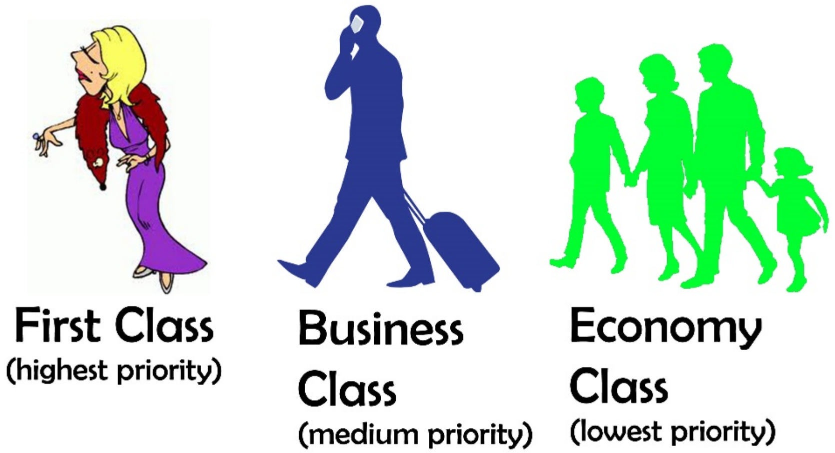
Priority Queues - Binary Heap

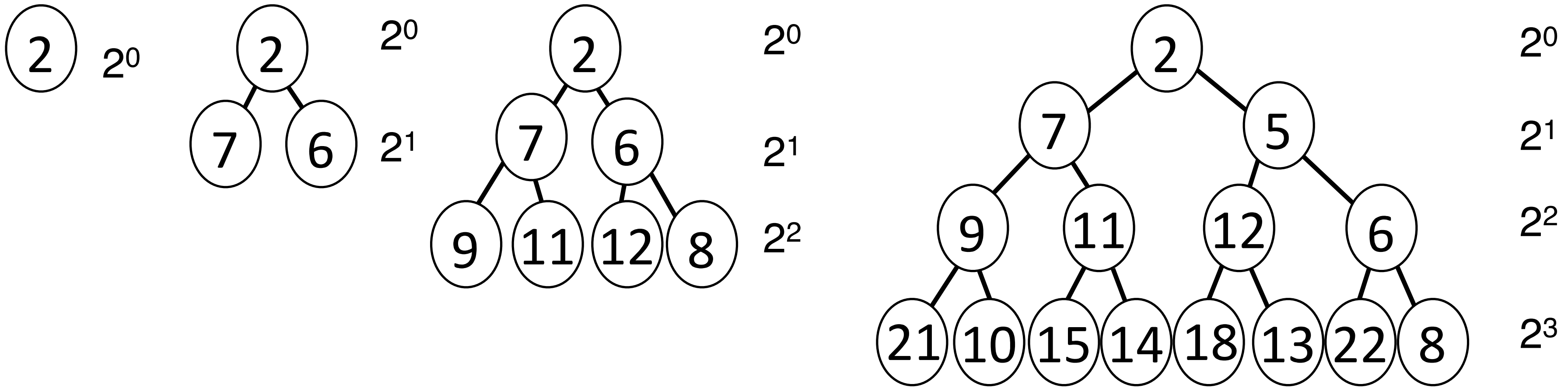
Priority of airline passengers



Priority of patients

Full Binary Tree

A binary tree is **full** if all its leaves are on the same level.
The number of nodes in level *k* of a full binary tree is 2^k

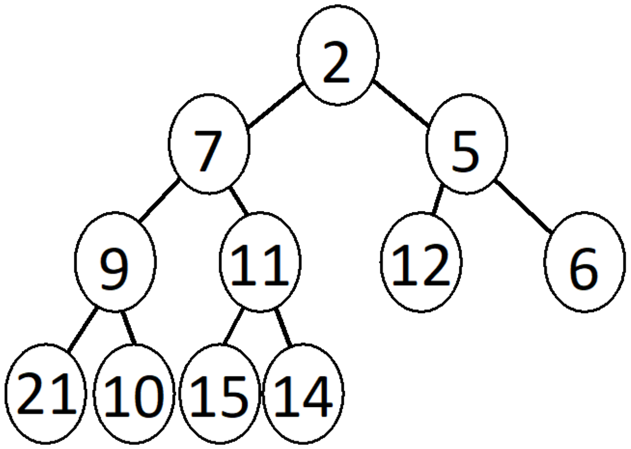


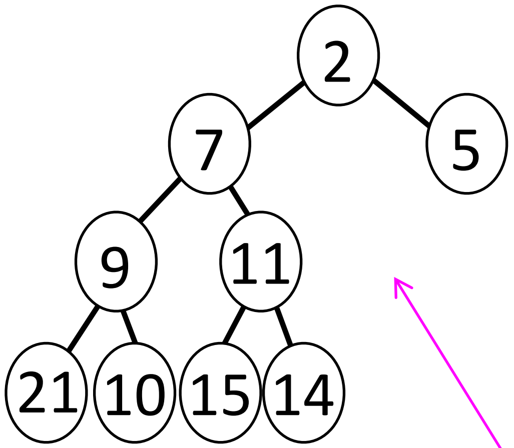
QUESTION:

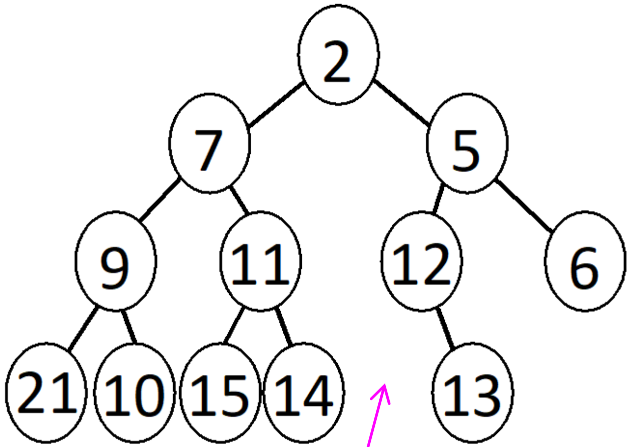
How many nodes does a full binary tree of height *h* have? **$2^{h+1} - 1$**

Complete Binary Tree

A **complete** binary tree has all levels full except the last one.
The last level is filled from the left.





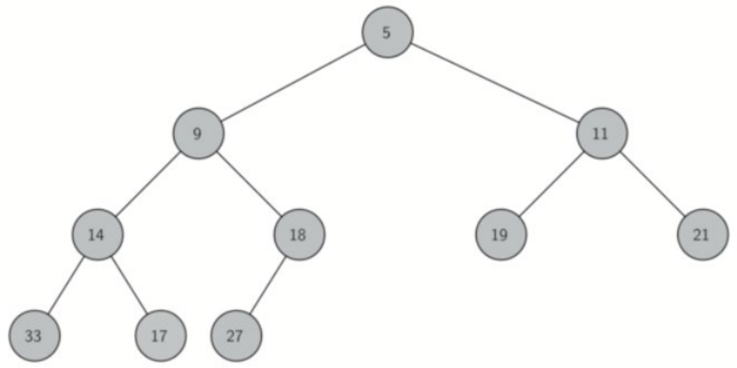


Complete

Not complete

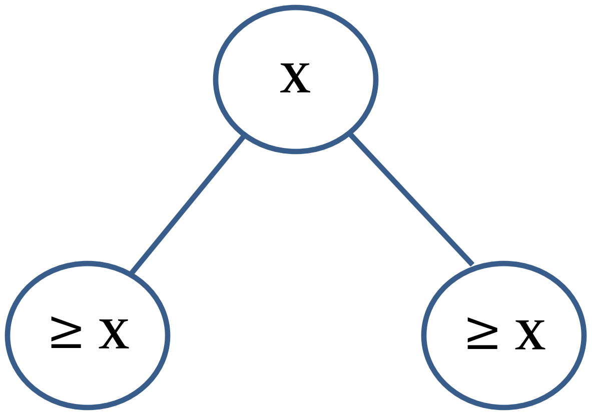
Binary heap

- The classic way to implement a priority queue is with a **binary heap**
- A binary heap has two special properties:



a *complete* binary tree
Each level has all possible nodes, except for the bottom level which is filled from left to right

shape property

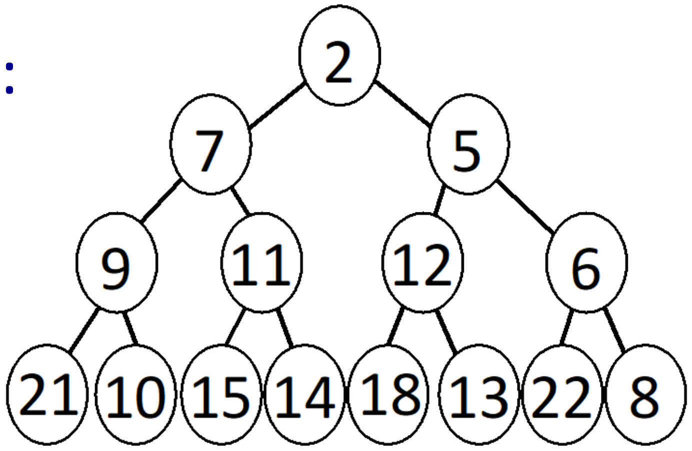


a **partial** ordering over nodes
the key at every parent node is less than or equal to **both** of its children

order property

Binary Heap – Stored in List

A complete binary tree is efficiently stored using a list:



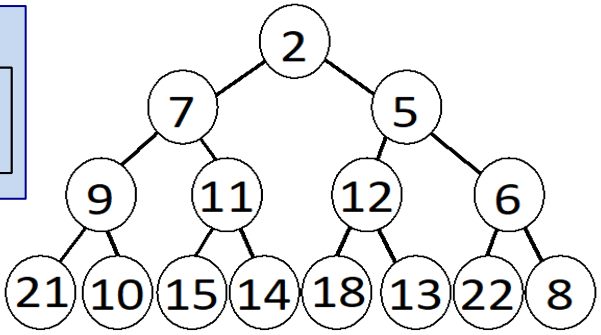
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	7	5	9	11	12	6	21	10	15	14	18	13	22	8

For convenience we are going to leave the first element blank and store the root element in position 1:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	2	7	5	9	11	12	6	21	10	15	14	18	13	22	8

Binary Heap

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	2	7	5	9	11	12	6	21	10	15	14	18	13	22	8



QUESTIONS

What indices are the children of node at index 6 in the list?

What indices are the children of node at index i in the list?

What is the index of the parent of the node at index 6?

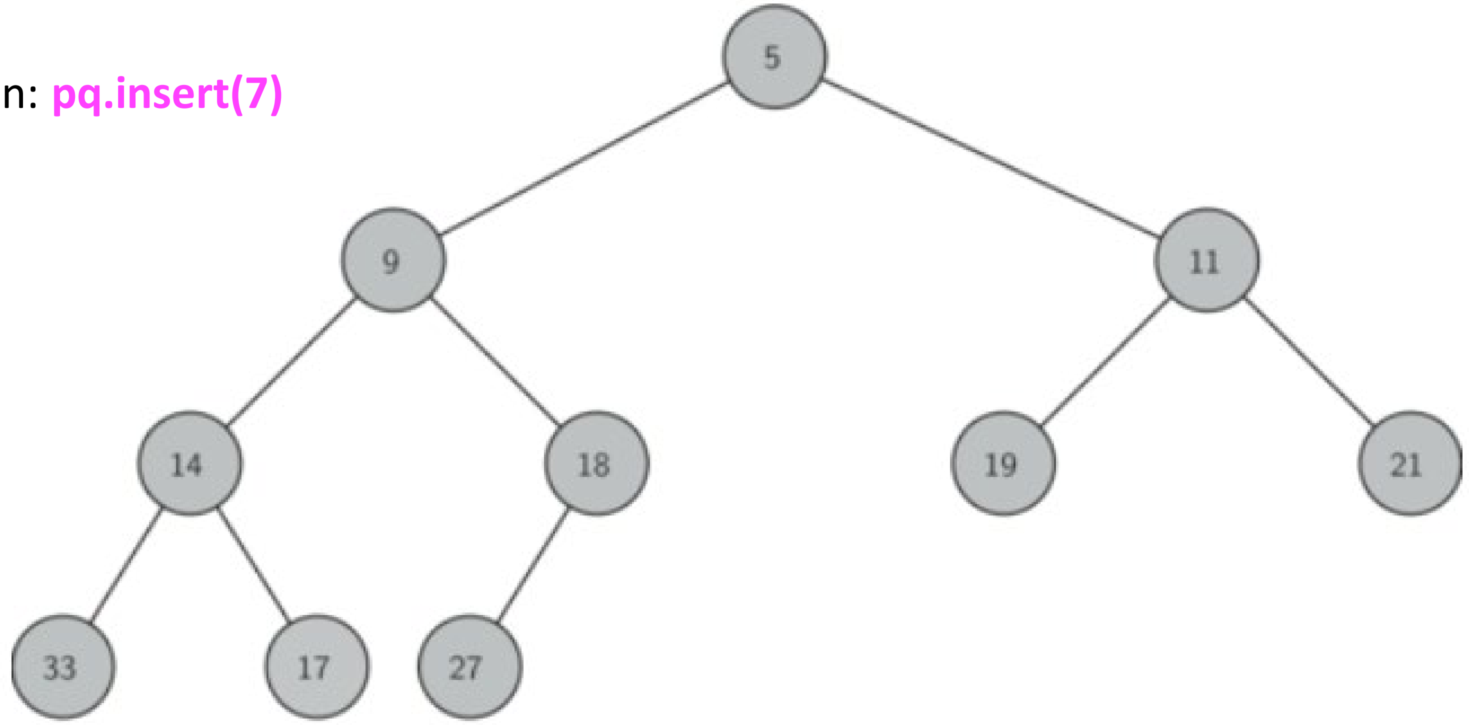
Children of node L[i] are L[2i] and L[2i+1]

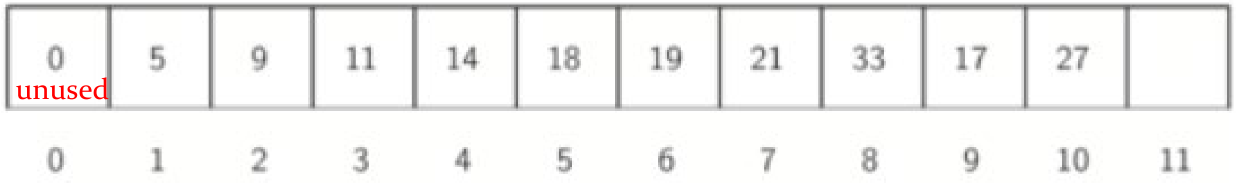
Parent of node L[i] is L[i // 2]

Heap operations

- Let’s look at the important heap operations (**insert** and **delete_minimum**) in the context of the following heap

Insertion: pq.insert(7)

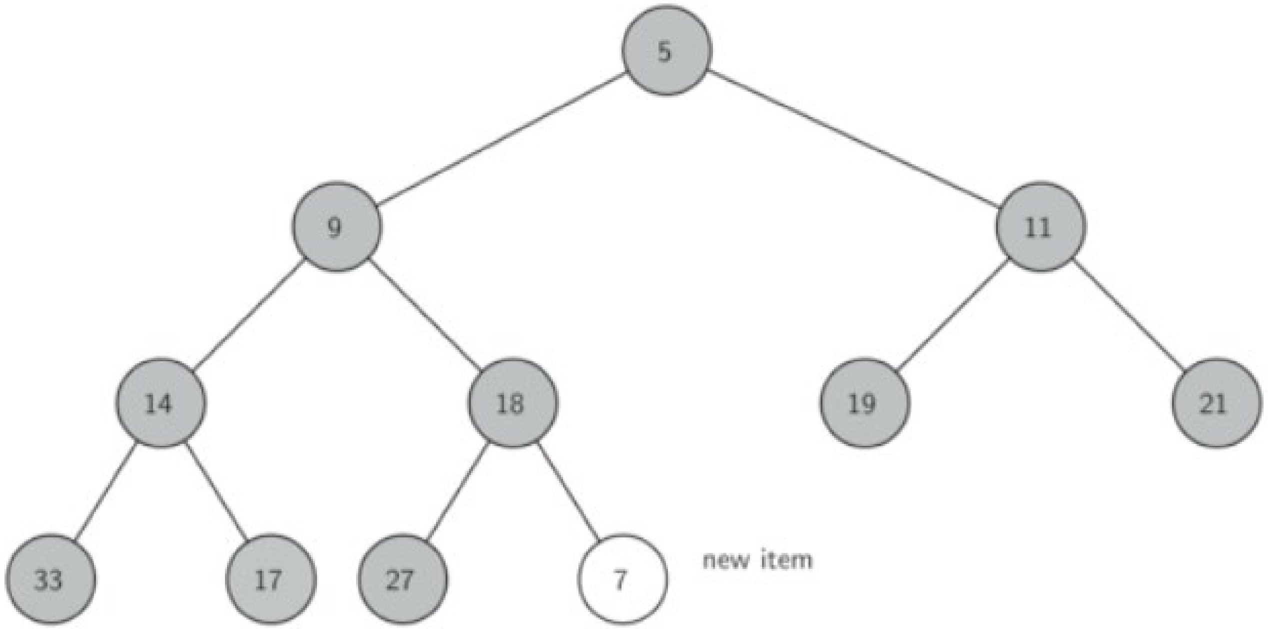




0 unused	5	9	11	14	18	19	21	33	17	27	
0	1	2	3	4	5	6	7	8	9	10	11

Heap operations

- Insertion: **pq.insert(7)**

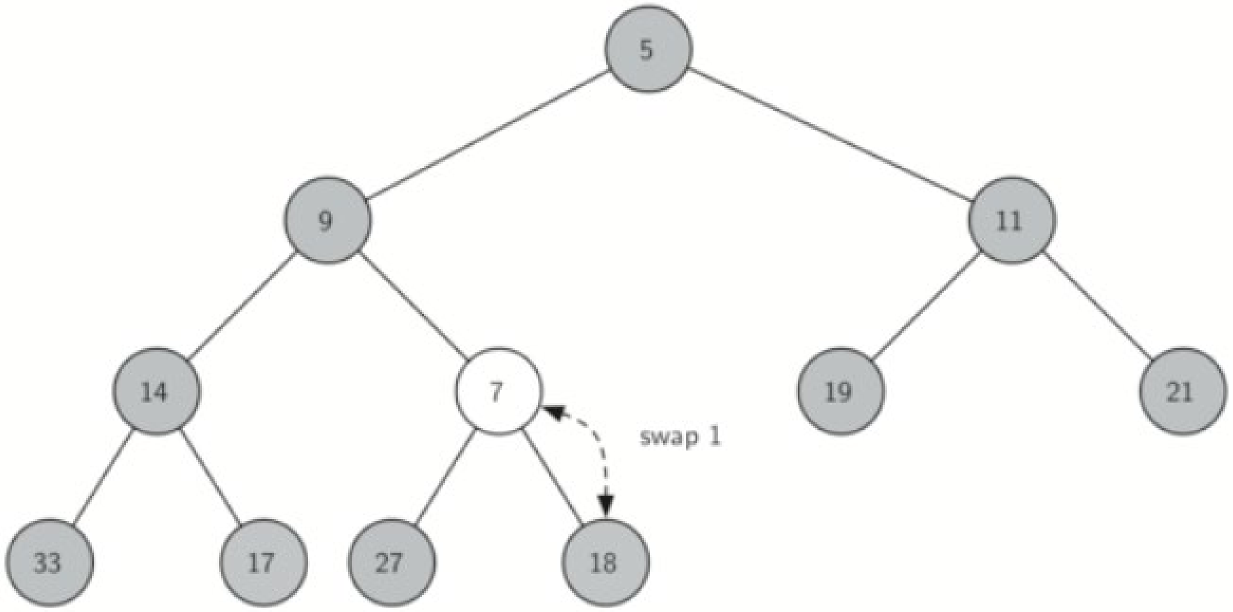


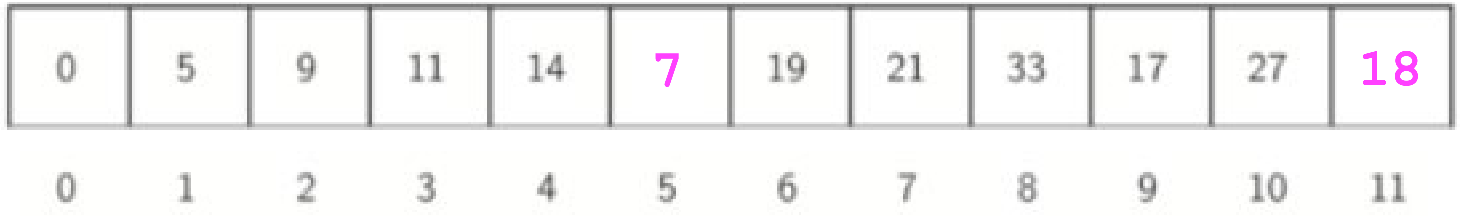
```
self.__binary_heap.append(7)
```

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

Heap operations

- Insertion: **pq.insert(7)**

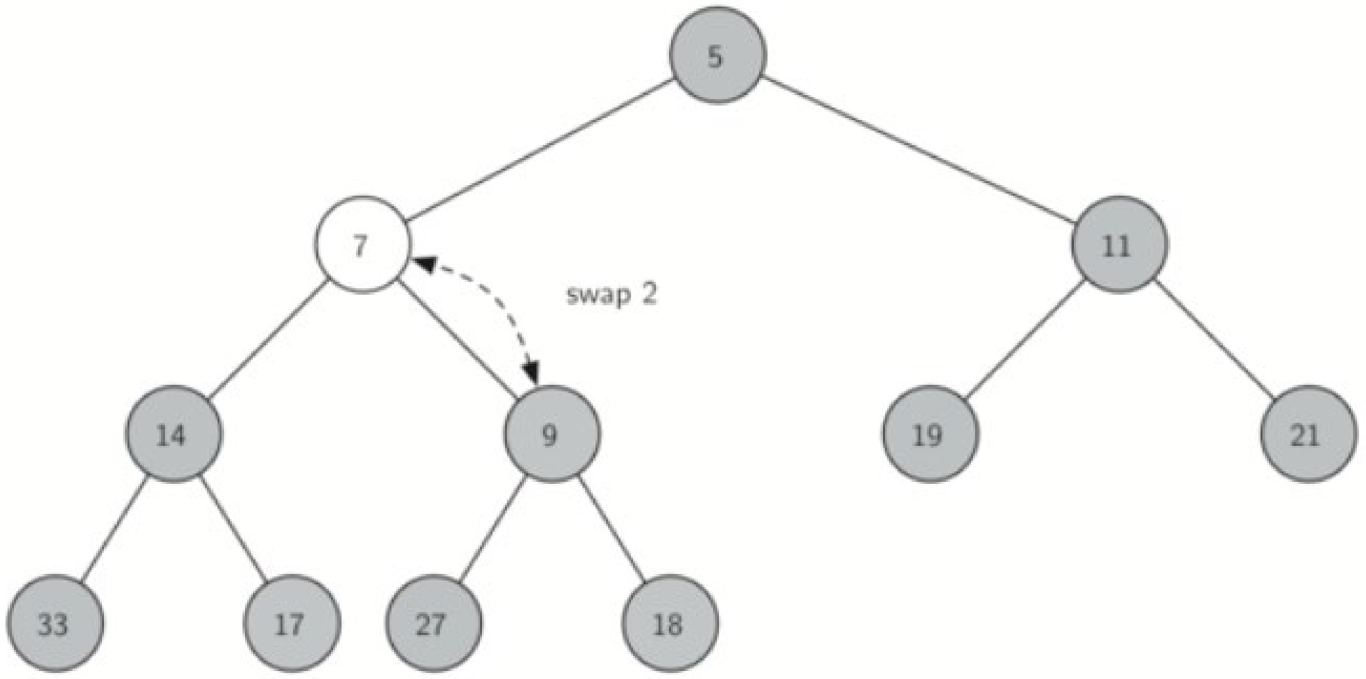




0	5	9	11	14	7	19	21	33	17	27	18
0	1	2	3	4	5	6	7	8	9	10	11

Heap operations

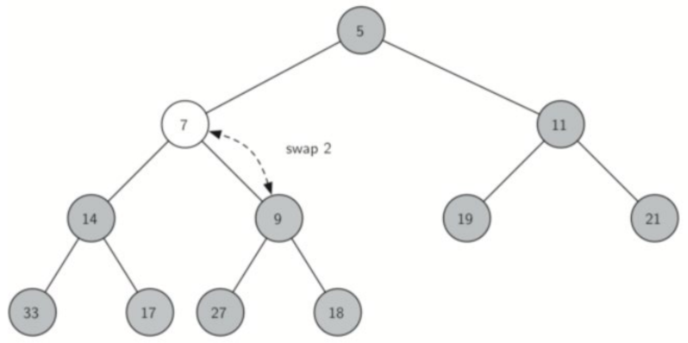
- Insertion: **pq.insert(7)**



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

Heap operations

- Insertion: **pq.insert(7)**



```
def percolate_up(self,i):
  while i // 2 > 0:
    if self.__binary_heap[i] < self.__binary_heap[i // 2]:
```

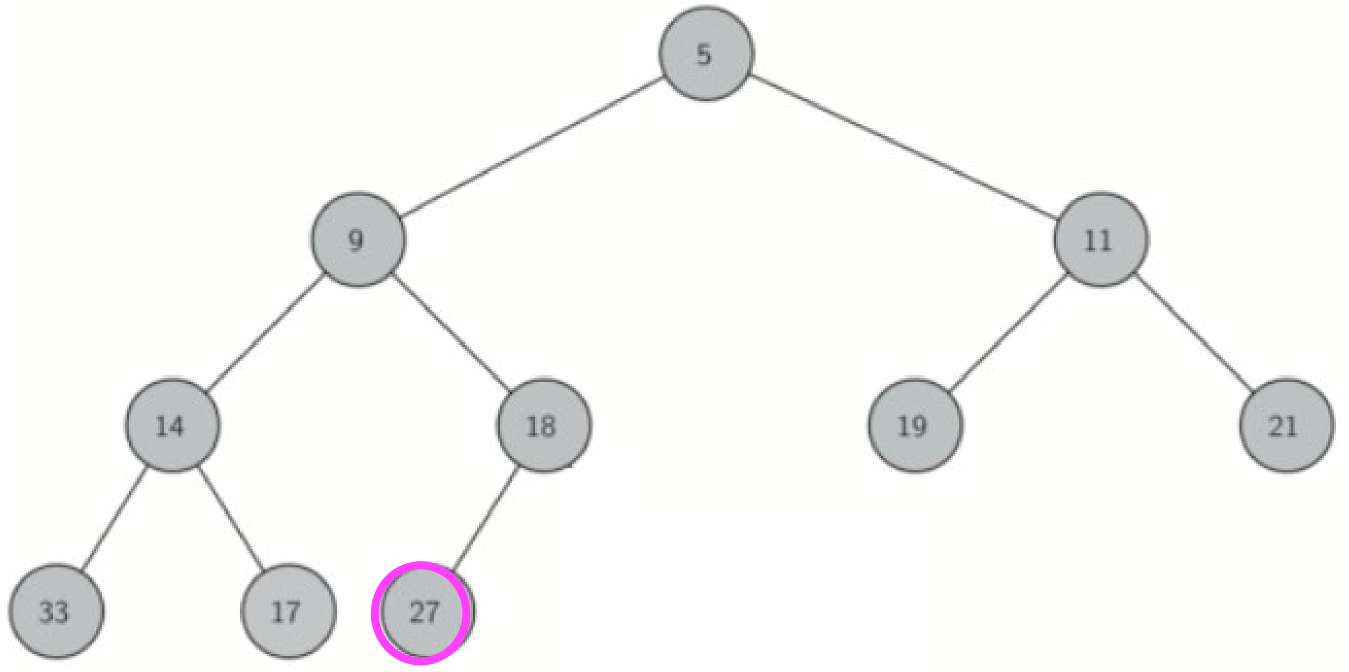
swap **self.__binary_heap**[i//2] and **self.__binary_heap**[i]

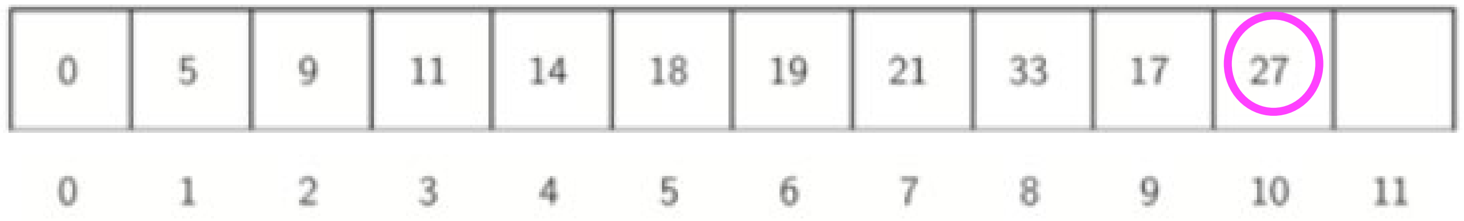
```
    i = i // 2

def insert(self,k):
  self__binary_heap.append(k)
  self.__size = self.__size + 1
  self.percolate_up(self.__size)
```

Heap operations

- Deletion : **pq.delete_minimum ()**

To maintain the **shape property**, we have to replace the **root value** with the **very bottom leaf** node



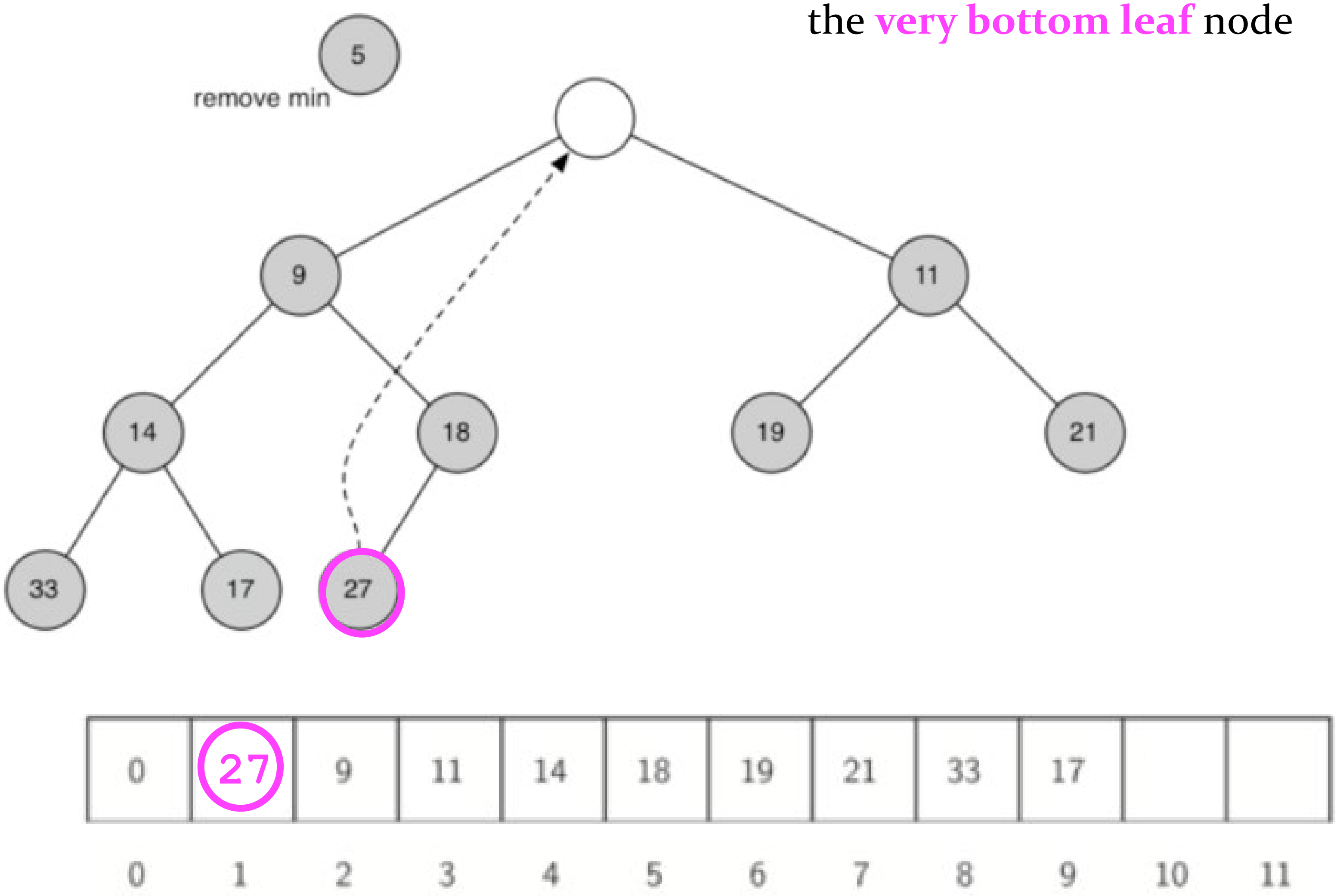


0	5	9	11	14	18	19	21	33	17	27	
0	1	2	3	4	5	6	7	8	9	10	11

Heap operations

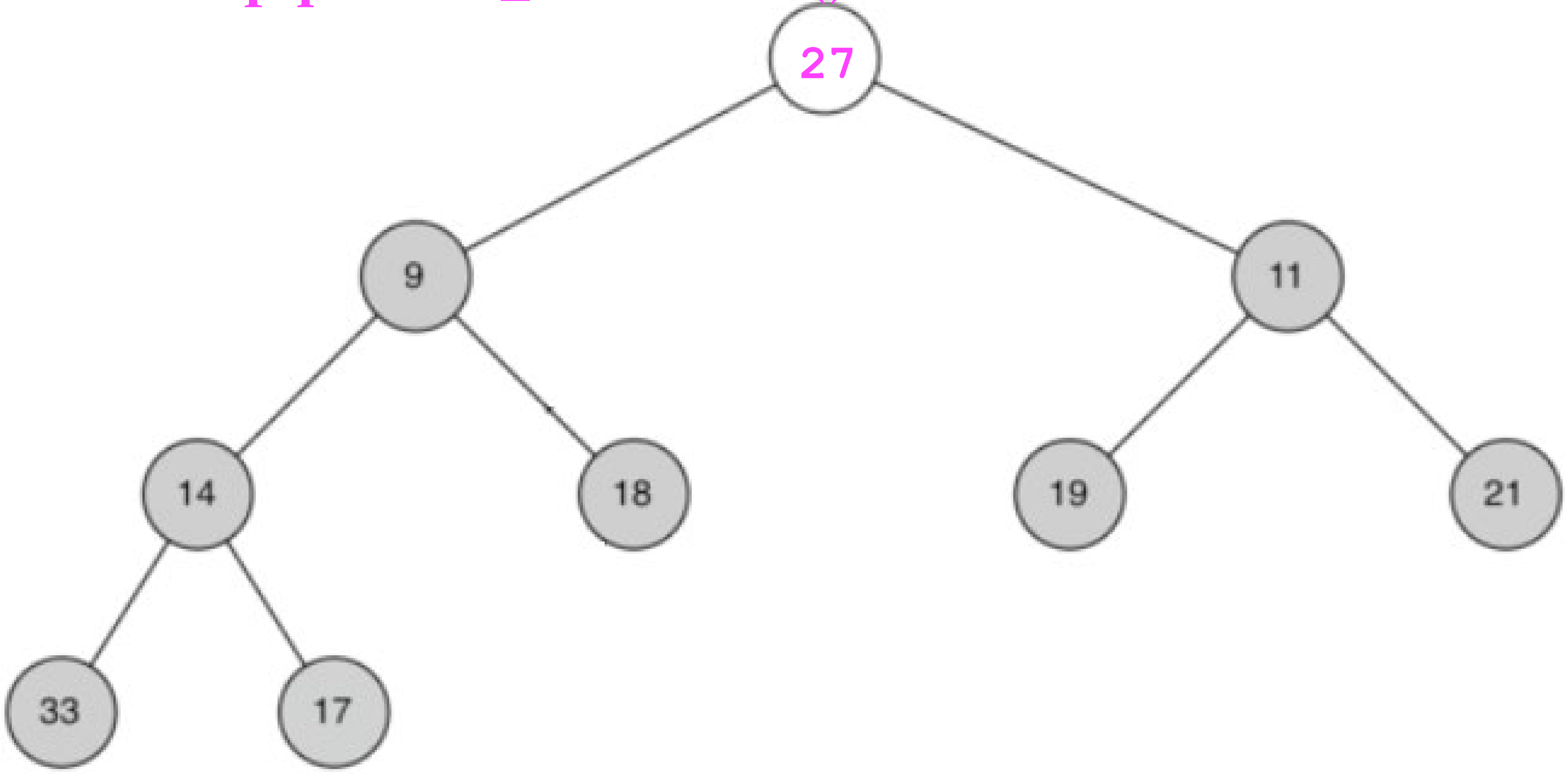
- Deletion : **pq.delete_minimum()**

To maintain the **shape property**, we have to replace the **root value** with the **very bottom leaf** node



Heap operations

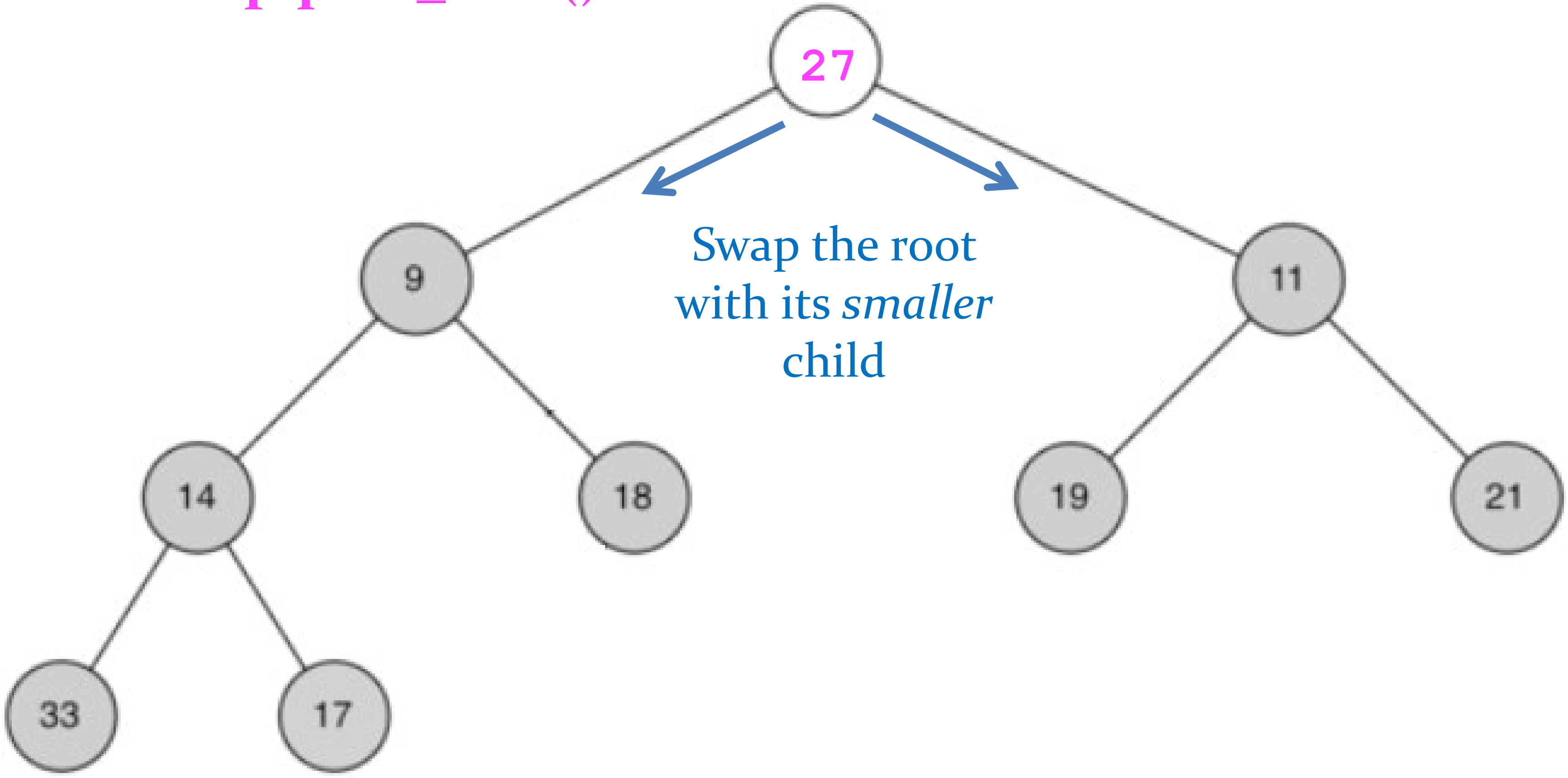
- Deletion : **pq.delete_minimum()**



We have maintained the shape property, but now we have broken the **order property**. *How can we restore it?*

Heap operations

- Deletion : **pq.del_min()**

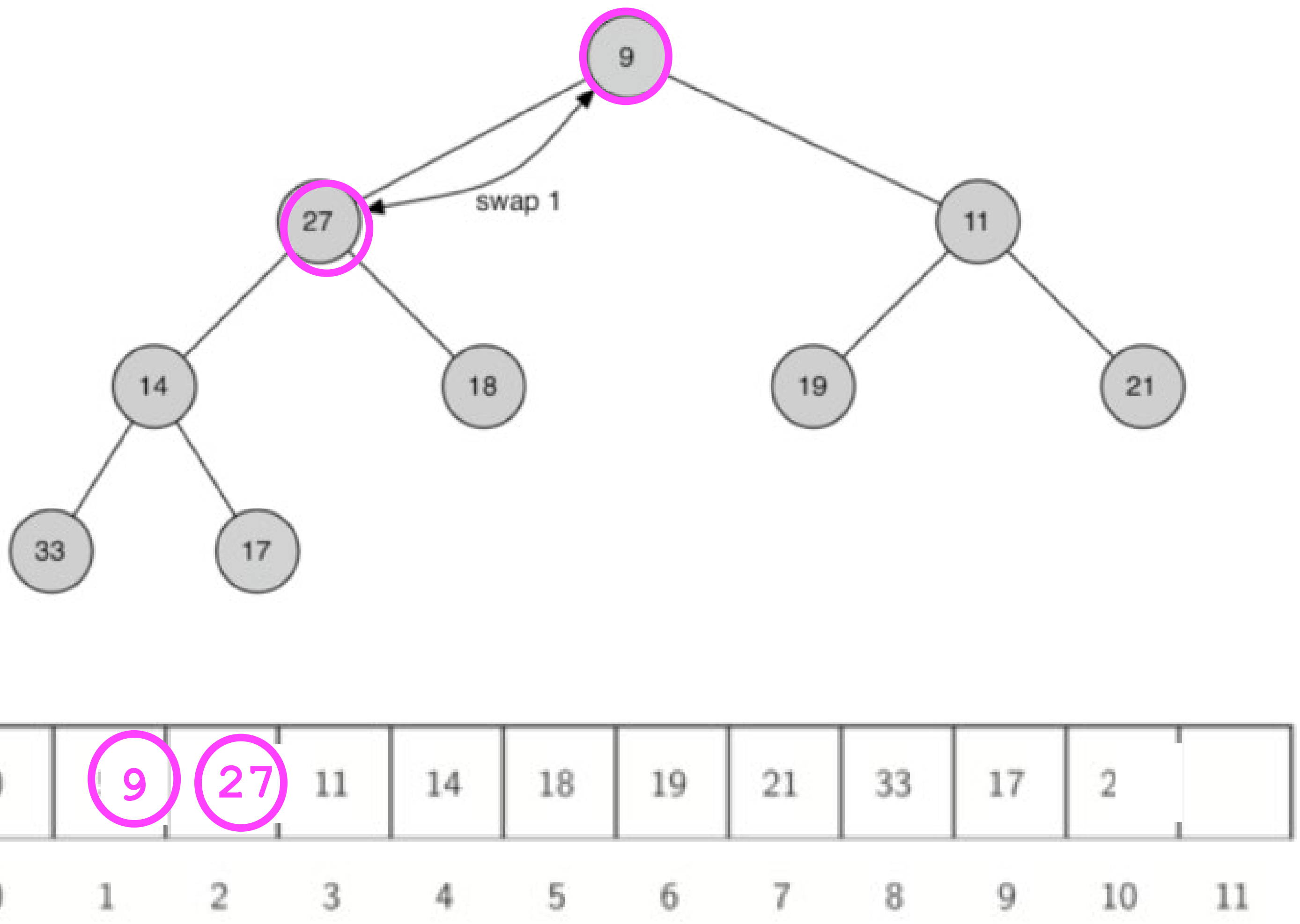


Swap the root with its *smaller* child

We have maintained the shape property, but now we have broken the **order property**. *How can we restore it?*

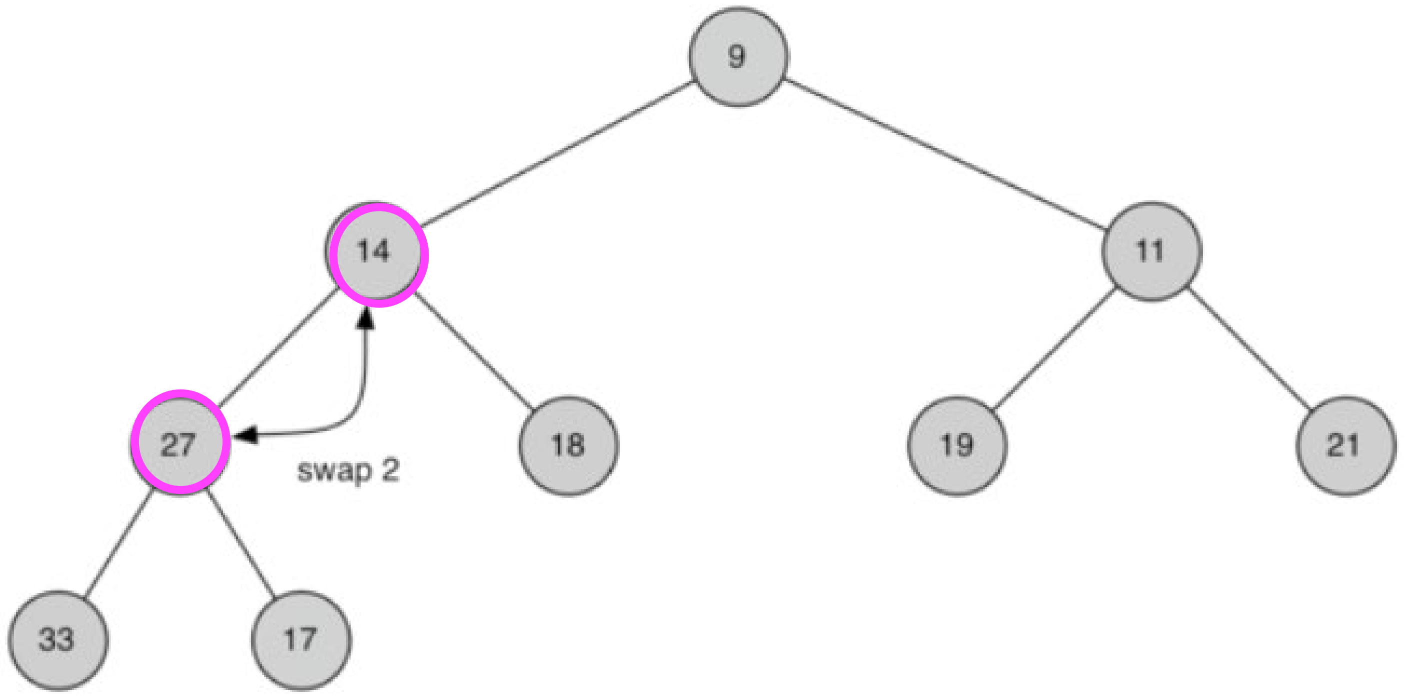
Heap operations

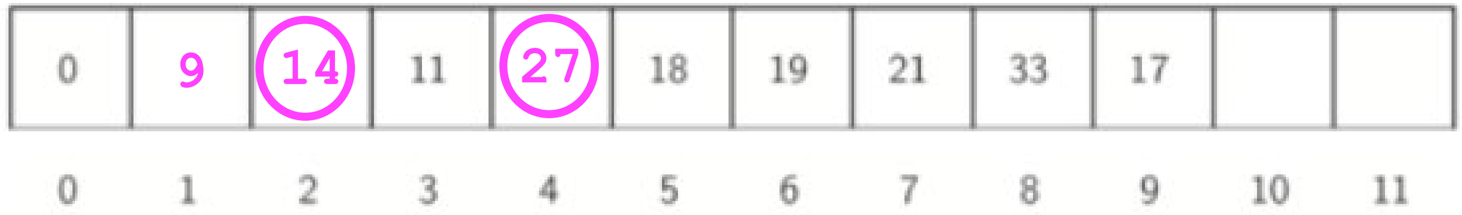
- Deletion : **pq.delete_minimum()**



Heap operations

- Deletion : **pq.delete_minimum()**

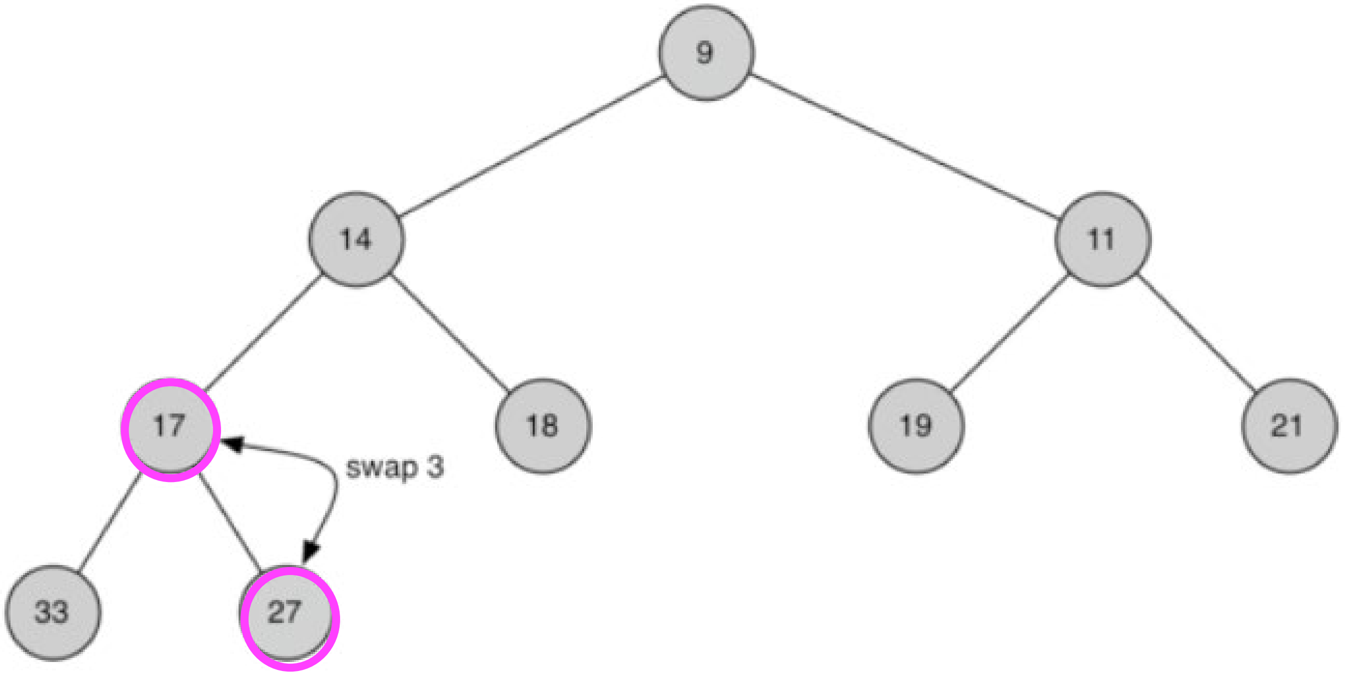




0	9	14	11	27	18	19	21	33	17		
0	1	2	3	4	5	6	7	8	9	10	11

Heap operations

- Deletion : **pq.delete_minimum()**



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