The Art of Data Structures

Binary Heaps

Alan Beadle
CSC 162: The Art of Data Structures
UNIVERSITY of ROCHESTER

Agenda

- Binary Heap Operations
- Binary Heap Implementation

Binary Heaps

Binary Heap
Specification

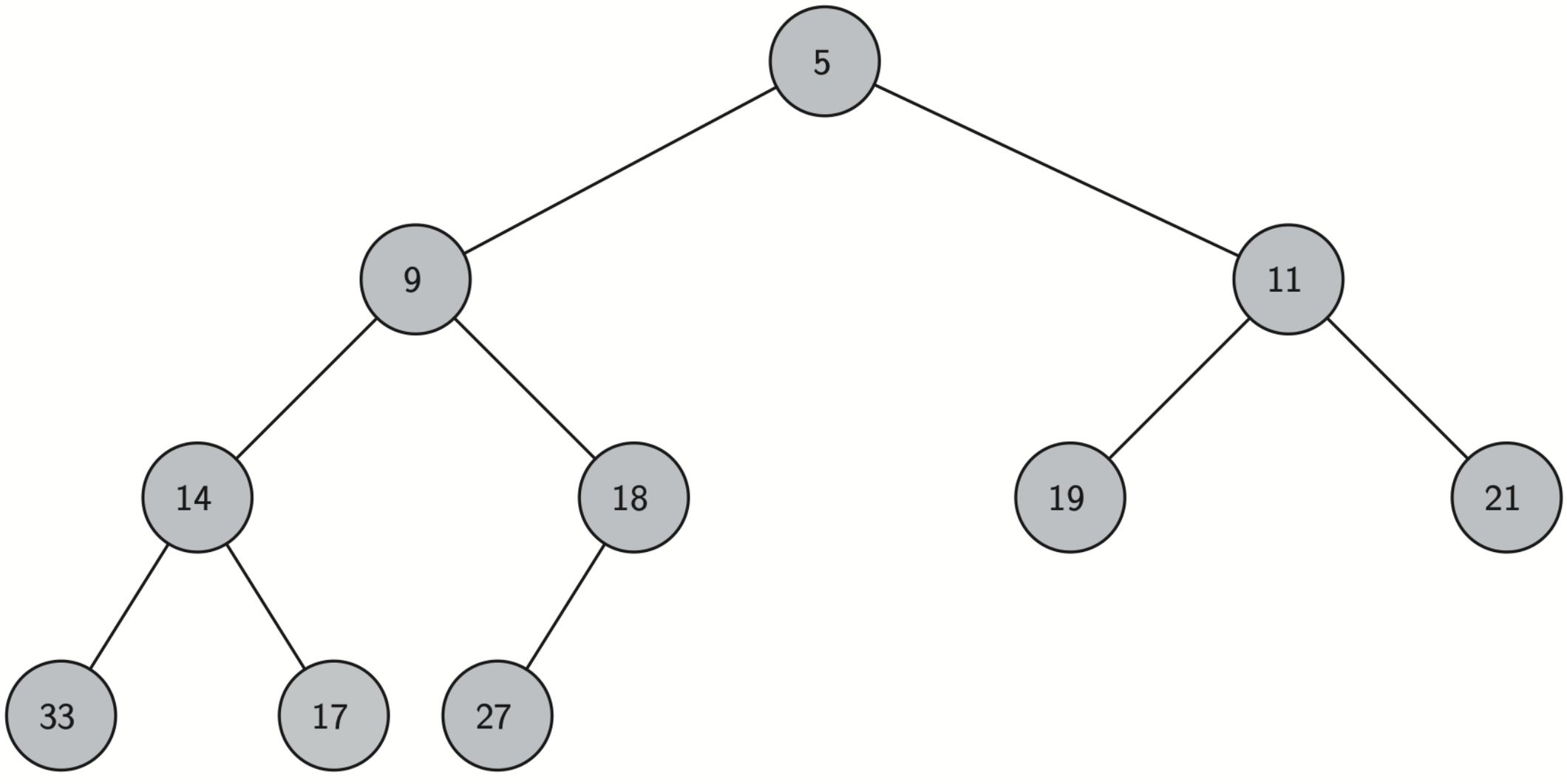
- `BinaryHeap()` creates a new binary heap
- `insert(k)` adds a new item to the heap
- `find_min()` returns the item with the minimum key value, leaving item in the heap
- `del_min()` returns the item with the minimum key value, removing the item from the heap

Binary Heap
Specification (cont.)

- `is_empty()` returns true if the heap is empty, false otherwise
- `size()` returns the number of items in the heap
- `buid_heap(list)` builds a new heap from a list of keys
- `decrease_key(k)` finds a key in the heap and updates its key value to a new lower value

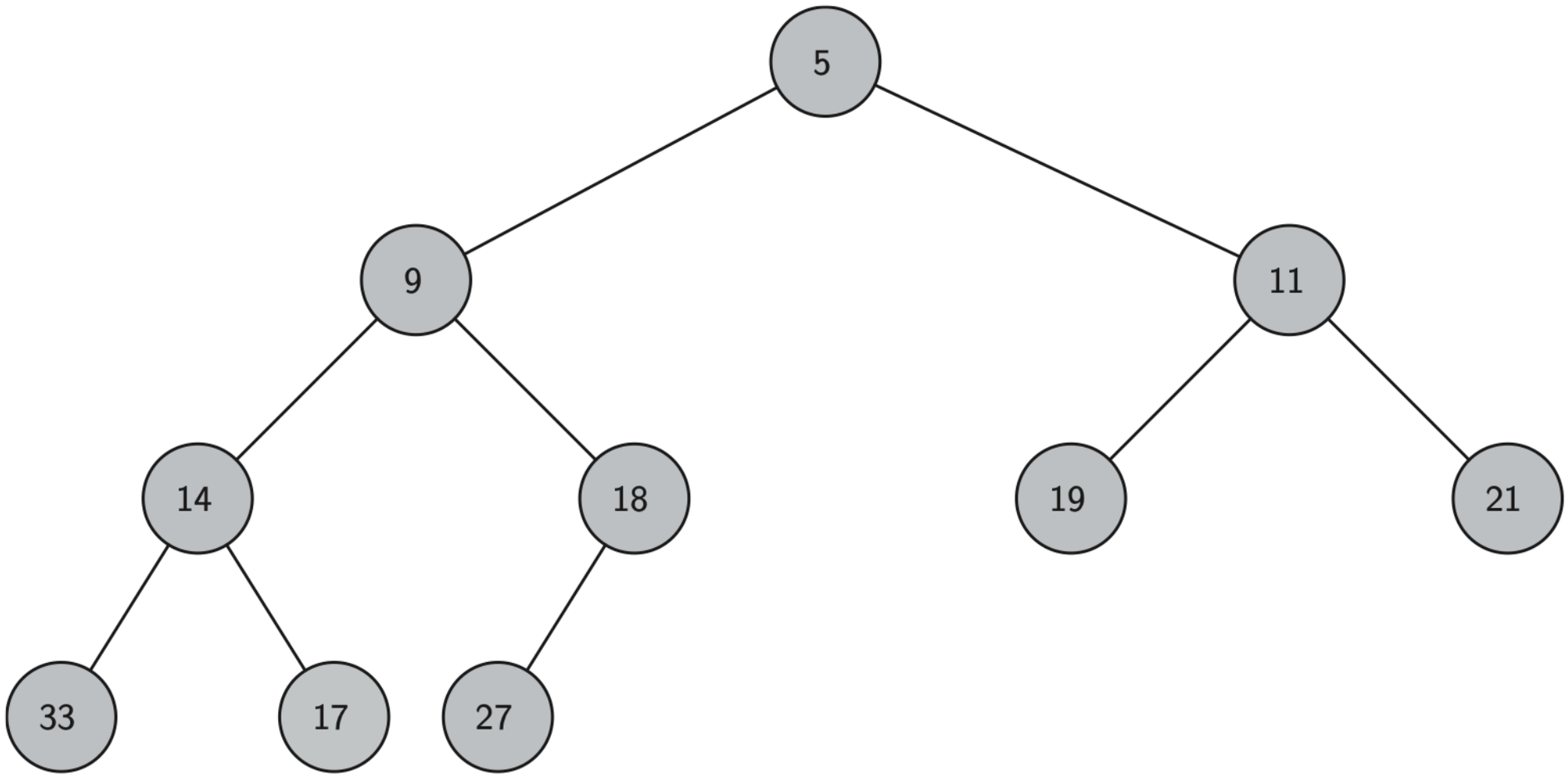
Binary Heap

A Complete Binary Tree



Binary Heap

Complete Binary Tree with List Representation



0	5	9	11	14	18	19	21	33	17	27	
0	1	2	3	4	5	6	7	8	9	10	11

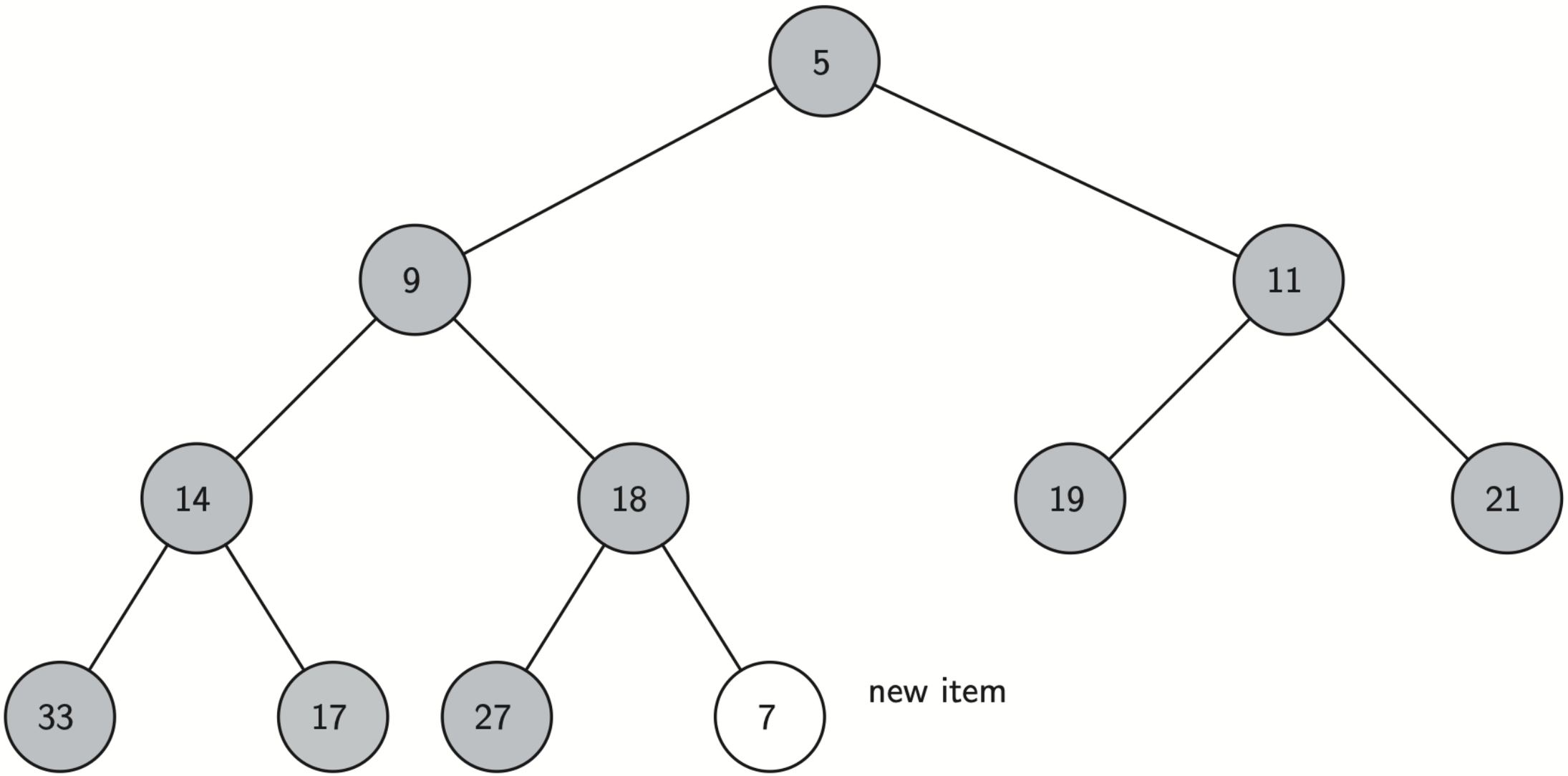
Binary Heap

Create a New Binary Heap

```
class BinHeap:
    def __init__(self):
        self.heap_list = [0]
        self.current_size = 0
```

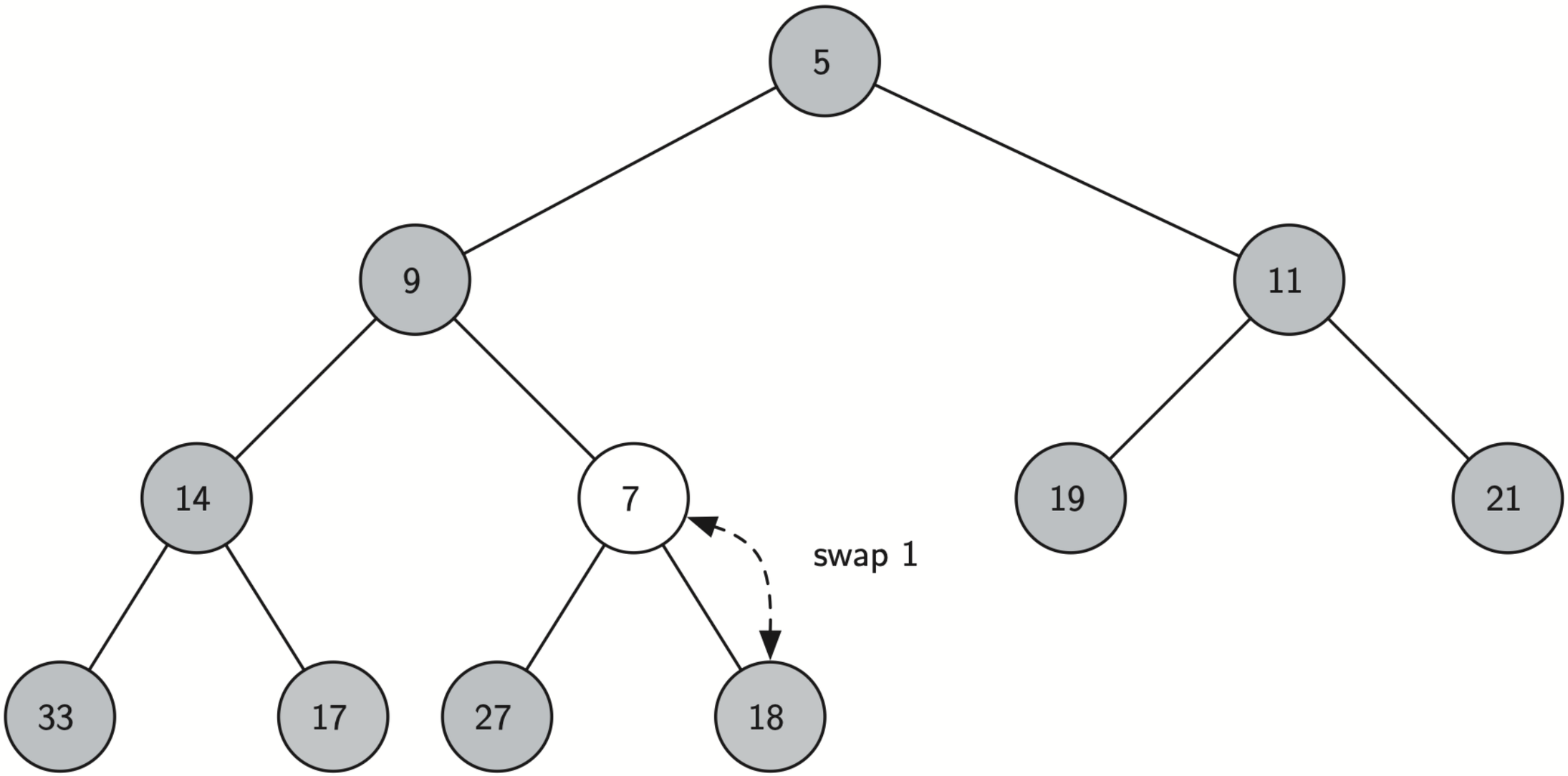
Binary Heap

Percolate the New Node up to its Proper Position



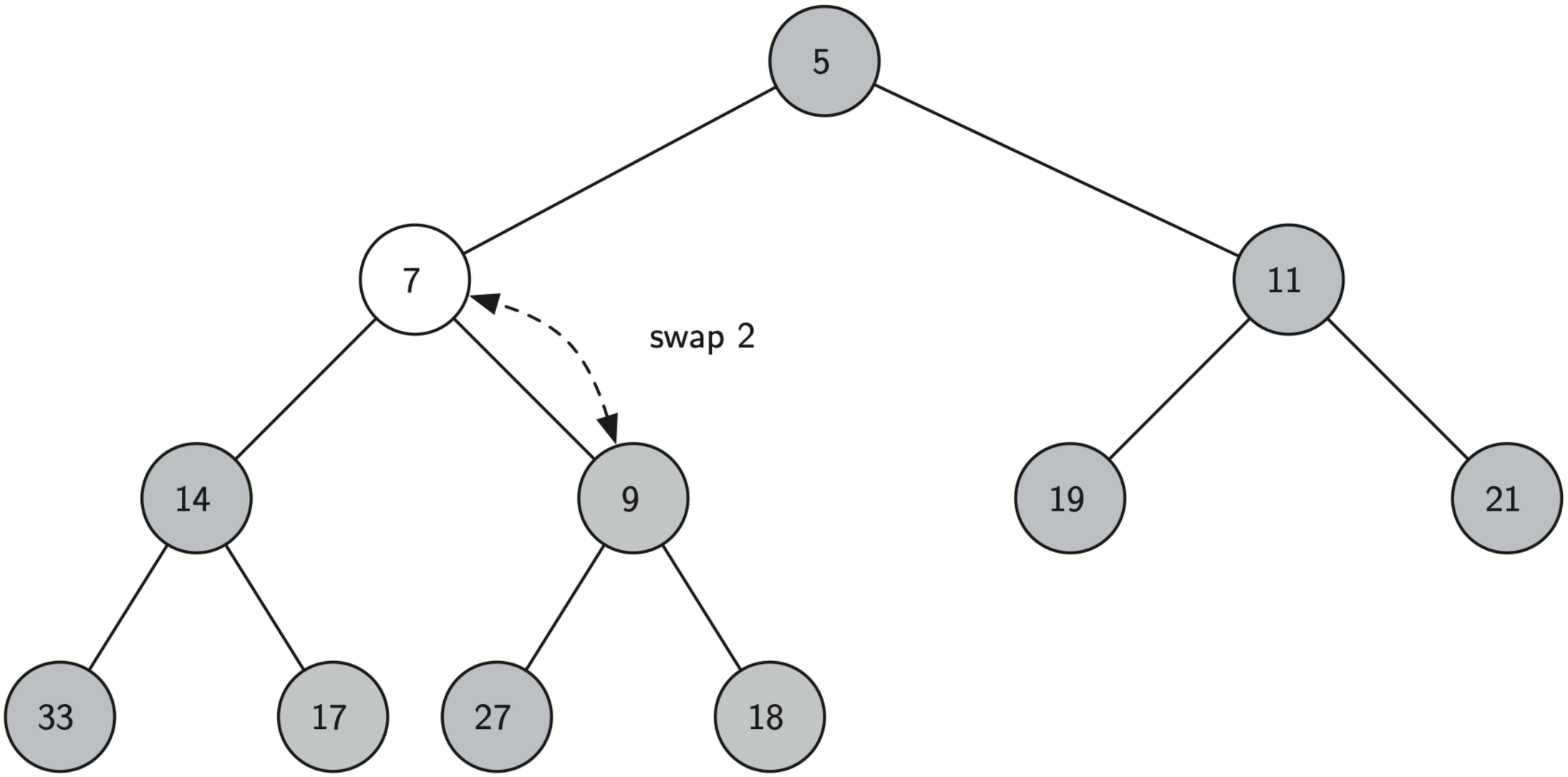
Binary Heap

Percolate the New Node up to its Proper Position



Binary Heap

Percolate the New Node up to its Proper Position



Binary Heap

The `percUp` *Method*

```
def perc_up(self,i):
    while i // 2 > 0:
      if self.heap_list[i] < self.heap_list[i // 2]:
         tmp = self.heap_list[i // 2]
         self.heap_list[i // 2] = self.heap_list[i]
         self.heap_list[i] = tmp
      i = i // 2
```

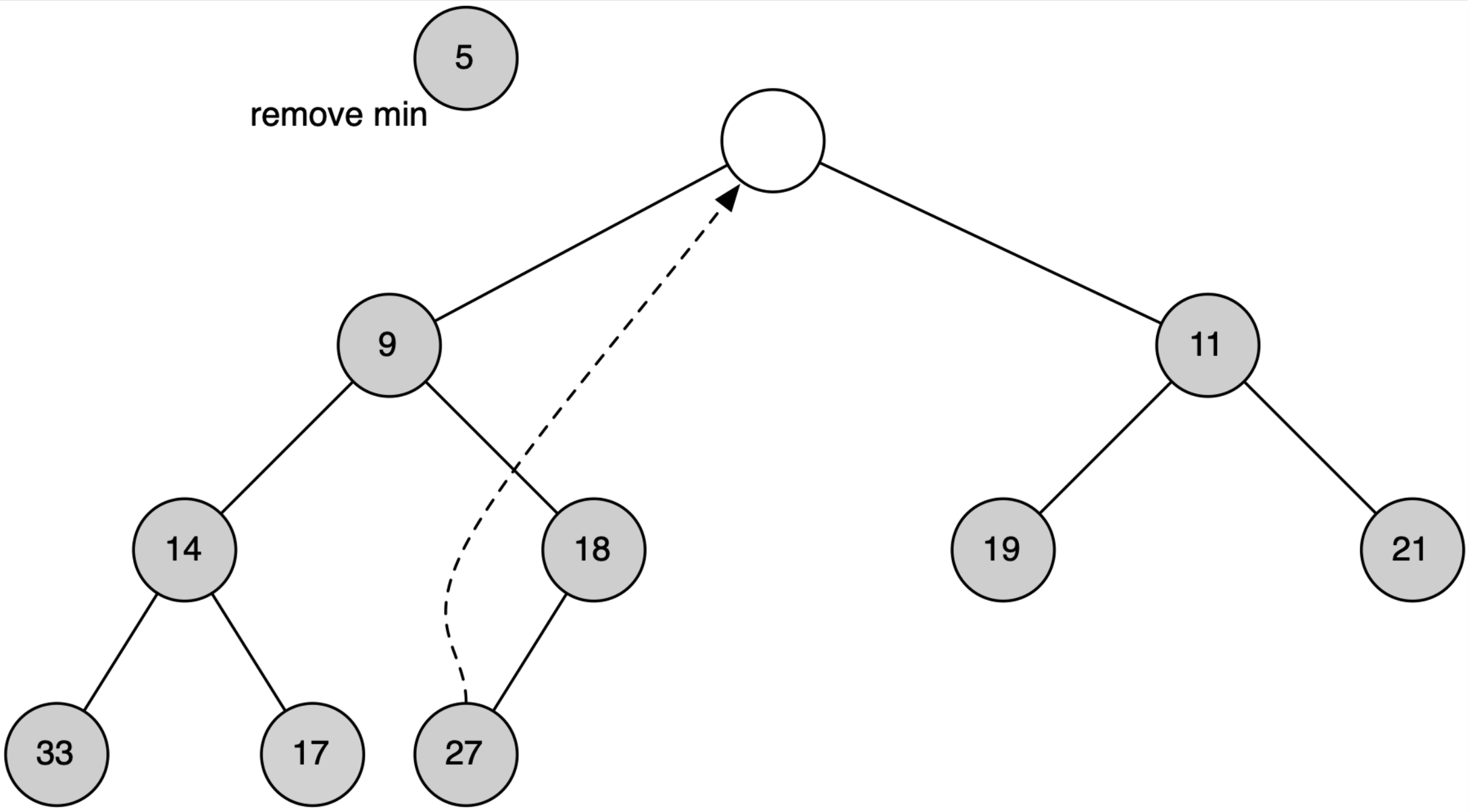
Binary Heap

Adding a New Item to the Binary Heap

```
def insert(self, k):
    self.heap_list.append(k)
    self.current_size = self.current_size + 1
    self.perc_up(self.current_size)
```

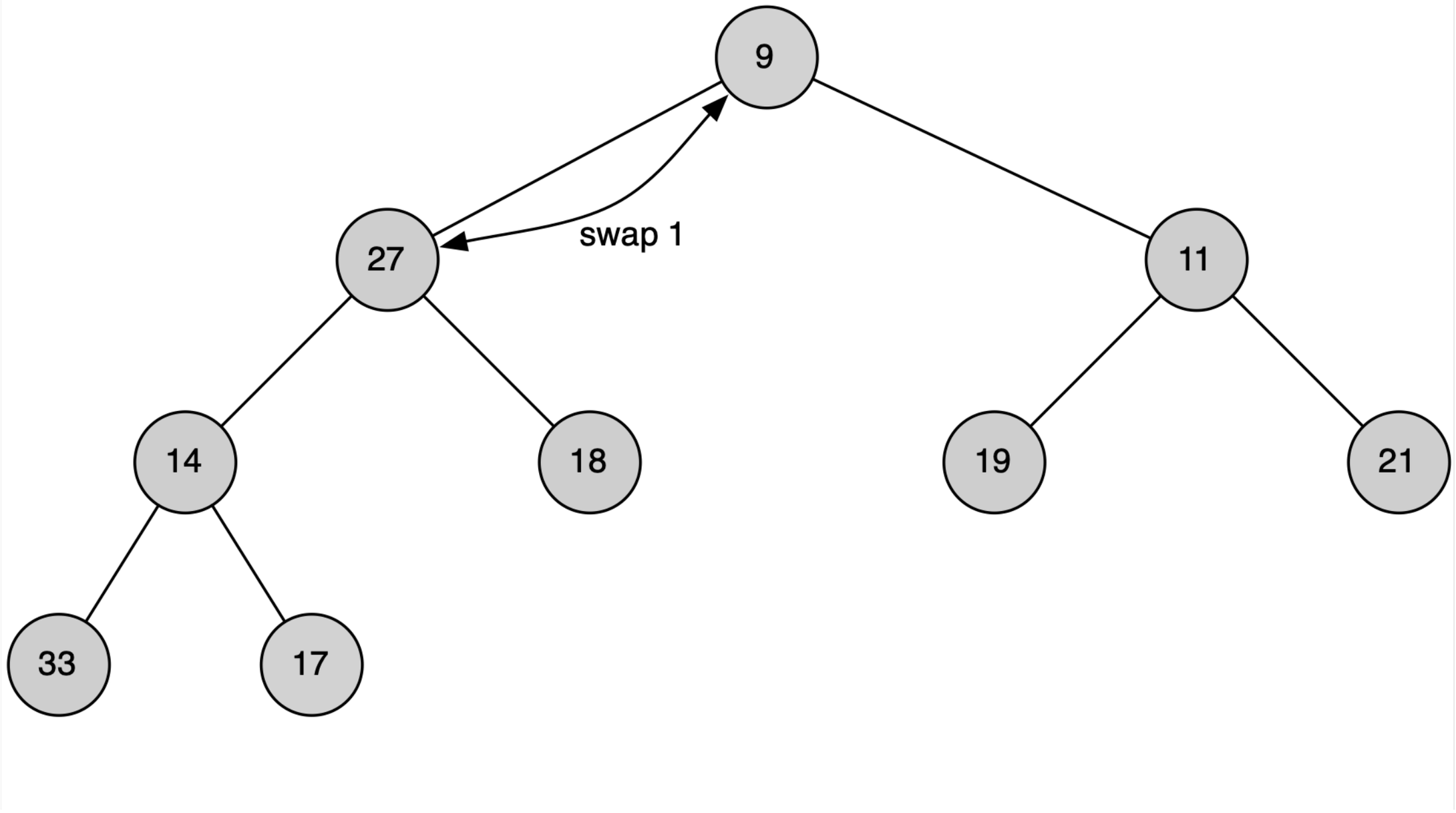
Binary Heap

Percolating the Root Node down the Tree



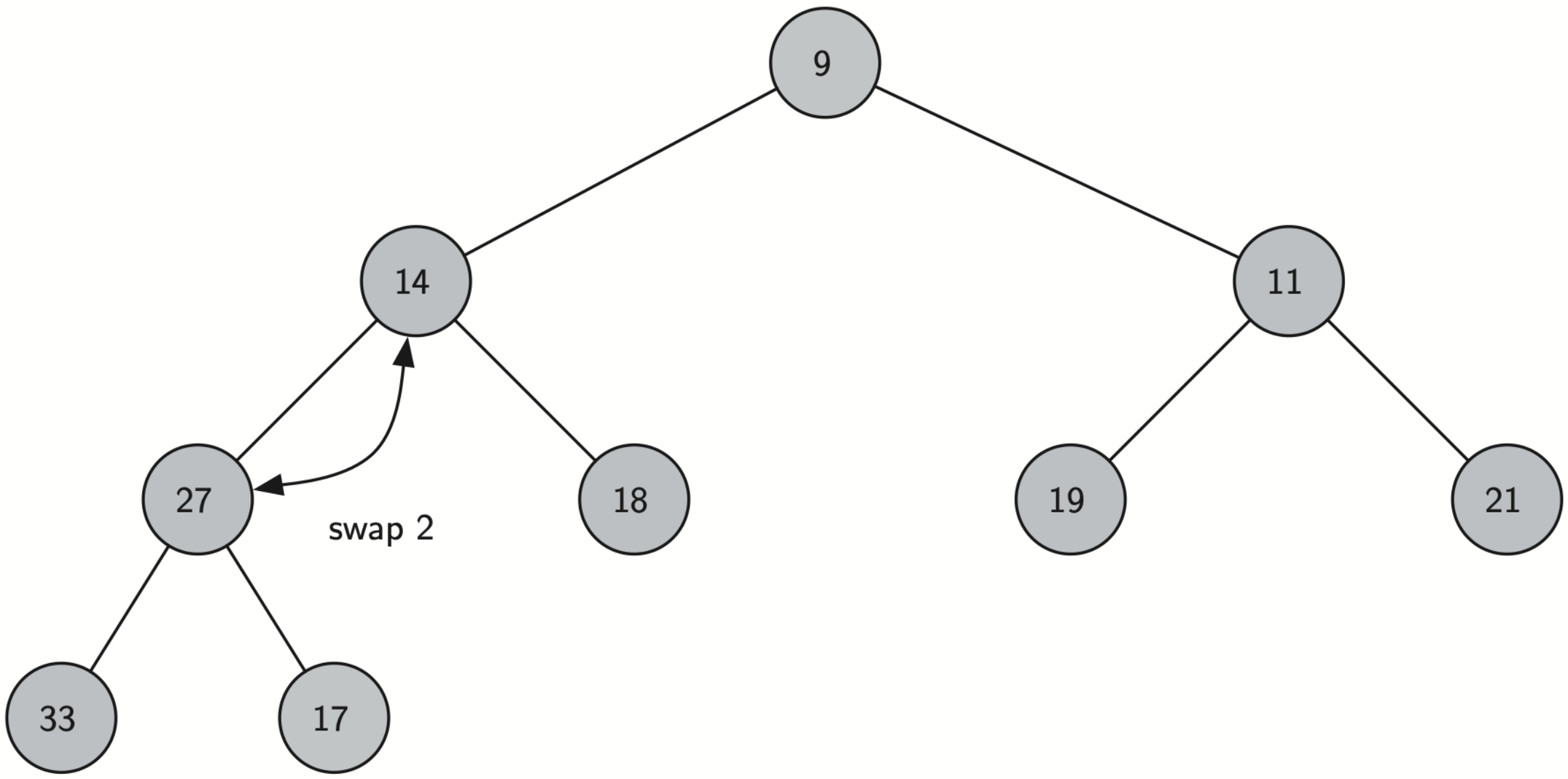
Binary Heap

Percolating the Root Node down the Tree



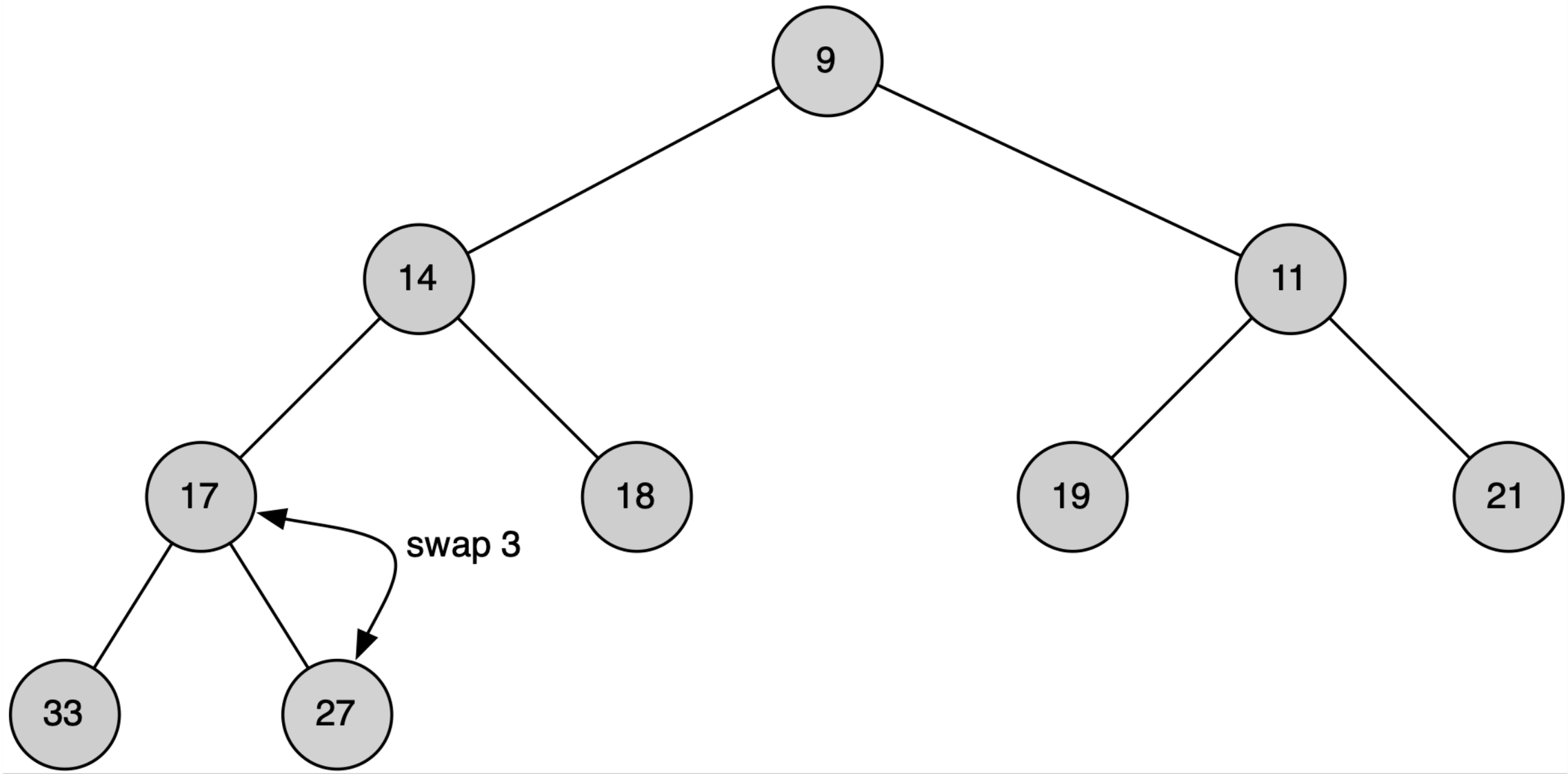
Binary Heap

Percolating the Root Node down the Tree



Binary Heap

Percolating the Root Node down the Tree



Binary Heap

The `percDown` *Method*

```
def perc_down(self, i):
    while (i * 2) <= self.current_size:
        mc = self.min_child(i)
        if self.heap_list[i] > self.heap_list[mc]:
            tmp = self.heap_list[i]
            self.heap_list[i] = self.heap_list[mc]
            self.heap_list[mc] = tmp
        i = mc
```

Binary Heap

The `minChild` *Method*

```
def min_child(self,i):
    if i * 2 + 1 > self.current_size:
        return i * 2
    else:
        if self.heap_list[i*2] < self.heap_list[i*2+1]:
            return i * 2
        else:
            return i * 2 + 1
```

Binary Heap

Deleting the Minimum Item from the Binary Heap

```
def del_min(self):
    retval = self.heap_list[1]
    self.heap_list[1] = self.heap_list[self.current_size]
    self.current_size = self.current_size - 1
    self.heap_list.pop()
    self.percDown(1)
    return retval
```

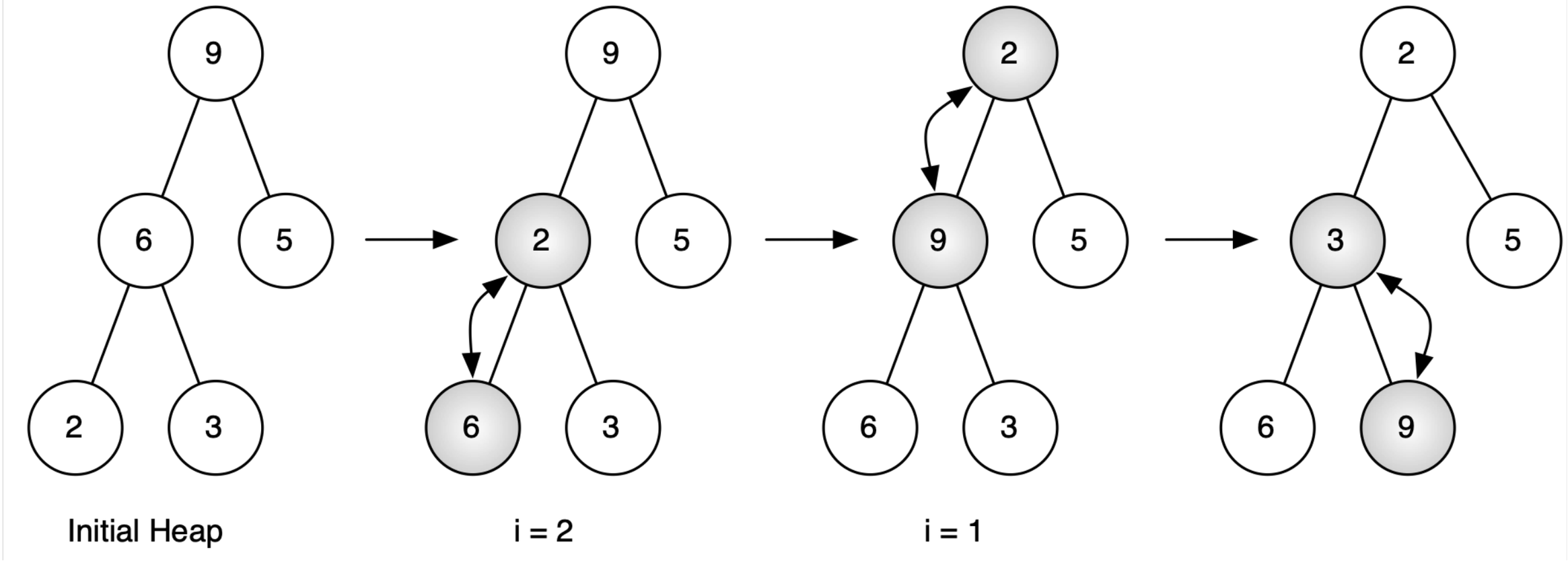
Binary Heap

Building a New Heap from a List of Items

```
def build_heap(self, alist):
    i = len(alist) // 2
    self.current_size = len(alist)
    self.heap_list = [0] + alist[:]
    while (i > 0):
        self.perc_down(i)
        i = i - 1
```

Binary Heap

Building a New Heap from a List of Items



Binary Heap

Building a Heap from the [9, 5, 6, 2, 3]

```
i=2 [0,9,5,6,2,3]
i=1 [0,9,2,6,5,3]
i=0 [0,2,3,6,5,9]
```

Binary Heap

Building a Heap from the [9, 5, 6, 2, 3]

```
bh = BinHeap()
bh.build_heap([9,5,6,2,3])

print(bh.del_min())
print(bh.del_min())
print(bh.del_min())
print(bh.del_min())
print(bh.del_min())
```

Questions?