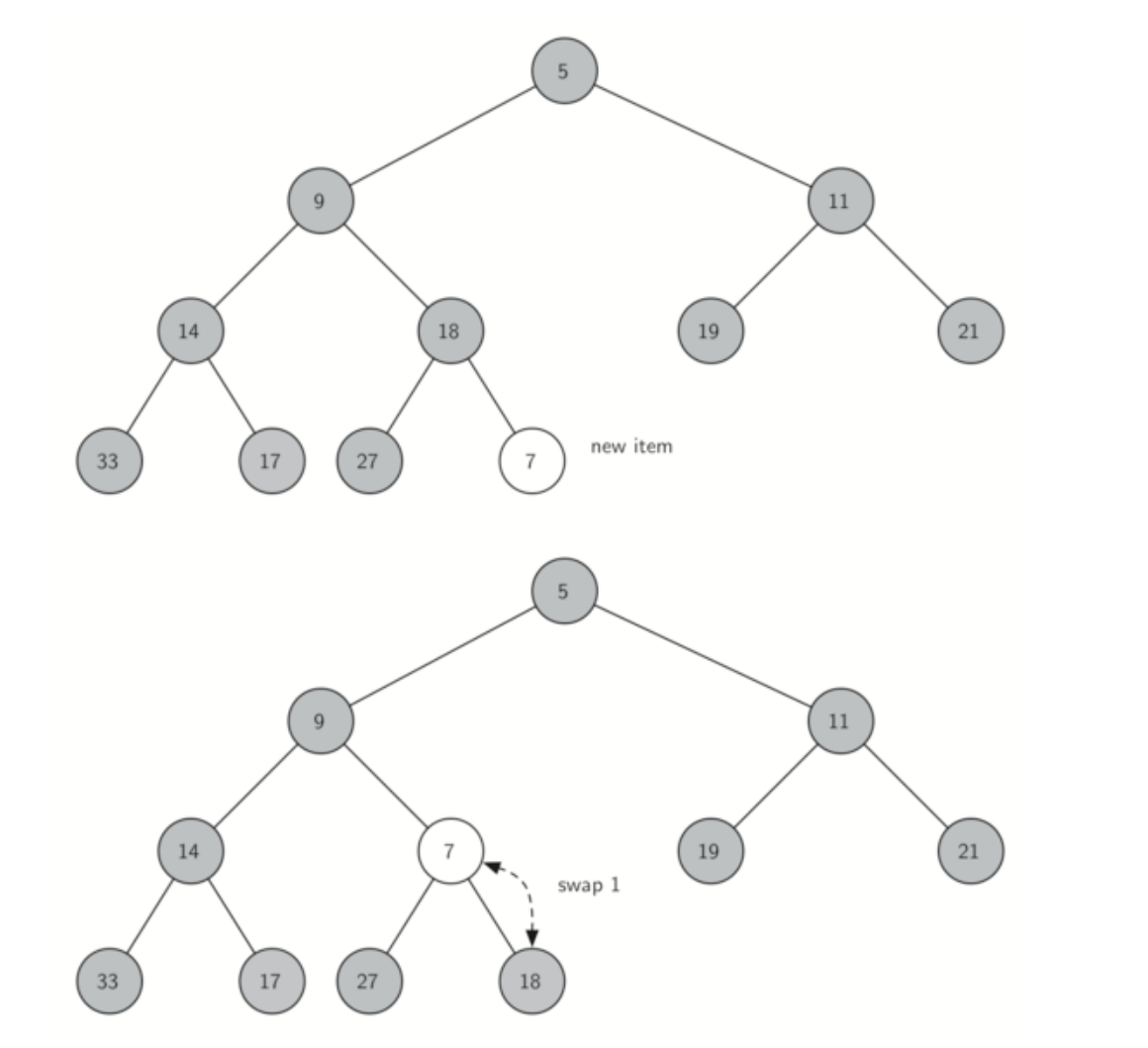
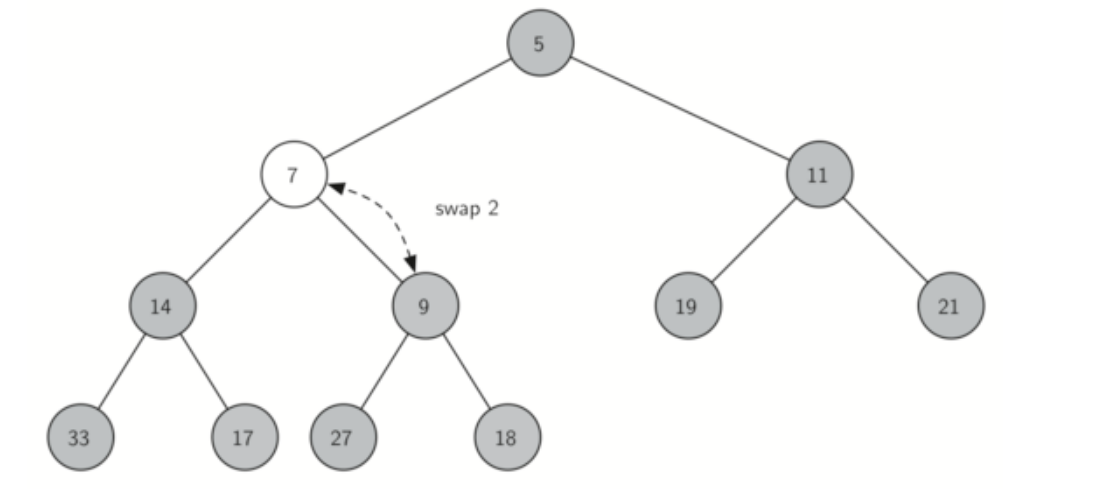
2.

insert():

The method we will implement is insert. Adds the new item to the last node of the binary tree. But we will very likely violate the heap structure property. However, it is possible to write a method that will allow us to regain the heap structure property by comparing the newly added node with its parent. If the newly added node’s value is less than its parent’s value, then we can swap the node’s value with its parent’s value. The diagram below shows the series of swaps needed to percolate the newly added node up to its proper position in the tree.

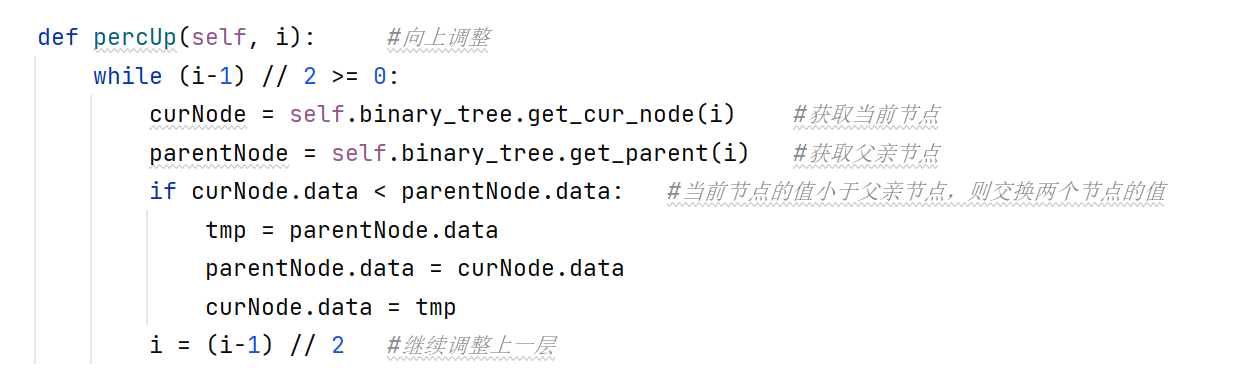
If the newly added node’s value is very small, we may still need to swap it up another level. In fact, we may need to keep swapping until we get to the top of the tree.





Percolate the new node up to its proper position

The code below shows the percUp method, which percolates a new node as far up in the tree as it needs to go to maintain the heap property.

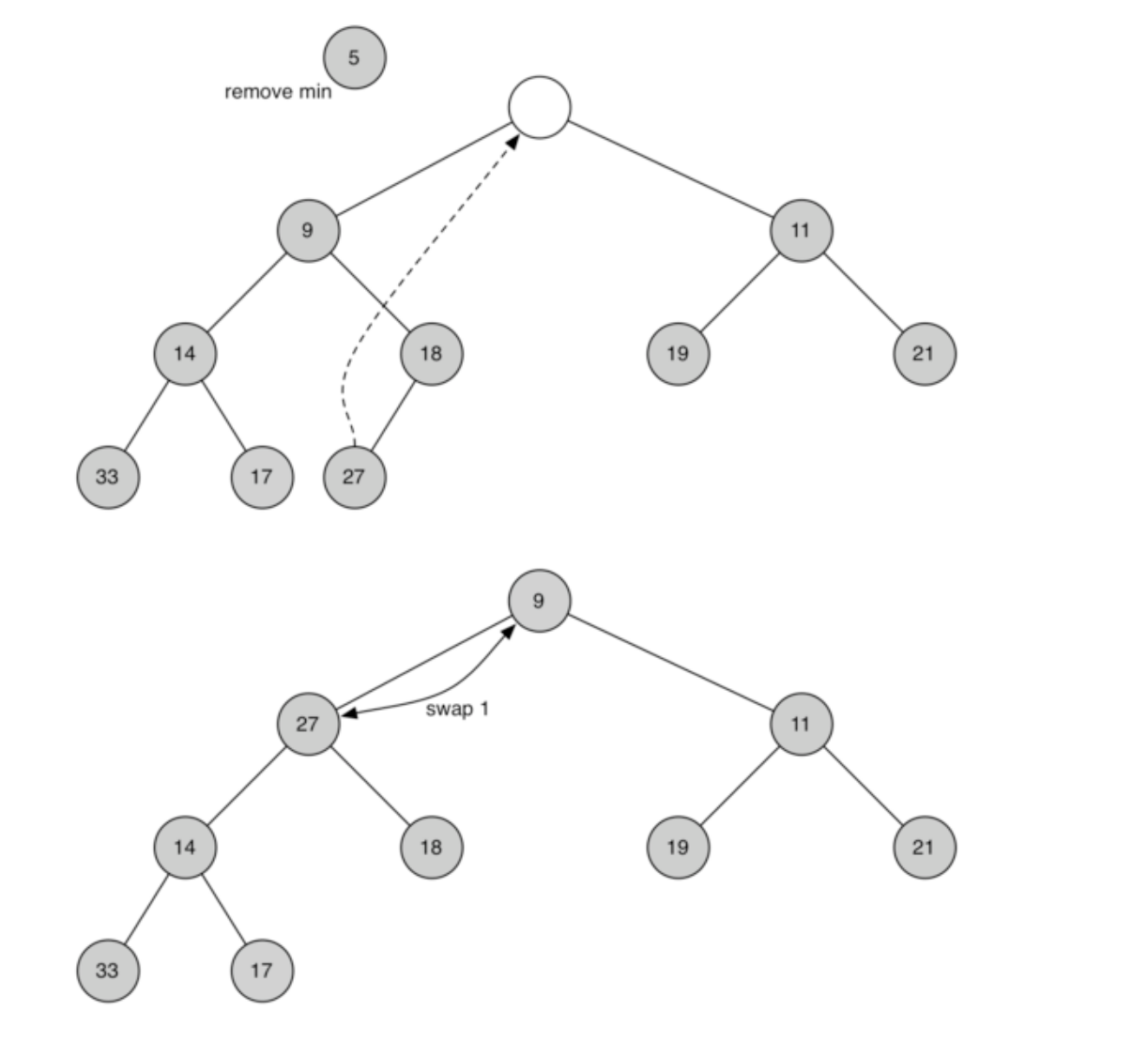


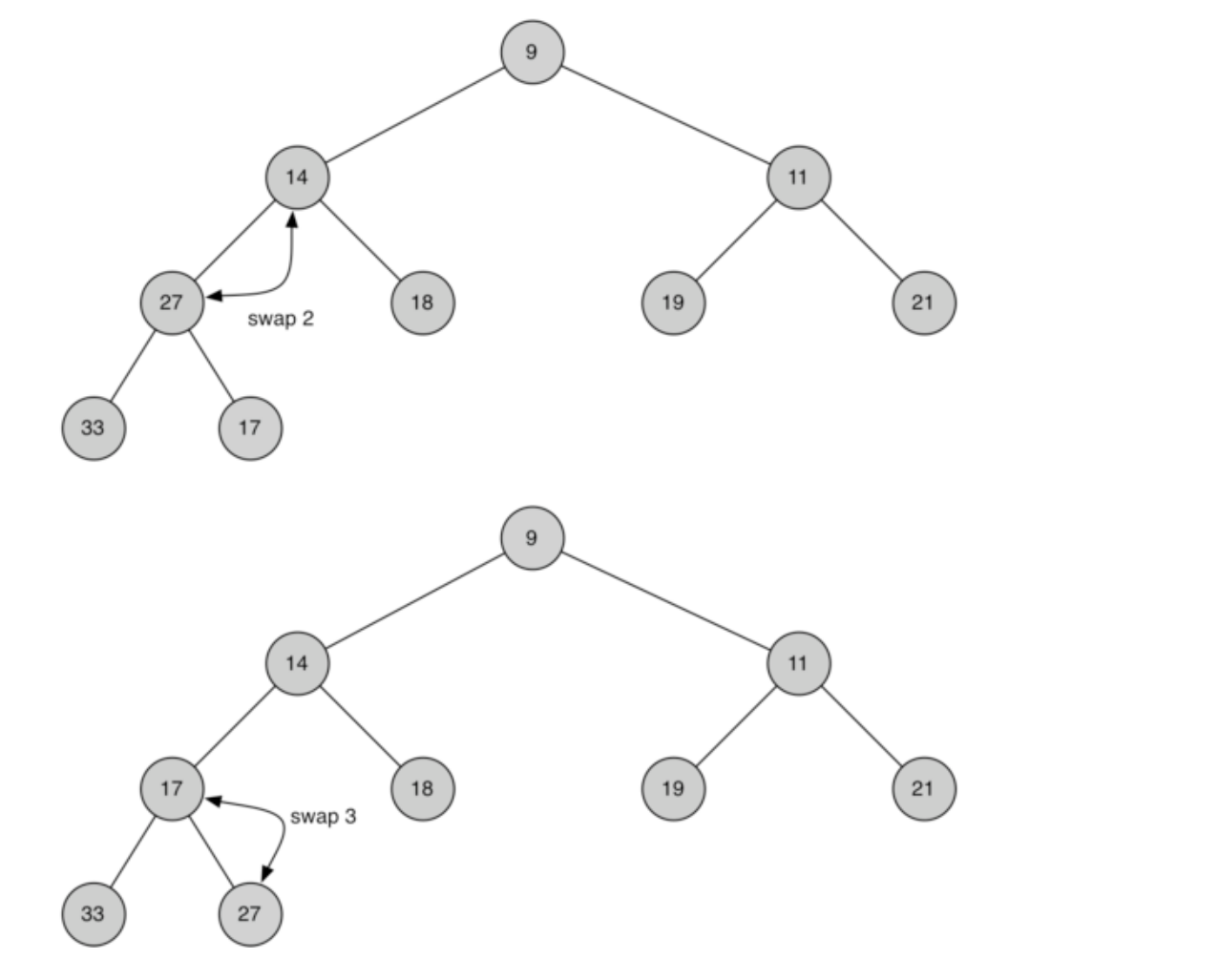
The insert method is as follows:



delMin():

Since the heap property requires that the root of the tree be the smallest node in the tree, finding the minimum item is easy. The hard part of delMin is restoring full compliance with the heap structure and heap order properties after the root has been removed. We can restore our heap in two steps. First, we will restore the root node by taking the last node in the tree and moving it to the root position. Moving the last node maintains our heap structure property. However, we have probably destroyed the heap order property of our binary heap. Second, we will restore the heap order property by pushing the new root node down the tree to its proper position. The diagram shows the series of swaps needed to move the new root node to its proper position in the heap.





Percolating the root node down the tree

In order to maintain the heap order property, all we need to do is swap the root with its smallest child less than the root. After the initial swap, we may repeat the swapping process with a node and its children until the node is swapped into a position on the tree where it is already less than both children. The code for percolating a node down the tree is found in the percDown and minChild methods below.



The code for the delete\_min operation is below.



3.

The minimum priority queue has four methods: empty(),put(),get(),pop()

empty(): its function is to determine whether the queue is empty, directly return the minimum heap size, time complexity is O(1)

put(): The goal is to insert an element into the queue, using the insert() method of the smallest heap. Suppose there are n elements in the queue, the time complexity of obtaining the parent node of the new node is O(n), and the time of adjusting upward is log(n) times. And the time complexity of getting the current node and the parent node of the current node is O(n) in the percUp() method, so the total time complexity of the percUp() is O(nlogn), so the total time complexity of the put() method is O(n)+O(nlogn) = O(nlogn).

get(): Its goal is to get the minimum value of the queue and return directly to the root node of the minimum heap. The time complexity is O(1).

pop(): The dequeuing operation, which removes the root node from the heap, is implemented using the delMin() method of the minimum heap. Suppose there are n elements in the queue, the time complexity when exchanging the value of the node and deleting the last node is O(n),and the times of adjusting down is log(n), and the time complexity of obtaining the current node and the child node of the current node is O(n). Therefore, the total time complexity of downward adjustment is O(nlogn), so the total time degree of pop method is O(n)+O(nlogn) = O(nlogn).