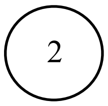
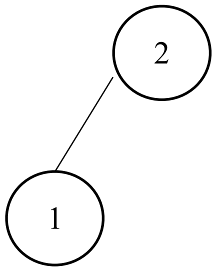
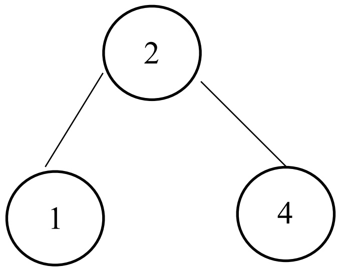
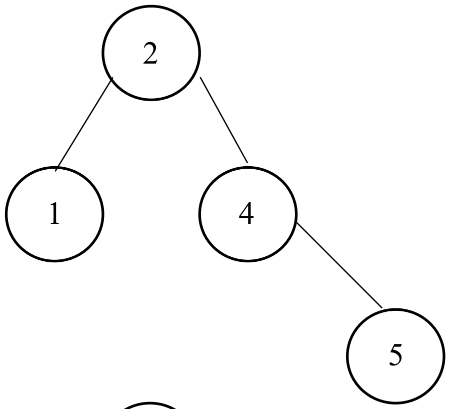
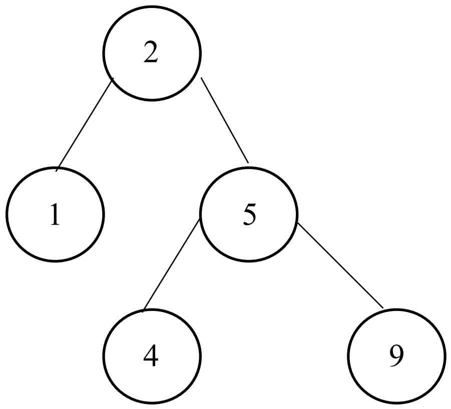
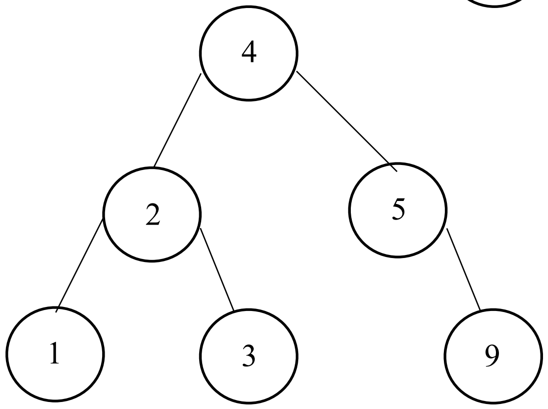
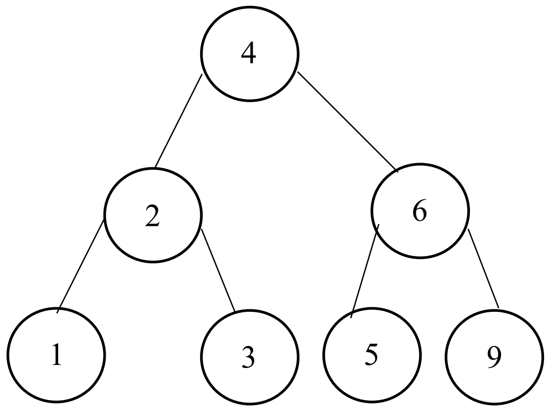
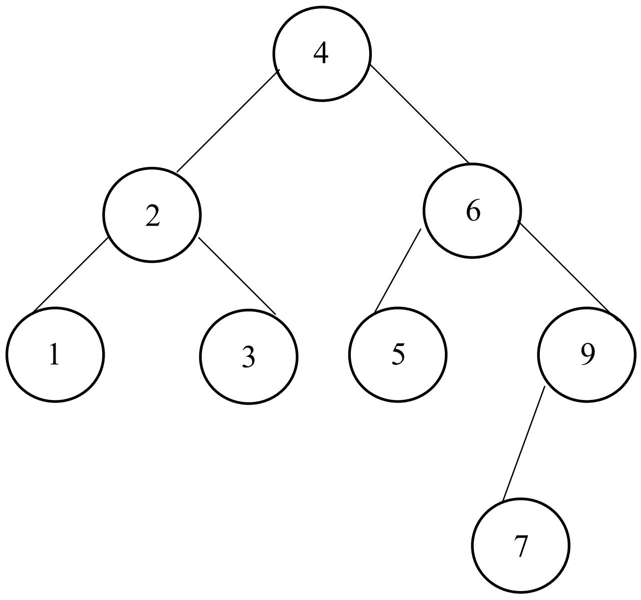
University of Regina
Department of Computer Science

Bao Cao
200363431

CS 340 - Assignment 3

Exercise 4.19

University of Regina
Department of Computer Science

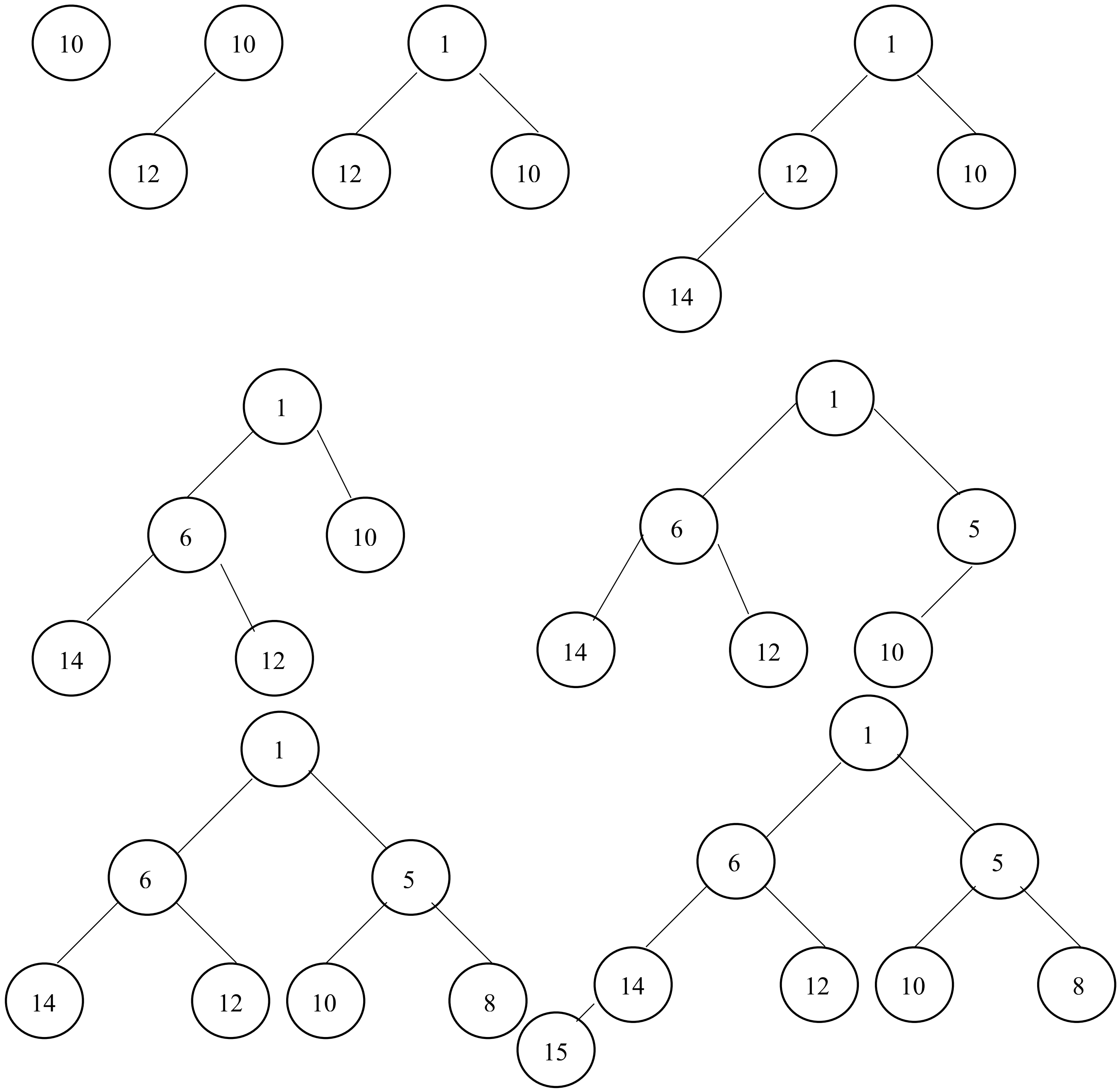
Bao Cao
200363431

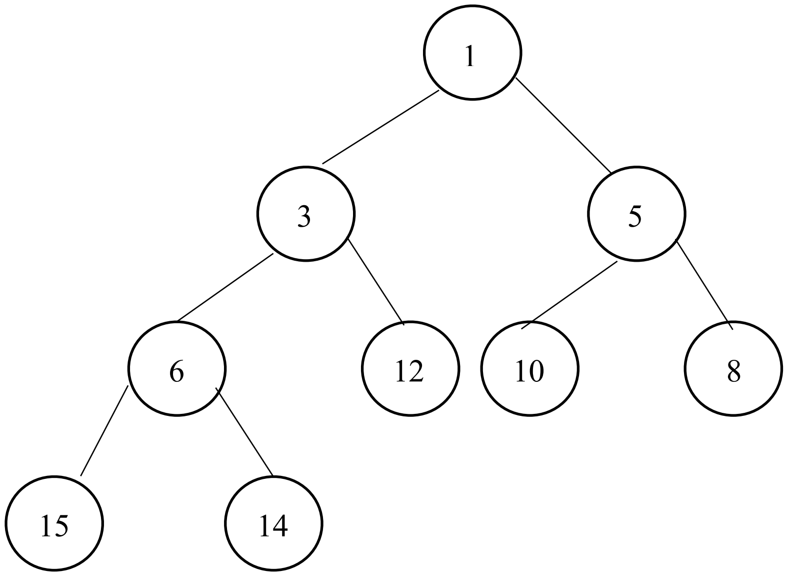
Exercise 4.25

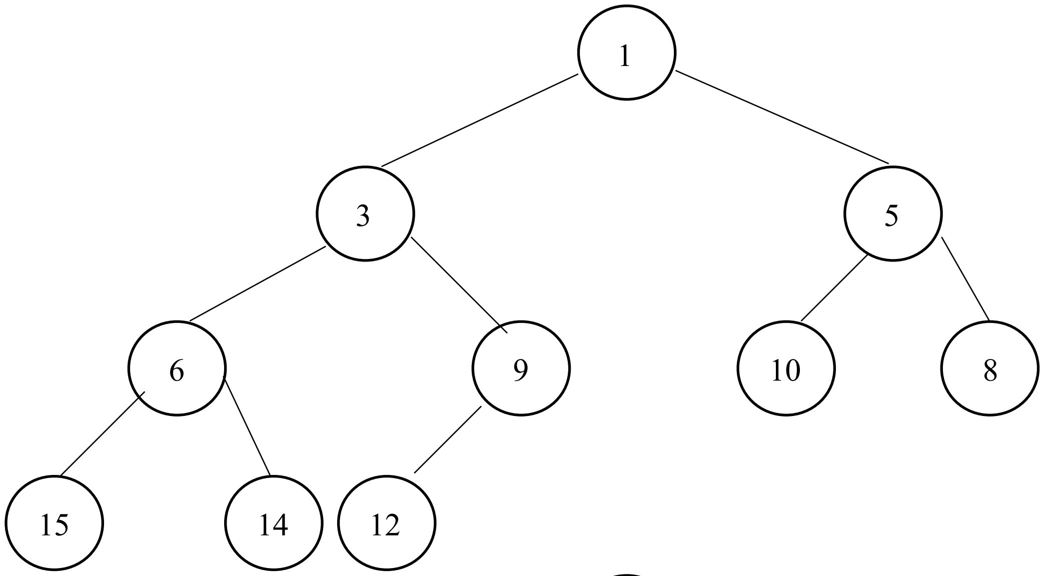
a. The maximum height of an N-node AVL tree is O(logN). Therefore, O(log(logN)) is the number of bits are required per node to store the height of a node. (log is logarithm base 2).
b. The smallest AVL tree that overflows an 8-bit height counter is AVL tree of height 127 (8-bit integers range from -128 to 127).

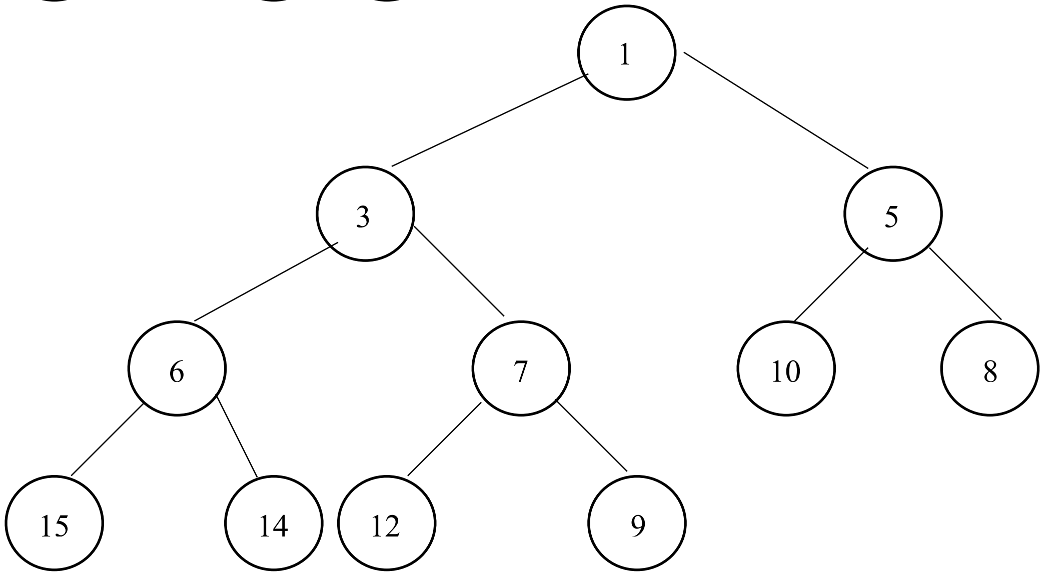
Exercise 6.2

a.

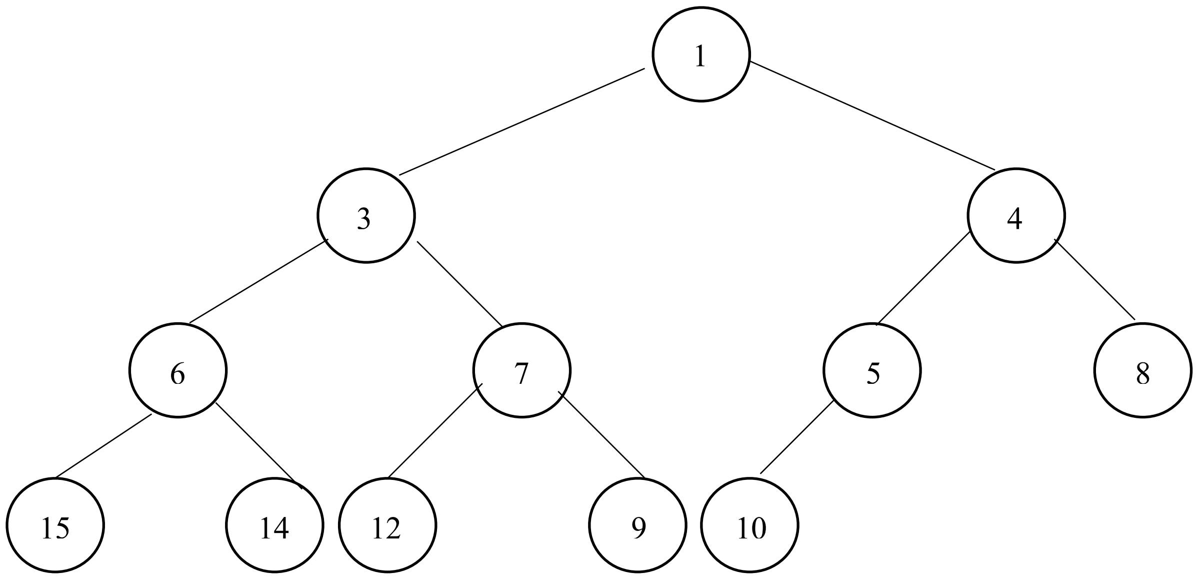


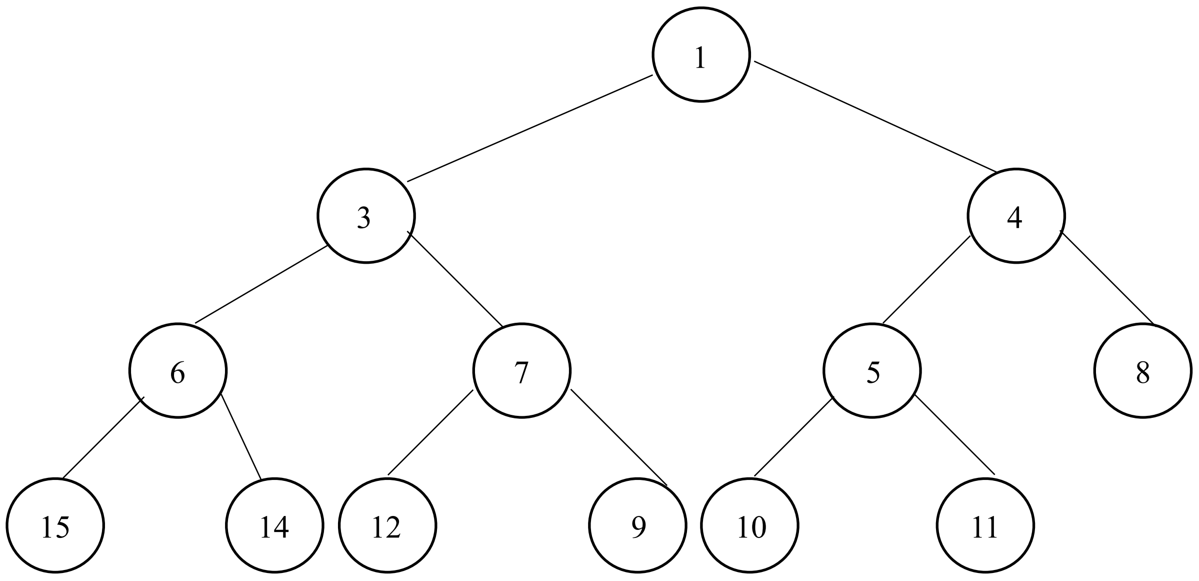

1
3
5
6
12
10
8
15
14

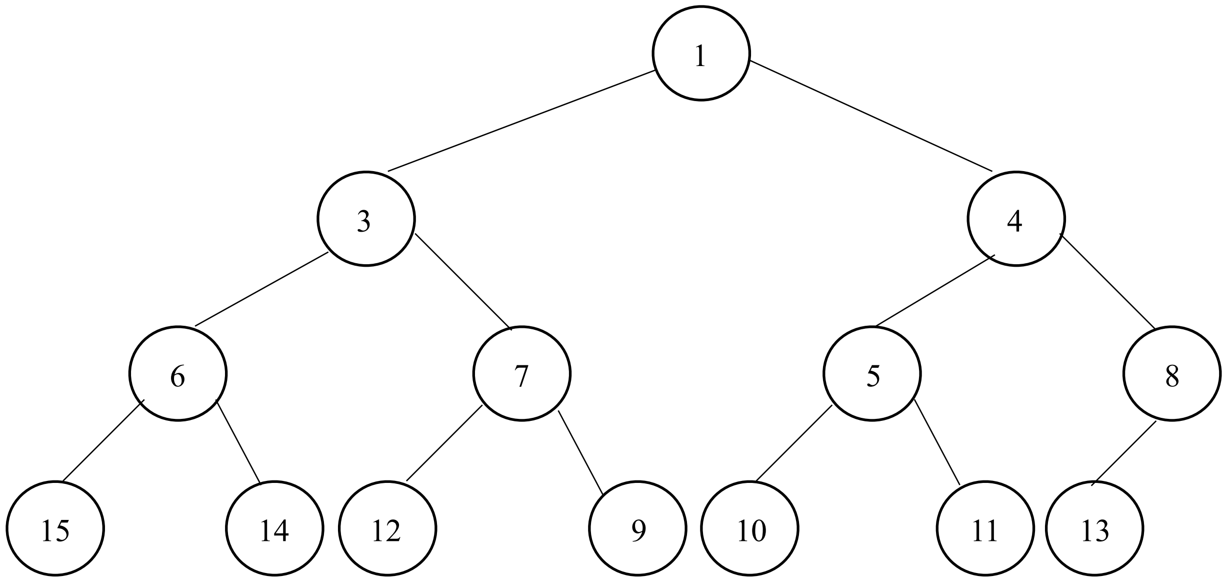

1
3
5
6
9
10
8
15
14
12

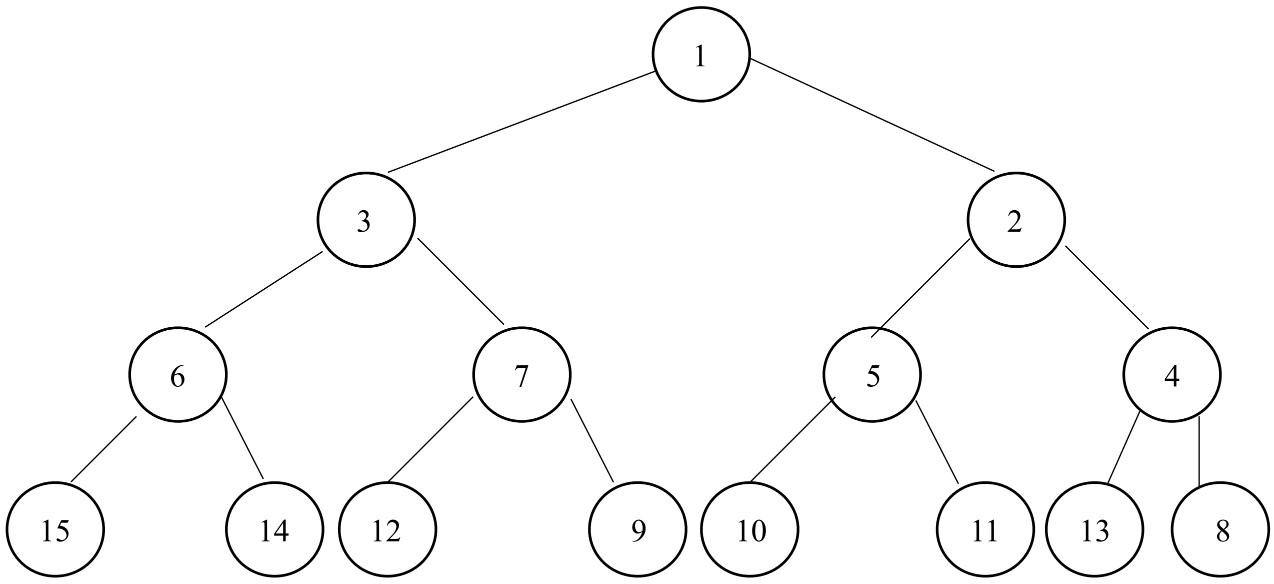

1
3
5
6
7
10
8
15
14
12
9

Bao Cao
200363431

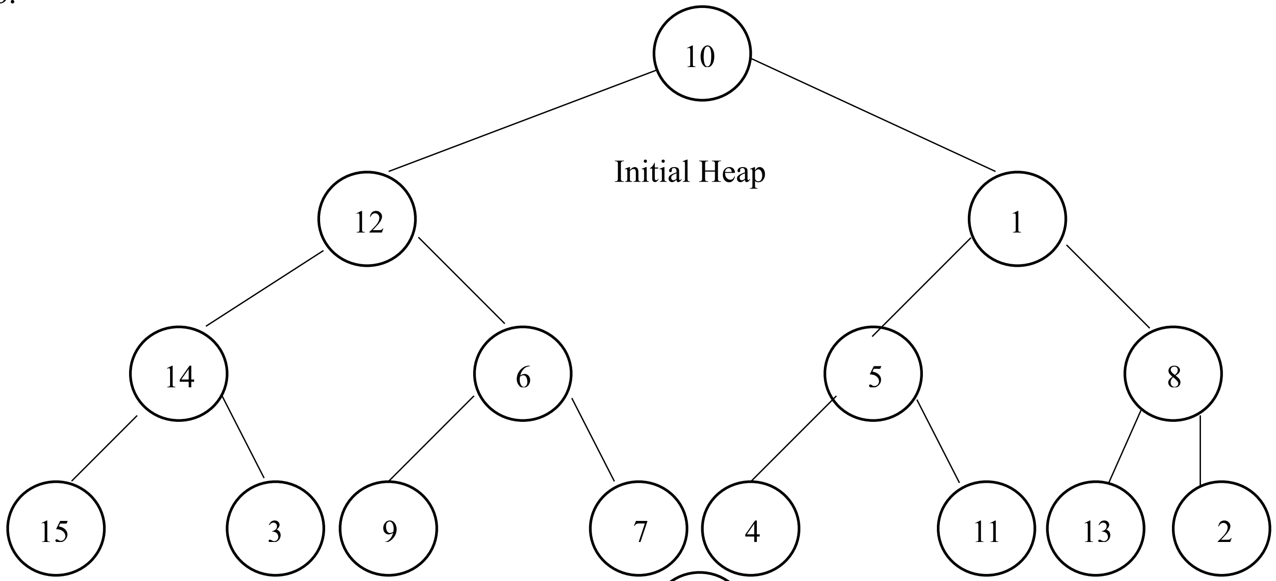


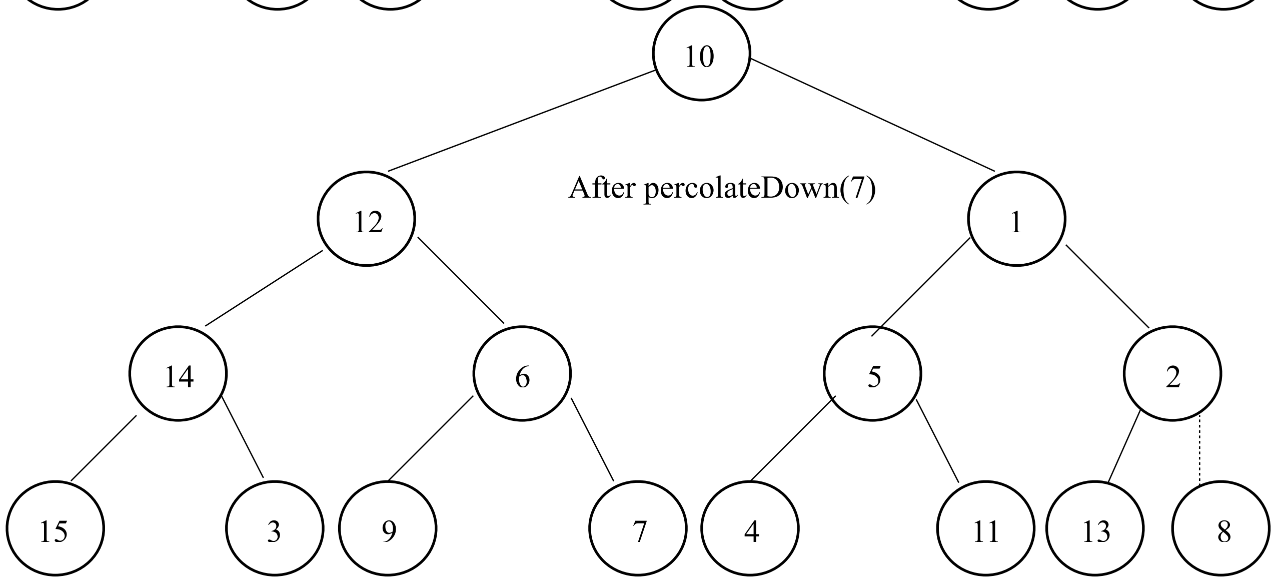




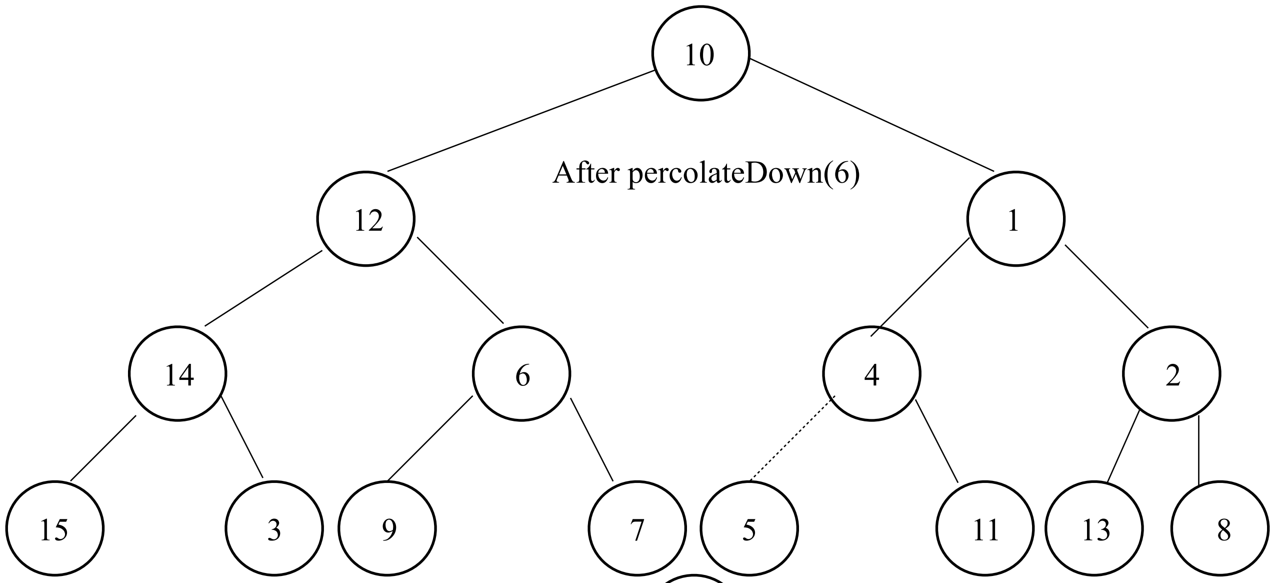

1
3
2
6
7
5
4
15
14
12
9
10
11
13
8

b.

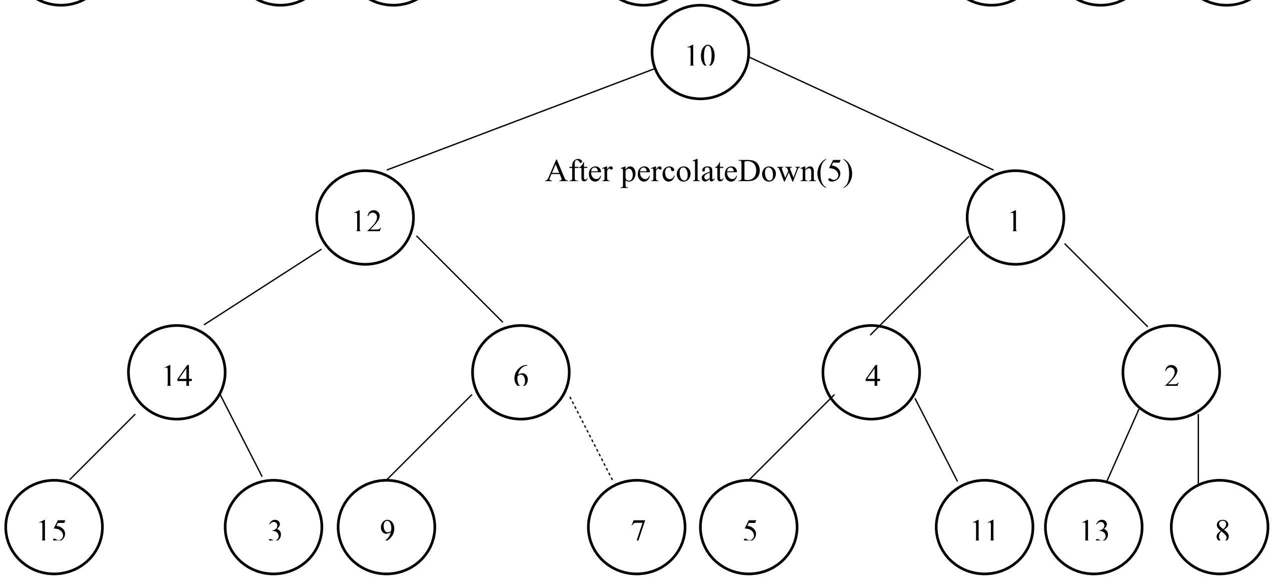

10
Initial Heap
12
1
14
6
5
8
15
3
9
7
4
11
13
2


10
After percolateDown(7)
12
1
14
6
5
2
15
3
9
7
4
11
13
8

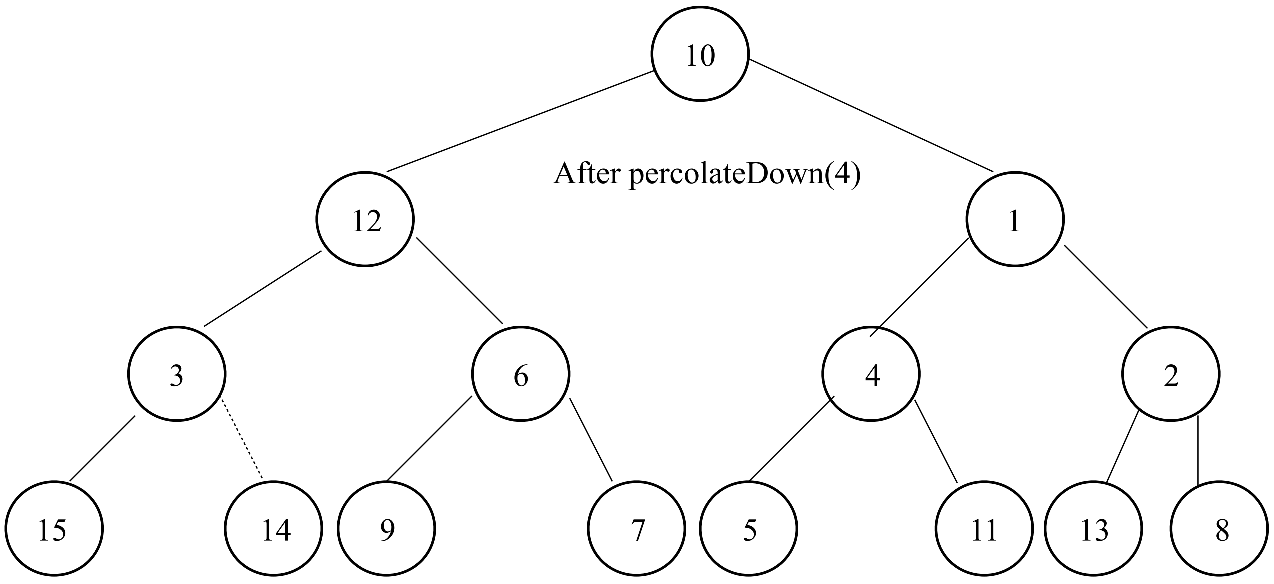
Bao Cao
200363431



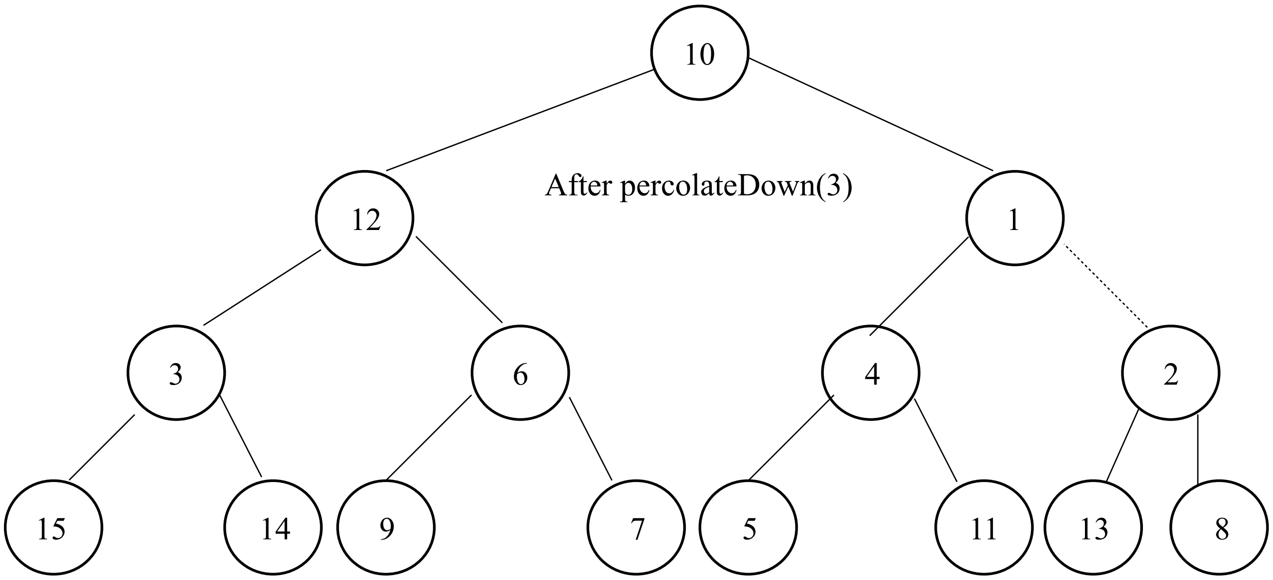
After percolateDown(6)

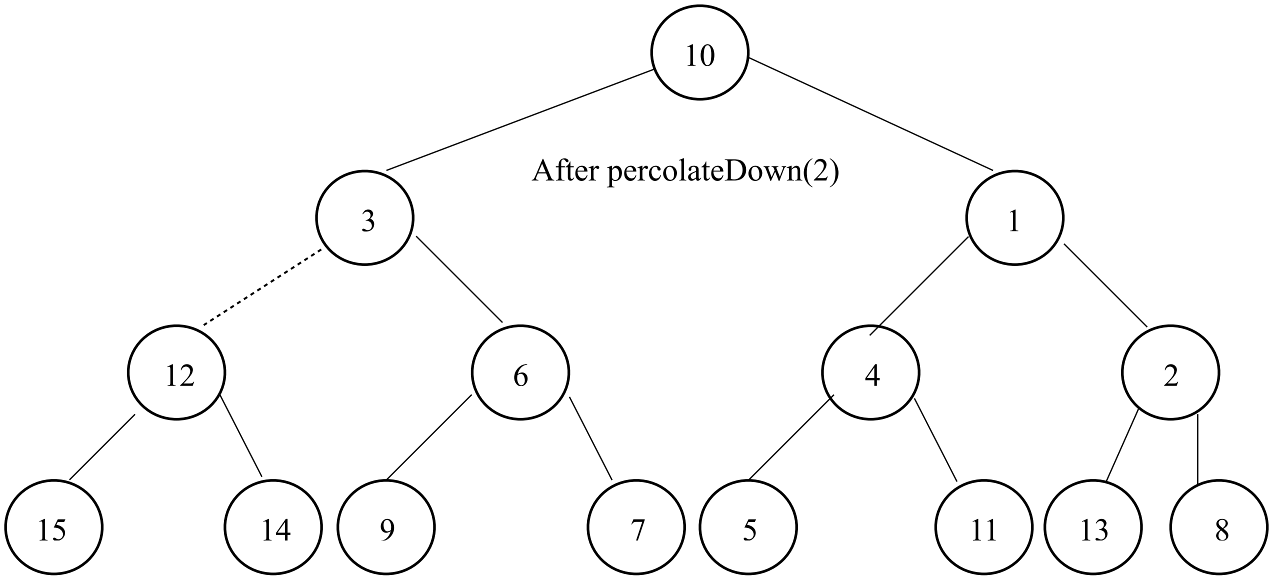


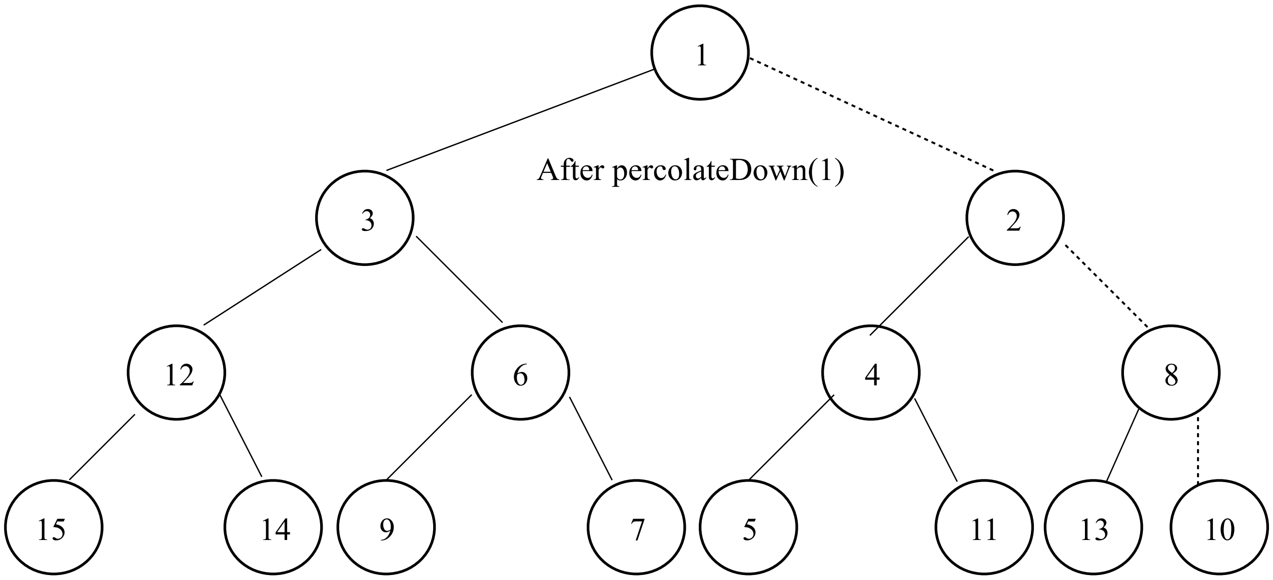
After percolateDown(5)



After percolateDown(4)

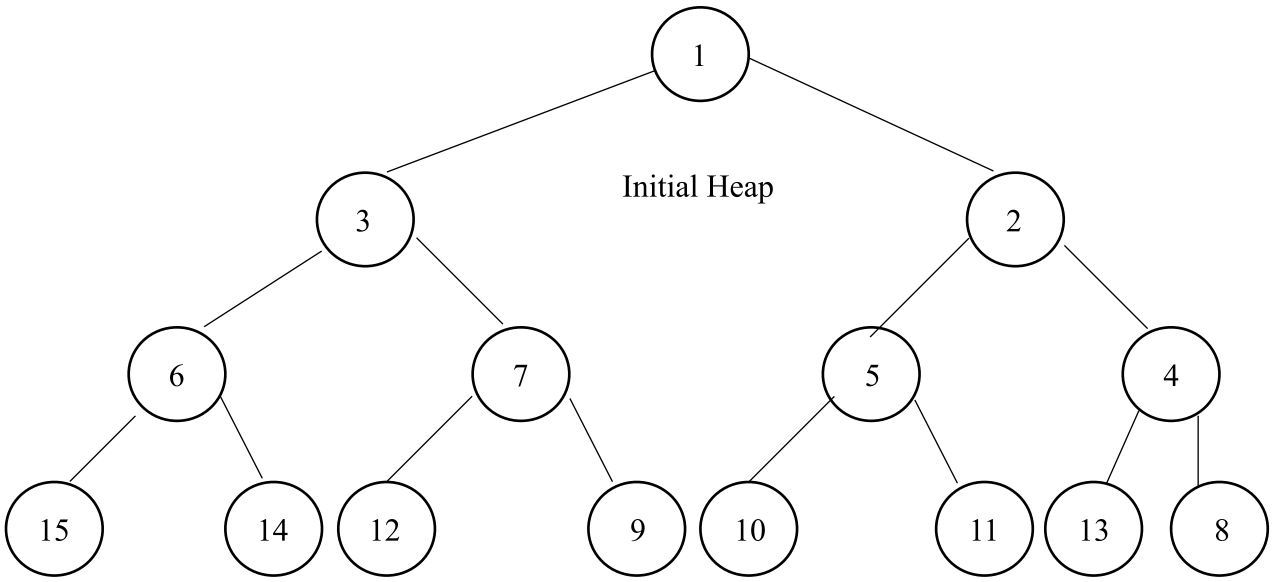

After percolateDown(3)


After percolateDown(2)

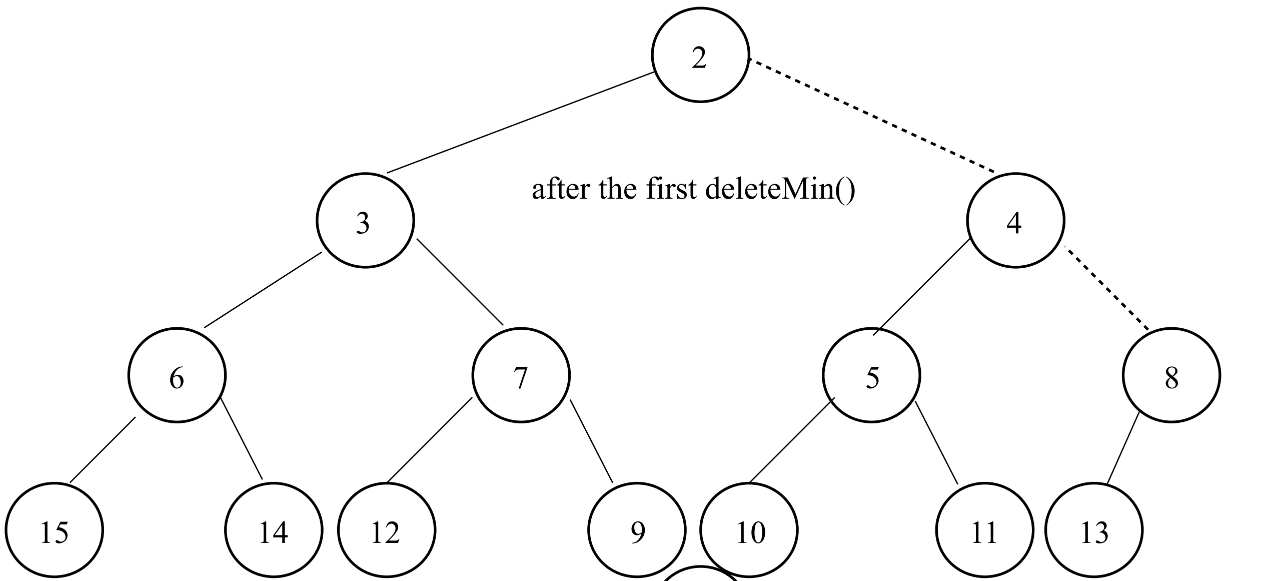

After percolateDown(1)

Bao Cao
200363431

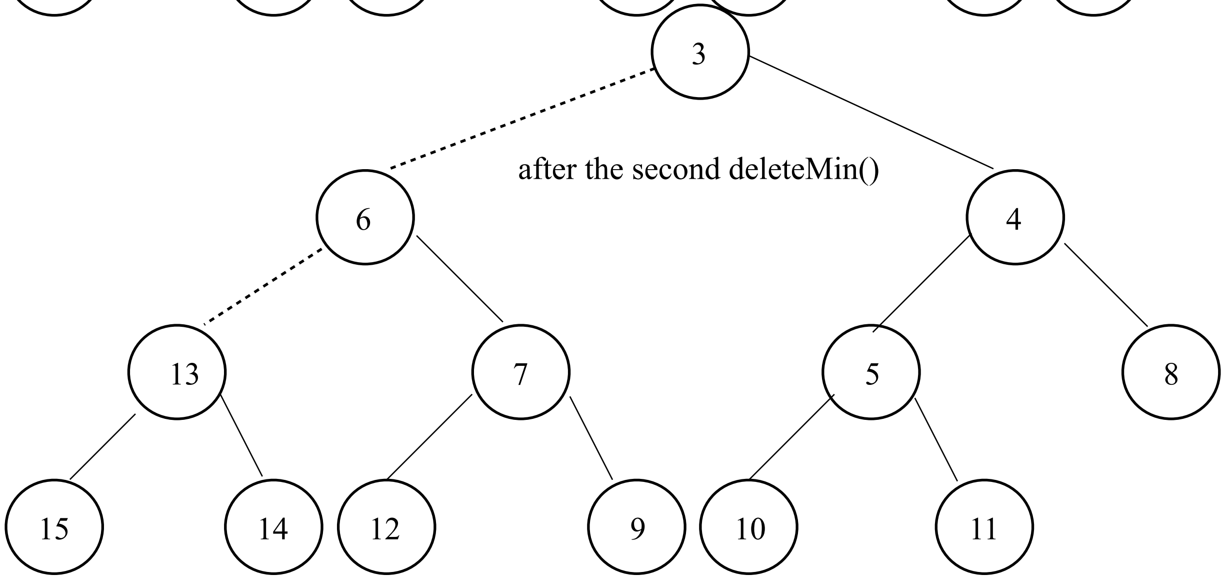
Question 6.3



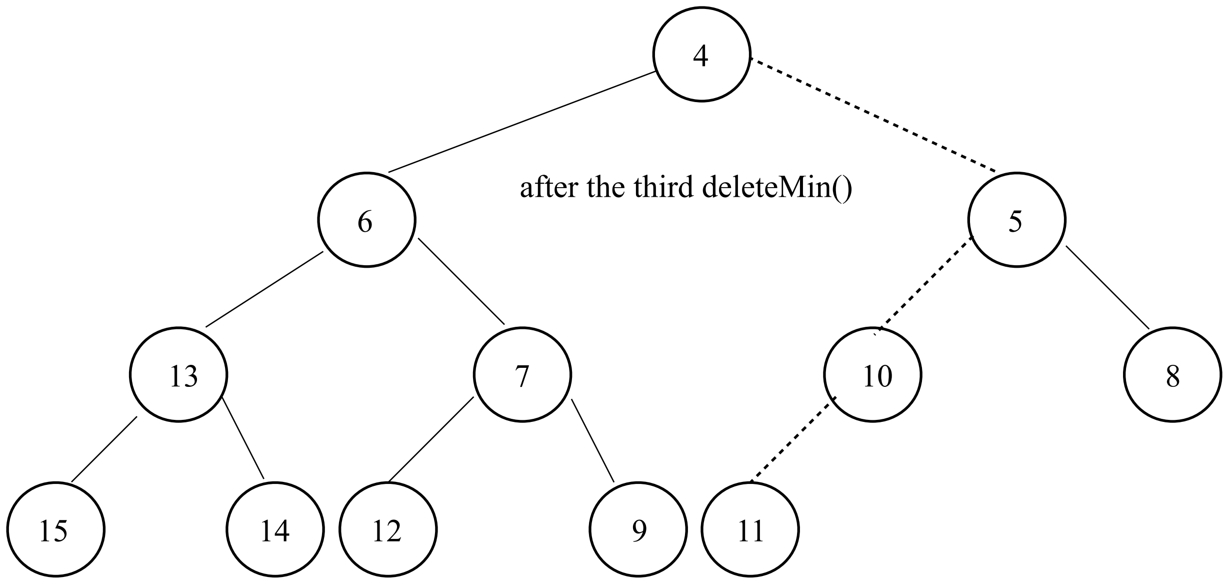
Initial Heap



after the first deleteMin()



after the second deleteMin()



Question 6.6

There are 1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 - 15*2 = 225 nodes

Question 6.18

a. Minimum element in a min-max heap is the root. Maximum element in a min-max heap is the largest element in the second level.
b. Insert a new node into min-max heap:
   The main steps that happen in function Insert():
   - Increase the size of the heap by one which will be the new spot to insert the new value.
   - Assign new value to that spot.
   - Percolate the value up if there is any violation (the details of percolation is below).

```
void PercolateUp(Heap heap, int index){
    int level = floor(log(index)); // Determine whether the node is in odd level or even level
    int parent = index / 2;           // Get the parent of the new inserted node
    int grandparent = parent / 2;  // Get the grandparent of the new inserted node
    while(grandparent > 0) {
        // If the level is even, the node should be smaller than parent and larger than
grandparent. If not, swap it to parent or grandparents depends on the case.
        if (level is even) {
            if (heap[index] > heap[parent]) {
                swap(index, parent);
                index = parent;
            }
            else if (heap[index] < heap[parent] AND heap[index] < heap[grandparent]) {
                swap(index, grandparent);
                index = grandparent;
            }
            else // if the inserted node doesn't violate anything
                return;
        }
        // If the node is at odd level, it should be larger than parent and smaller than
grandparent. If not, swap it to its parent or grandparent depends on the case.
        else {
            if (heap[index] < heap[parent]) {
                swap(index, parent);
                index = parent;
            }
            else if (heap[index] > heap[parent] AND heap[index] > heap[grandparent]) {
                swap(index, grandparent);
                index = grandparent;
            }
            else // if the inserted node doesn't violate anything
                return;
        }
        // Set the value of parent, grandparent and level if the inserted node is percolated up.
        parent = index / 2;
        grandparent = parent / 2;
        level = floor(log(index));
    }
    return;
}


void Insert(int element, Heap heap) {
    int i = heap->size + 1;
    heap[i] = element;
    PercolateUp(heap, i);
}
```

c. The main steps in DeletedMin and DeletedMax:
- Percolate down the deletedNode (the details for percolating down is inside the function itself).
- When the node can't be percolated down anymore, swap it to the last node.
- Delete the last node which contains the value of the deleted node.
- Since the swapped node can violate the heap, depends on its position on the heap, either percolate up the node or its chid (the details is inside the function).
Note: PercolateUp function used in DeleteMin and DeleteMax is the function from section b above.

```
void DeleteMin(Heap heap){
    int level = 2;
    int deletedIndex = 1; // index of the deleted node - the root
    while (pow(2, level) < heap->size) {
        int indexMin = pow(2, level);
        int min = heap[pow(2, level)];
        // Find minimum at the current level
        for (int i = pow(2, level); i < pow(2, level + 1) OR i < heap->size; i++) {
             if (heap[i] < min) {
                 min = heap[i];
                 indexMin = i;
             }
        } // END OF FOR LOOP
        // Swap deleted node and the minimum (percolate down the deleted node)
        swap(deletedIndex, indexMin);
        // Keep track of the index of the deleted node
        deletedIndex = indexMin;
        // Move down to the next even level;
        level += 2;
    } // END OF WHILE LOOP
    // After the root is percolated down to the highest even level
    // If the highest level of the heap is even, swap the last node to the deleted node, delete the
last node and percolate up the swapped position
    if (deletedIndex * 2 > heap->size) {
       swap(heap->size-1, deletedIndex);
       delete(heap[size-1]);
       PercolateUp(heap, deletedIndex);
    }
    // If the deleted position has one child, that one child is the last node, swap them, delete
the last node and percolate up the swapped node
    else if (deletedIndex * 2 + 1 > heap->size) {
        swap(heap->size - 1, deletedIndex);
        delete(heap[size-1]);
        PercolateUp(heap, deletedIndex);
    }
    // If the deleted position has 2 child, swap the node with the last node, delete the last node
then percolate up the smallest child of the deleted node
    else {
        swap(heap->size - 1, deletedIndex);
        delete(heap[size-1]);
        // C++ Syntax to Percolate up the smaller child of heap[deletedIndex];
        heap[deletedIndex * 2] < heap[deletedIndex * 2 + 1] ? PercolateUp(heap, deletedIndex
* 2) : PercolateUp(heap, deletedIndex * 2 + 1);
```

```
    }
} // END OF FUNCTION
```

// DeleteMax function will be similar to DeleteMin, except the level is changed to odd, and instead of finding minimum in the current level, find maximum. When the node can't be moved further down, if the node has two children, percolate up the larger one. Instead of the root, the largest child of the root will be the maximum therefore will be deleted

```
void DeleteMax(Heap heap){
    int level = 3;
    // deletedIndex is the index of the largest child
    int deletedIndex = heap[2] > heap[3] ? 2 : 3;
    while (pow(2, level) < heap->size) {
        int indexMax = pow(2, level);
        int max = heap[pow(2, level)];
        // Find maximum at the current level
        for (int i = pow(2, level); i < pow(2, level + 1) OR i < heap->size; i++) {
            if (heap[i] > max) {
                max = heap[i];
                indexMax = i;
            }
        } // END OF FOR LOOP
        // Swap deleted node and the maximum (percolate down the deleted node)
        swap(deletedIndex, indexMax);
        // Keep track of the index of the deleted node
        deletedIndex = indexMax;
        // Move down to the next odd level;
        level += 2;
    } // END OF WHILE LOOP
    // After the root is percolated down to the highest odd level
    // If the highest level of the heap is odd, swap the last node to the deleted node, delete the
last node and percolate up the swapped position
    if (deletedIndex * 2 > heap->size) {
        swap(heap->size-1, deletedIndex);
        delete(heap[size-1]);
        PercolateUp(heap, deletedIndex);
    }
    // If the deleted position has one child, that one child is the last node, swap them, delete
the last node and percolate up the swapped node
    else if (deletedIndex * 2 + 1 > heap->size) {
        swap(heap->size - 1, deletedIndex);
        delete(heap[size-1]);
```

```
            PercolateUp(heap, deletedIndex);
        }
        // If the deleted position has 2 child, swap the node with the last node, delete the last node
    then percolate up the largest child of the deleted node
        else {
            swap(heap->size - 1, deletedIndex);
            delete(heap[size-1]);
            // C++ Syntax to Percolate up the smaller child of heap[deletedIndex];
            heap[deletedIndex * 2] > heap[deletedIndex * 2 + 1] ? PercolateUp(heap, deletedIndex
    * 2) : PercolateUp(heap, deletedIndex * 2 + 1);
        }
    } // END OF FUNCTION
```