Trees
Special Binary Trees

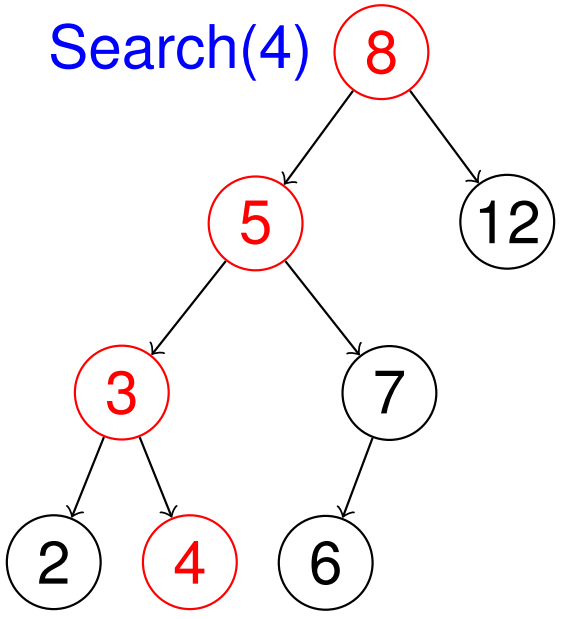
Dr. Nguyen Hua Phung

HCMC University of Technology, Viet Nam

01, 2020

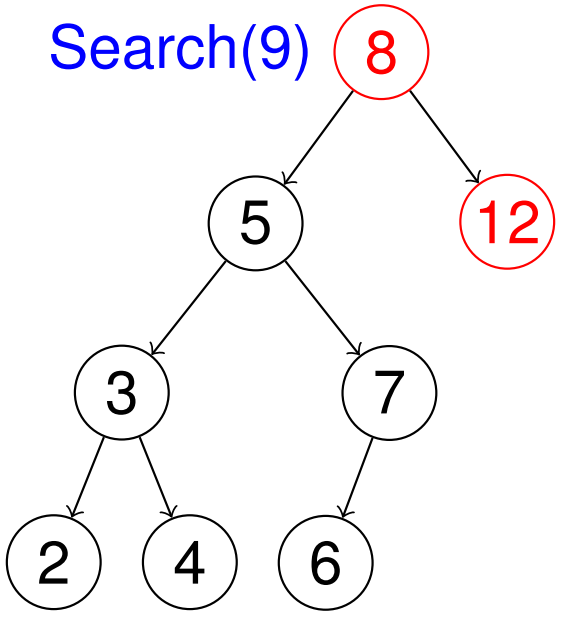
Outline

Binary Search Trees (BST)



- For a node whose value is K,
  - all nodes in its left subtree have key values $<$ K, and
  - all nodes in its right subtree have key values $\geq$ K
- LNR Traversal => ascending order
- RNL Traversal => descending order
- Search a value along the path of the tree

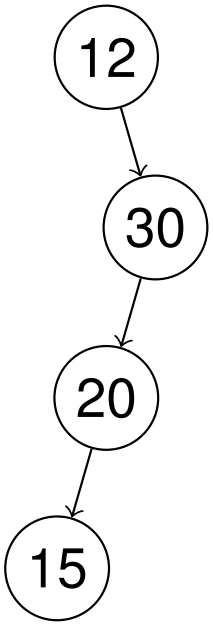
Binary Search Trees (BST)



- For a node whose value is K,
  - all nodes in its left subtree have key values $<$ K, and
  - all nodes in its right subtree have key values $\geq$ K
- LNR Traversal => ascending order
- RNL Traversal => descending order
- Search a value along the path of the tree

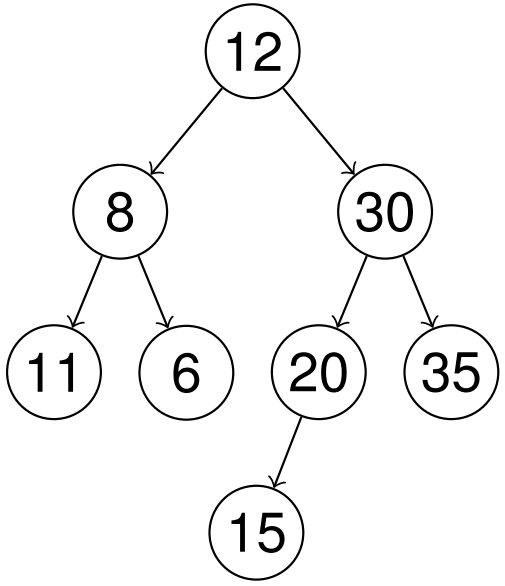
Types of BST

- Degenerate: only one child
- Balanced: mostly two children



Degenerate BST

Searching: O(n)



Balanced BST

Searching: O(log n)

BST Insertion

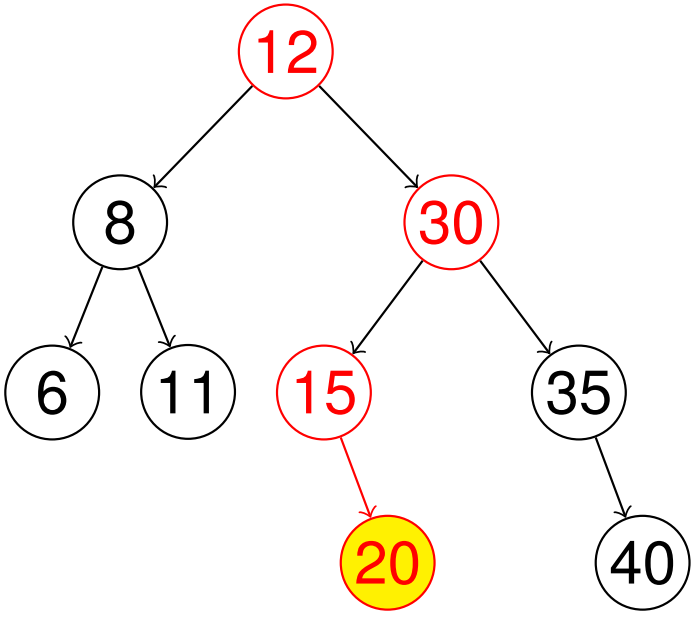
Algorithm 1: Insert Node whose value is K into BST

Input: New node whose value is K
Output: a BST with new node inserted

```
1  N ← root
2  while true do
3      if N's value ≤ K then
4          if N's right child is NULL then
5              N's right child ← New node
6              Break
7          else
8              N ← N's right child
9          end
10     else
11         if N's left child is NULL then
12             N's left child ← New node
13             Break
14         else
15             N ← N's left child
16         end
17     end
18 end
19 return
```

BST Insertion Example

- Insert(20)



- $12 < 20$, right
- $30 > 20$, left
- $15 < 20$, right

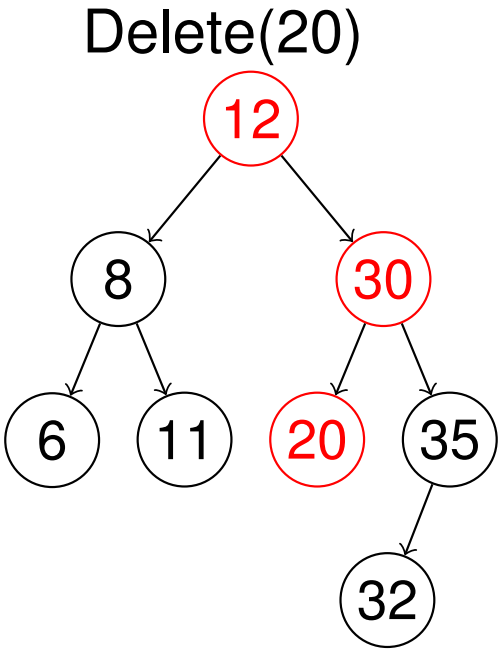
BST Node Deletion

Algorithm 2: Delete Node whose value K in BST

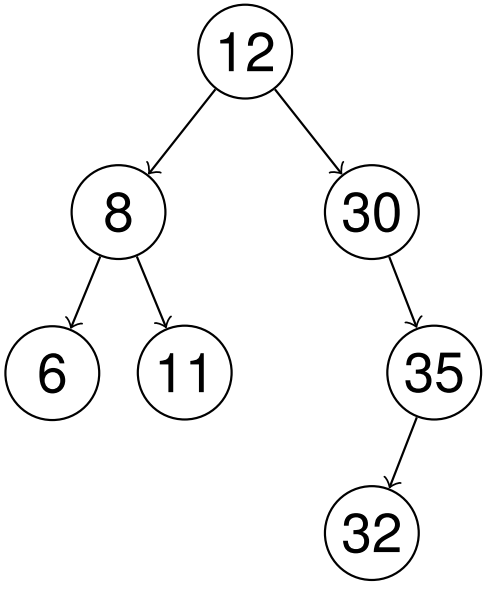
Input: Value K
Output: a BST node whose value is K is removed

```
1  Find a node whose value is K
2  if found then
3  |   N ← found node
4  |   if N is leaf then
5  |   |   delete node N
6  |   else
7  |   |   L ← leftmost of N's right subtree
8  |   |   N's value ← L's value
9  |   |   replace L with its right child and delete node L
10 |   end
11 return
```

Deleted Node is a leaf



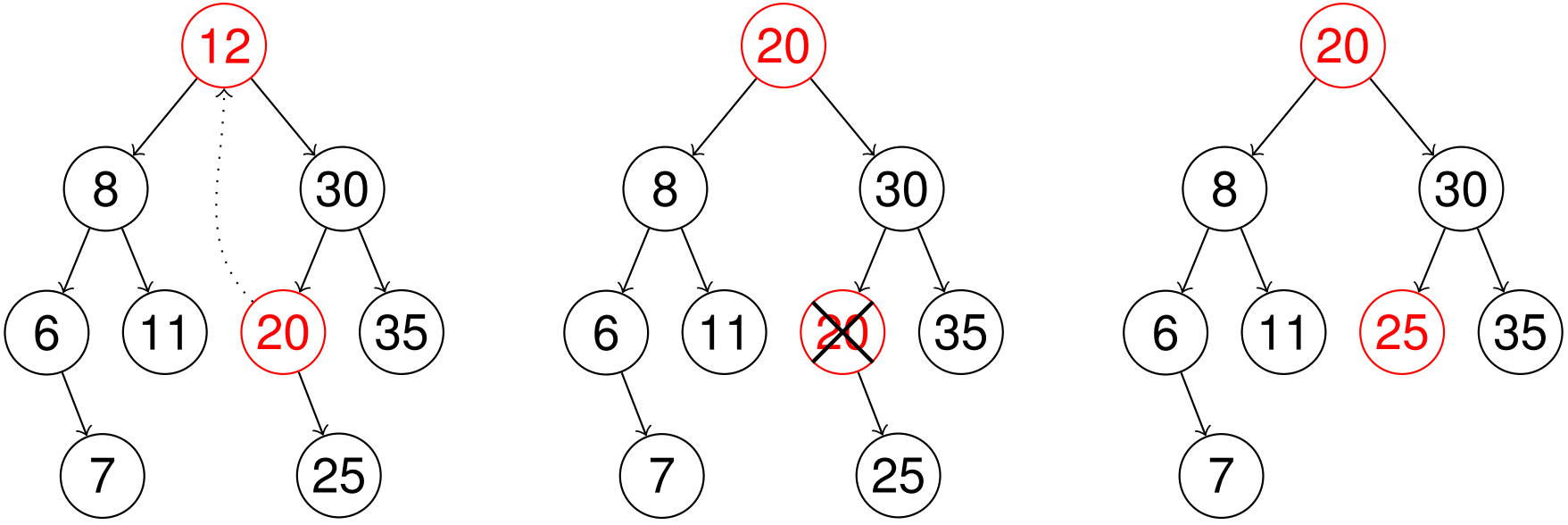
- 12 $<$ 20, right
- 30 $>$ 20, left
- 20 = 20, delete



Deleted node is an internal

Delete(12)

- find node whose value is 12 => internal
- find leftmost of its right subtree
- replace value
- delete node/replace with right child of leftmost node

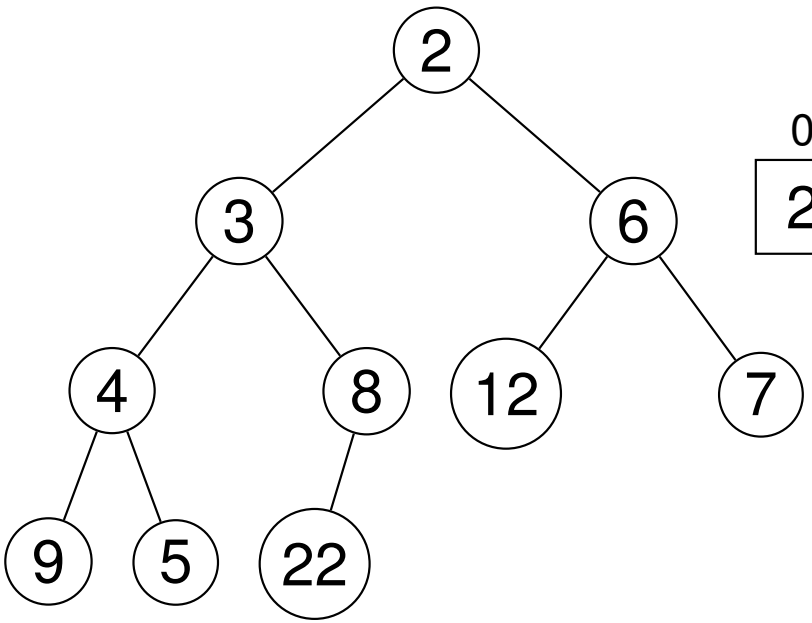


Heaps and Priority Queues

- Normal queues: FIFO
- Priority queues: Highest priority removed first
  - Most critical patient is treated first
  - Highest priority task is executed first
- Heaps are used to implement priority queues

Heaps

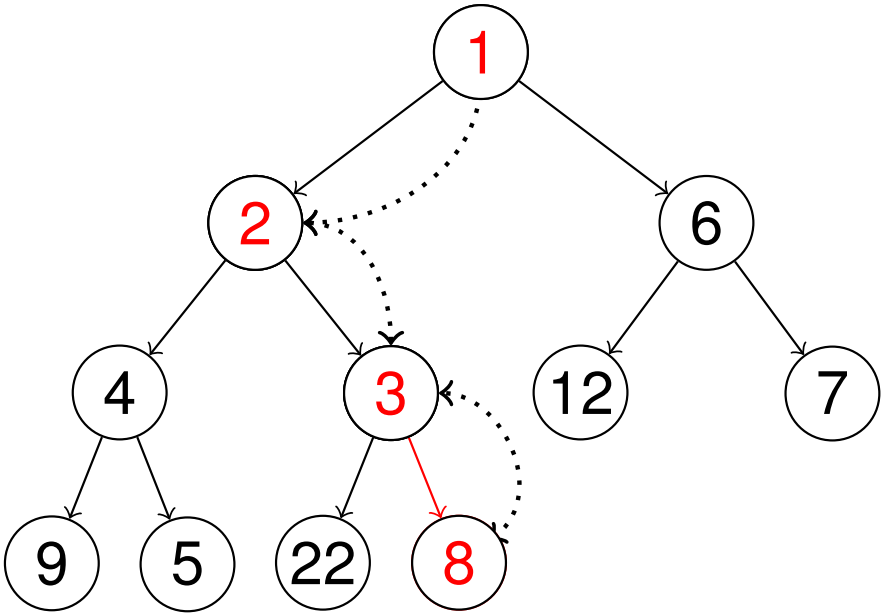
- Complete binary tree
- Partially order
  - Max-heap: parent's value $>=$ its children's value
  - Min-heap: parent's value $<=$ its children's value



0	1	2	3	4	5	6	7	8	9	10
2	3	6	4	8	12	7	9	5	22	

Heap Insertion

- Insert(1)



Heap Insertion

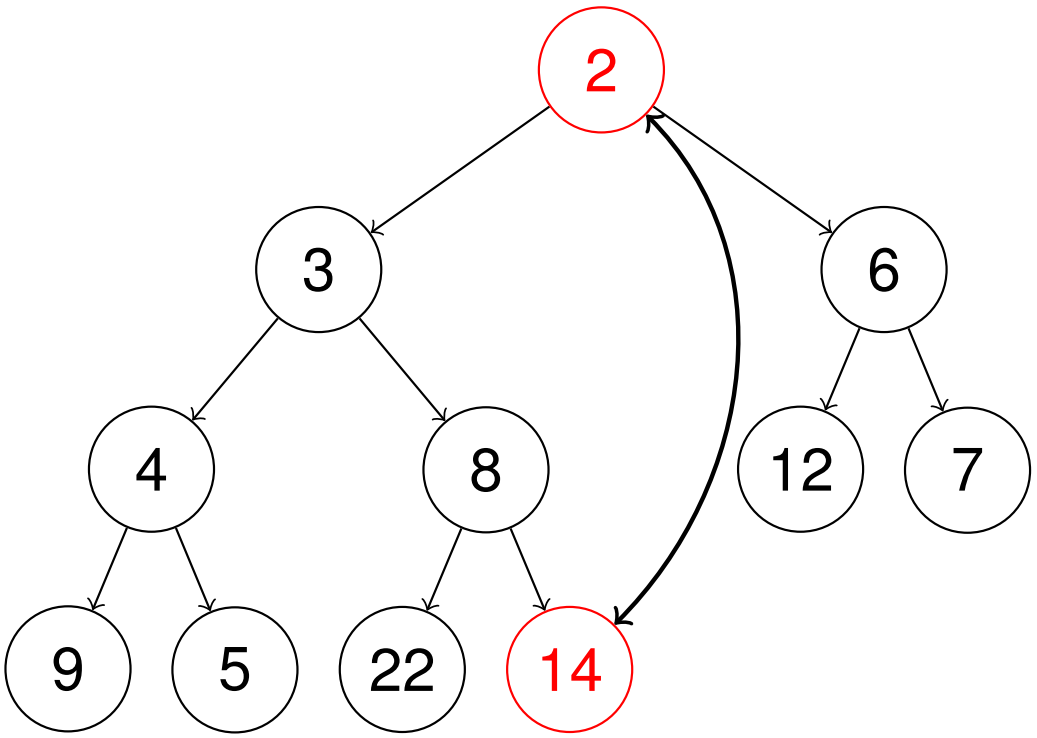
Algorithm 3: Heap Insertion

Input: New value V

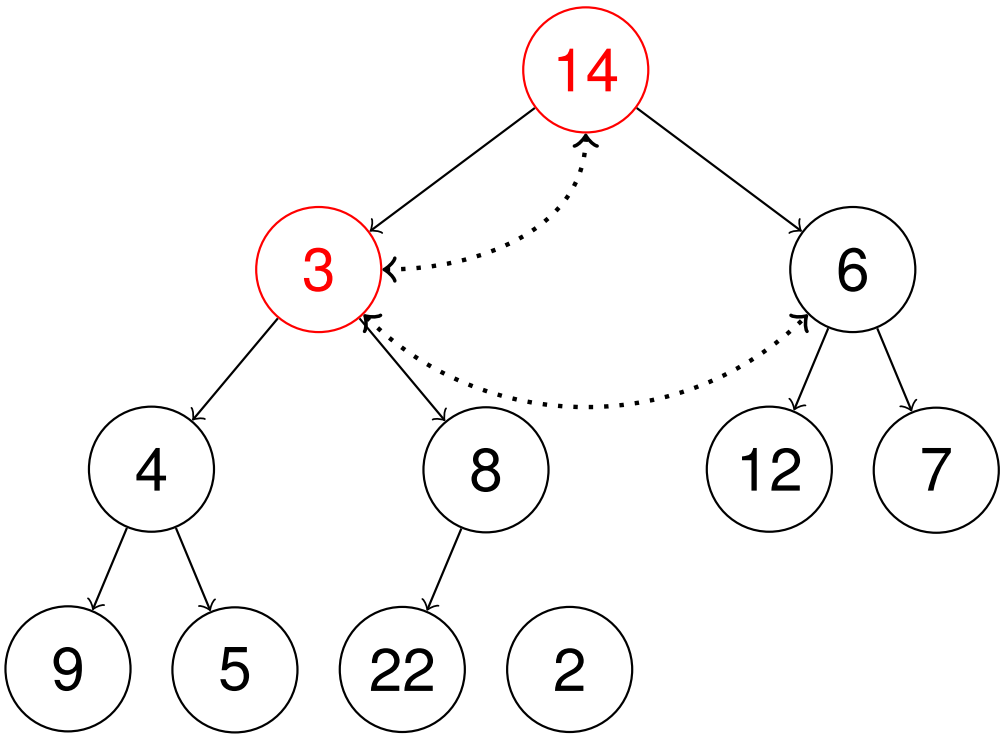
Output: New element of V is inserted into heap

```
1 Increase heap size by 1 to have room for new value;
2 Assign V to new element ;
3 i ← index of new element ;
4 while i is not index of root and V < value at parent of i do
5 |   swap V and value at parent of i ;
6 |   i ← index of parent of i ;
7 end
```

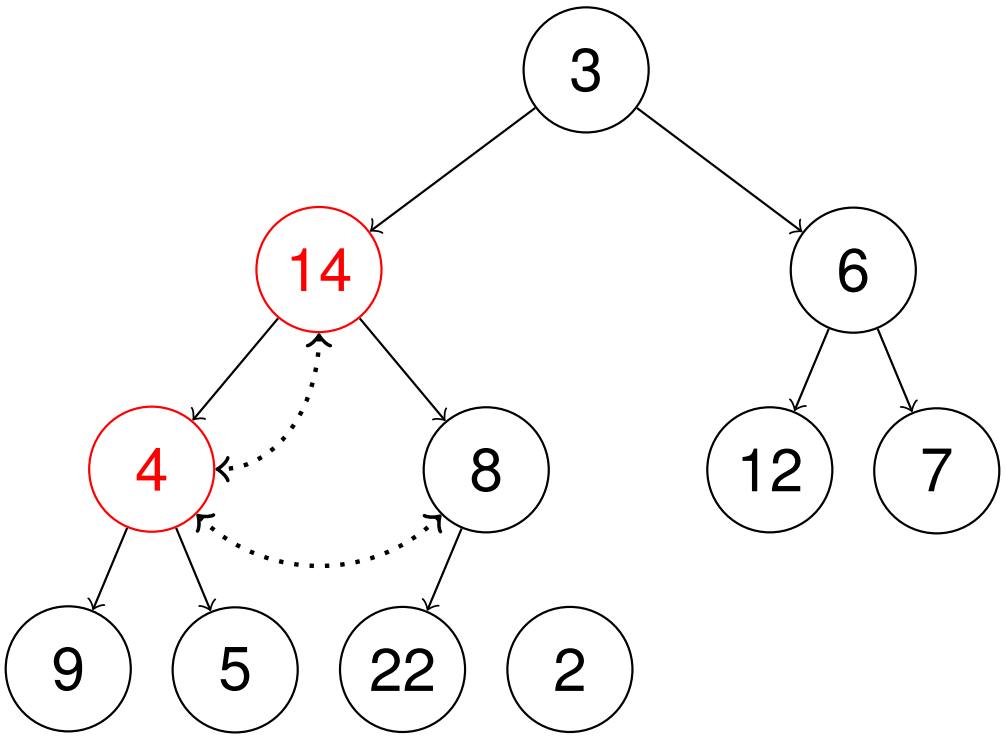
Heap Deletion



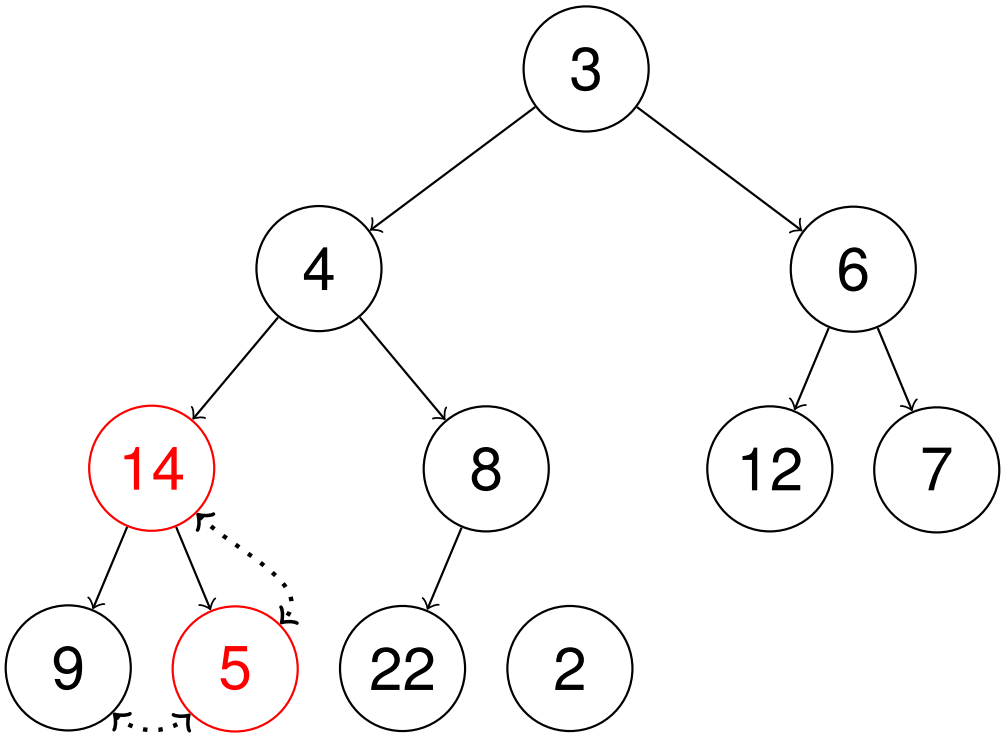
Heap Deletion



Heap Deletion



Heap Deletion



Heap Deletion