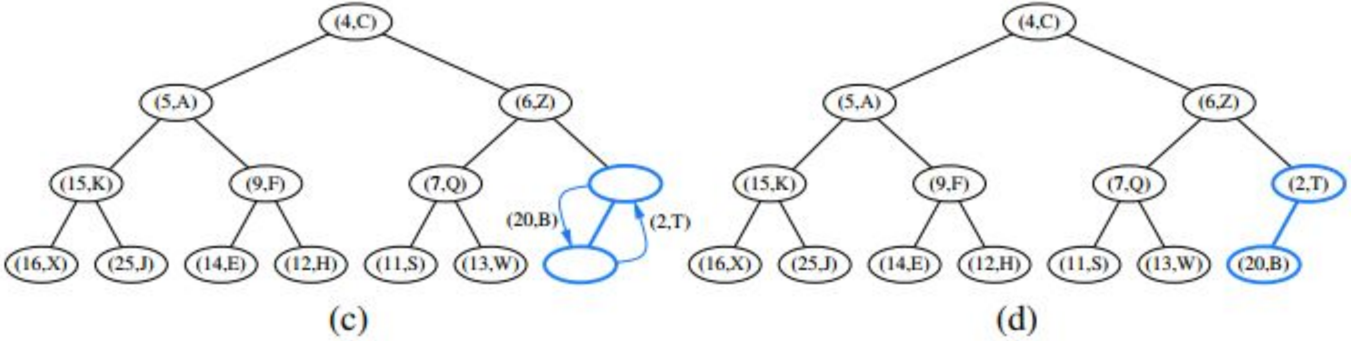
up-heap bubbling (bubble-up)

- Occurs when a new element is added or when an element's priority is increased.
- The new or updated element may violate the heap property.
  - It can break the min-heap, max-heap property.
- So, we compare the element to its parent node.
  - If it violates the property, we swap it with the parent.
  - This continues until element is in a position where the heap property is no longer violated.
- This is used when inserting a new element into heap or when an existing element's priority has been increased (for a max heap) or decreased (for a min heap)

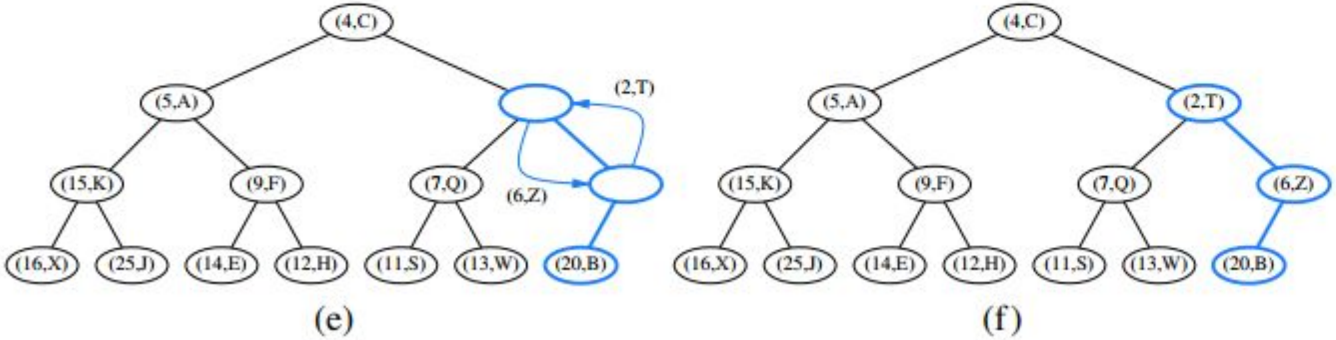
Adding a new entry with key 2 to min heap



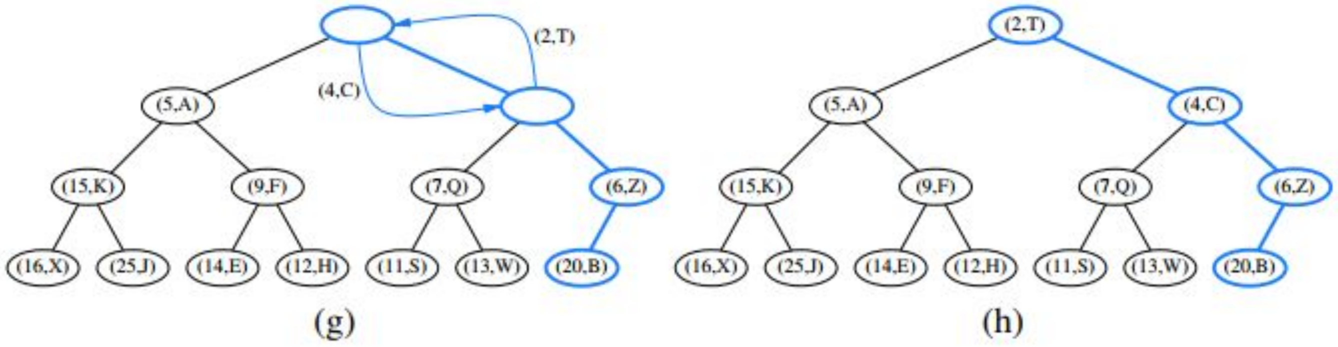
Adding a new entry with key 2 to min heap



Adding a new entry with key 2 to min heap



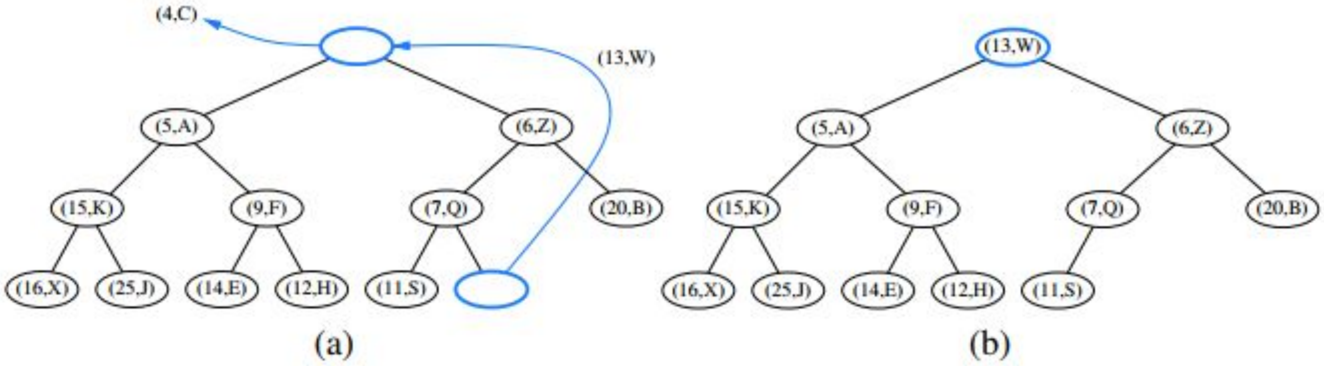
Adding a new entry with key 2 to min heap



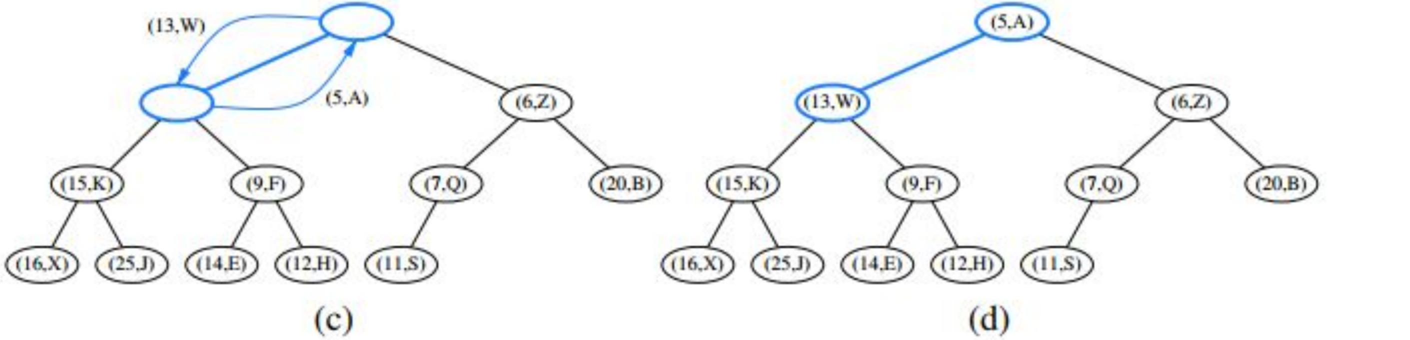
down-heap bubbling

- Used when an element is removed from the heap or an elements priority is decreased.
  - In a min-heap, this typically involves the root element being removed.
  - In max-heap, the largest element.
- After the root is removed, the last element in the heap is moved to the root position.
  - This is compared to its children.
  - If it violates the heap property, it is swapped with one of its children.
  - This is repeated until the heap property is restored.

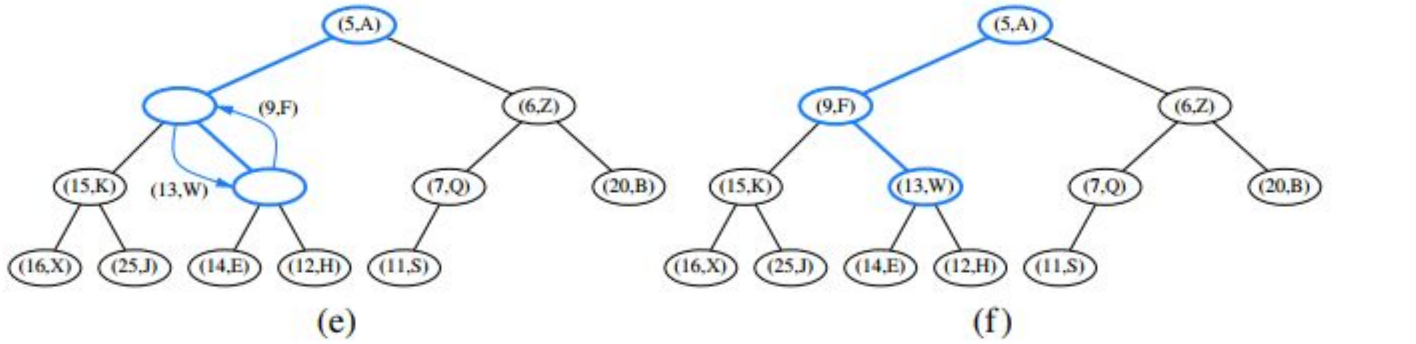
Removing the entry with smallest key



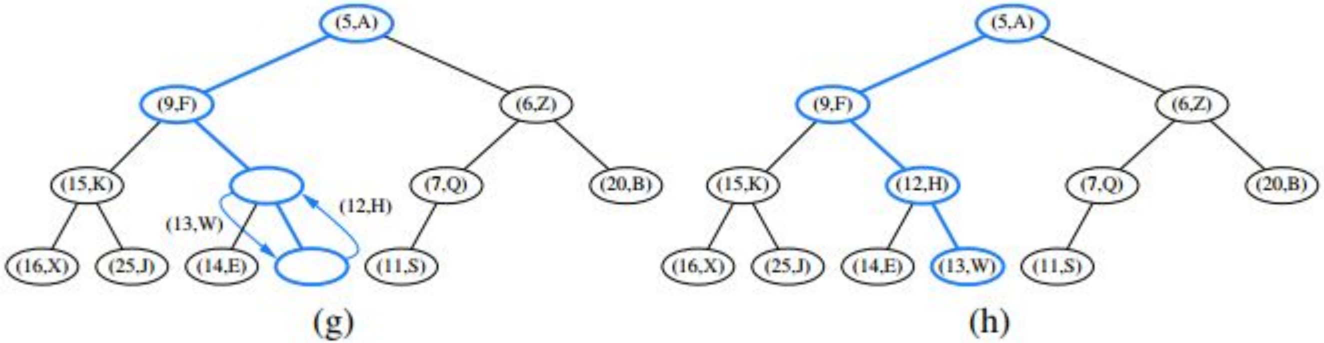
Removing the entry with smallest key



Removing the entry with smallest key

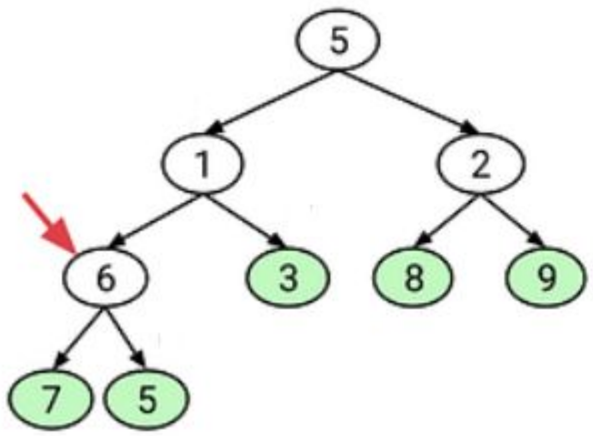


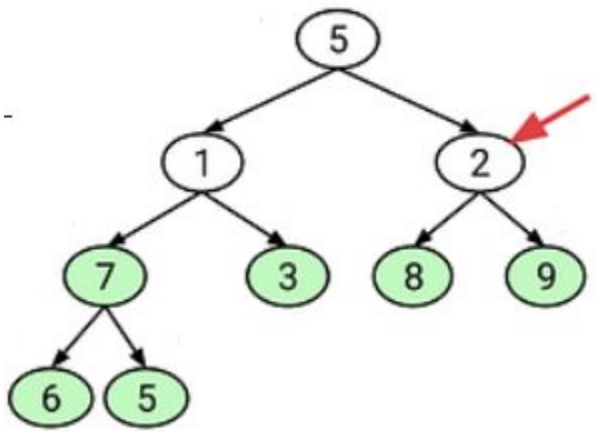
Removing the entry with smallest key



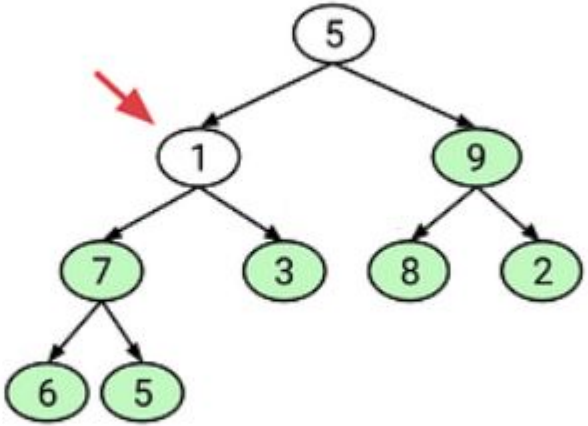
Creating a heap with bottom up method

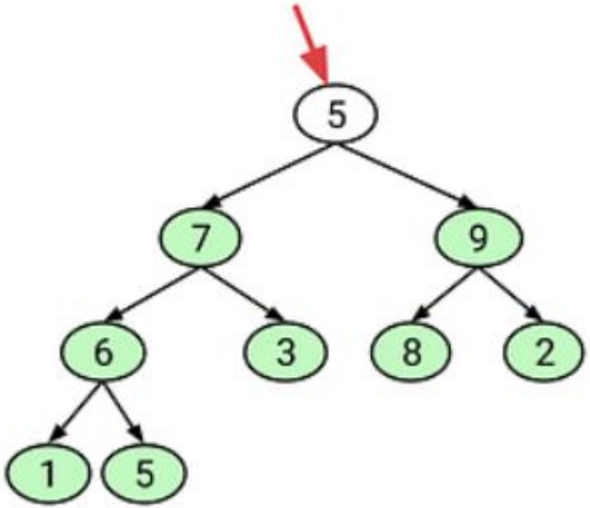
- Create a *max heap* with bottom up method:
- {5,1,2,6,3,8,9,7,5}
- Lets go step by step



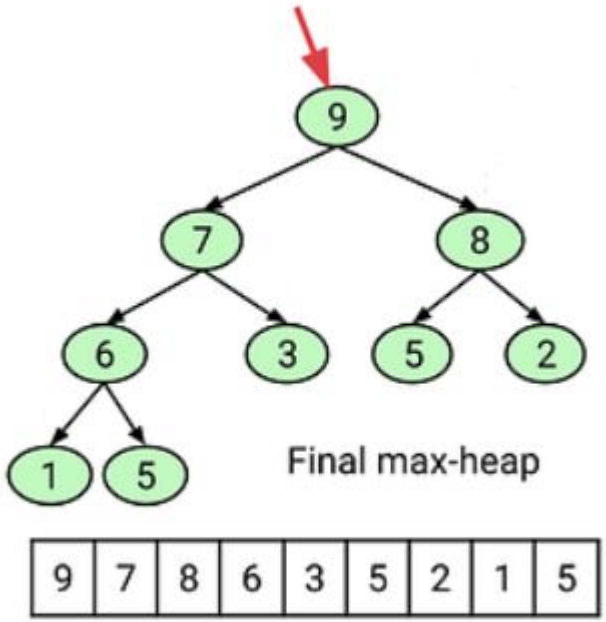


Creating a heap with bottom up method





Final



I expect you to know...

- Which bubbling is used when an element's priority is decreased?
- Which bubbling is used when an element is inserted to a priority queue?
- Given a tree, can you say whether it is min-heap or max-heap?
- Given a tree, write it in array form.
- Removing a tree from a heap tree
- Adding an element to a heap tree
- Create a min-heap or max-heap from a given unsorted array