Prepared by Julian García, based on material by
David Albrecht and María García de la Banda

Lecture 34
Heaps

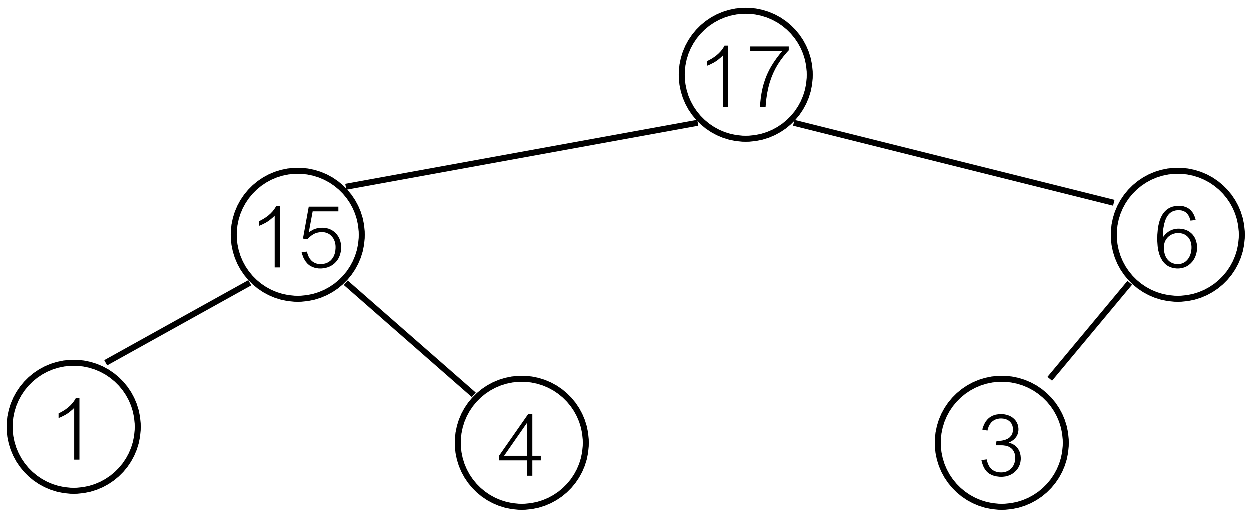
FIT 1008&2085
Introduction to Computer Science

MONASH University
Information Technology

Objectives

- Revise basics of **Heaps** and Heap-based Priority Queue
- To understand a simple **implementation of Heaps**
- To be able to reason about the complexity of its operations
- Heap Sort

Heap (**Max**-Heap)



For **every** node:

- The values of the children are **<u>smaller or equal</u>** to its value.
- **All the levels are filled**, except possibly the last one, which is filled left to right.

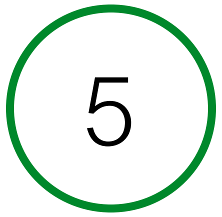
Note: The **maximum** is always at the root of the tree.

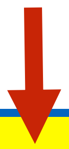
add:
- put at the bottom
- while order is broken, rise.

get_max:
- swap root with last item
- remove last item
- while order is broken, sink.

Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an empty **MAX**heap

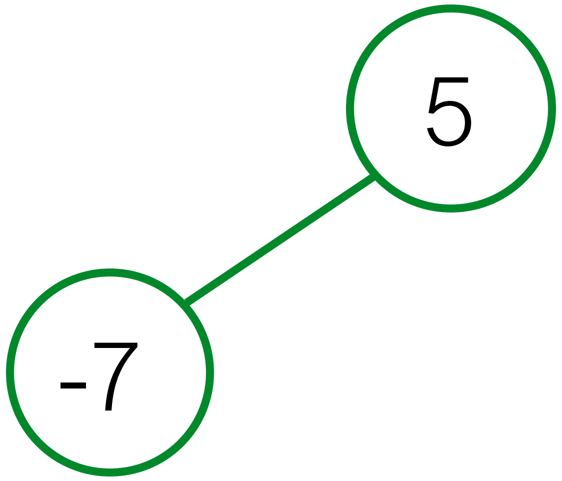
Heap (maxheap)





Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an empty heap

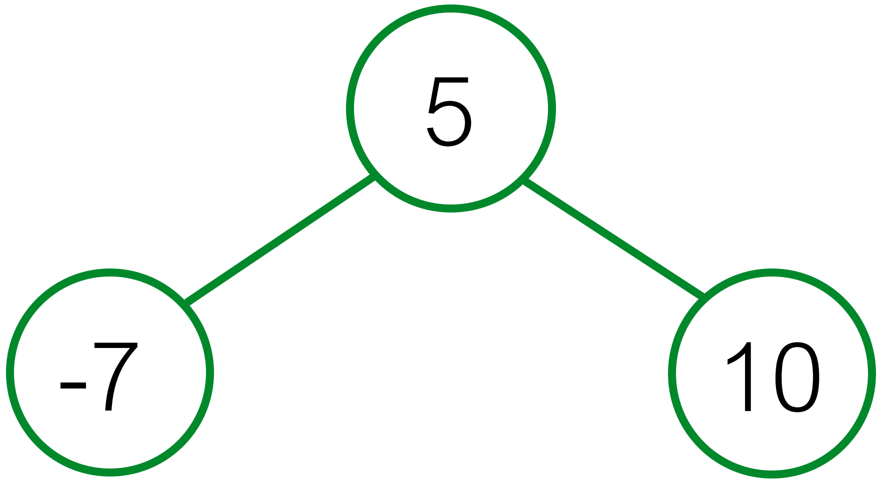
Heap



-7 < root

Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an empty heap

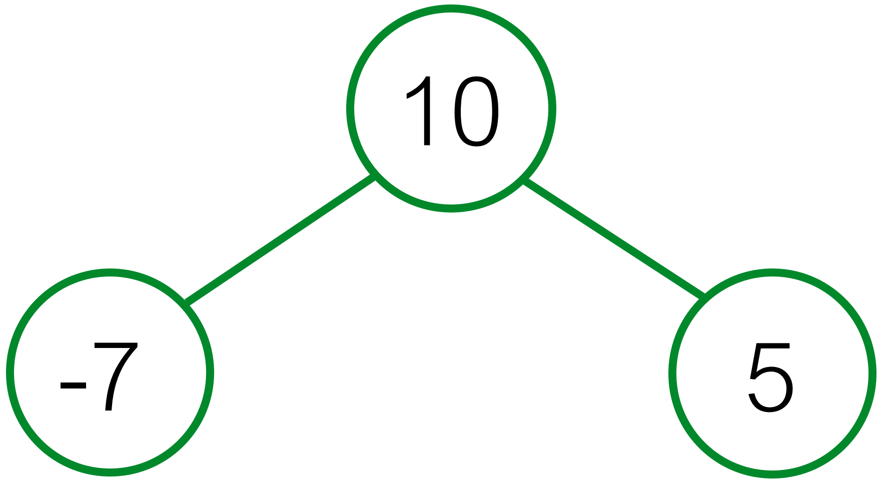
Heap



10 > root

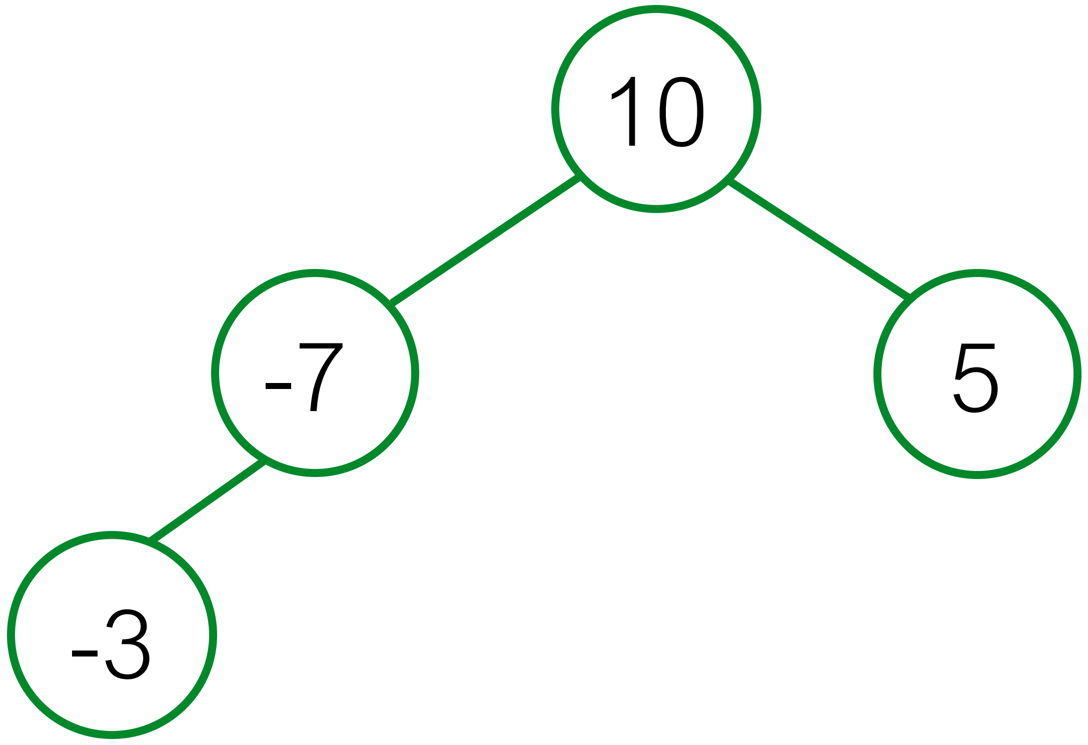
Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an empty heap

Heap



Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an empty heap

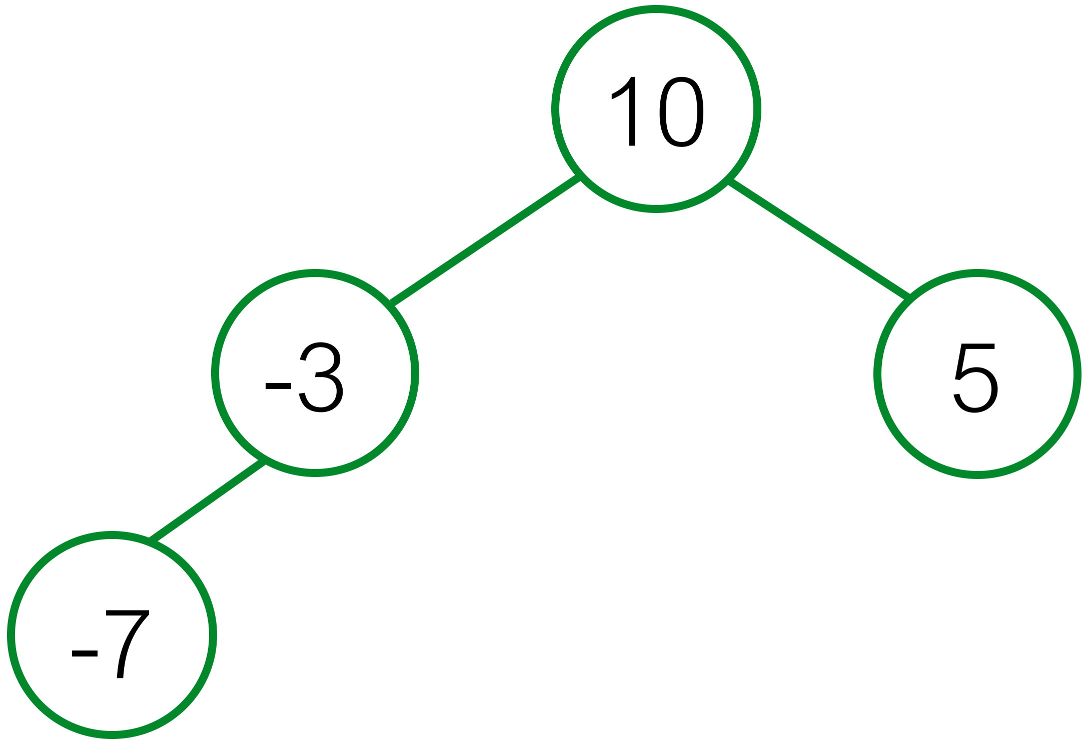
Heap



-3 > -7

Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an empty heap

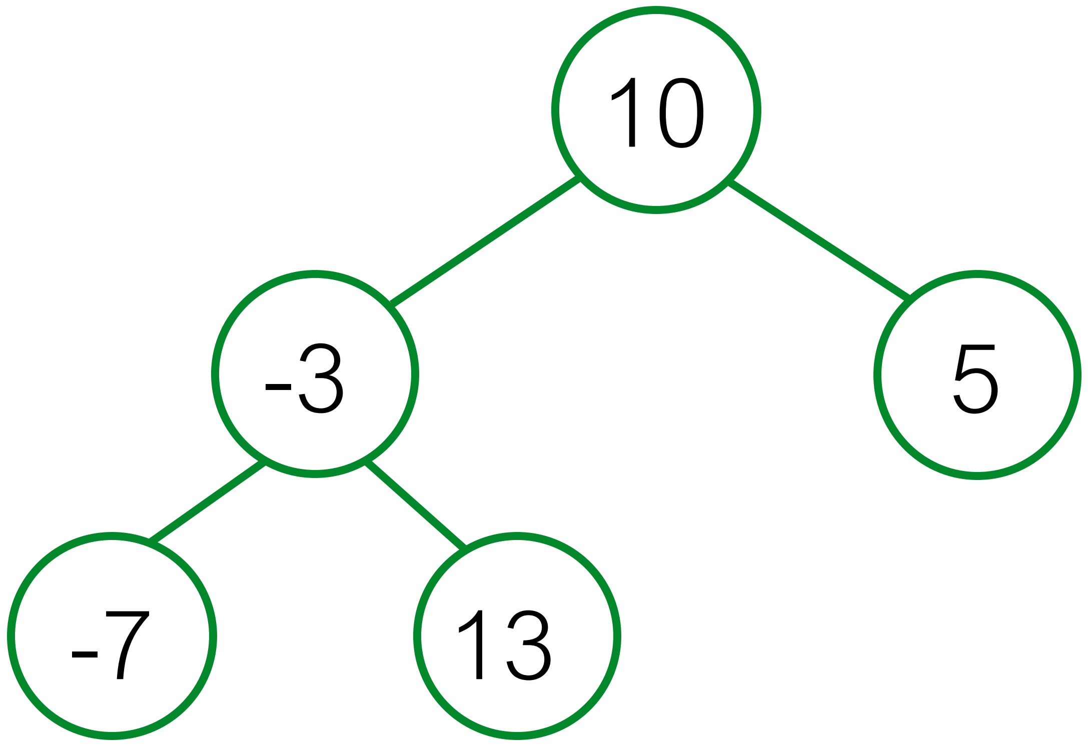
Heap



-3 < 10

Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an empty heap

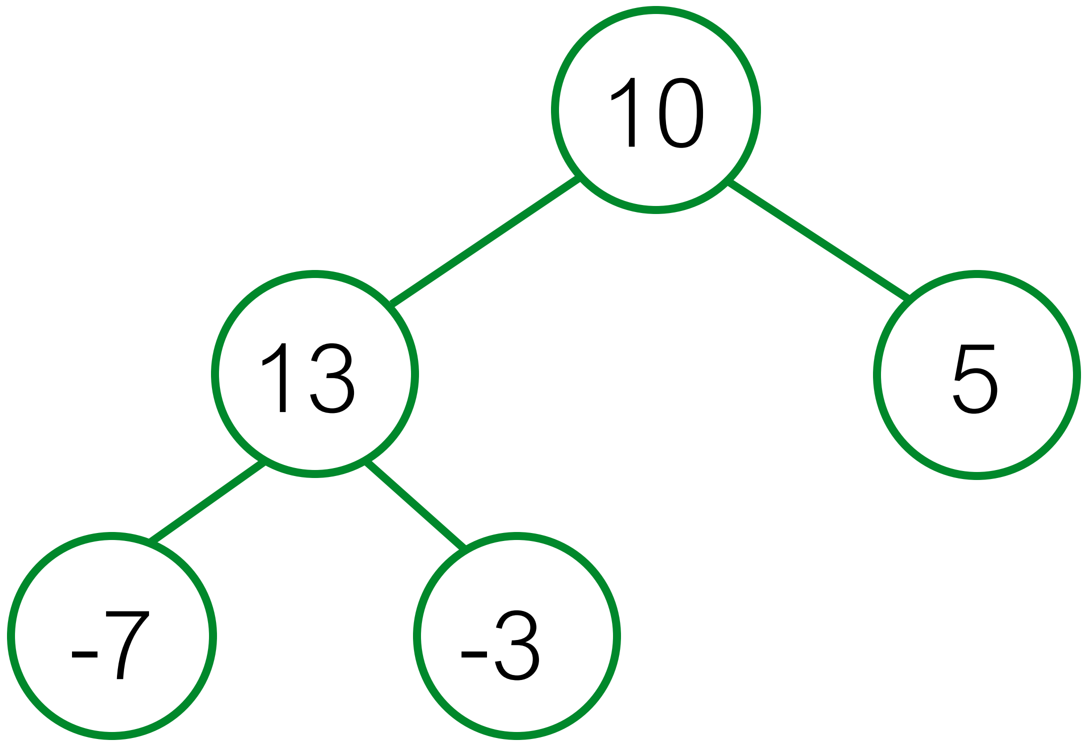
Heap



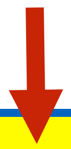
13 > -3

Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an empty heap

Heap

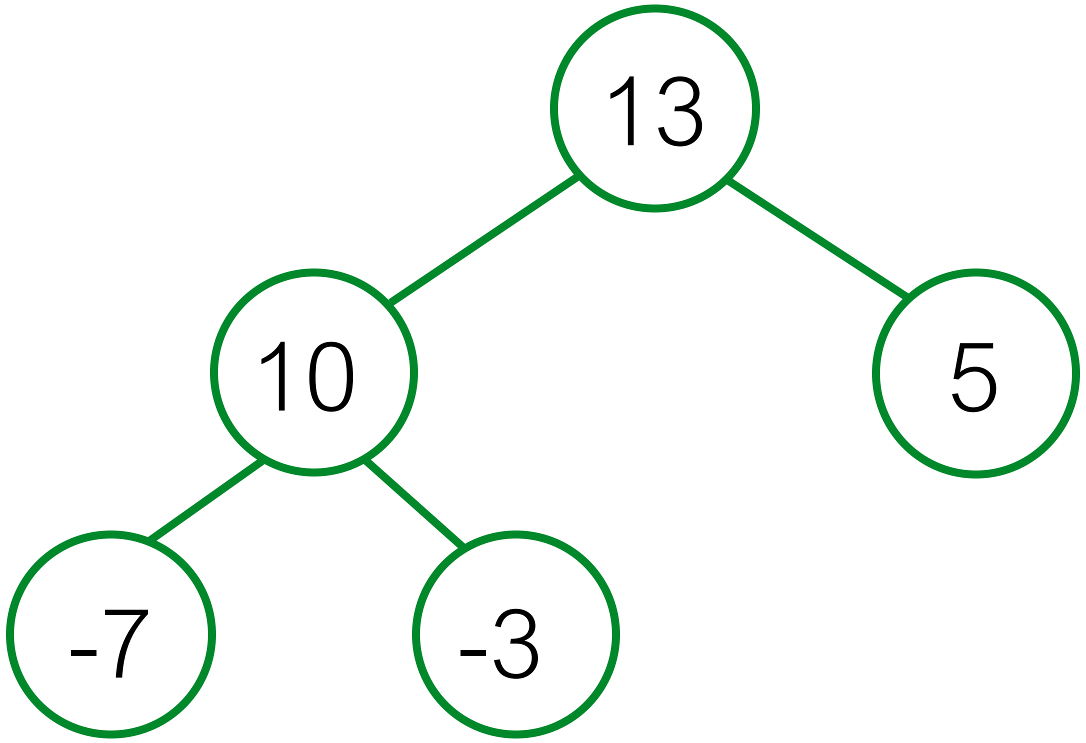


13 > 10



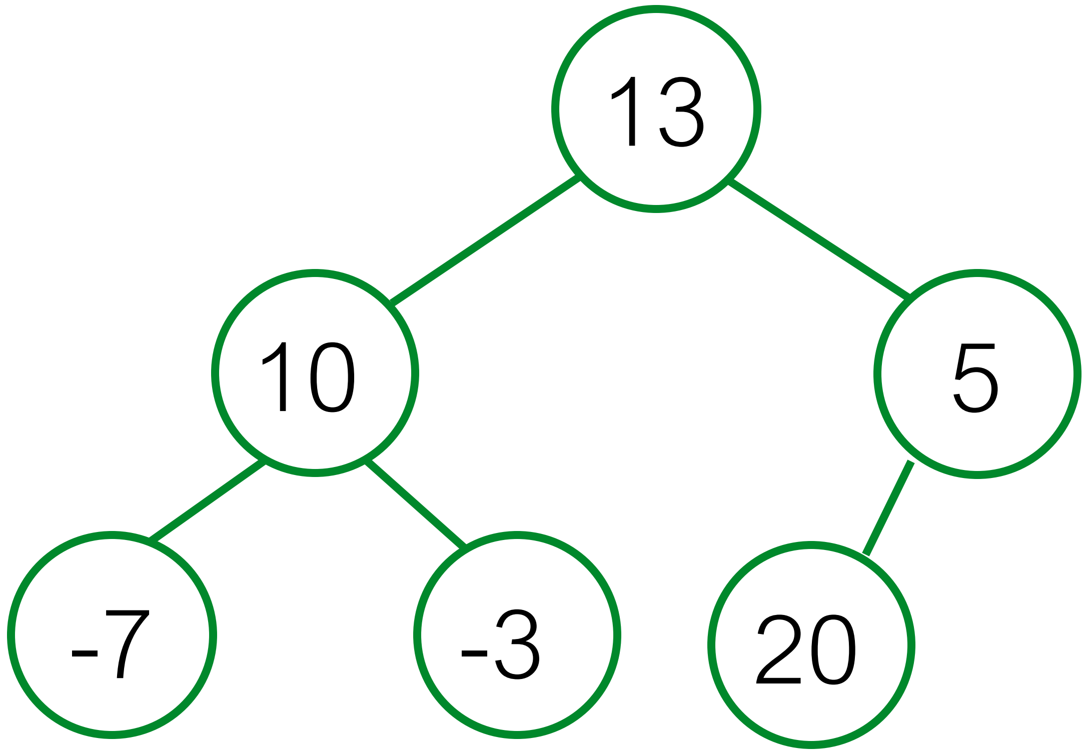
Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an empty heap

Heap



Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an empty heap

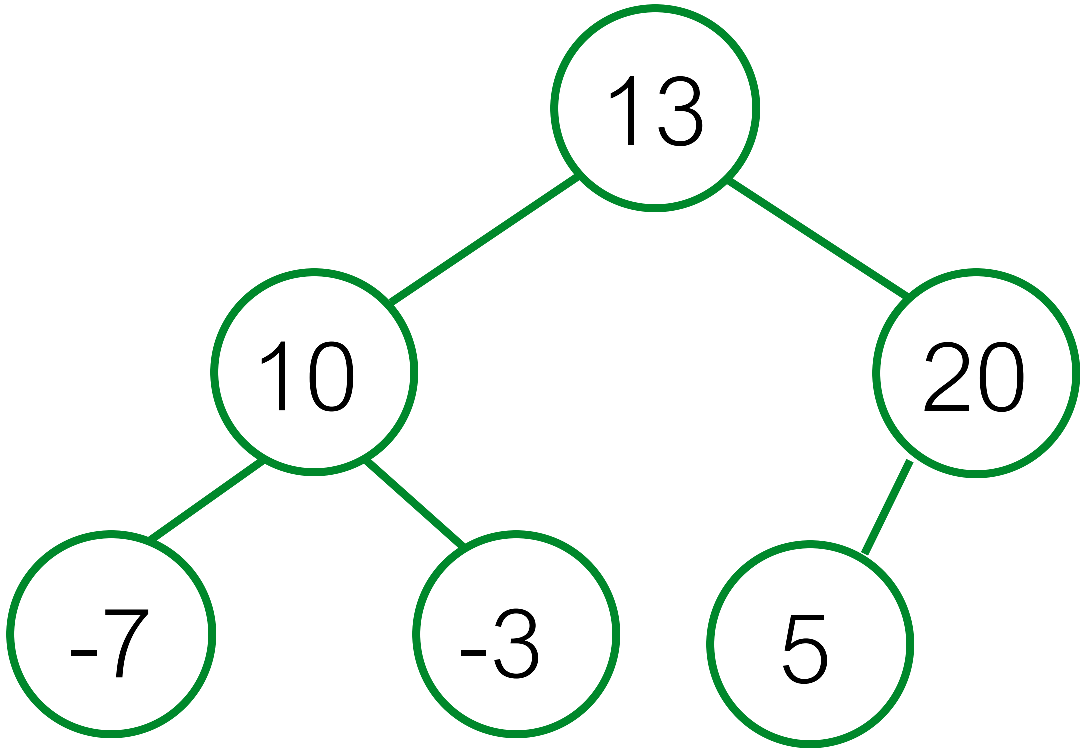
Heap



20 > 5

Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an empty heap

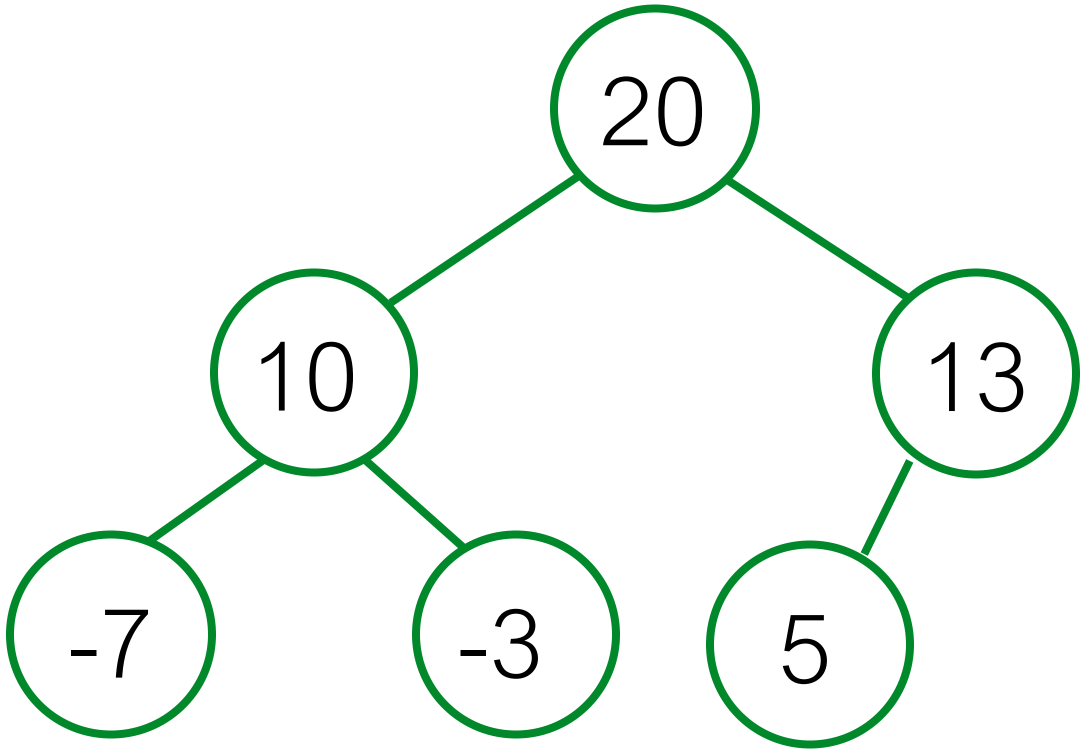
Heap

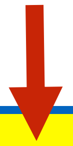


20 > 13

Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an empty heap

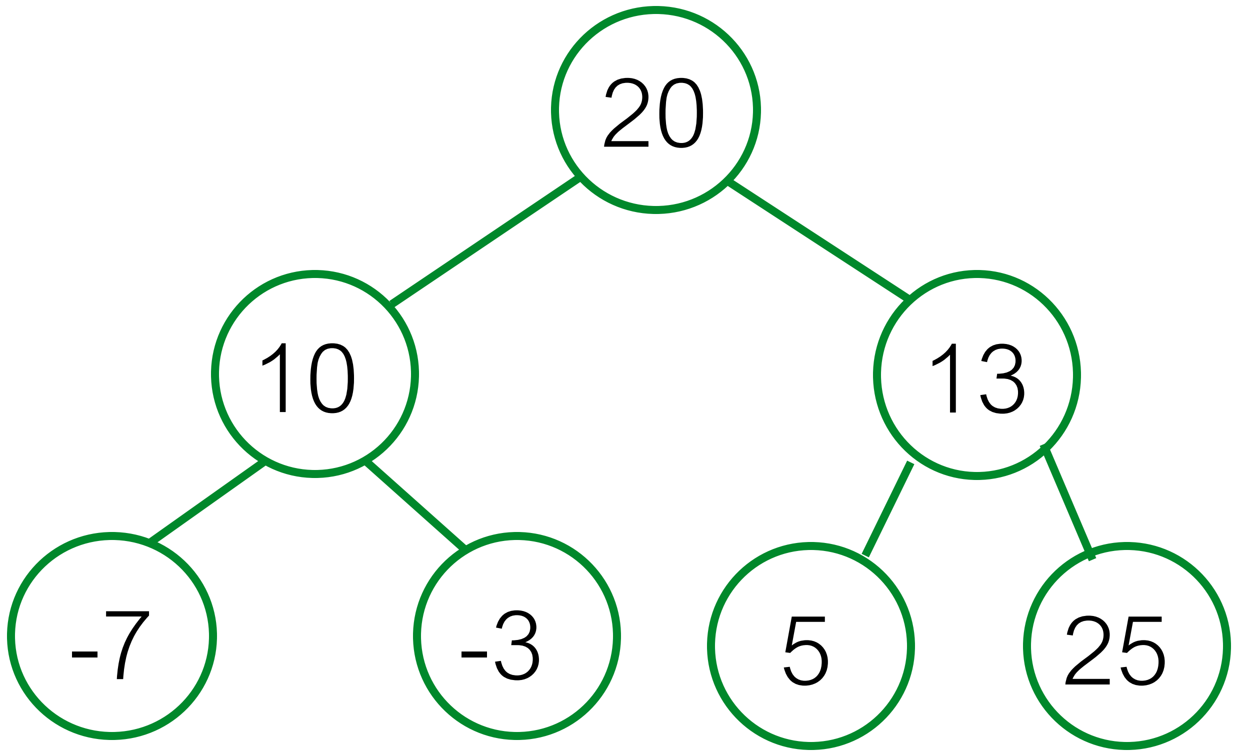
Heap





Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1]
into an empty heap

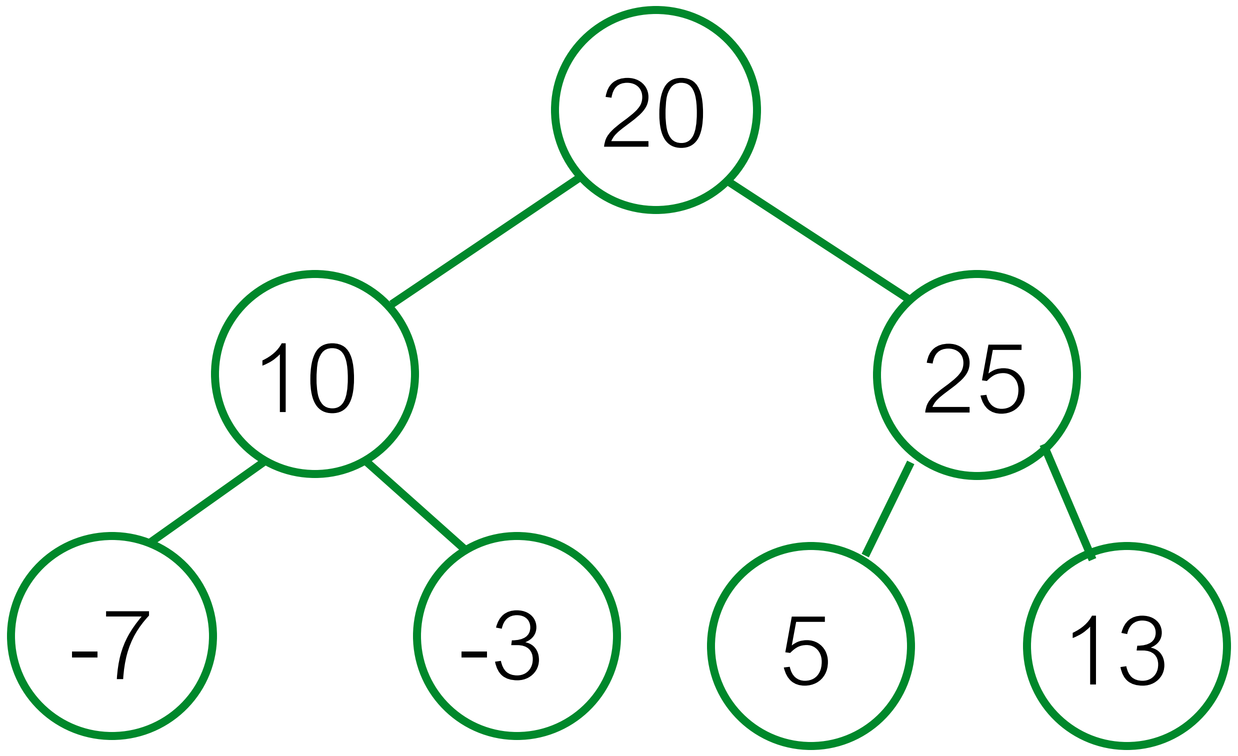
Heap



25 > 13

Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an empty heap

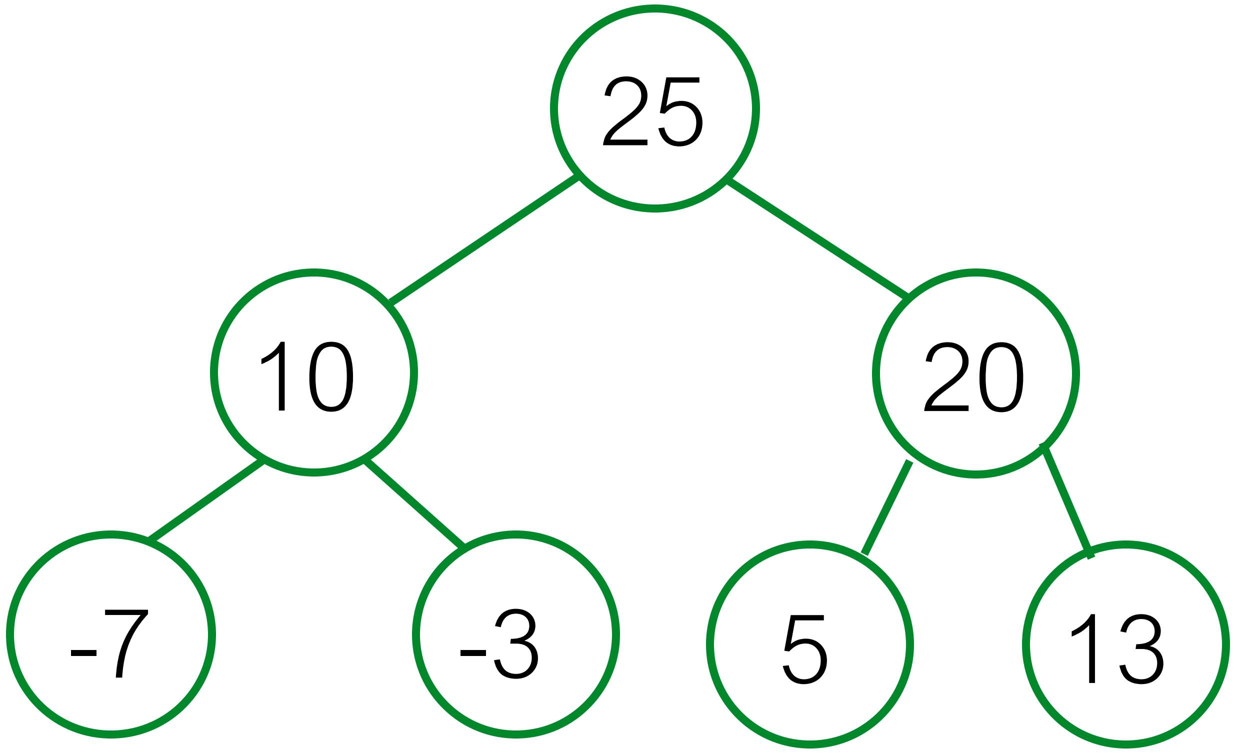
Heap



25 > 20

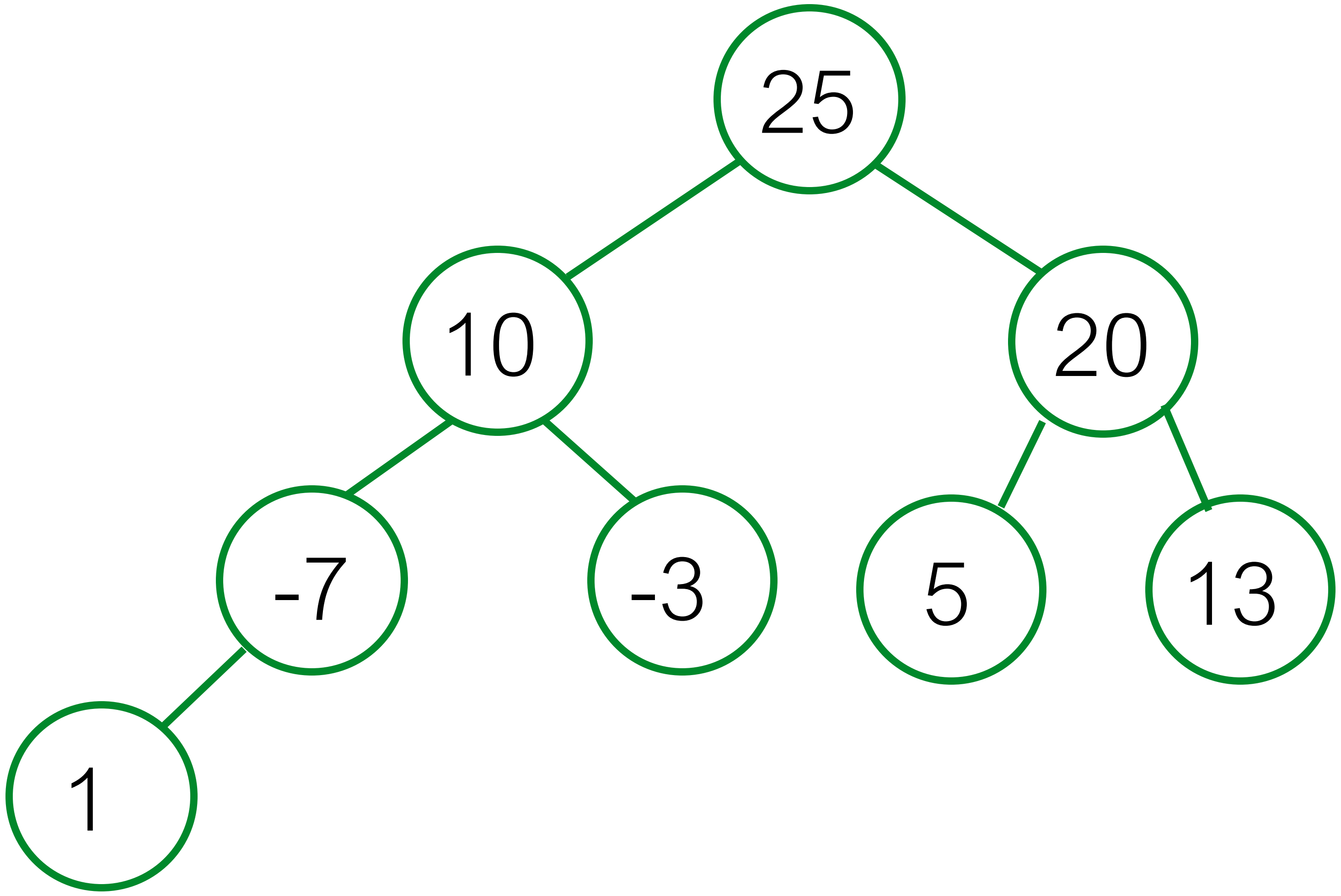
Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an empty heap

Heap



Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an empty heap

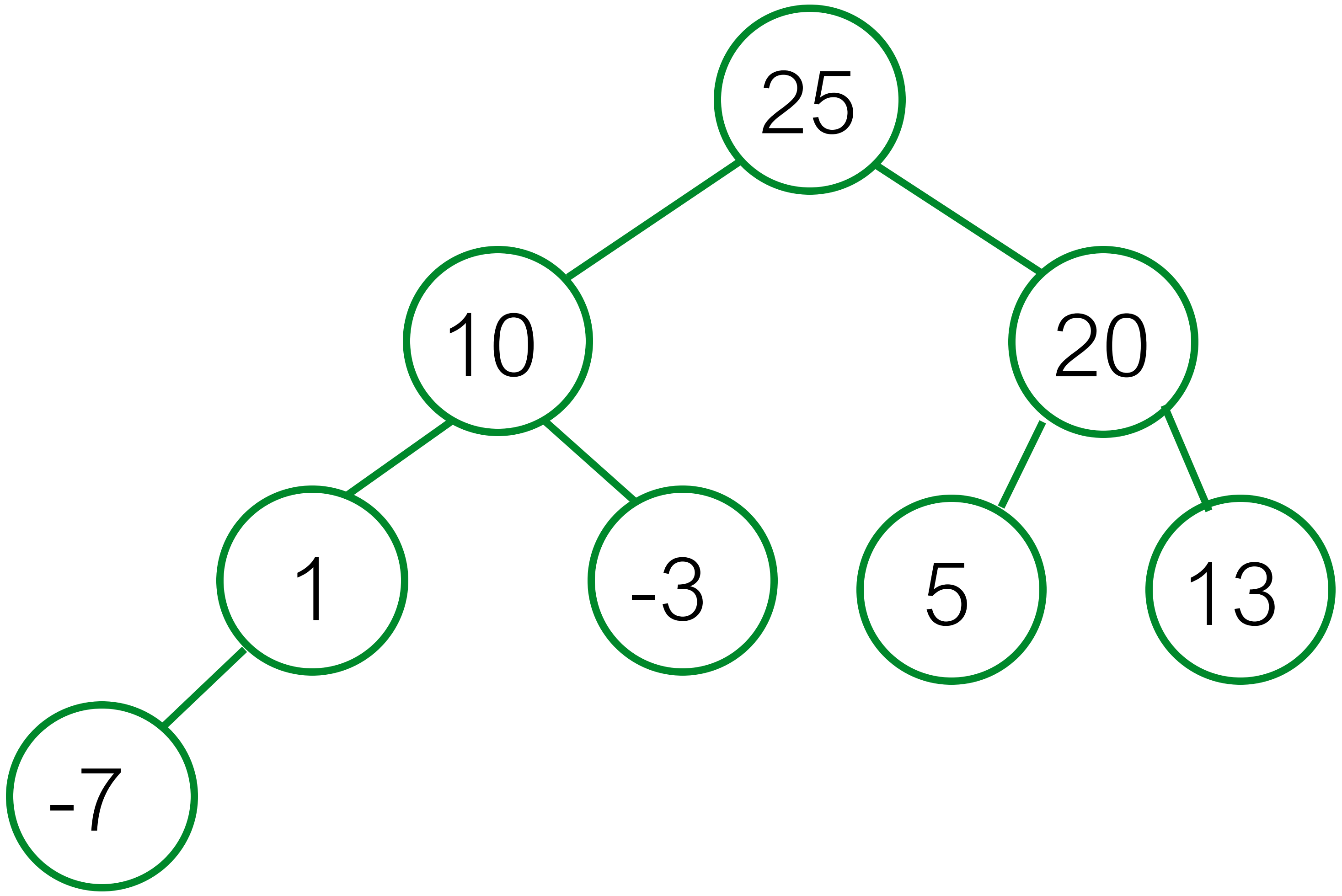
Heap



1 > -7

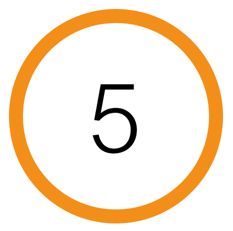
Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an empty heap

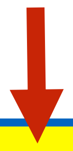
Heap



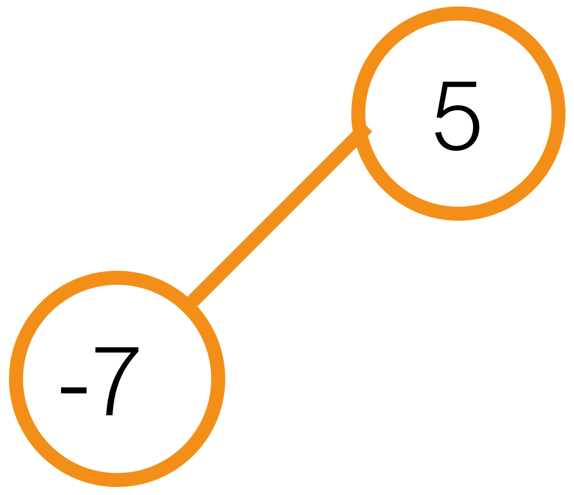
Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an empty heap

Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an **Binary Search Tree**

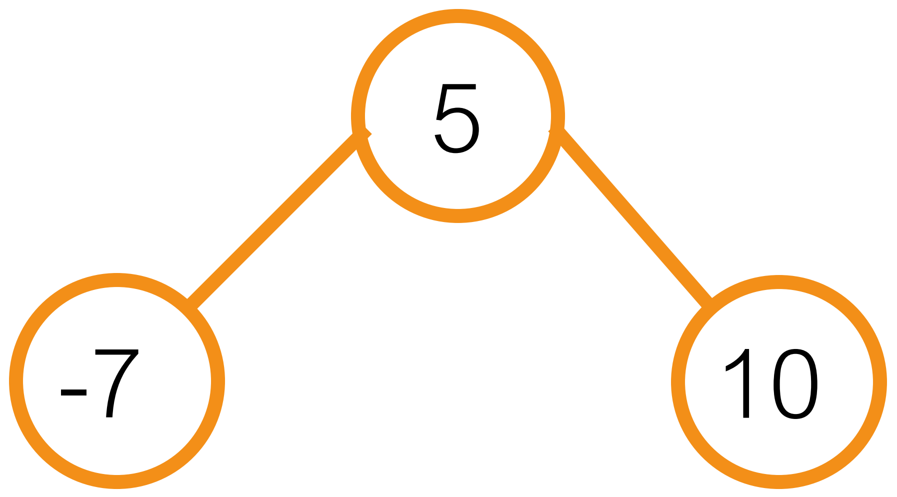




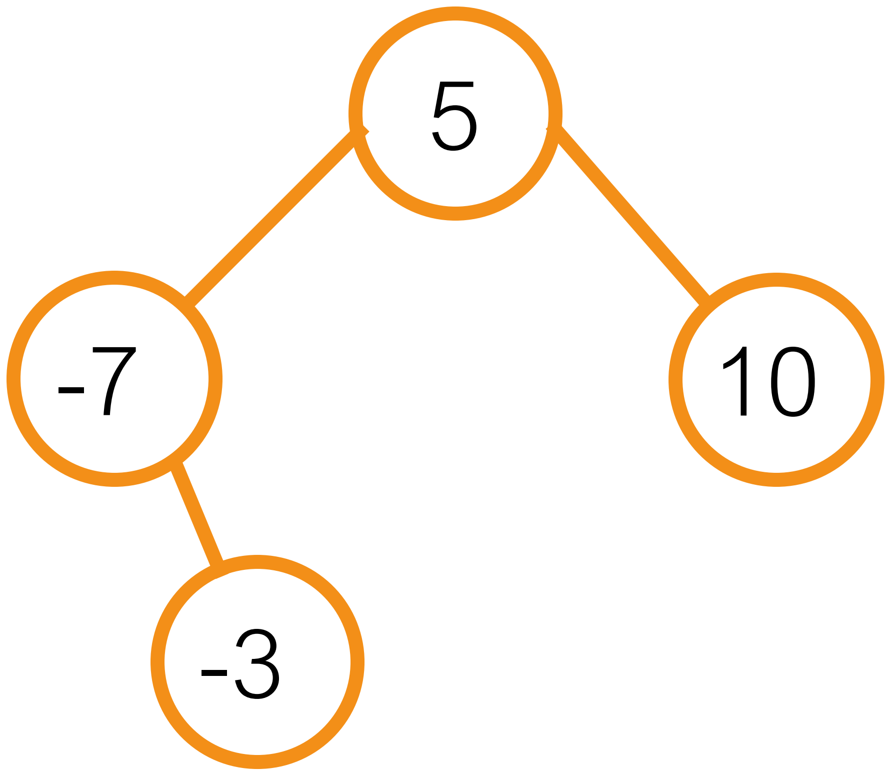
Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an **Binary Search Tree**



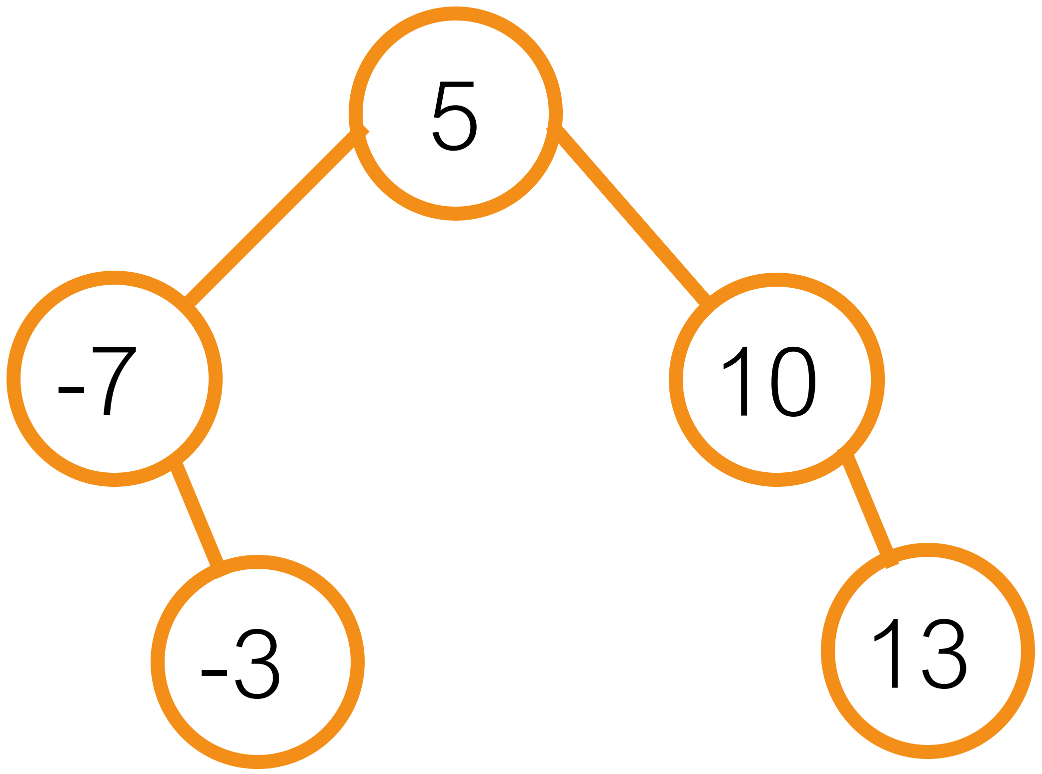
Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an **Binary Search Tree**



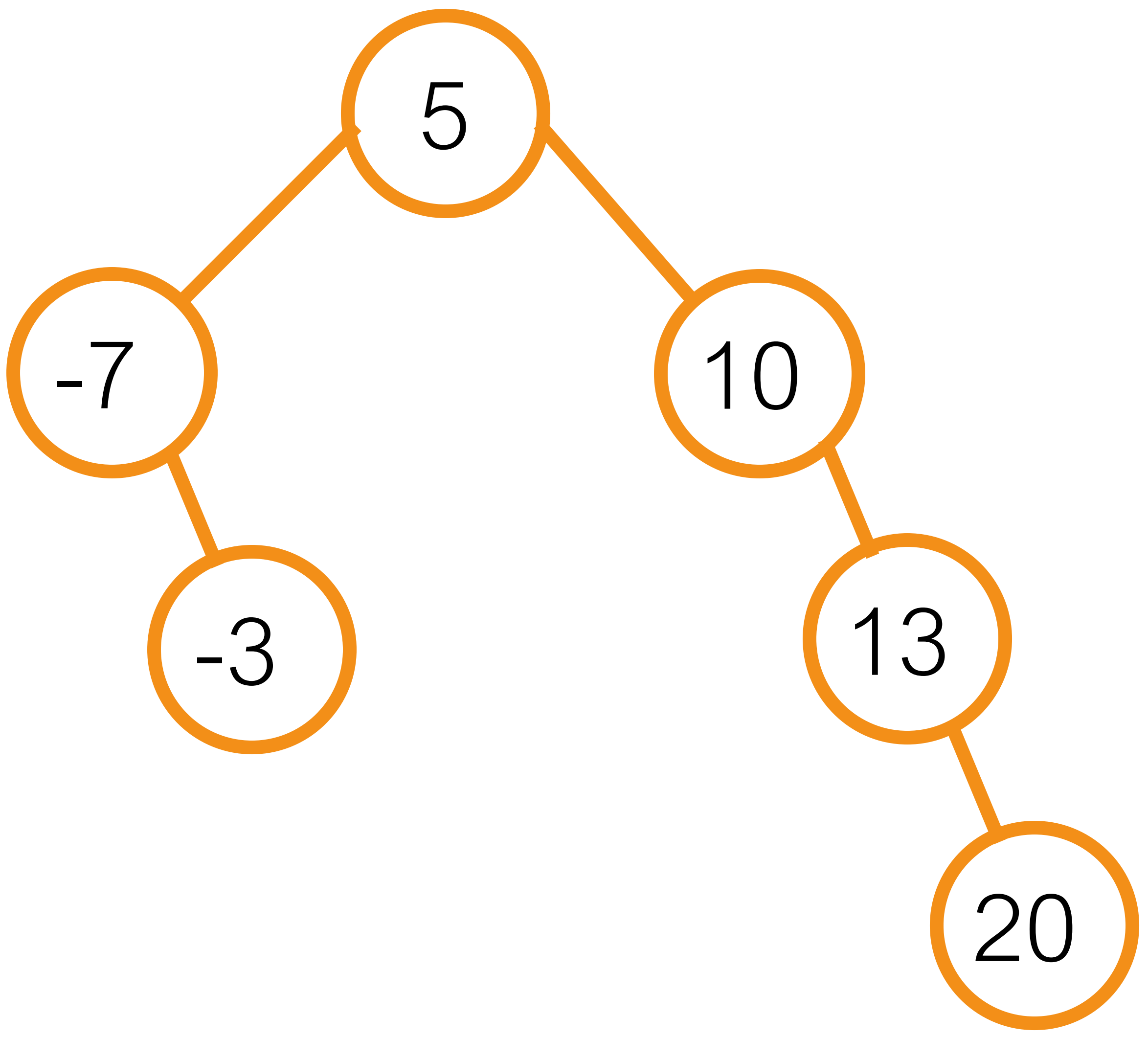
Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an **Binary Search Tree**



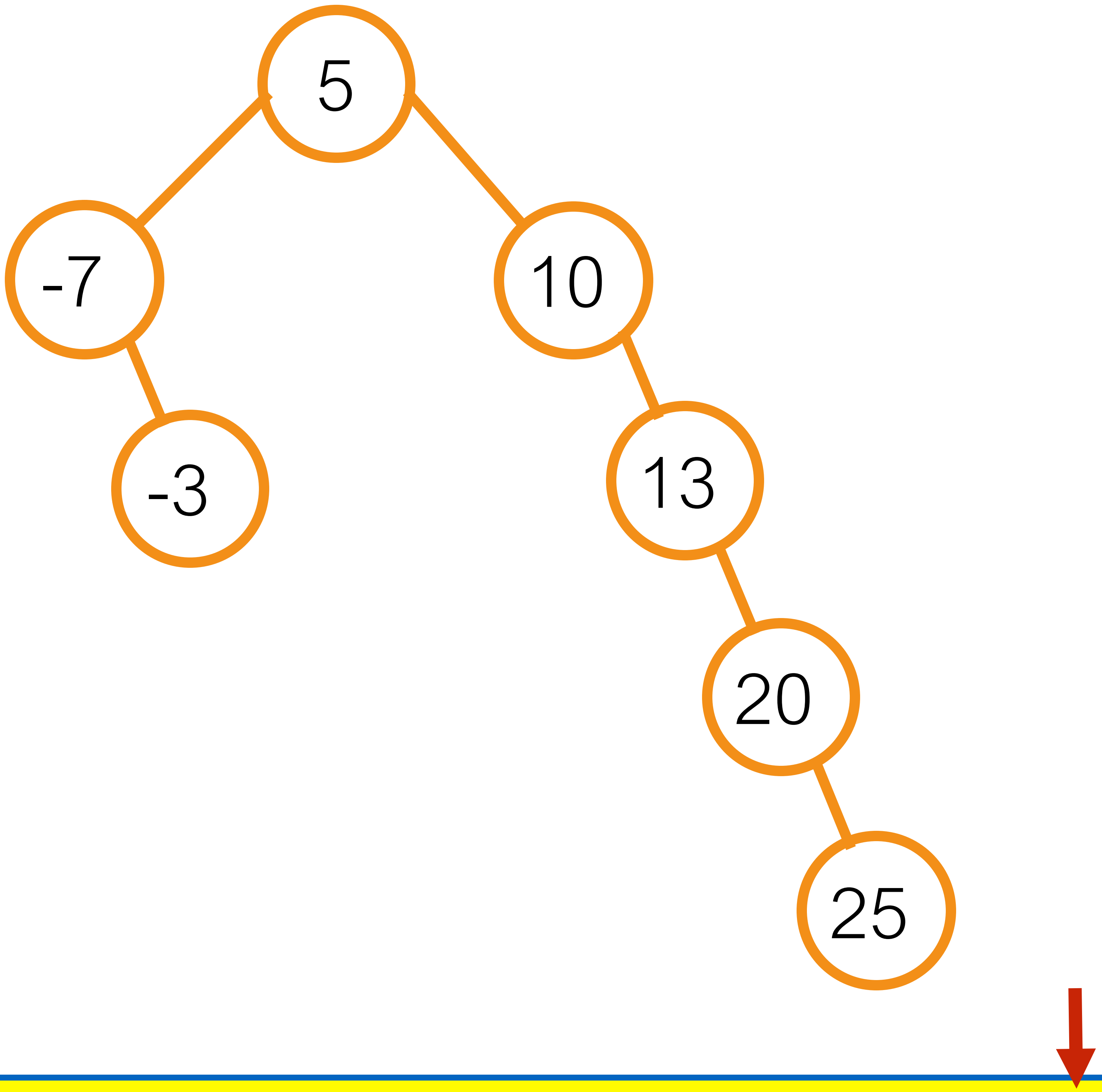
Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an **Binary Search Tree**



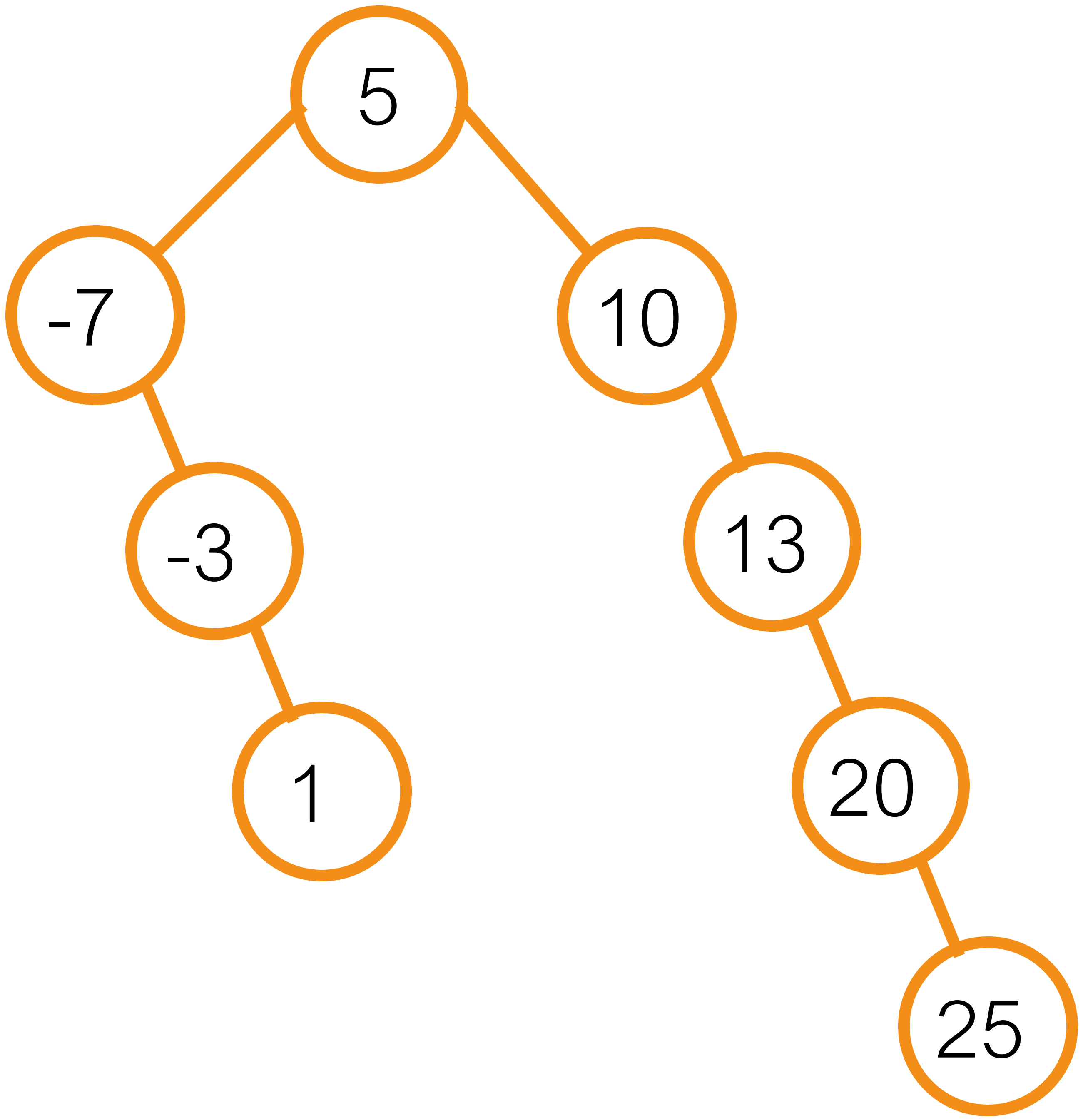
Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an **Binary Search Tree**



Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an **Binary Search Tree**

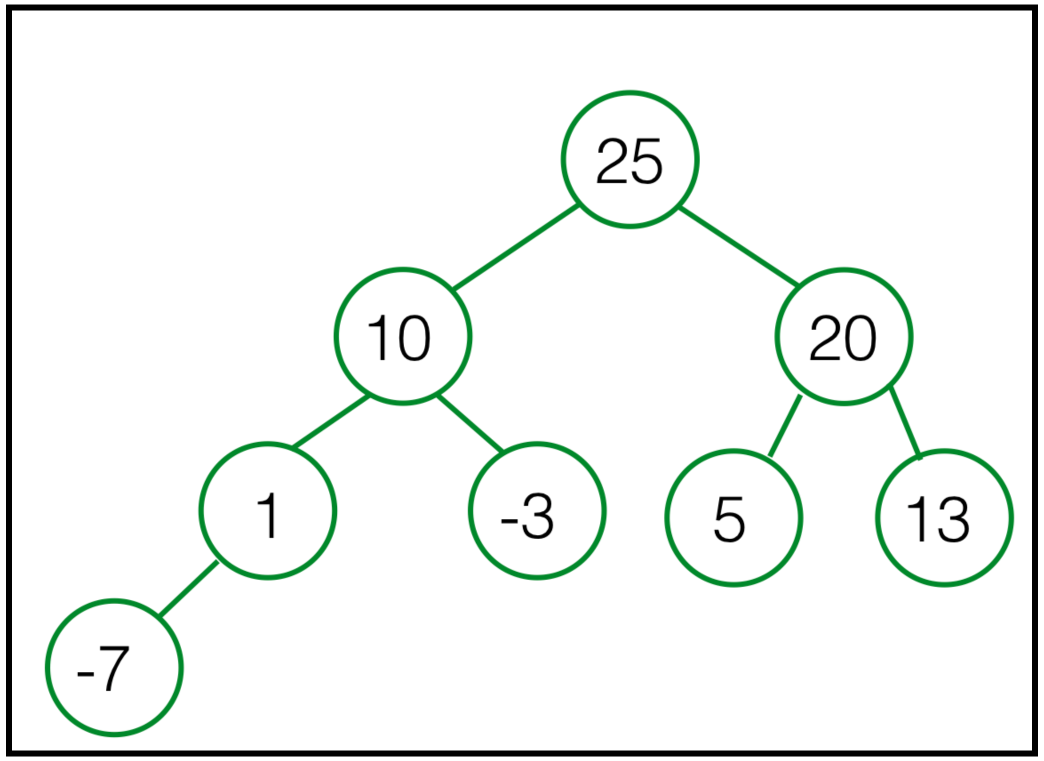


Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an **Binary Search Tree**

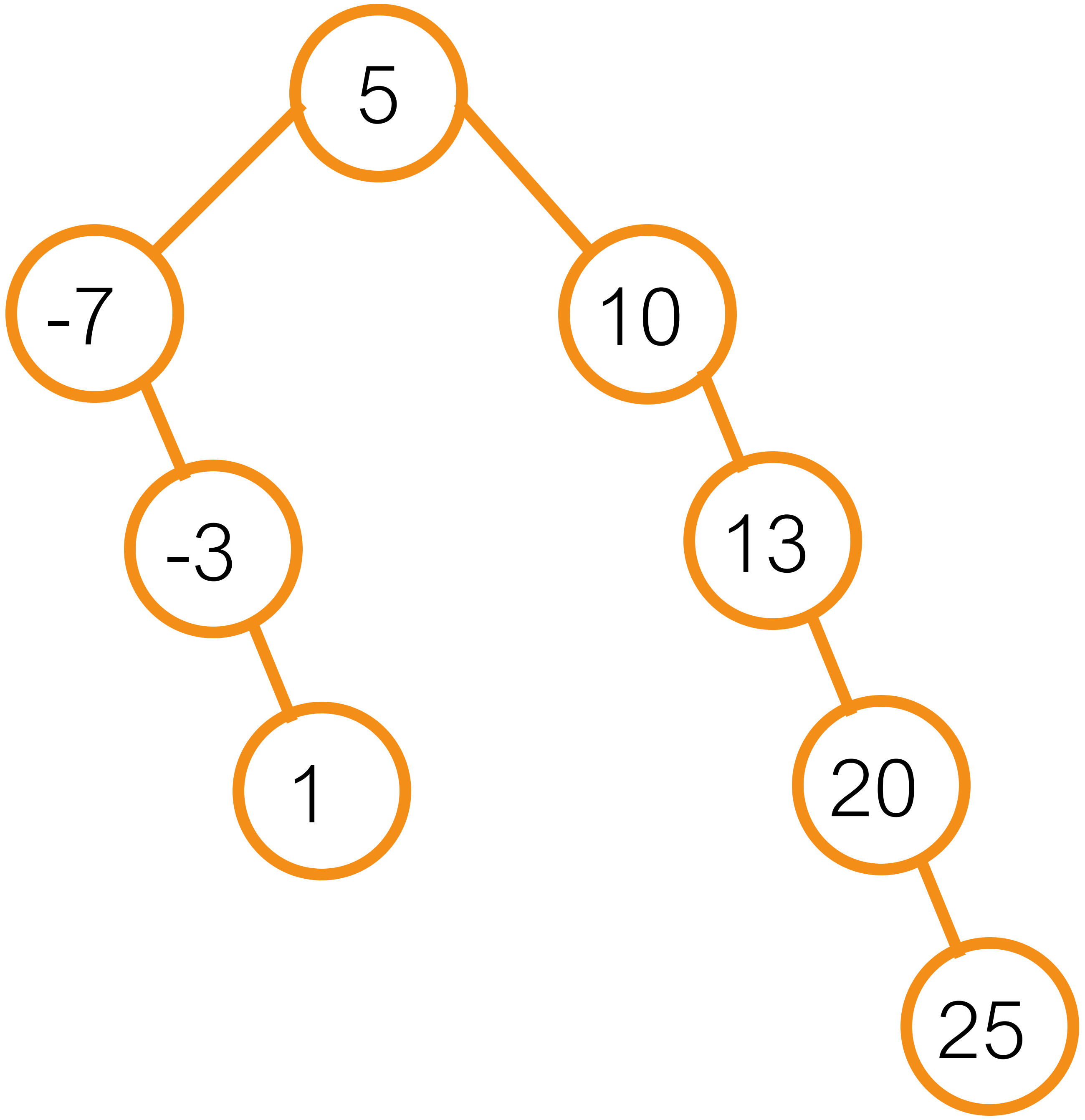


Lets insert the numbers [5, -7, 10, -3, 13, 20, 25,1] into an **Binary Search Tree**

Heap vs Binary Search Tree



Very different!



Implementation of Heaps?

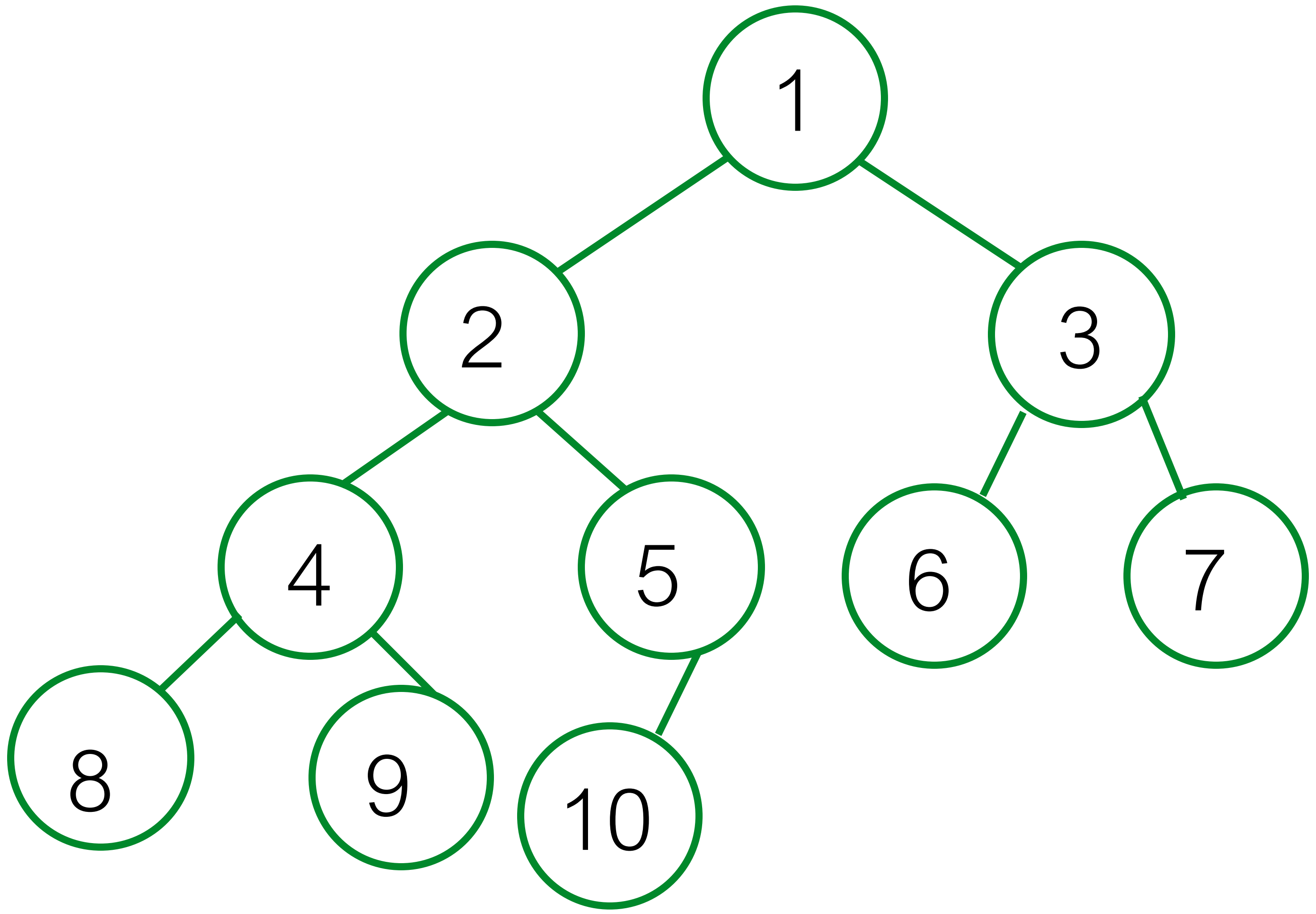
Implementation

Alternative 1: Binary tree of <u>linked nodes</u>

- **Downside**: complex -- requires extra references to move up the tree (rise a node)
- Extra memory.

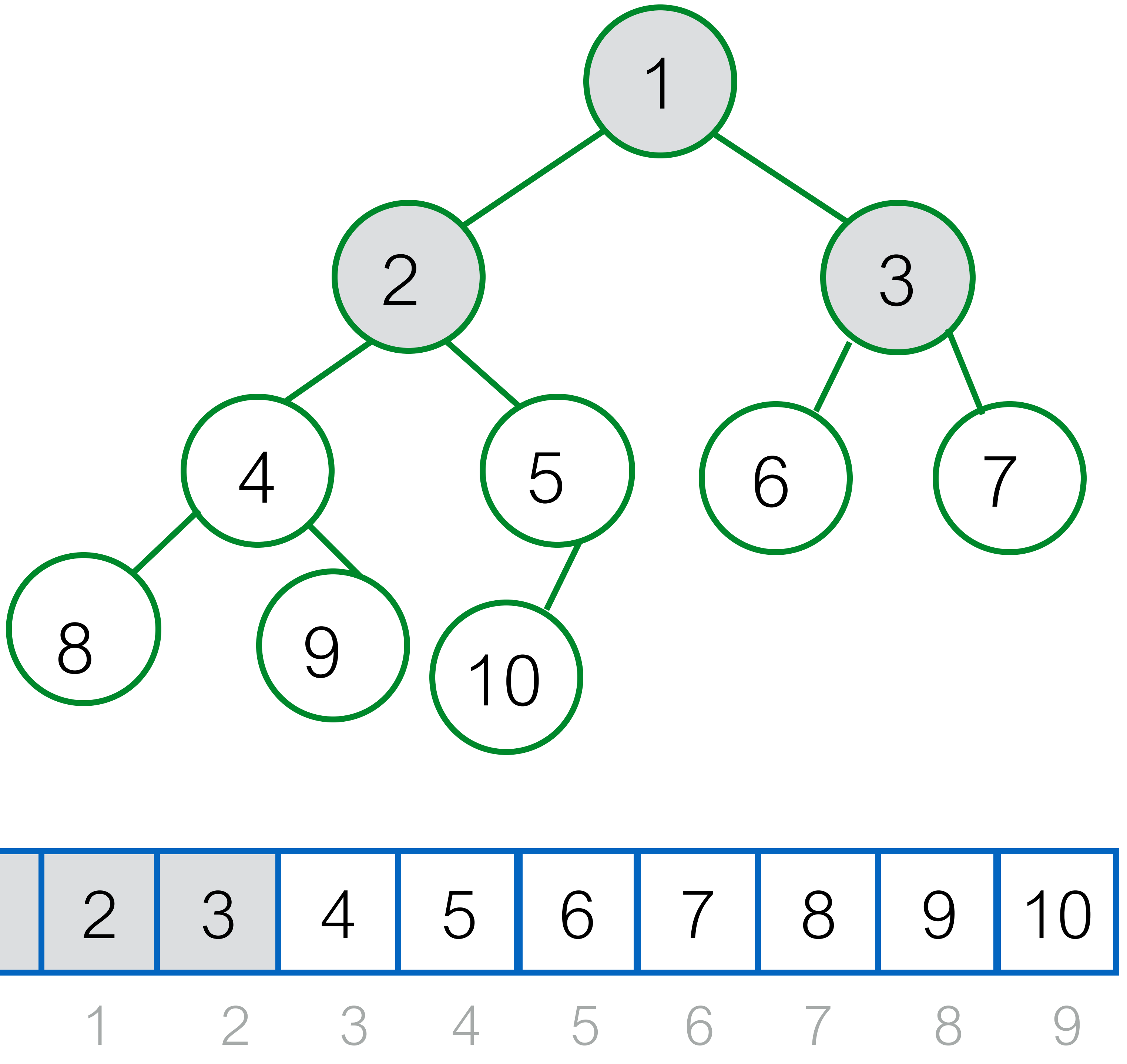
Alternative 2: With an <u>array</u>

- Possible due to completeness of the binary tree.
- **Advantages**: Very compact

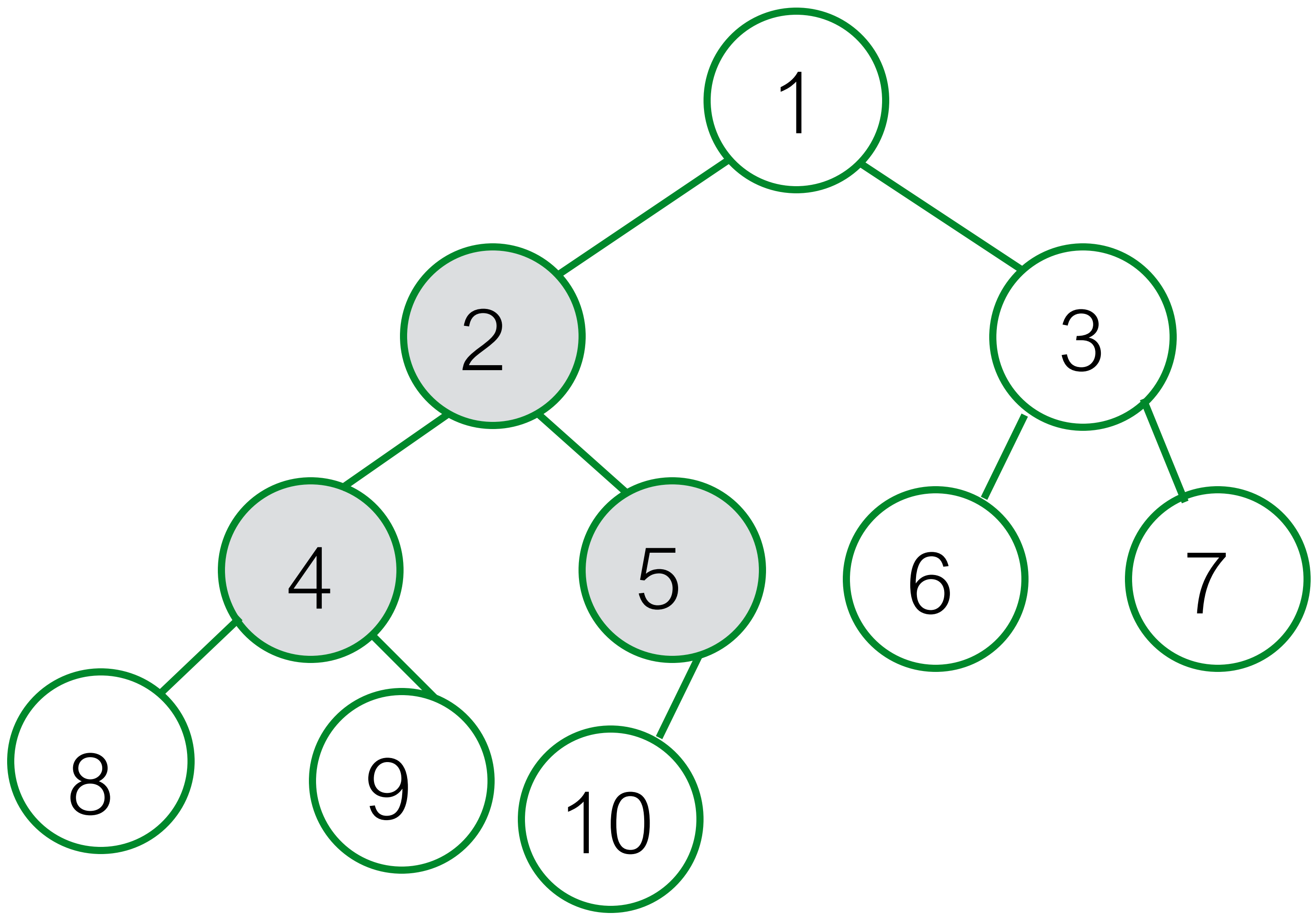


Parent Position	Child Left	Child Right

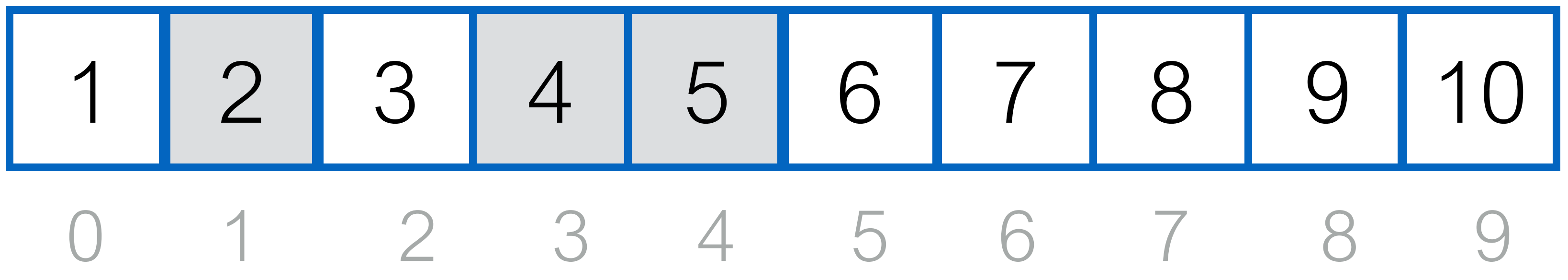
1	2	3	4	5	6	7	8	9	10
0	1	2	3	4	5	6	7	8	9

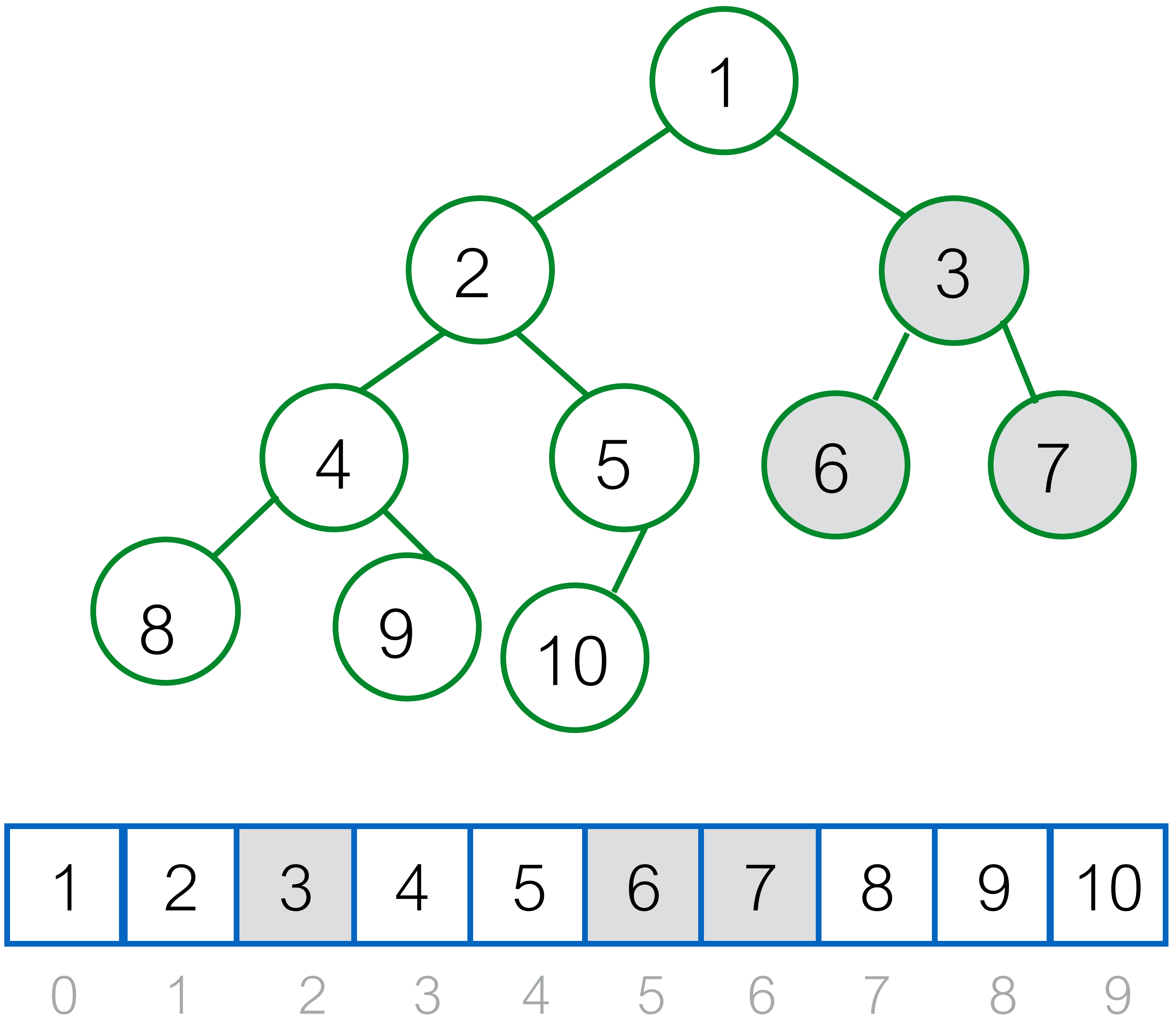


Parent Position	Child Left	Child Right
0	1	2

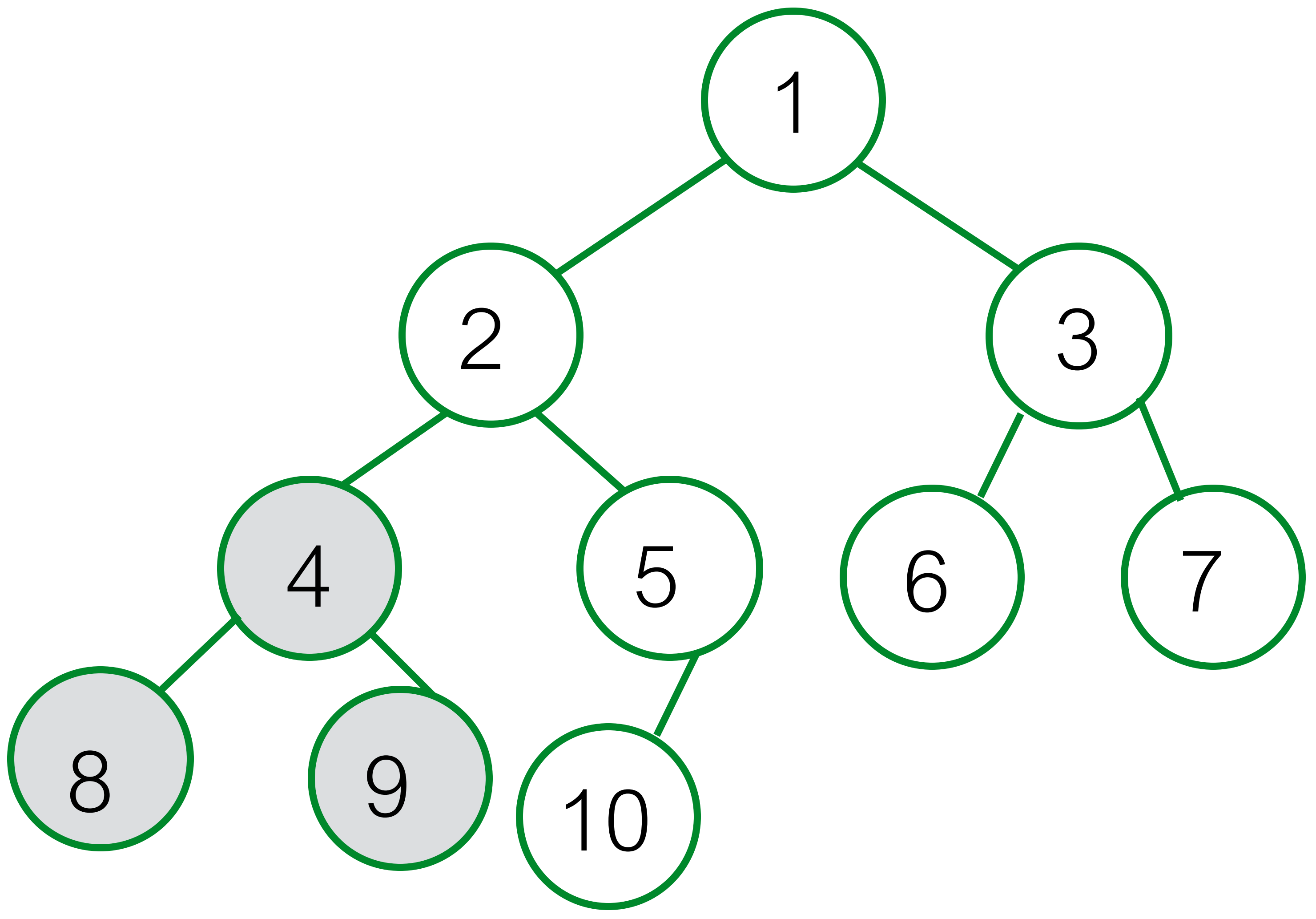


Parent Position	Child Left	Child Right
0	1	2
1	3	4

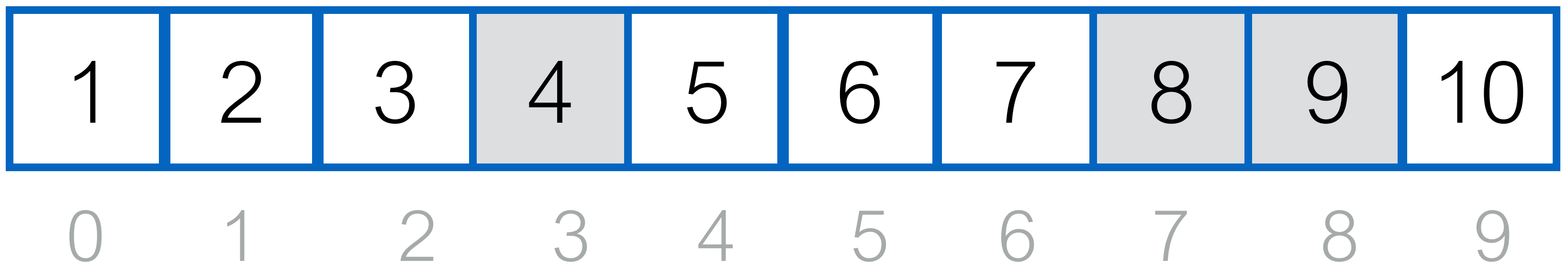


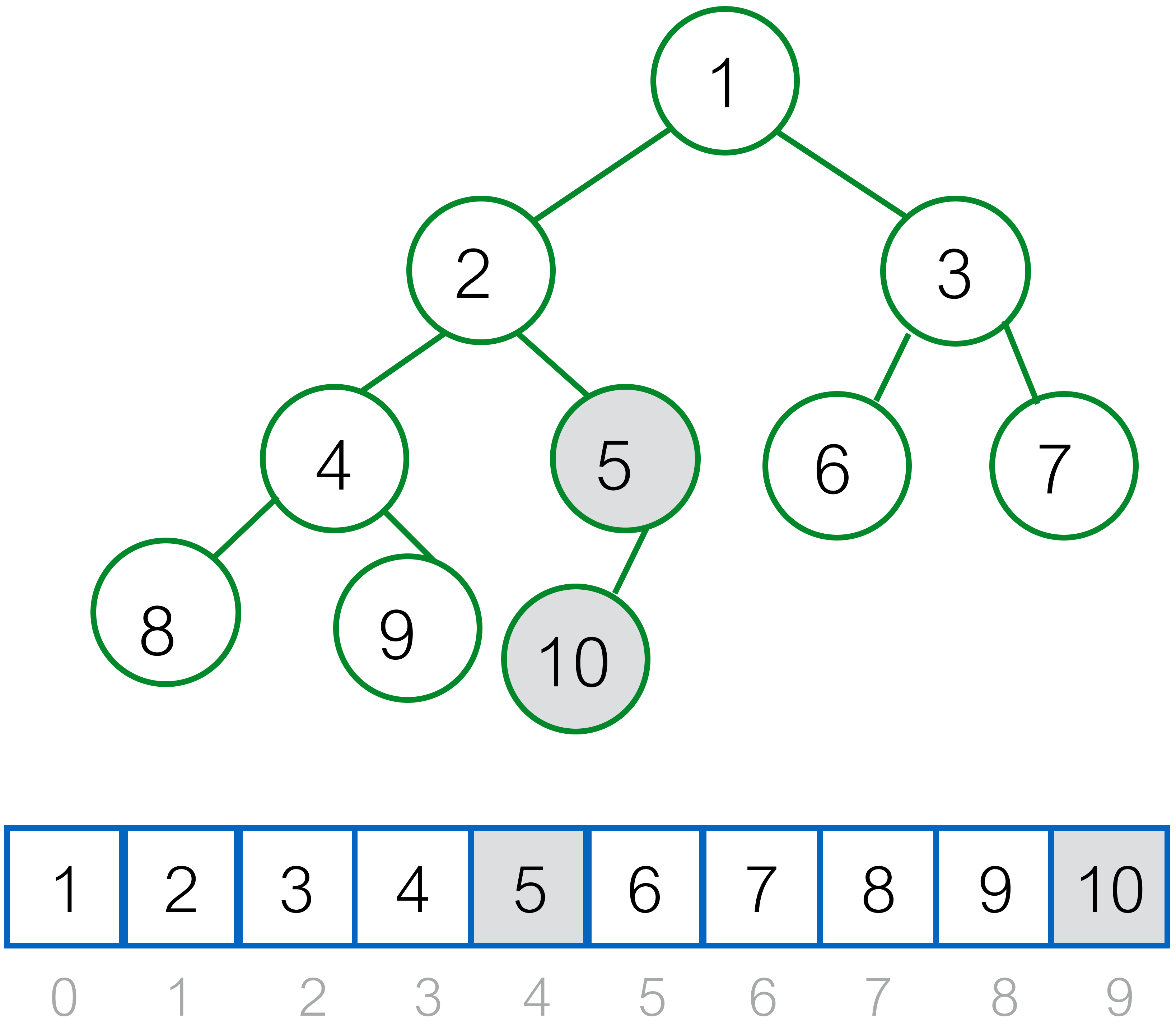


Parent Position	Child Left	Child Right
0	1	2
1	3	4
2	5	6

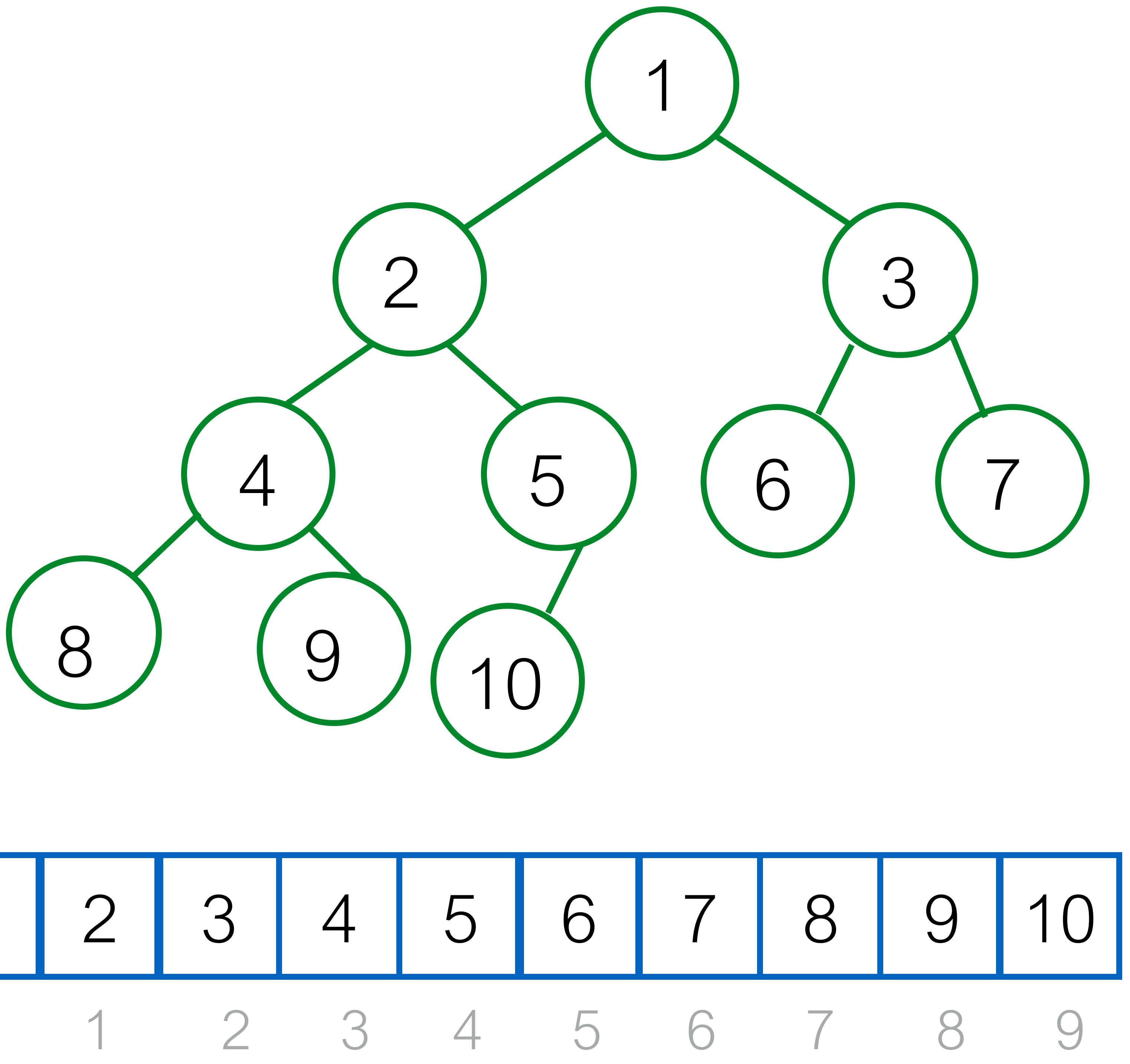


Parent Position	Child Left	Child Right
0	1	2
1	3	4
2	5	6
3	7	8



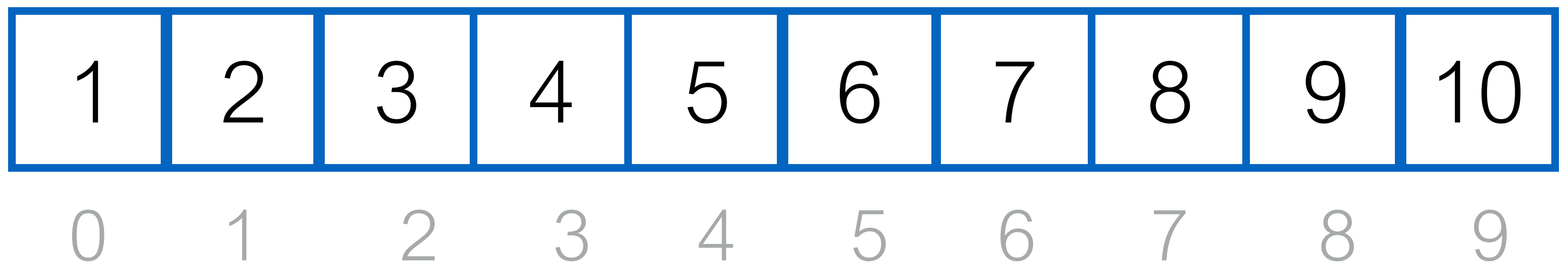


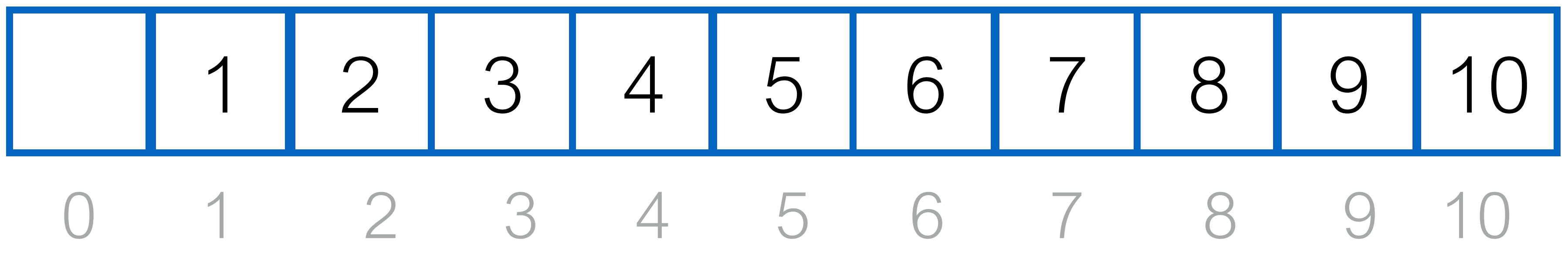
Parent Position	Child Left	Child Right
0	1	2
1	3	4
2	5	6
3	7	8
4	9	

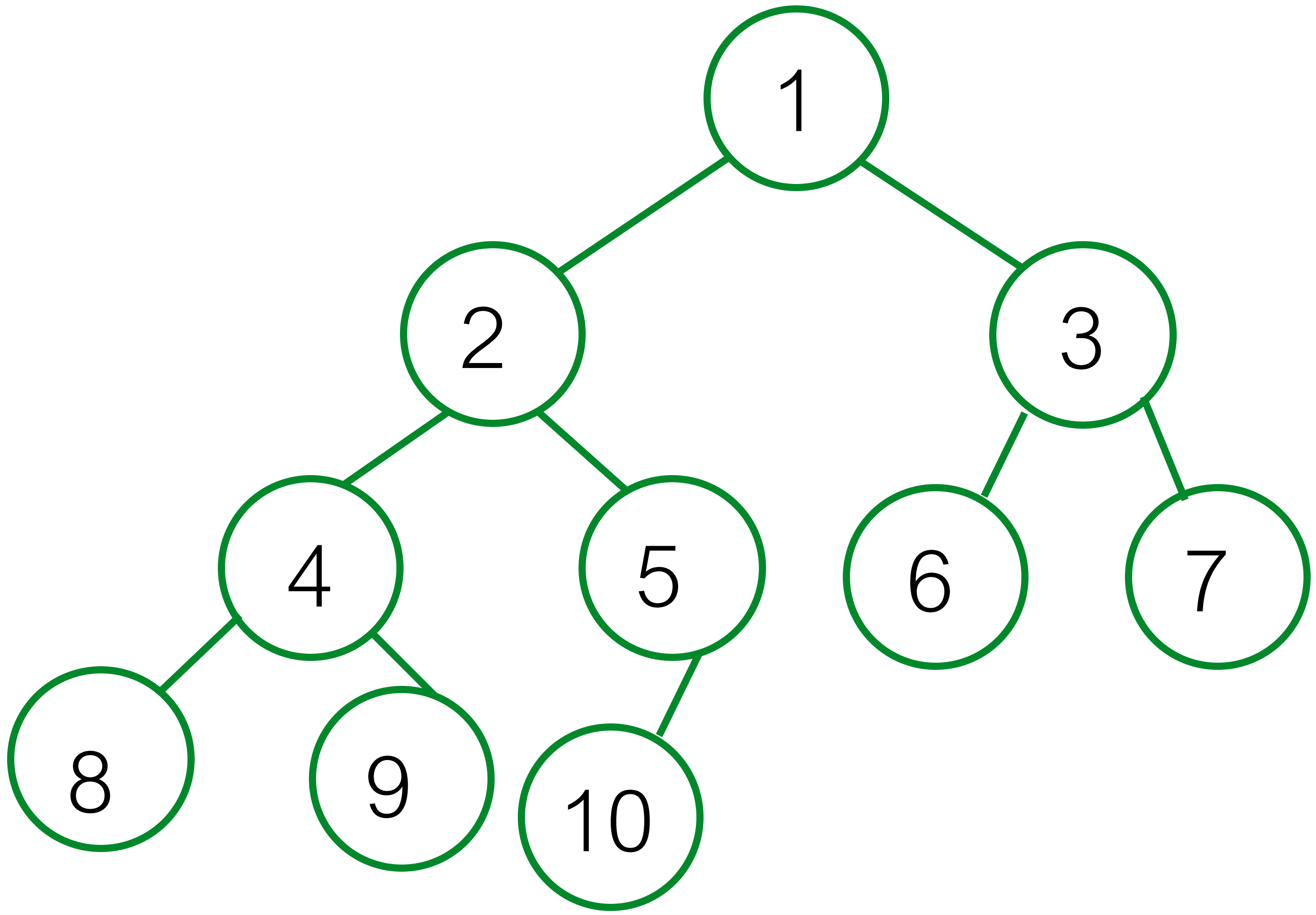


Parent Position	Child Left	Child Right
0	1	2
1	3	4
2	5	6
3	7	8
4	9	
k	**?**	**?**

shift

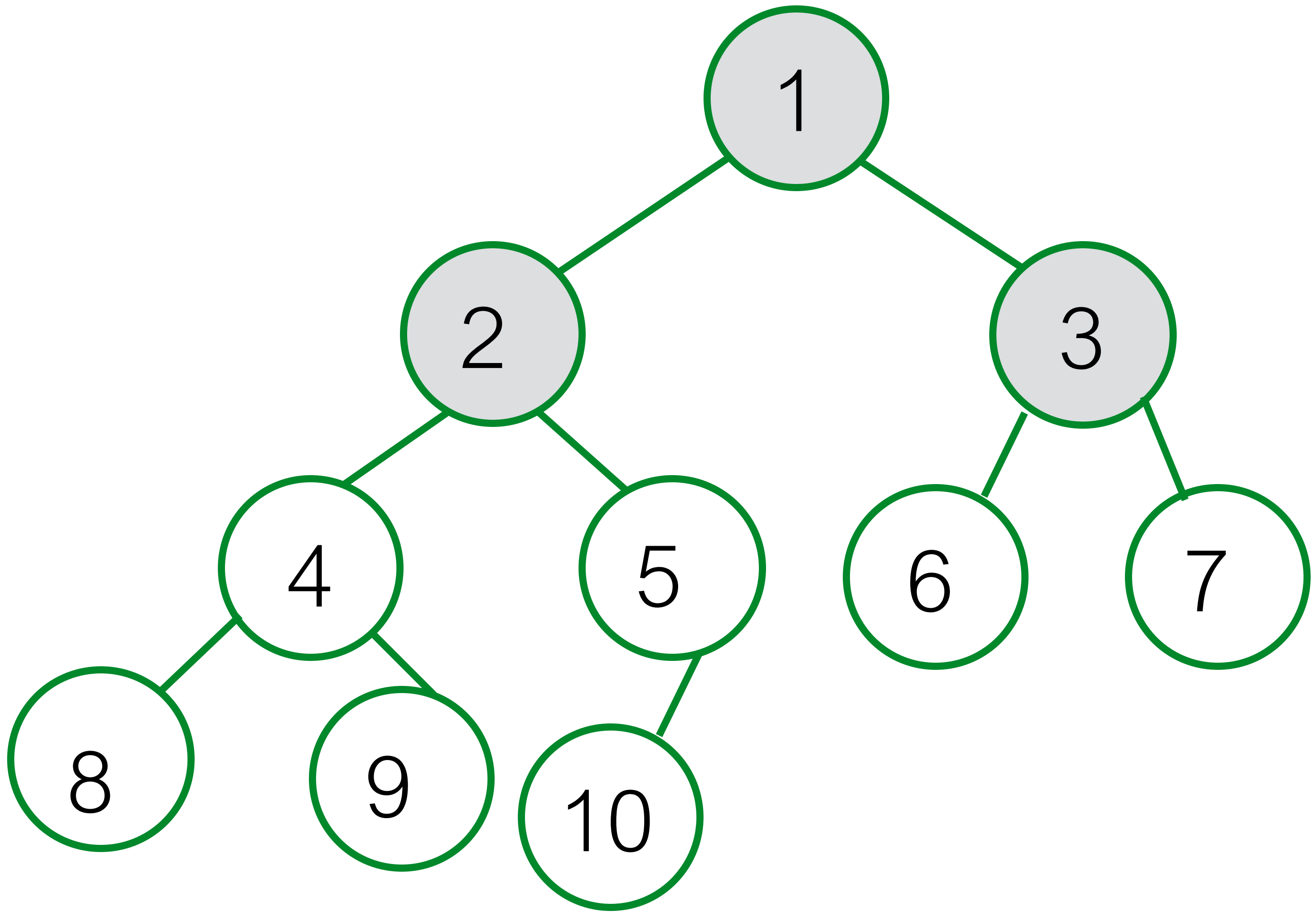






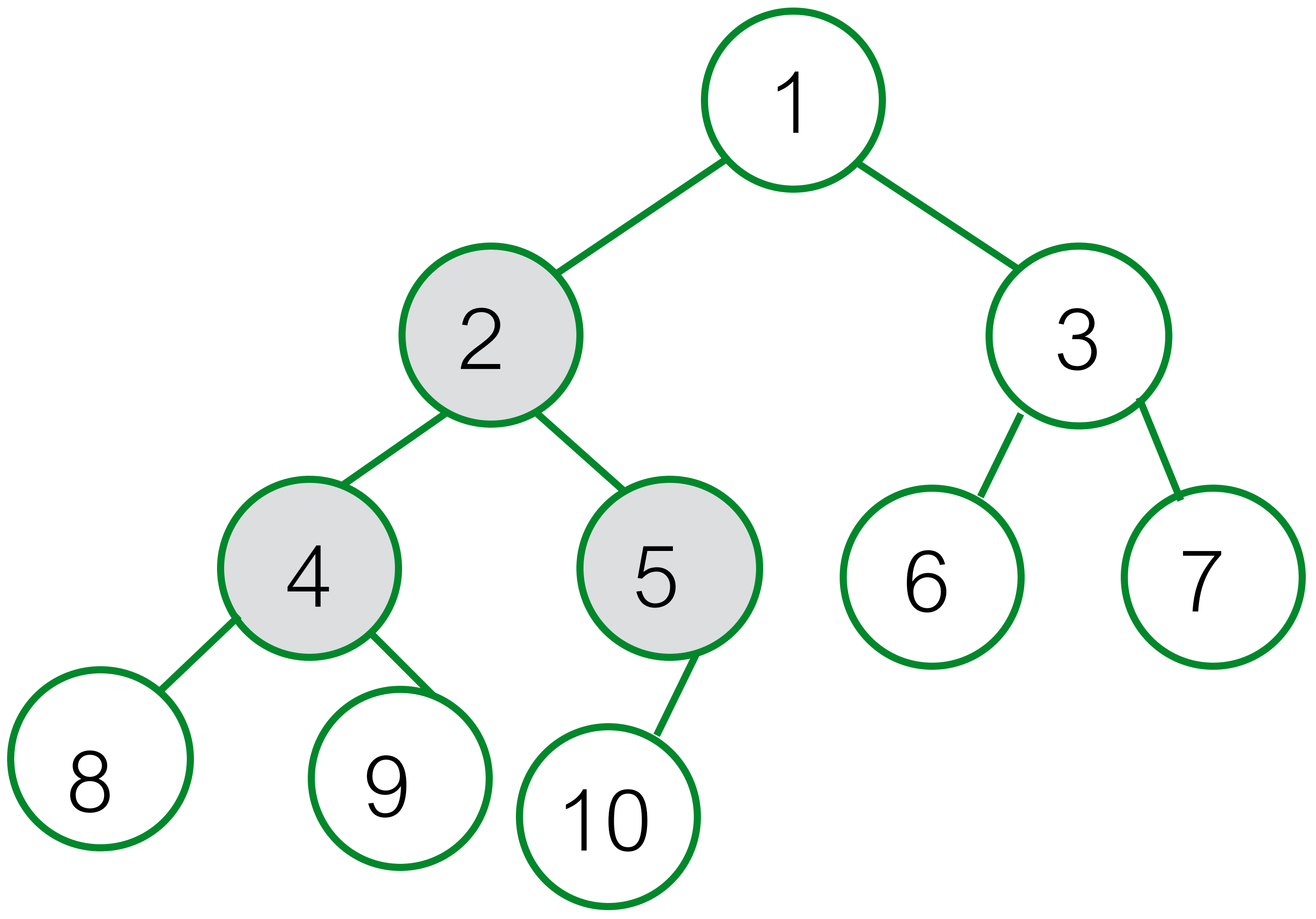
1	2	3	4	5	6	7	8	9	10
0	1	2	3	4	5	6	7	8	9	10

Parent Position	Child Left	Child Right

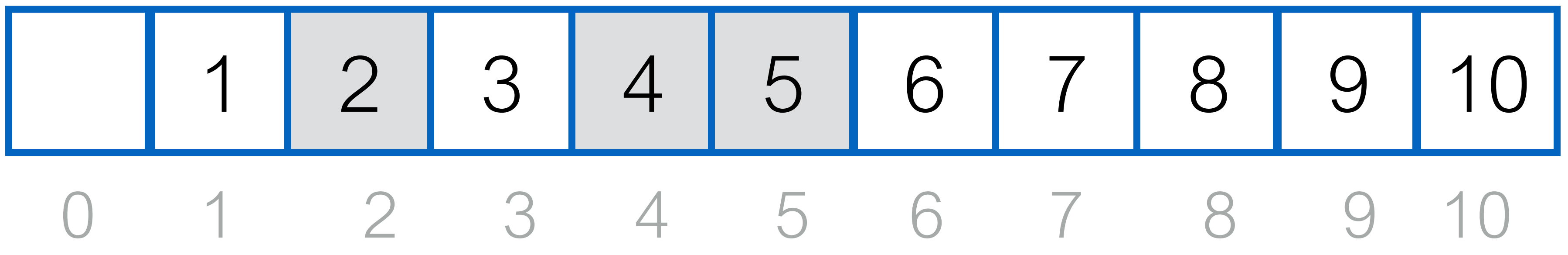


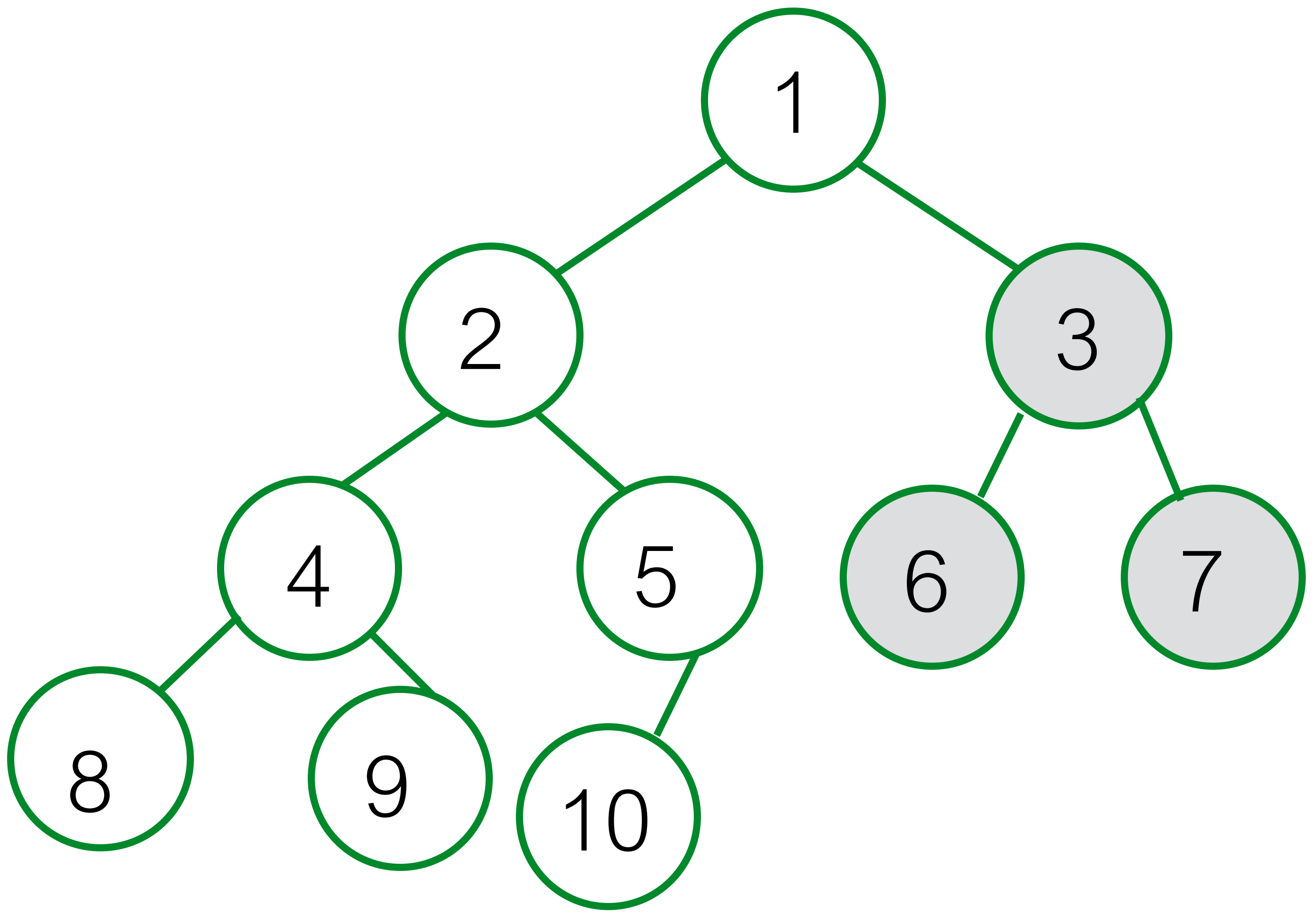
	1	2	3	4	5	6	7	8	9	10
0	1	2	3	4	5	6	7	8	9	10

Parent Position	Child Left	Child Right
1	2	3



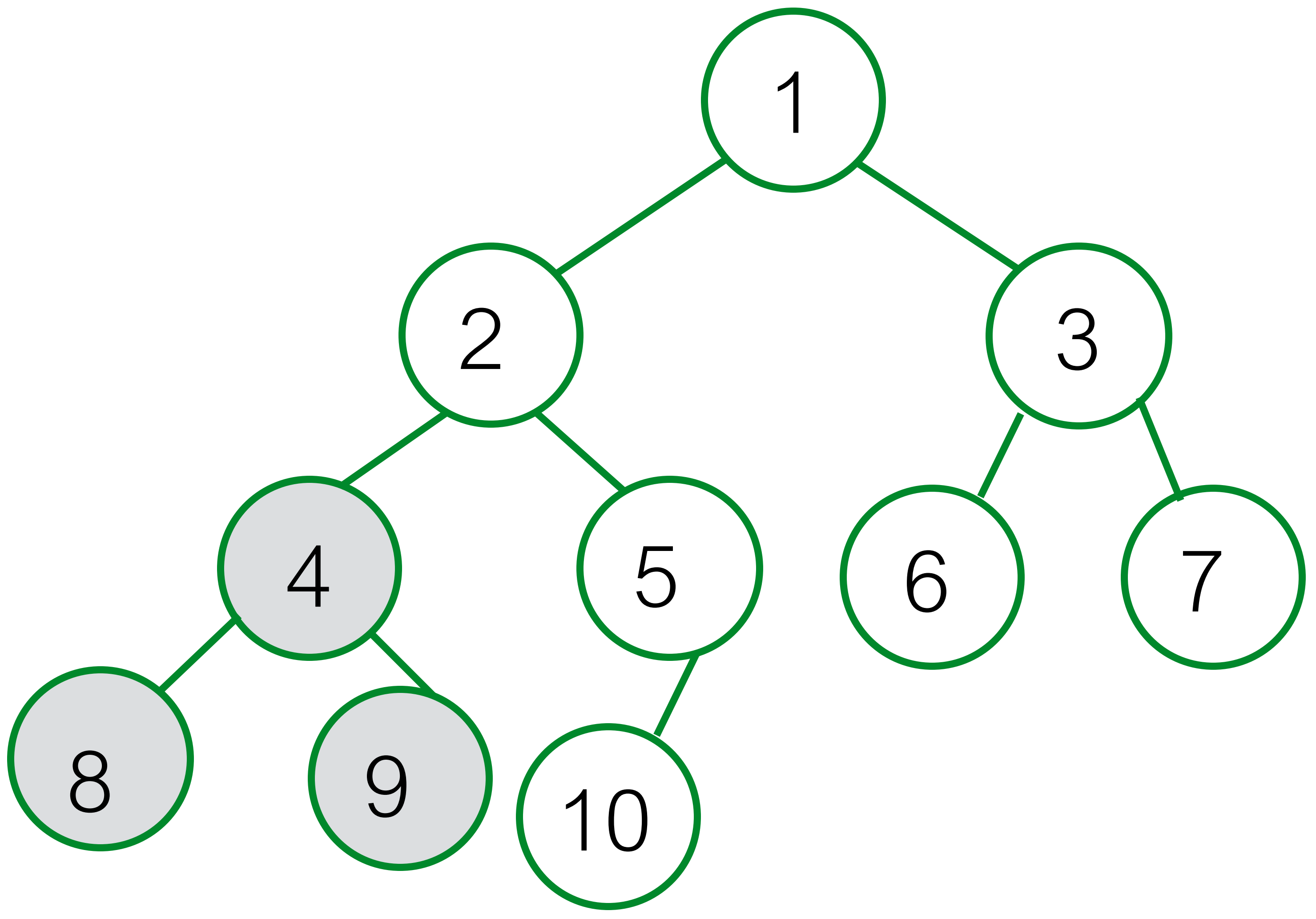
Parent Position	Child Left	Child Right
1	2	3
2	4	5



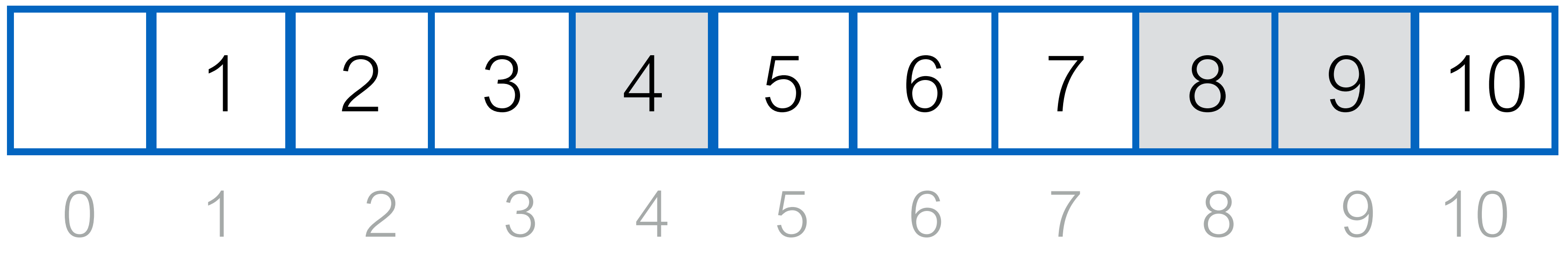


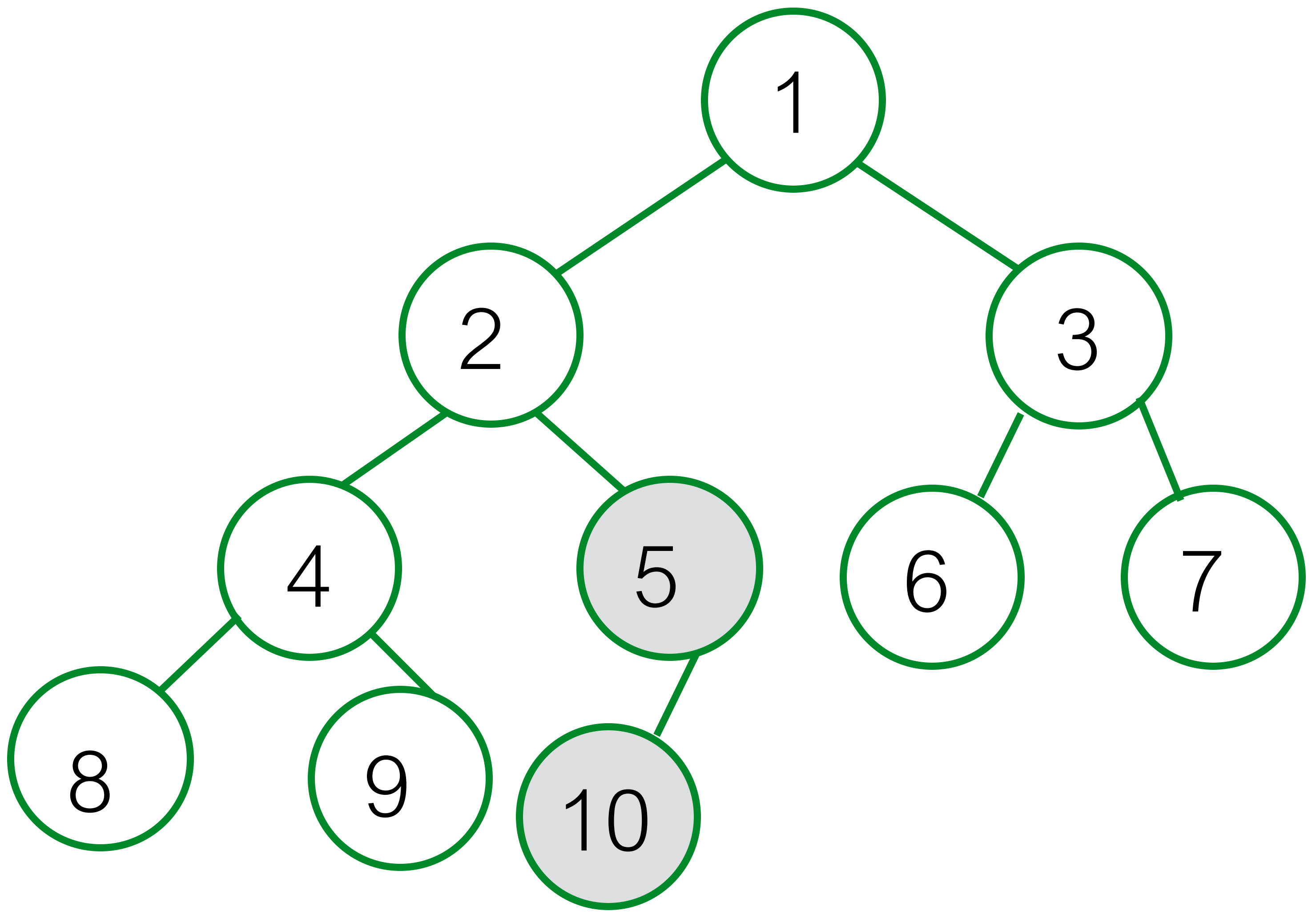
Parent Position	Child Left	Child Right
1	2	3
2	4	5
3	6	7

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

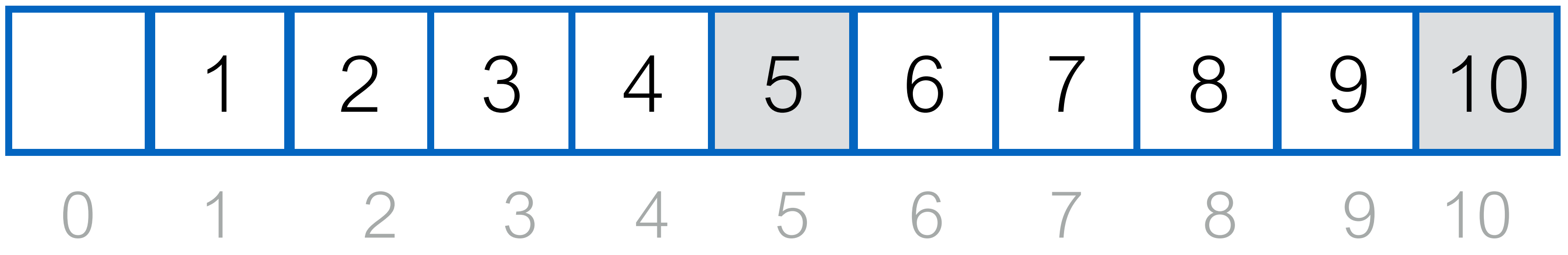


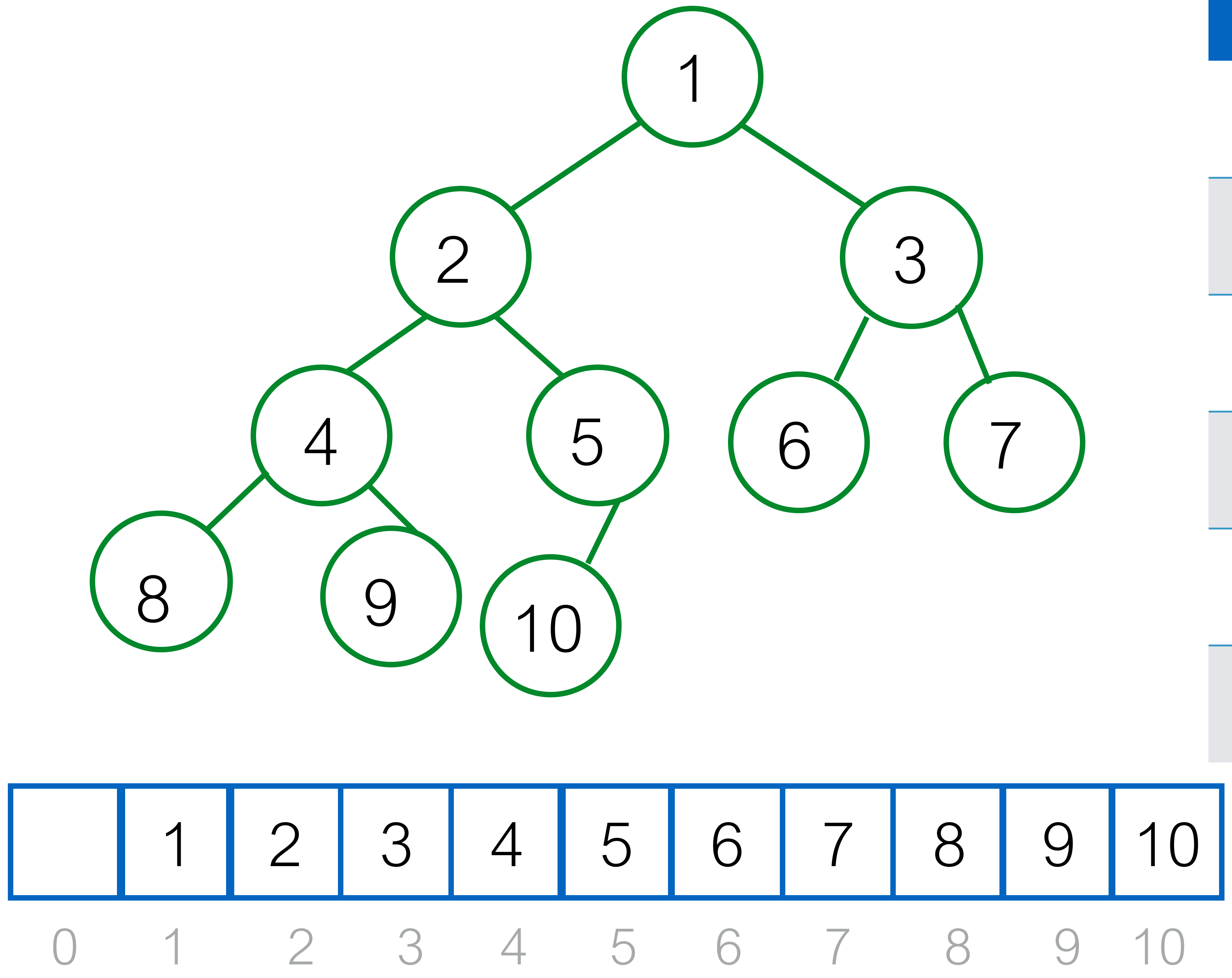
Parent Position	Child Left	Child Right
1	2	3
2	4	5
3	6	7
4	8	9



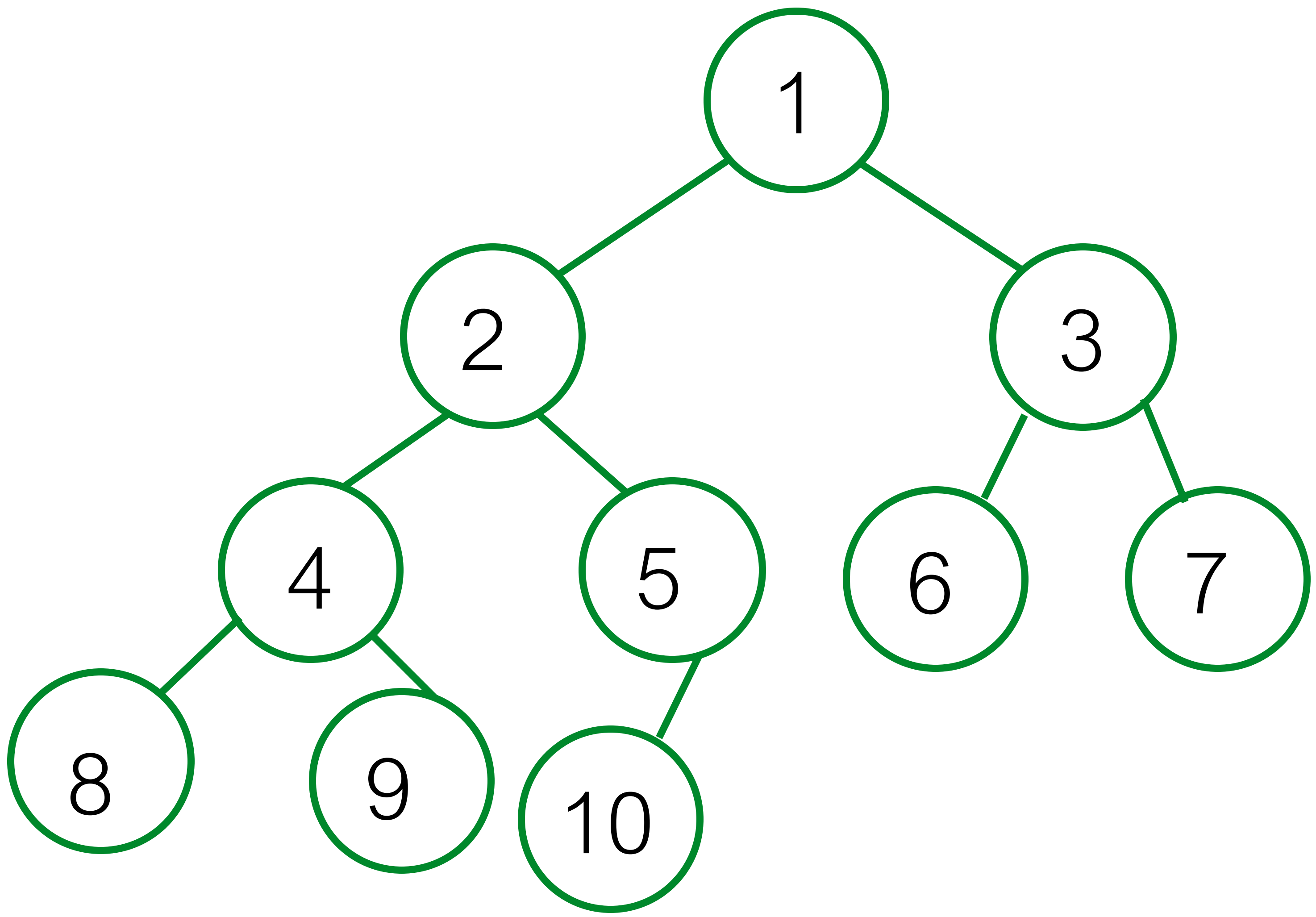


Parent Position	Child Left	Child Right
1	2	3
2	4	5
3	6	7
4	8	9
5	10	

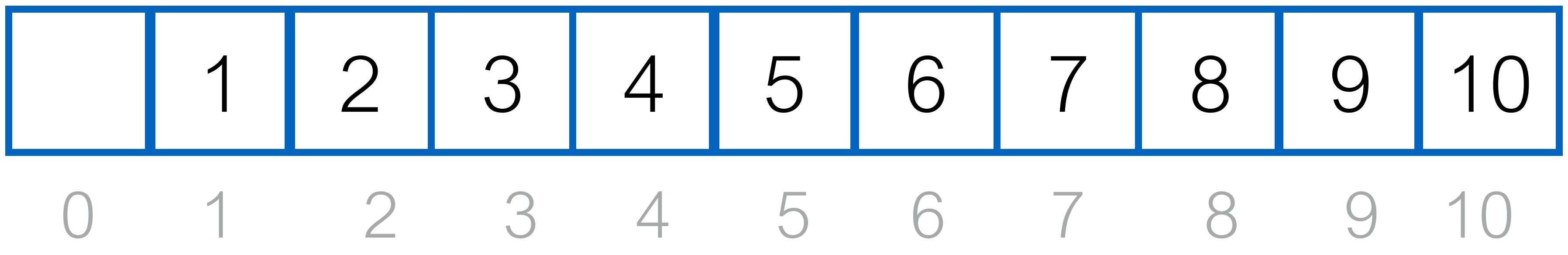


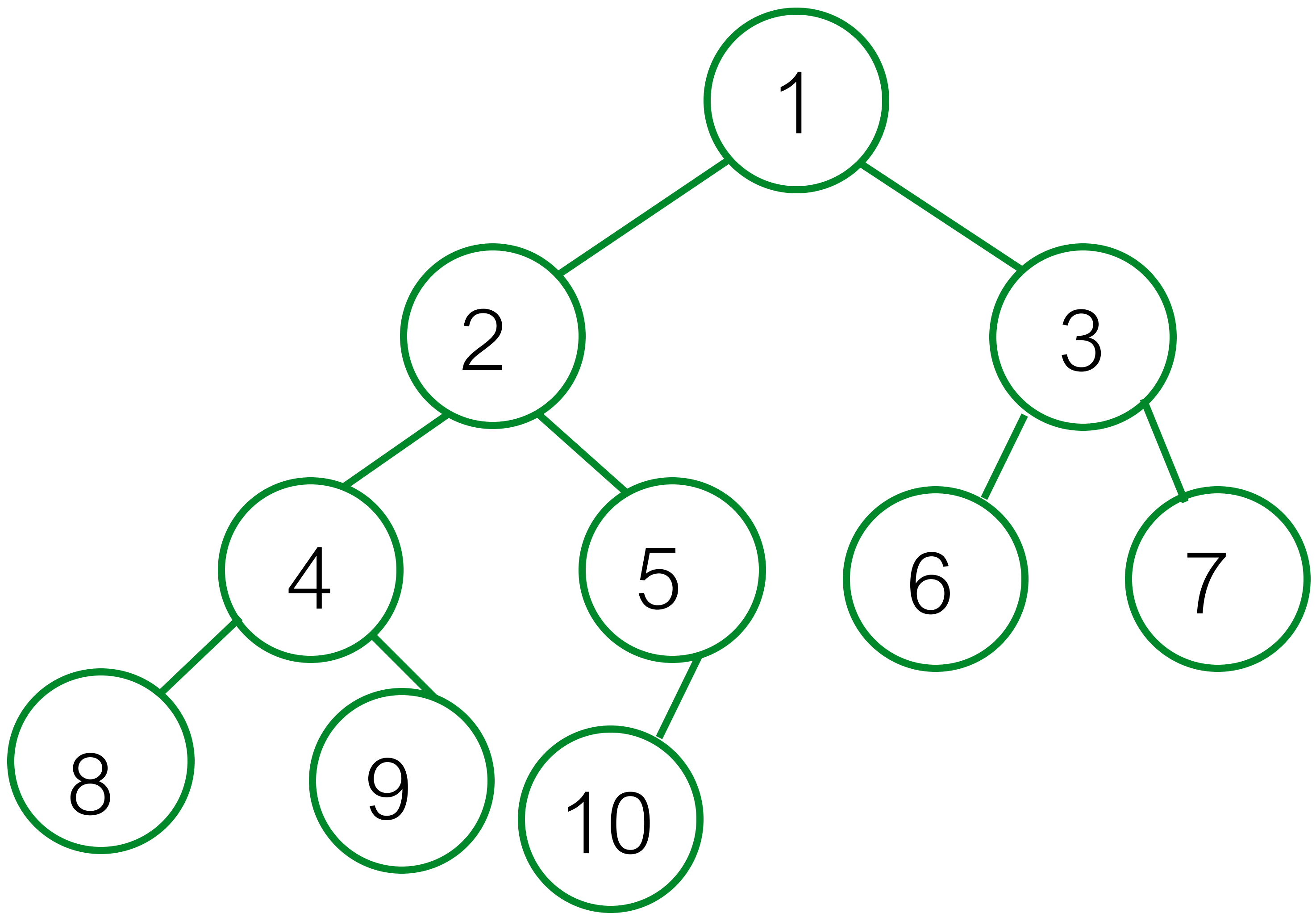


Parent Position	Child Left	Child Right
1	2	3
2	4	5
3	6	7
4	8	9
5	10	
k		

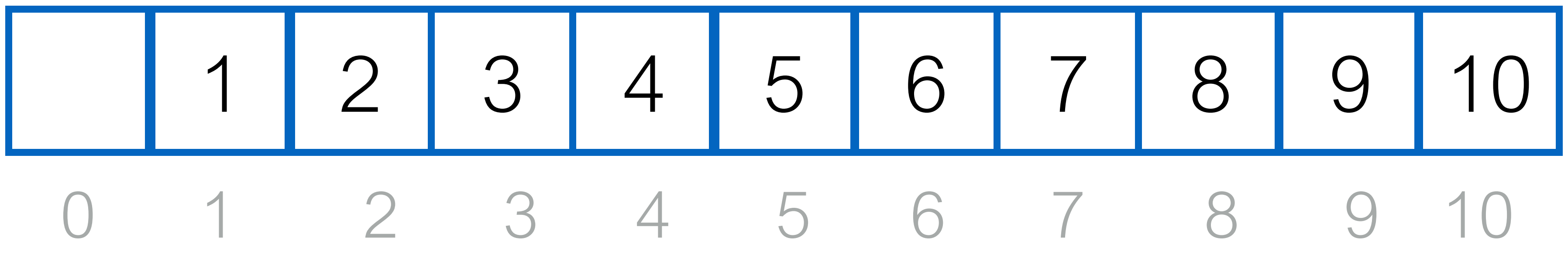


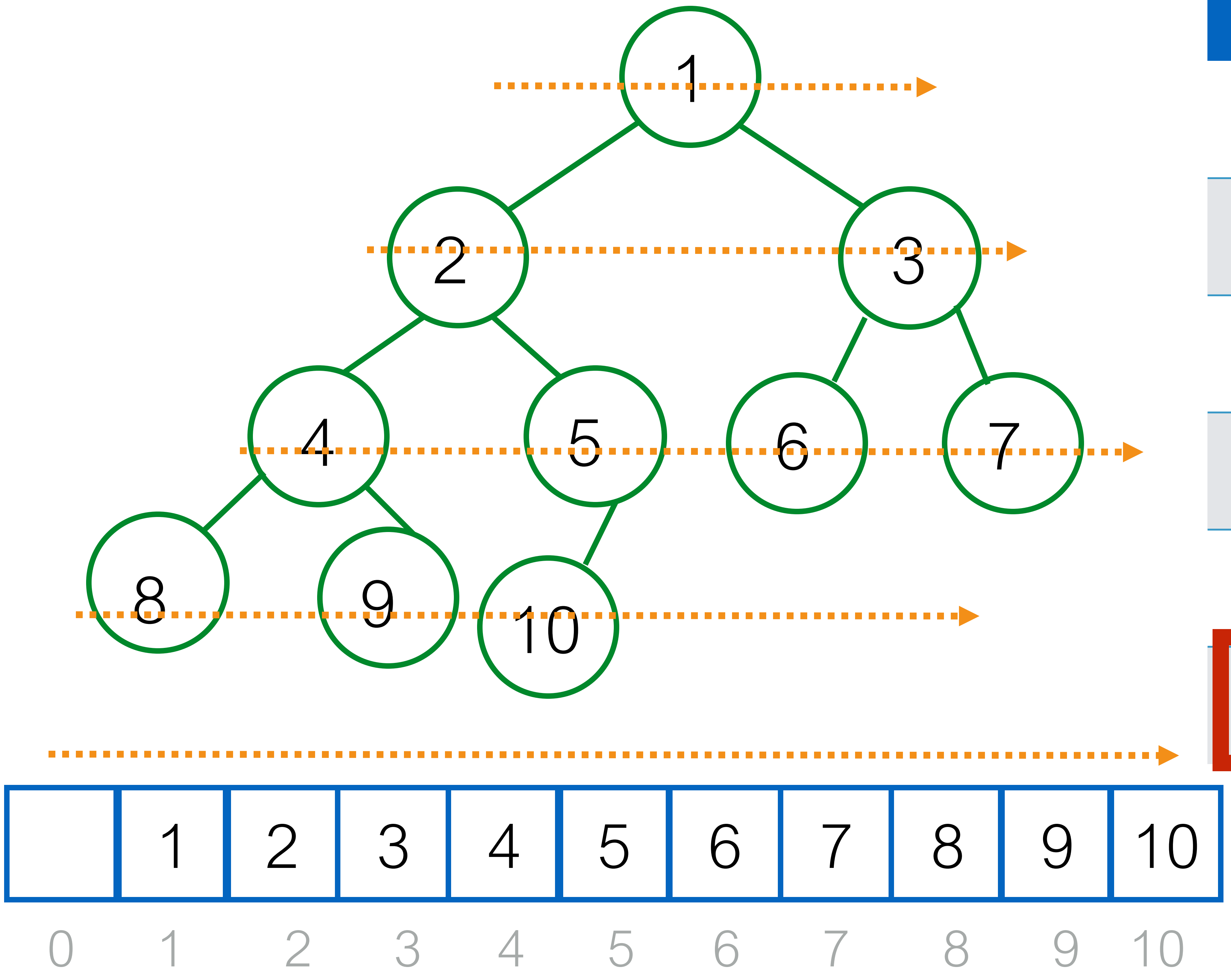
Parent Position	Child Left	Child Right
1	2	3
2	4	5
3	6	7
4	8	9
5	10	
k	**2*k**	





Parent Position	Child Left	Child Right
1	2	3
2	4	5
3	6	7
4	8	9
5	10	
k	**2*k**	**2*k+1**



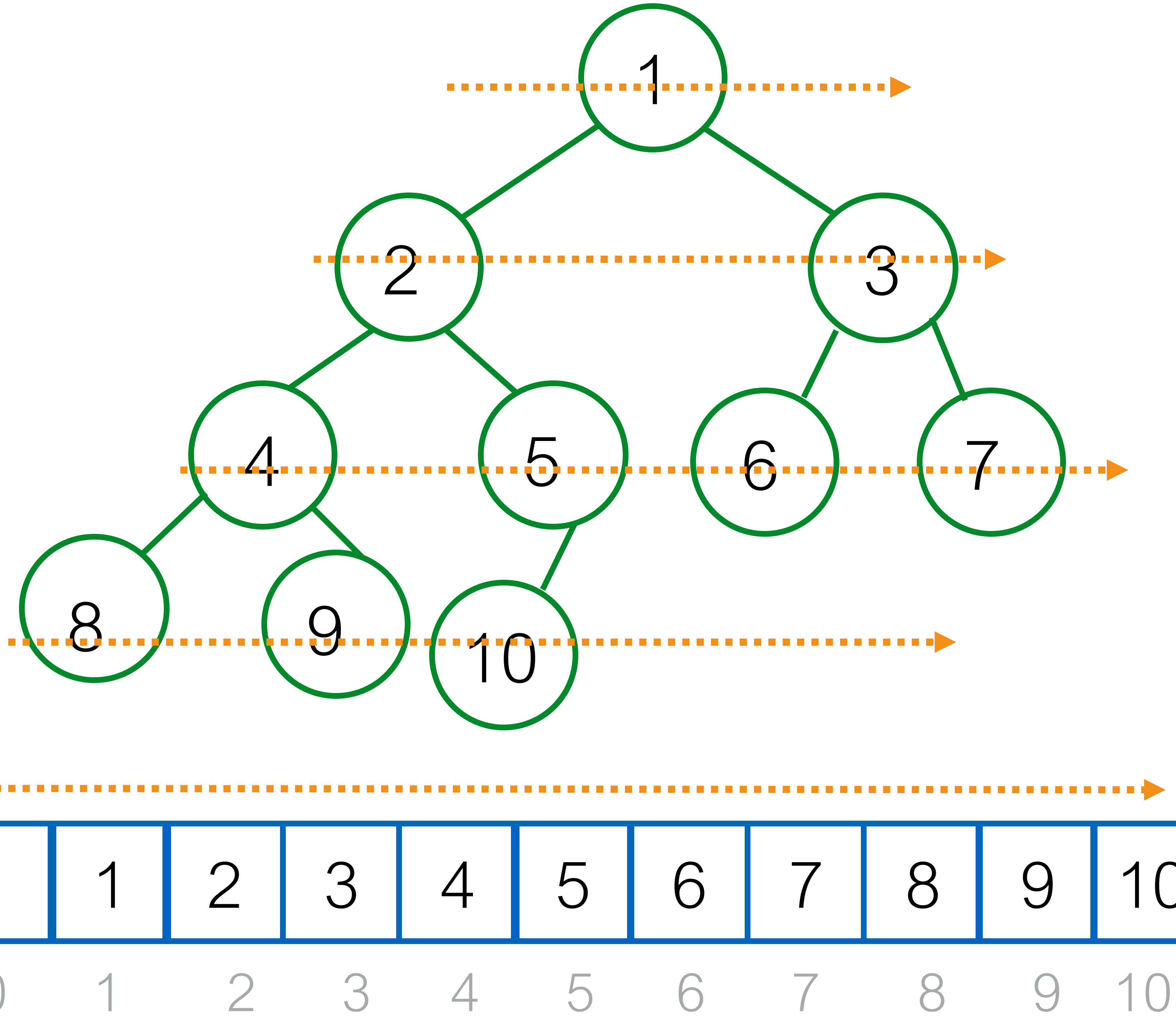


Parent Position	Child Left	Child Right
1	2	3
2	4	5
3	6	7
4	8	9
5	10	
k	**2*k**	**2*k+1**

Root at position 1

Children of k: 2*k 2*k+1 (if they exist)

Parent of k: position k//2 (except for root)



Parent Position	**Child Left**	**Child Right**
1	2	3
2	4	5
3	6	7
4	8	9
5	10	
k	**2*k**	**2*k+1**

A concrete implementation

```
from referential_array import build_array


class Heap:

    def __init__(self):
        self.count = 0
        self.array = build_array(100)
```

Initial capacity will be 100, we'll resize as required…

Operations

add:
- put at the bottom
- while order is broken, rise.

get_max:
- swap root with last item
- remove last item
- while order is broken, sink.

a.k.a Priority

```
def add(self, key, value):
    item = (key, value)
    if self.count + 1 < len(self.array):
        # there is space
        self.array[self.count+1] = item
    else:
        self._resize()
        self.array[self.count+1] = item
    # update counter
    self.count += 1
    self.rise(self.count)
```

We have done this before…

rise the last element -
swap with parent while order is broken

```
# Rise item at index k to its correct position
# Precondition: 1<= k <= self.count
def rise(self, k):
```

```
# Rise item at index k to its correct position
# Precondition: 1<= k <= self.count
def rise(self, k):
    while k > 1 and self.array[k] > self.array[k//2]:
```

the item is not at the root

node’s value

parent’s value

order is broken

```python
# Rise item at index k to its correct position
# Precondition: 1<= k <= self.count
def rise(self, k):
    while k > 1 and self.array[k] > self.array[k//2]:
        self.swap(k, k//2)
```

swap with the parent

```python
# Rise item at index k to its correct position
# Precondition: 1<= k <= self.count
def rise(self, k):
    while k > 1 and self.array[k] > self.array[k//2]:
        self.swap(k, k//2)
        k //= 2
```

update position of the node

```python
# Rise item at index k to its correct position
# Precondition: 1<= k <= self.count
def rise(self, k):
    while k > 1 and self.array[k] > self.array[k//2]:
        self.swap(k, k//2)
        k //= 2

def add(self, key, value):
    item = (key, value)
    if self.count + 1 < len(self.array):
        # there is space
        self.array[self.count+1] = item
    else:
        self._resize()
        self.array[self.count+1] = item
    # update counter
    self.count += 1
    self.rise(self.count)
```

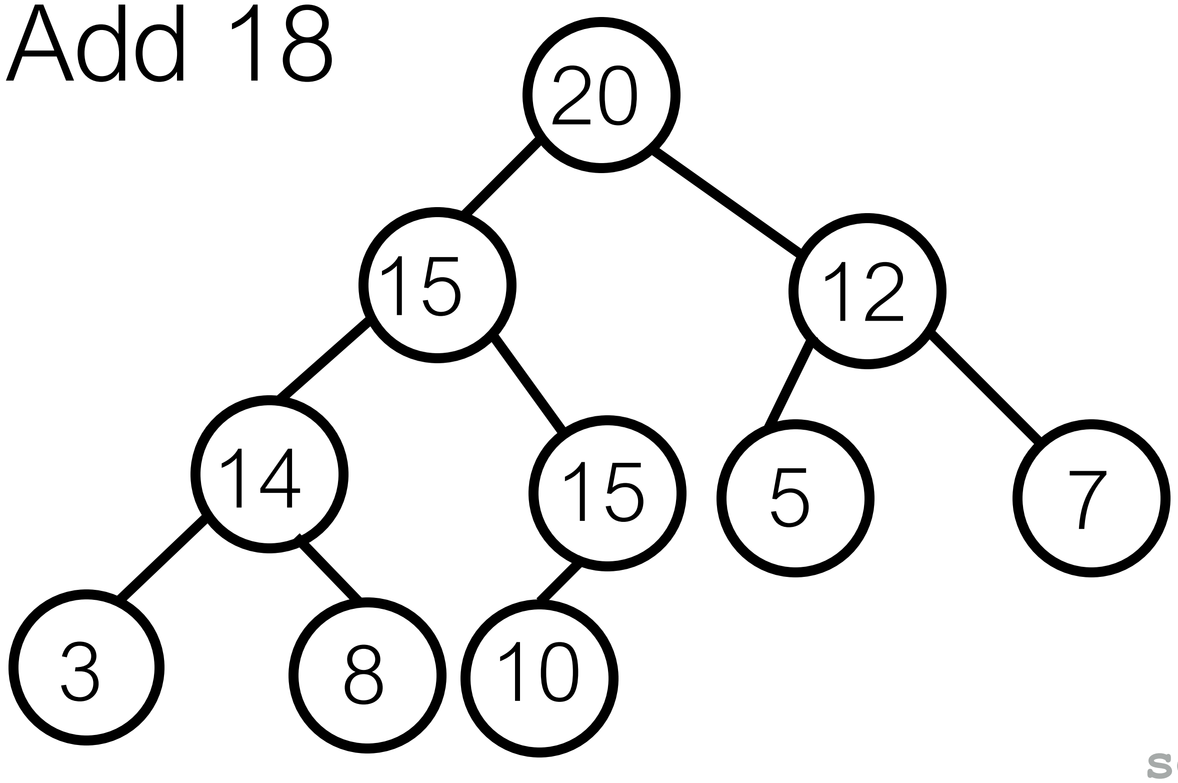
```
 def swap(self, i, j):
     self.array[i], self.array[j] = self.array[j], self.array[i]


# Rise item at index k to its correct position
# Precondition: 1<= k <= self.count
def rise(self, k):
    while k > 1 and self.array[k] > self.array[k//2]:
        self.swap(k, k//2)
        k //= 2

def add(self, key, value):
    item = (key, value)
    if self.count + 1 < len(self.array):
        # there is space
        self.array[self.count+1] = item
    else:
        self._resize()
        self.array[self.count+1] = item
    # update counter
    self.count += 1
    self.rise(self.count)
```

```
def _resize(self):
    new_array = build_array(2*len(self.array))
    for i in range(len(self.array)):
        new_array[i] = self.array[i]
    self.array = new_array
```

Add 18



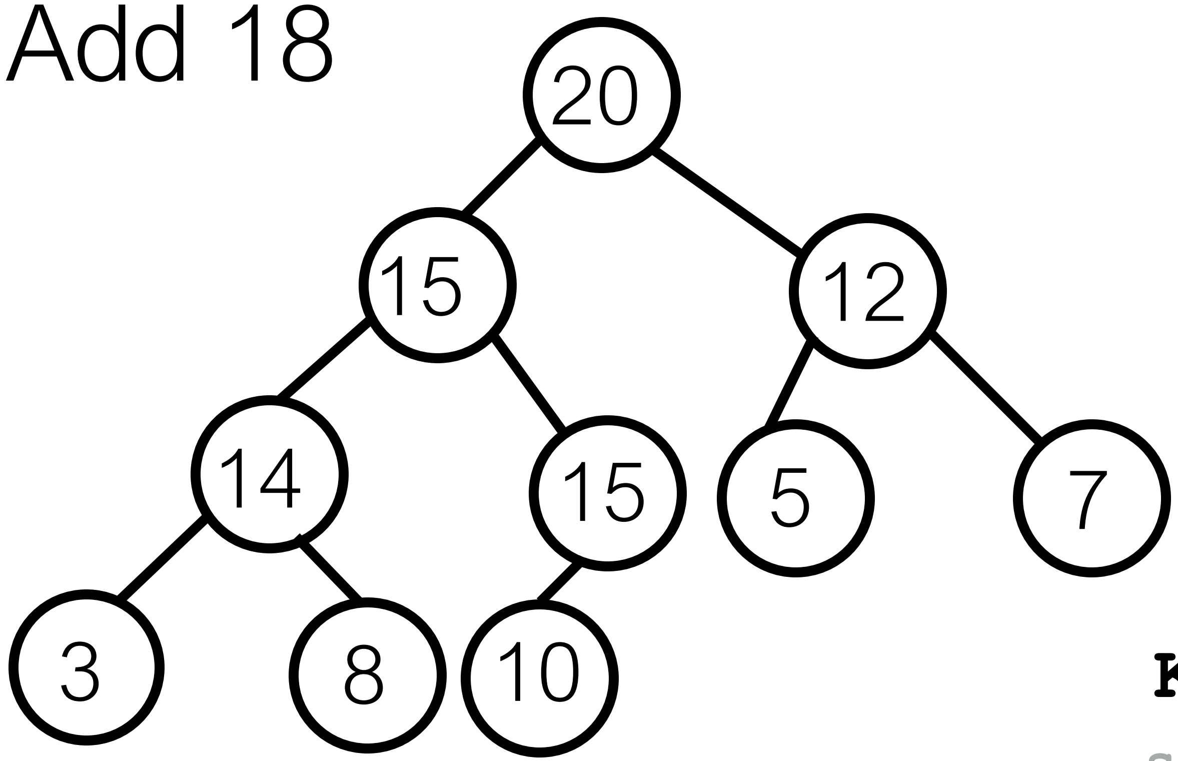
self.array

	20	15	12	14	15	5	7	3	8	10
0	1	2	3	4	5	6	7	8	9	10

```
def rise(self, k):
    while k > 1 and self.array[k//2][0] < self.array[k][0]:
        self.swap(k, k//2)
        k //= 2

def add(self, key, value):
    item = (key, value)
    if self.count + 1 < len(self.array):
        self.array[self.count+1] = item
    else:
        self._resize()
        self.array[self.count+1] = item
    self.count += 1
    self.rise(self.count)
```

Add 18



Key = 18 **self.count = 10**

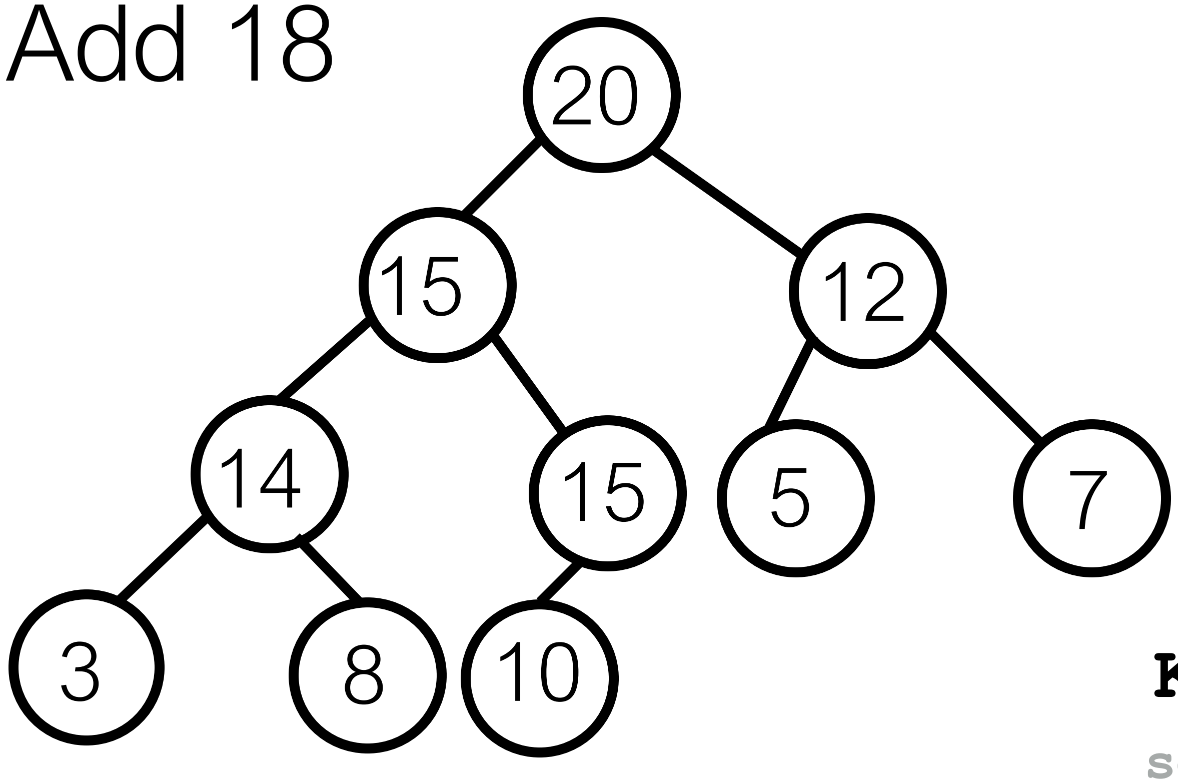
self.array

	20	15	12	14	15	5	7	3	8	10
0	1	2	3	4	5	6	7	8	9	10

```
def rise(self, k):
    while k > 1 and self.array[k//2][0] < self.array[k][0]:
        self.swap(k, k//2)
        k //= 2

def add(self, key, value):
    item = (key, value)
    if self.count + 1 < len(self.array):
        self.array[self.count+1] = item
    else:
        self._resize()
        self.array[self.count+1] = item
    self.count += 1
    self.rise(self.count)
```

Add 18



Key = 18   **self.count = 10**

self.array

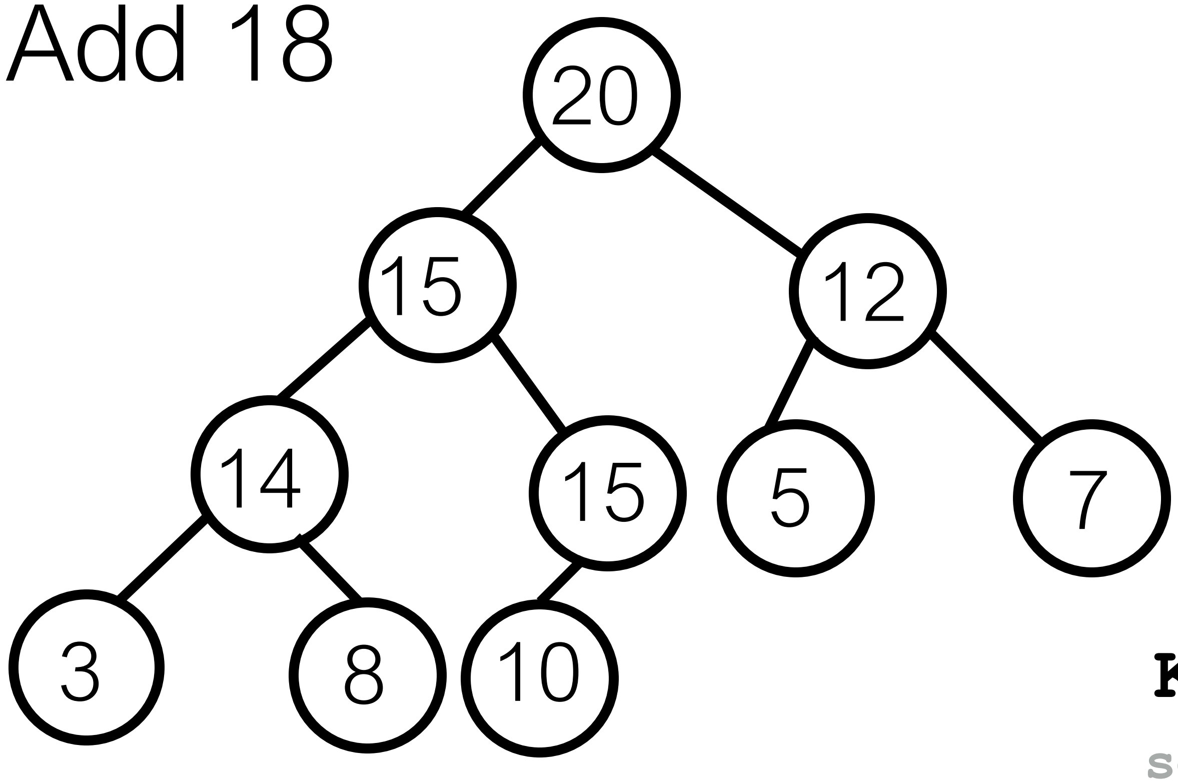
	20	15	12	14	15	5	7	3	8	10
0	1	2	3	4	5	6	7	8	9	10

```
def rise(self, k):
    while k > 1 and self.array[k//2][0] < self.array[k][0]:
        self.swap(k, k//2)
        k //= 2

def add(self, key, value):
    item = (key, value)
    if self.count + 1 < len(self.array):
        self.array[self.count+1] = item
    else:
        self._resize()
        self.array[self.count+1] = item
    self.count += 1
    self.rise(self.count)
```

10 + 1 < 11? False

Add 18



Key = 18   **self.count = 10**

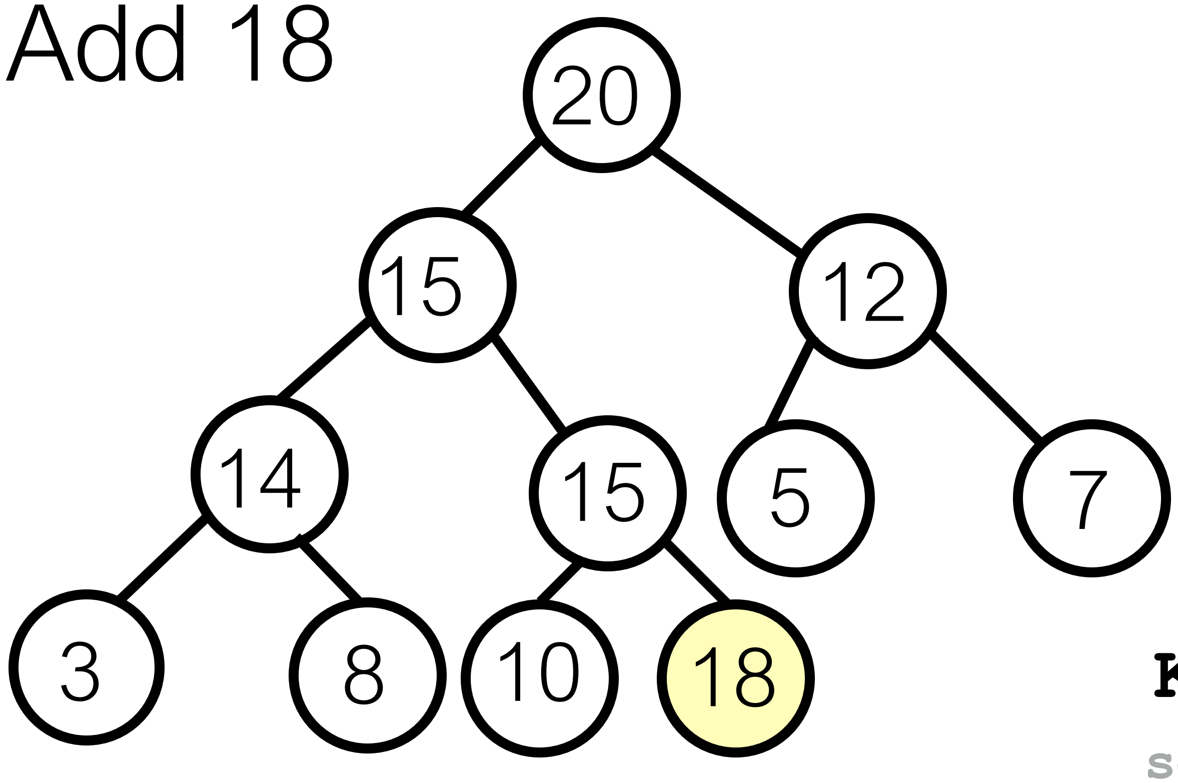
self.array

	20	15	12	14	15	5	7	3	8	10
0	1	2	3	4	5	6	7	8	9	10

```
def rise(self, k):
    while k > 1 and self.array[k//2][0] < self.array[k][0]:
        self.swap(k, k//2)
        k //= 2

def add(self, key, value):
    item = (key, value)
    if self.count + 1 < len(self.array):
        self.array[self.count+1] = item
    else:
        self._resize()
        self.array[self.count+1] = item
    self.count += 1
    self.rise(self.count)
```

Add 18



Key = 18 **self.count = 10**

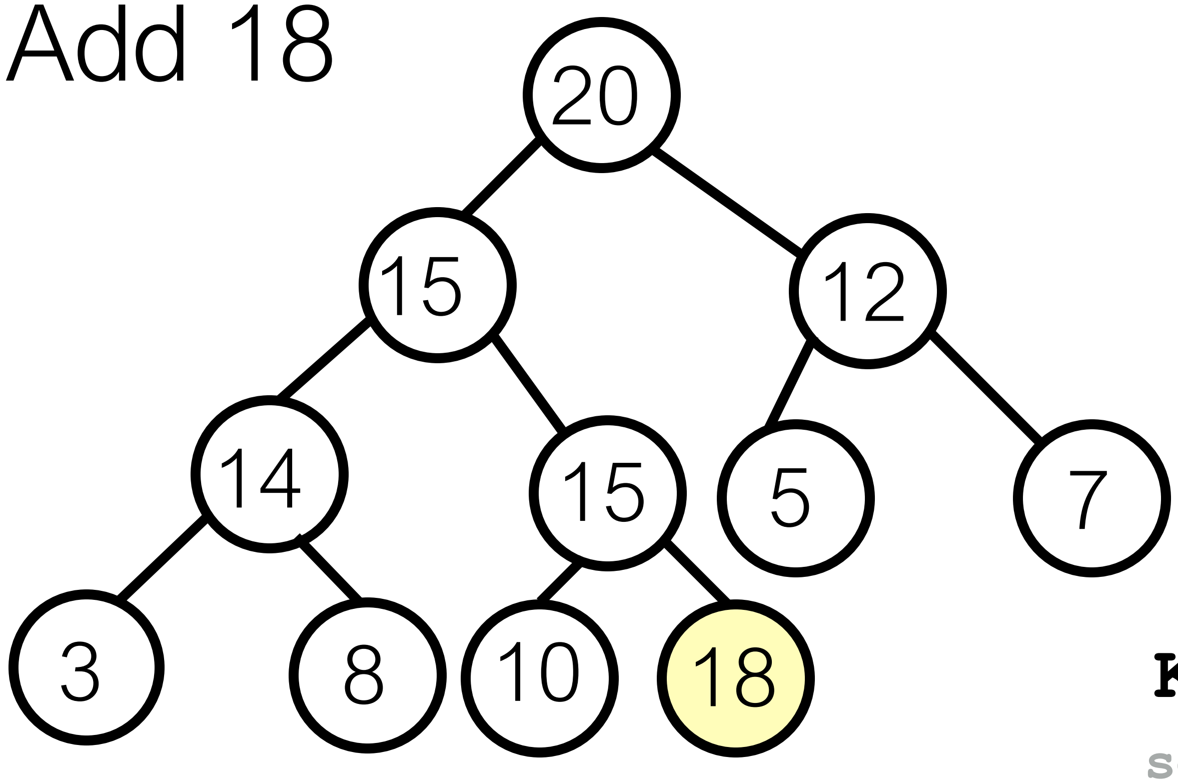
self.array

0	1	2	3	4	5	6	7	8	9	10	11
	20	15	12	14	15	5	7	3	8	10	18

```
def rise(self, k):
    while k > 1 and self.array[k//2][0] < self.array[k][0]:
        self.swap(k, k//2)
        k //= 2

def add(self, key, value):
    item = (key, value)
    if self.count + 1 < len(self.array):
        self.array[self.count+1] = item
    else:
        self._resize()
        self.array[self.count+1] = item
    self.count += 1
    self.rise(self.count)
```

Add 18



Key = 18 **self.count = 11**

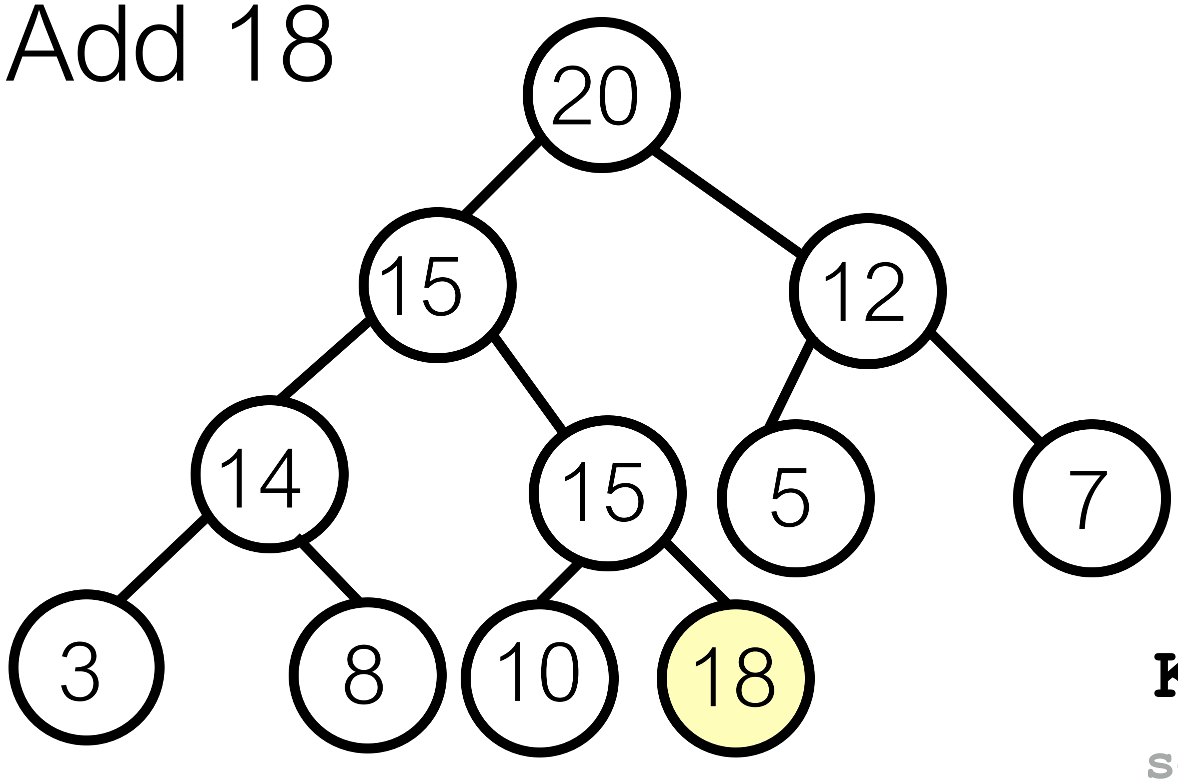
self.array

	20	15	12	14	15	5	7	3	8	10	18
0	1	2	3	4	5	6	7	8	9	10	11

```
def rise(self, k):
    while k > 1 and self.array[k//2][0] < self.array[k][0]:
        self.swap(k, k//2)
        k //= 2

def add(self, key, value):
    item = (key, value)
    if self.count + 1 < len(self.array):
        self.array[self.count+1] = item
    else:
        self._resize()
        self.array[self.count+1] = item
    self.count += 1
    self.rise(self.count)
```

Add 18



Key = 18 **self.count = 11**

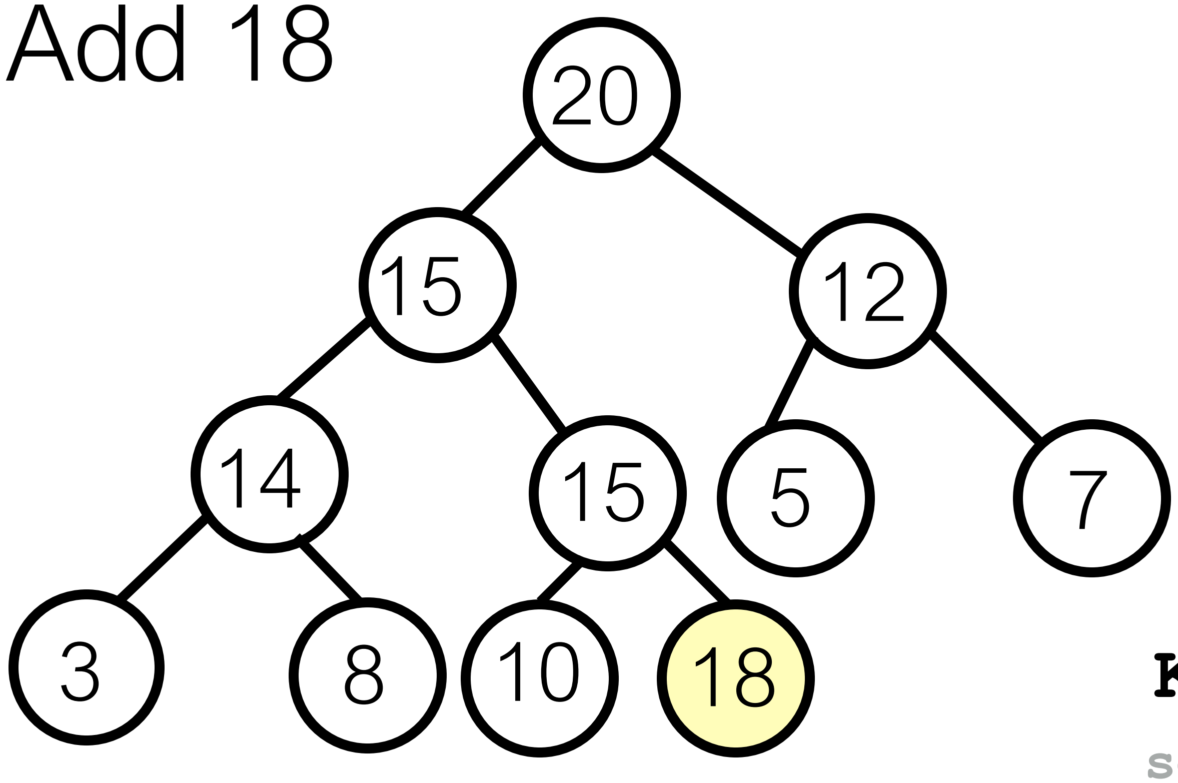
self.array

	20	15	12	14	15	5	7	3	8	10	18
0	1	2	3	4	5	6	7	8	9	10	11

```
def rise(self, k):
    while k > 1 and self.array[k//2][0] < self.array[k][0]:
        self.swap(k, k//2)
        k //= 2

def add(self, key, value):
    item = (key, value)
    if self.count + 1 < len(self.array):
        self.array[self.count+1] = item
    else:
        self._resize()
        self.array[self.count+1] = item
    self.count += 1
    self.rise(self.count)
```

Add 18



Key = 18 **self.count = 11**

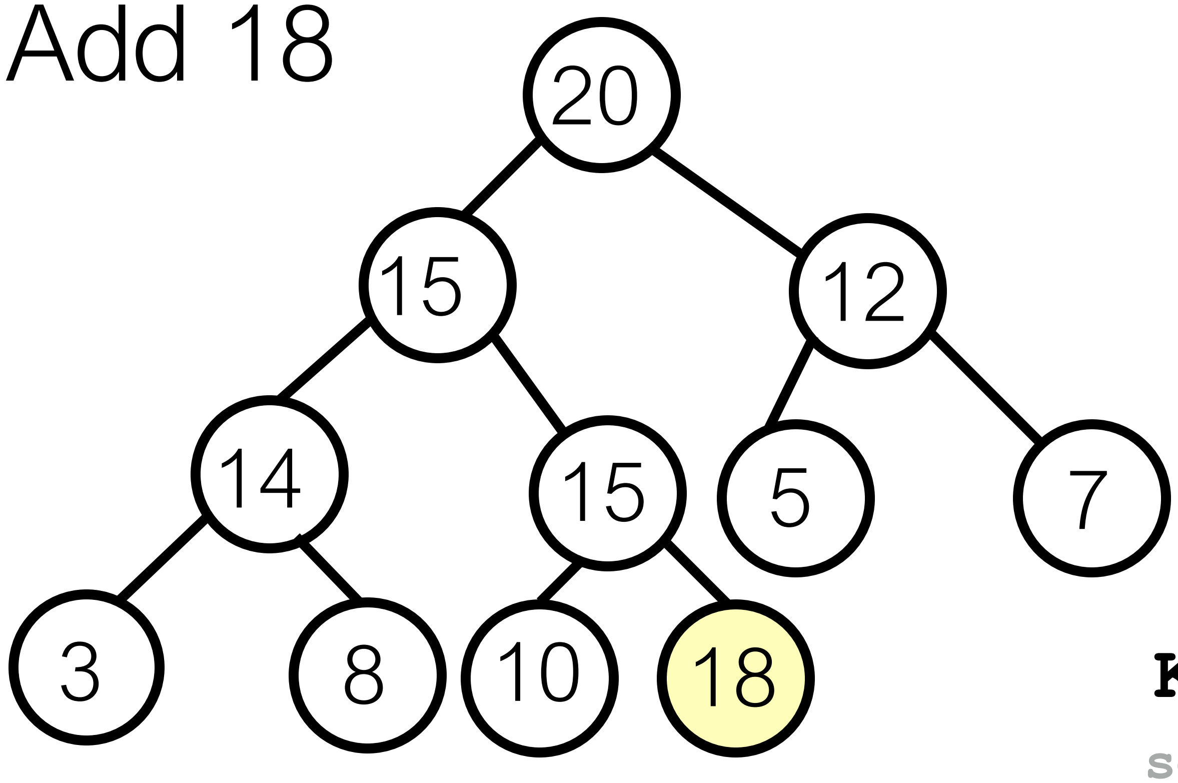
self.array

	20	15	12	14	15	5	7	3	8	10	18
0	1	2	3	4	5	6	7	8	9	10	11

```
def rise(self, k):
    while k > 1 and self.array[k//2][0] < self.array[k][0]:
        self.swap(k, k//2)
        k //= 2

def add(self, key, value):
    item = (key, value)
    if self.count + 1 < len(self.array):
        self.array[self.count+1] = item
    else:
        self._resize()
        self.array[self.count+1] = item
    self.count += 1
    self.rise(self.count)
```

Add 18



Key = 18 **self.count = 11**

self.array

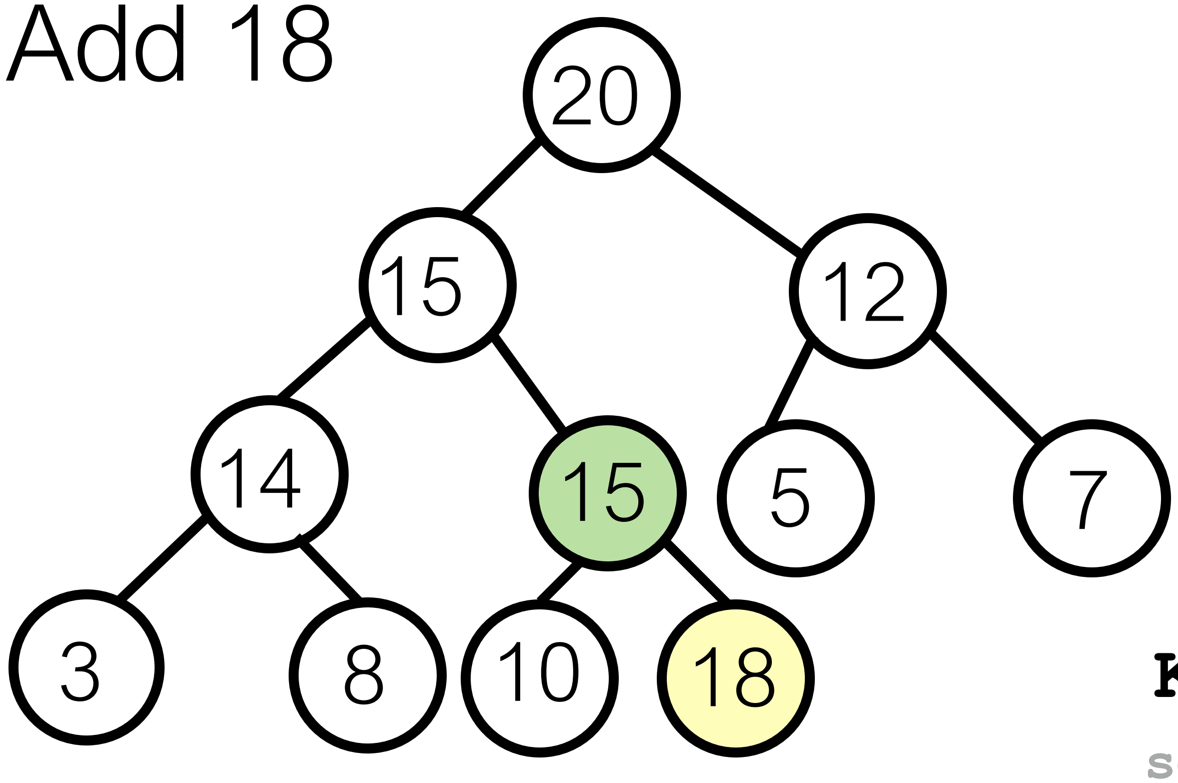
	20	15	12	14	15	5	7	3	8	10	18
0	1	2	3	4	5	6	7	8	9	10	11

```
def rise(self, k):
    while k > 1 and self.array[k//2][0] < self.array[k][0]:
        self.swap(k, k//2)
        k //= 2

def add(self, key, value):
    item = (key, value)
    if self.count + 1 < len(self.array):
        self.array[self.count+1] = item
    else:
        self._resize()
        self.array[self.count+1] = item
    self.count += 1
    self.rise(self.count)
```

k = 11

Add 18



Key = 18 **self.count = 11**

self.array

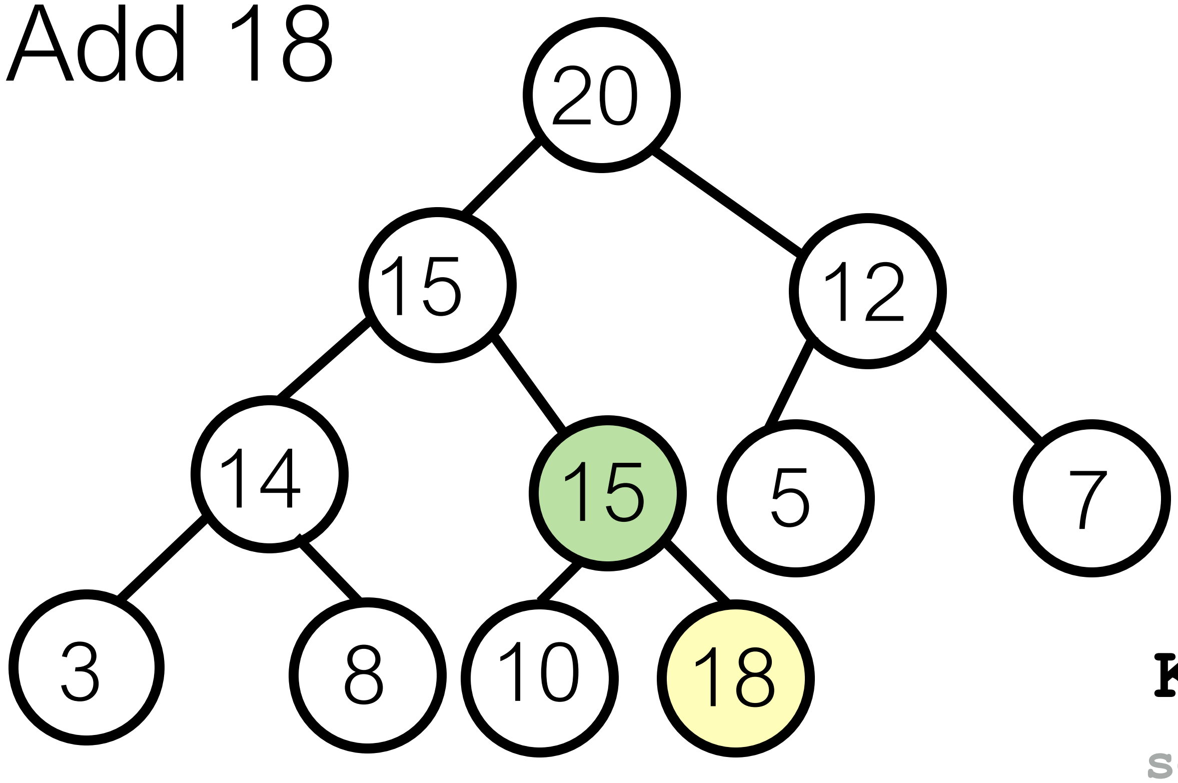
	20	15	12	14	15	5	7	3	8	10	18
0	1	2	3	4	5	6	7	8	9	10	11

```
def rise(self, k):
    while k > 1 and self.array[k//2][0] < self.array[k][0]:
        self.swap(k, k//2)
        k //= 2
```

```
def add(self, key, value):
    item = (key, value)
    if self.count + 1 < len(self.array):
        self.array[self.count+1] = item
    else:
        self._resize()
        self.array[self.count+1] = item
    self.count += 1
    self.rise(self.count)
```

k = 11

Add 18



Key = 18 **self.count = 11**

self.array

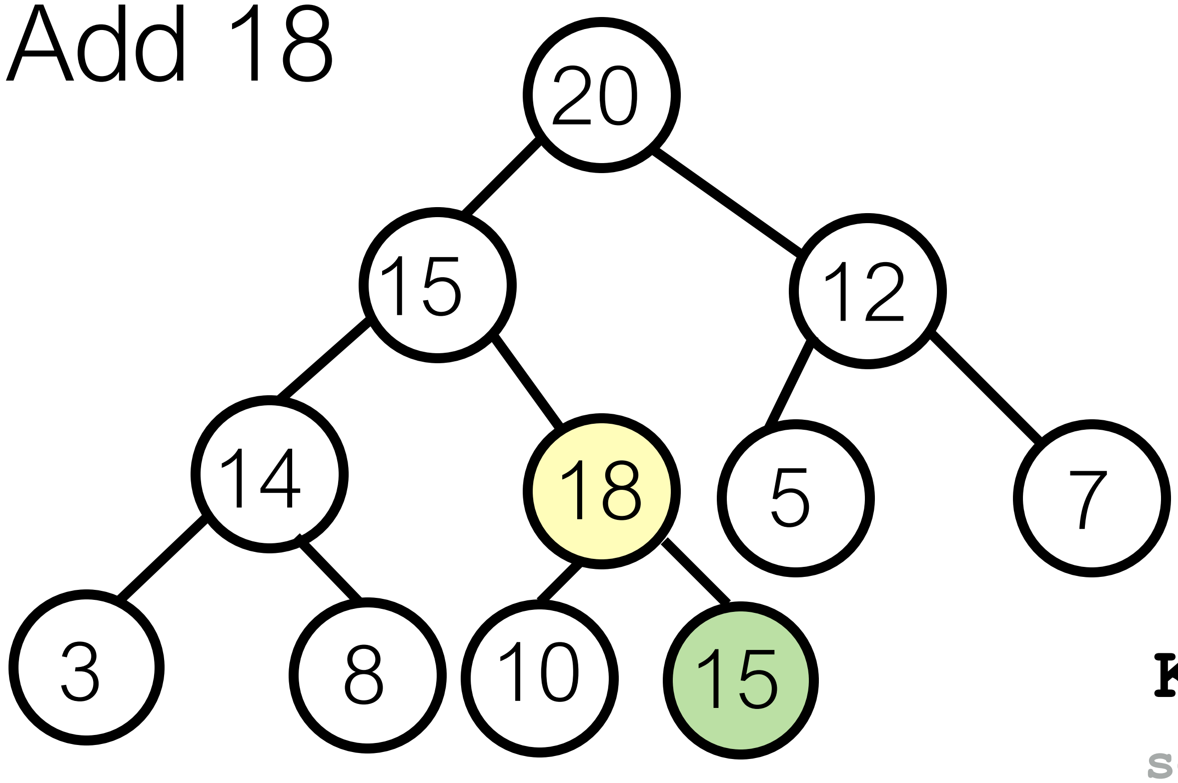
	20	15	12	14	15	5	7	3	8	10	18
0	1	2	3	4	5	6	7	8	9	10	11

```
def rise(self, k):
    while k > 1 and self.array[k//2][0] < self.array[k][0]:
        self.swap(k, k//2)
        k //= 2
```

```
def add(self, key, value):
    item = (key, value)
    if self.count + 1 < len(self.array):
        self.array[self.count+1] = item
    else:
        self._resize()
        self.array[self.count+1] = item
    self.count += 1
    self.rise(self.count)
```

k = 11

Add 18



Key = 18 **self.count = 11**

self.array

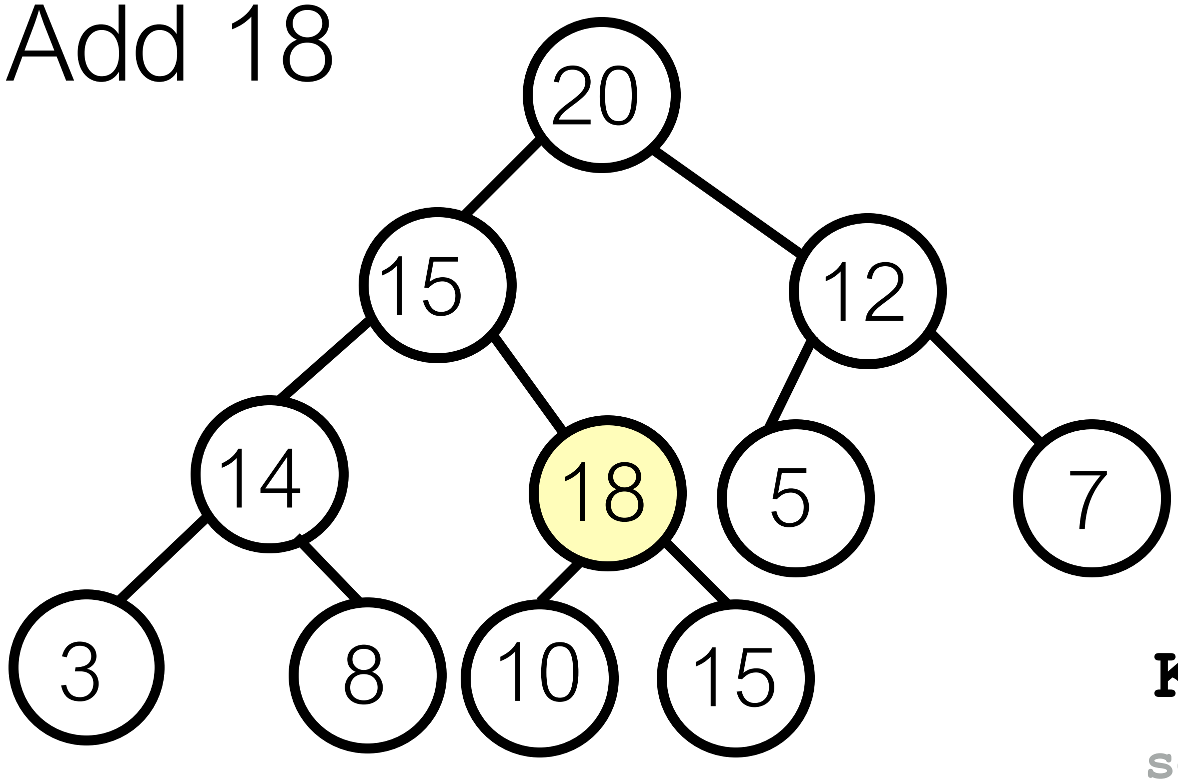
	20	15	12	14	18	5	7	3	8	10	15
0	1	2	3	4	5	6	7	8	9	10	11

```
def rise(self, k):
    while k > 1 and self.array[k//2][0] < self.array[k][0]:
        self.swap(k, k//2)
        k //= 2

def add(self, key, value):
    item = (key, value)
    if self.count + 1 < len(self.array):
        self.array[self.count+1] = item
    else:
        self._resize()
        self.array[self.count+1] = item
    self.count += 1
    self.rise(self.count)
```

k = 11

Add 18



Key = 18 **self.count = 11**

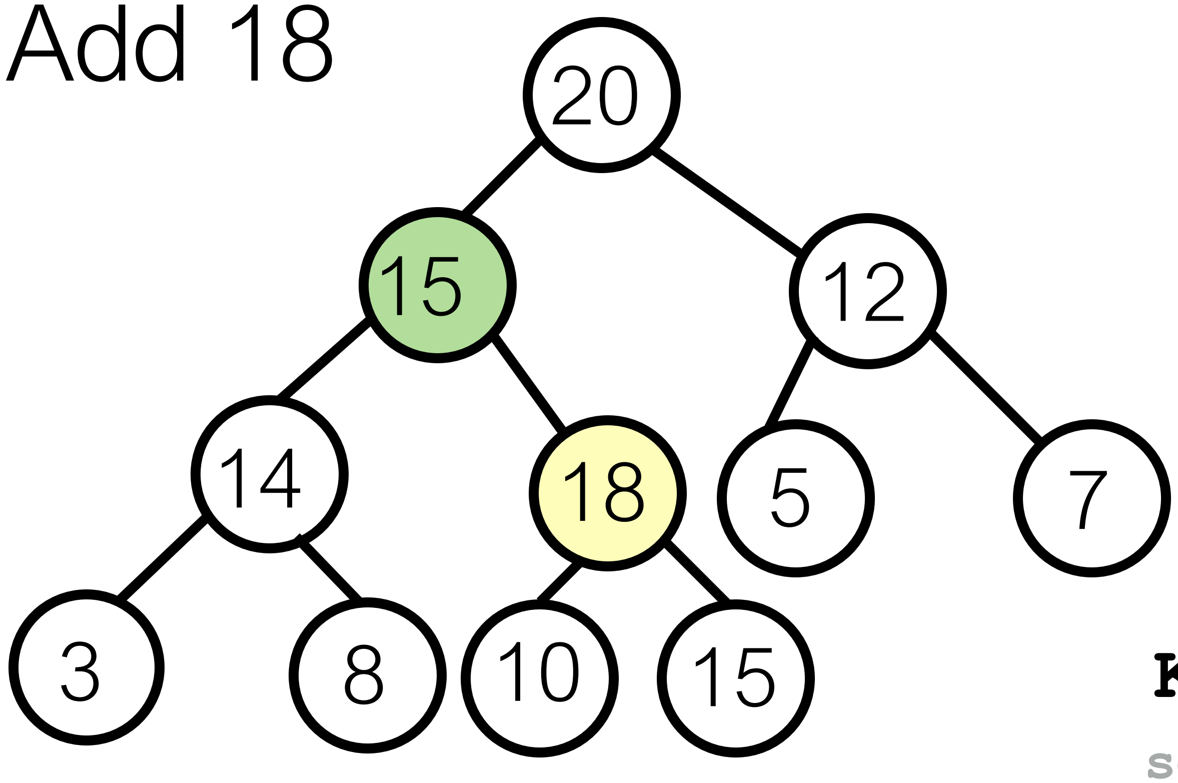
self.array

	20	15	12	14	18	5	7	3	8	10	15
0	1	2	3	4	5	6	7	8	9	10	11

```
def rise(self, k):
    while k > 1 and self.array[k//2][0] < self.array[k][0]:
        self.swap(k, k//2)
        k //= 2
def add(self, key, value):
    item = (key, value)
    if self.count + 1 < len(self.array):
        self.array[self.count+1] = item
    else:
        self._resize()
        self.array[self.count+1] = item
    self.count += 1
    self.rise(self.count)
```

~~k = 11~~
k = 5

Add 18



Key = 18 **self.count = 11**

self.array

	20	15	12	14	18	5	7	3	8	10	15
0	1	2	3	4	5	6	7	8	9	10	11

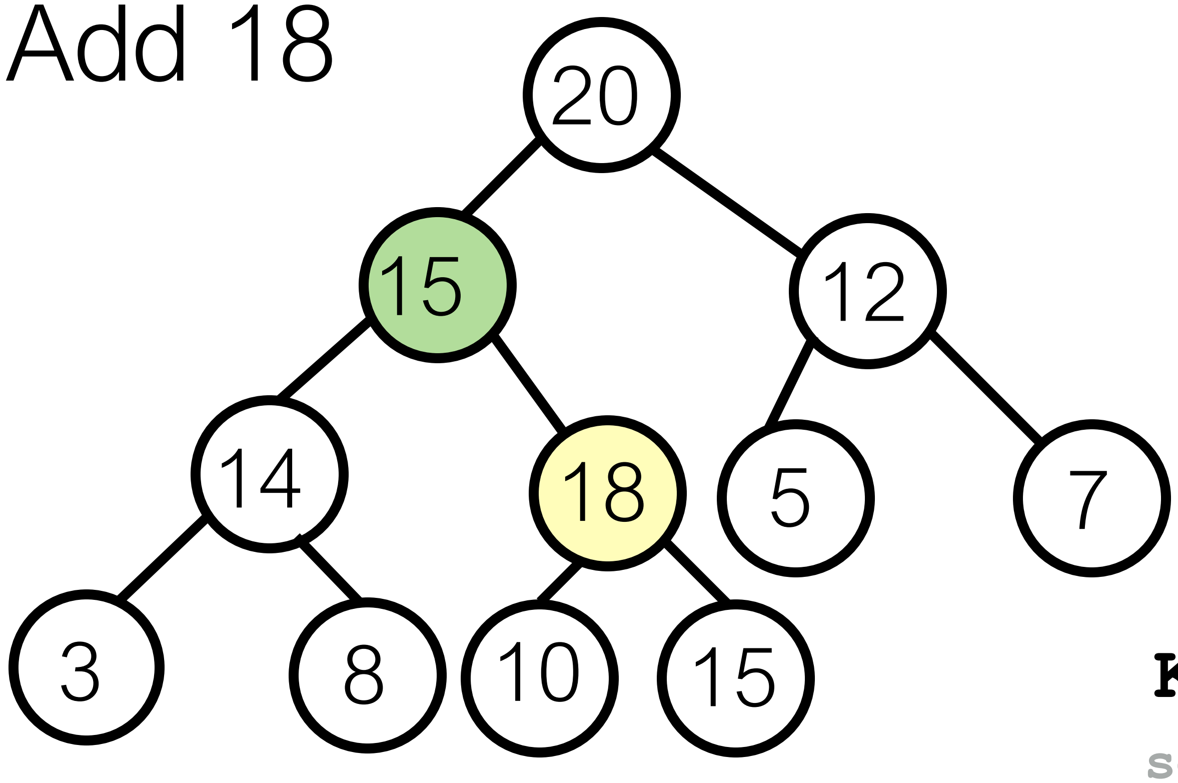
```
def rise(self, k):
    while k > 1 and self.array[k//2][0] < self.array[k][0]:
        self.swap(k, k//2)
        k //= 2
```

```
def add(self, key, value):
    item = (key, value)
    if self.count + 1 < len(self.array):
        self.array[self.count+1] = item
    else:
        self._resize()
        self.array[self.count+1] = item
    self.count += 1
    self.rise(self.count)
```

~~k = 11~~

k = 5

Add 18



Key = 18 **self.count = 11**

self.array

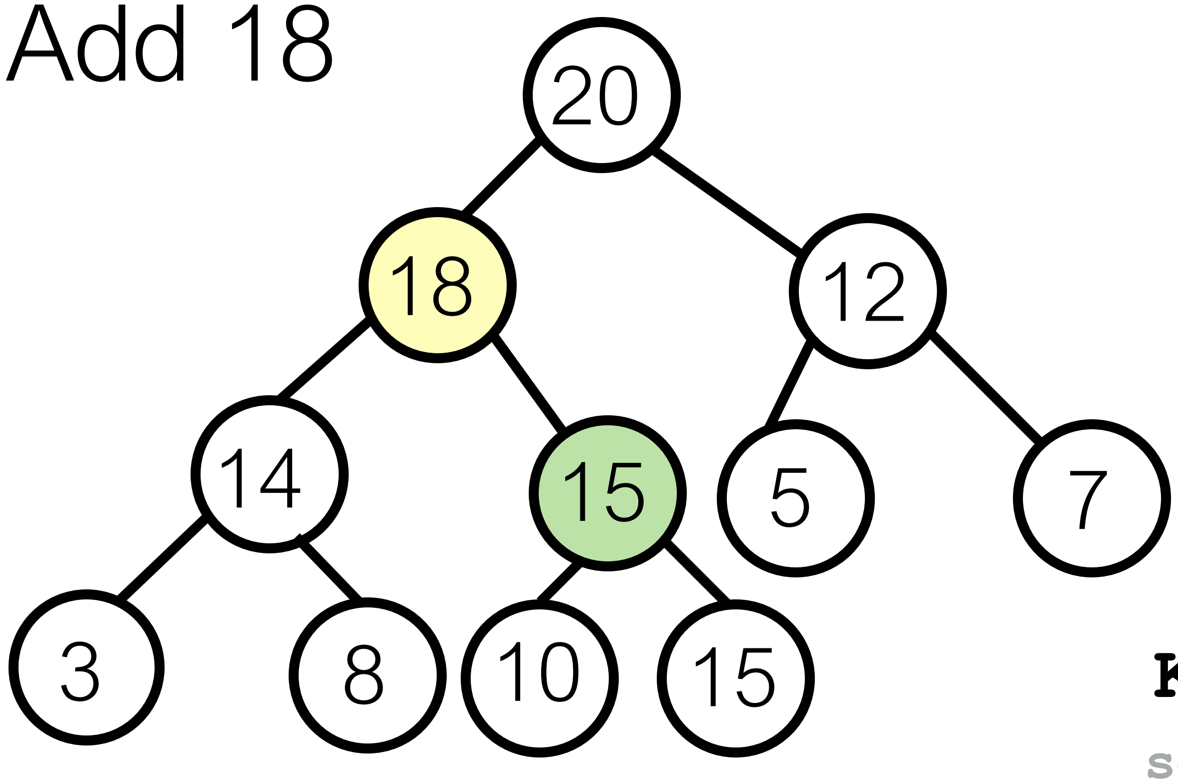
0	1	2	3	4	5	6	7	8	9	10	11
	20	15	12	14	18	5	7	3	8	10	15

```
def rise(self, k):
    while k > 1 and self.array[k//2][0] < self.array[k][0]:
        self.swap(k, k//2)
        k //= 2
```

```
def add(self, key, value):
    item = (key, value)
    if self.count + 1 < len(self.array):
        self.array[self.count+1] = item
    else:
        self._resize()
        self.array[self.count+1] = item
    self.count += 1
    self.rise(self.count)
```

~~k = 11~~
k = 5

Add 18



Key = 18 **self.count = 11**

self.array

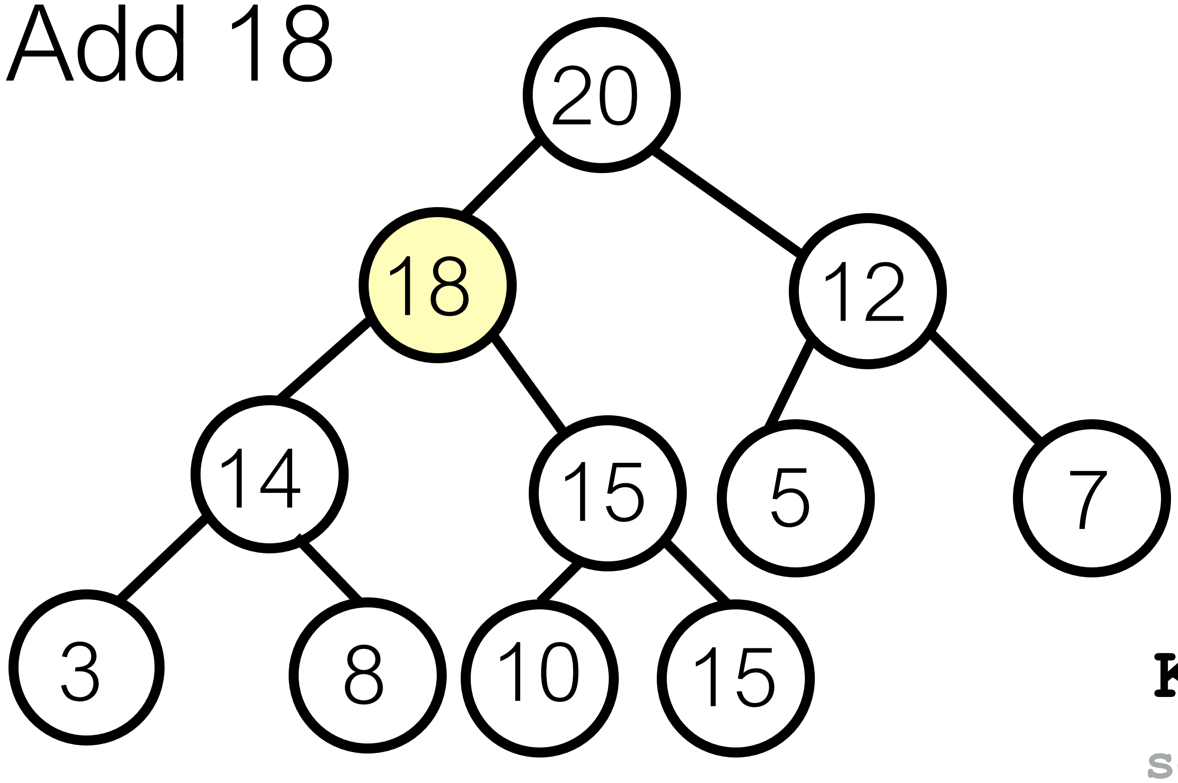
	20	18	12	14	15	5	7	3	8	10	15
0	1	2	3	4	5	6	7	8	9	10	11

```
def rise(self, k):
    while k > 1 and self.array[k//2][0] < self.array[k][0]:
        self.swap(k, k//2)
        k //= 2
```

```
def add(self, key, value):
    item = (key, value)
    if self.count + 1 < len(self.array):
        self.array[self.count+1] = item
    else:
        self._resize()
        self.array[self.count+1] = item
    self.count += 1
    self.rise(self.count)
```

~~k = 11~~
k = 5

Add 18



Key = 18 **self.count = 11**

self.array

	20	18	12	14	15	5	7	3	8	10	15
0	1	2	3	4	5	6	7	8	9	10	11

```
def rise(self, k):
    while k > 1 and self.array[k//2][0] < self.array[k][0]:
        self.swap(k, k//2)
        k //= 2
```

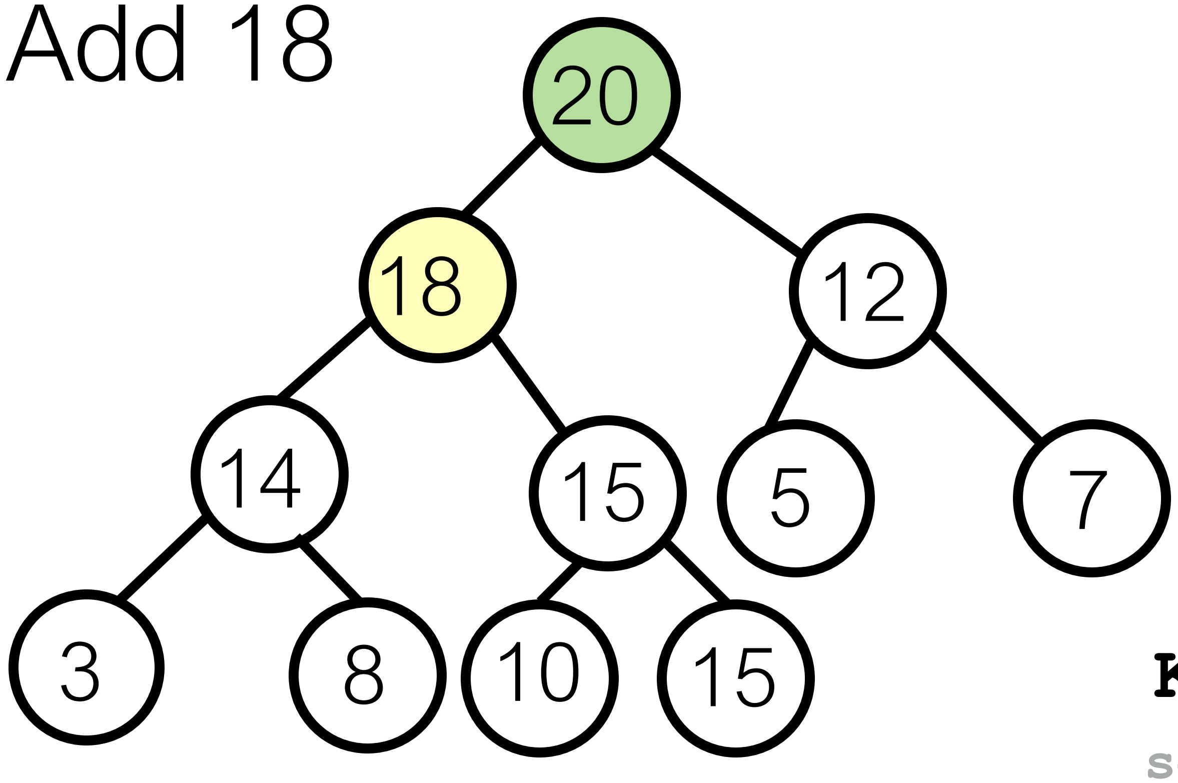
```
def add(self, key, value):
    item = (key, value)
    if self.count + 1 < len(self.array):
        self.array[self.count+1] = item
    else:
        self._resize()
        self.array[self.count+1] = item
    self.count += 1
    self.rise(self.count)
```

~~k = 11~~

~~k = 5~~

k = 2

Add 18



Key = 18 self.count = 11

self.array

0	1	2	3	4	5	6	7	8	9	10	11
	🟩 20	🟨 18	12	14	15	5	7	3	8	10	15

```
def rise(self, k):
    while k > 1 and self.array[k//2][0] < self.array[k][0]:
        self.swap(k, k//2)
        k //= 2
```

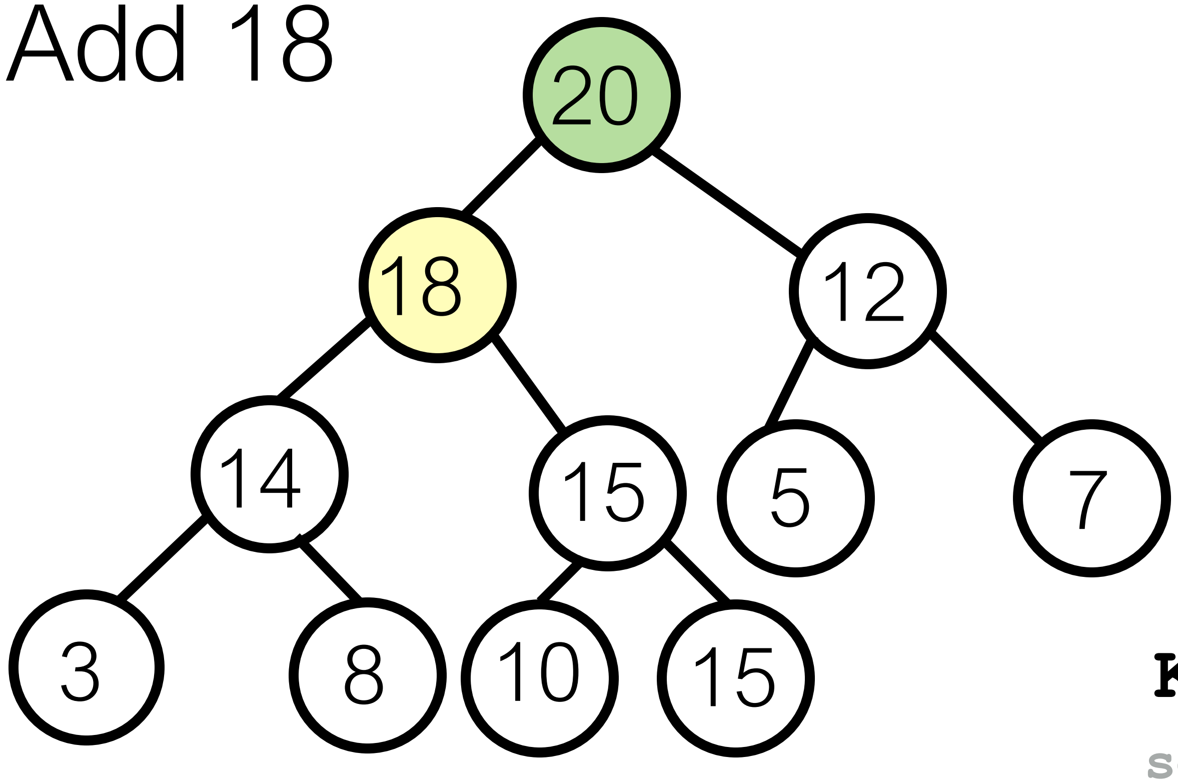
```
def add(self, key, value):
    item = (key, value)
    if self.count + 1 < len(self.array):
        self.array[self.count+1] = item
    else:
        self._resize()
        self.array[self.count+1] = item
    self.count += 1
    self.rise(self.count)
```

~~**k = 11**~~

~~**k = 5**~~

k = 2

Add 18



Key = 18 **self.count = 11**

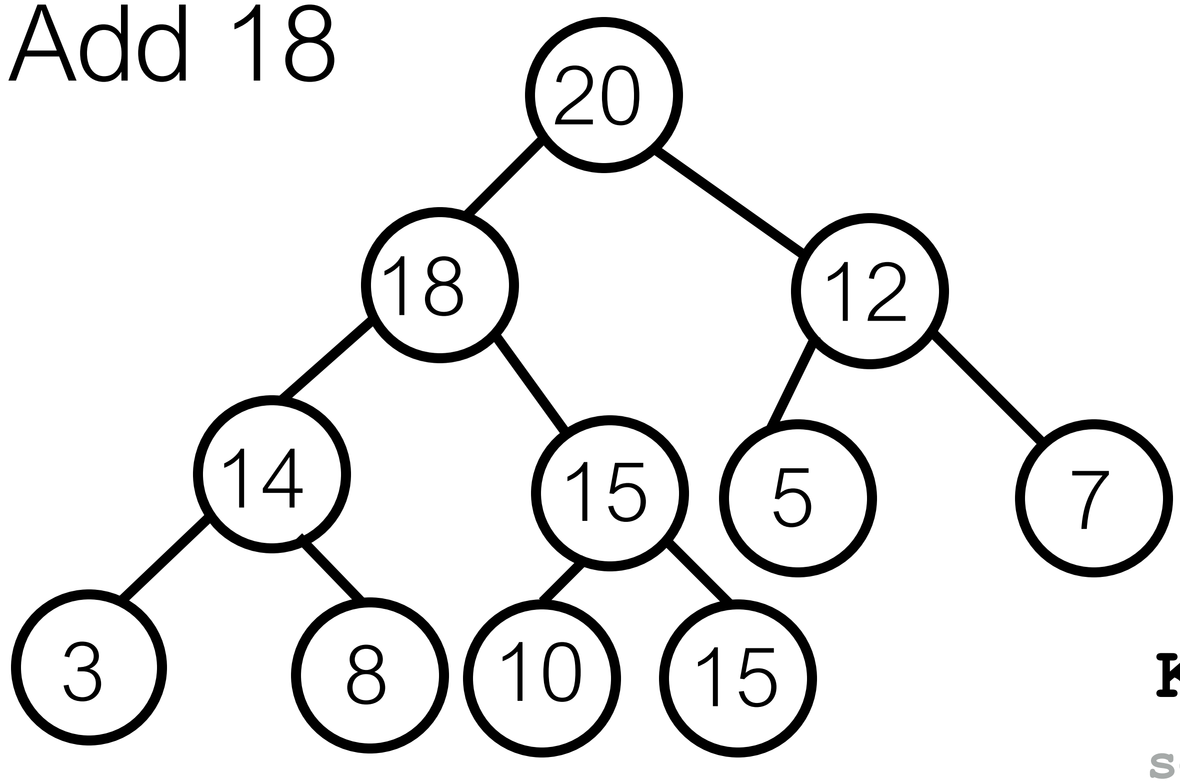
self.array

	20	18	12	14	15	5	7	3	8	10	15
0	1	2	3	4	5	6	7	8	9	10	11

```python
def rise(self, k):
    while k > 1 and self.array[k//2][0] < self.array[k][0]:
        self.swap(k, k//2)
        k //= 2

def add(self, key, value):
    item = (key, value)
    if self.count + 1 < len(self.array):
        self.array[self.count+1] = item
    else:
        self._resize()
        self.array[self.count+1] = item
    self.count += 1
    self.rise(self.count)
```

Add 18



Key = 18   **self.count = 11**

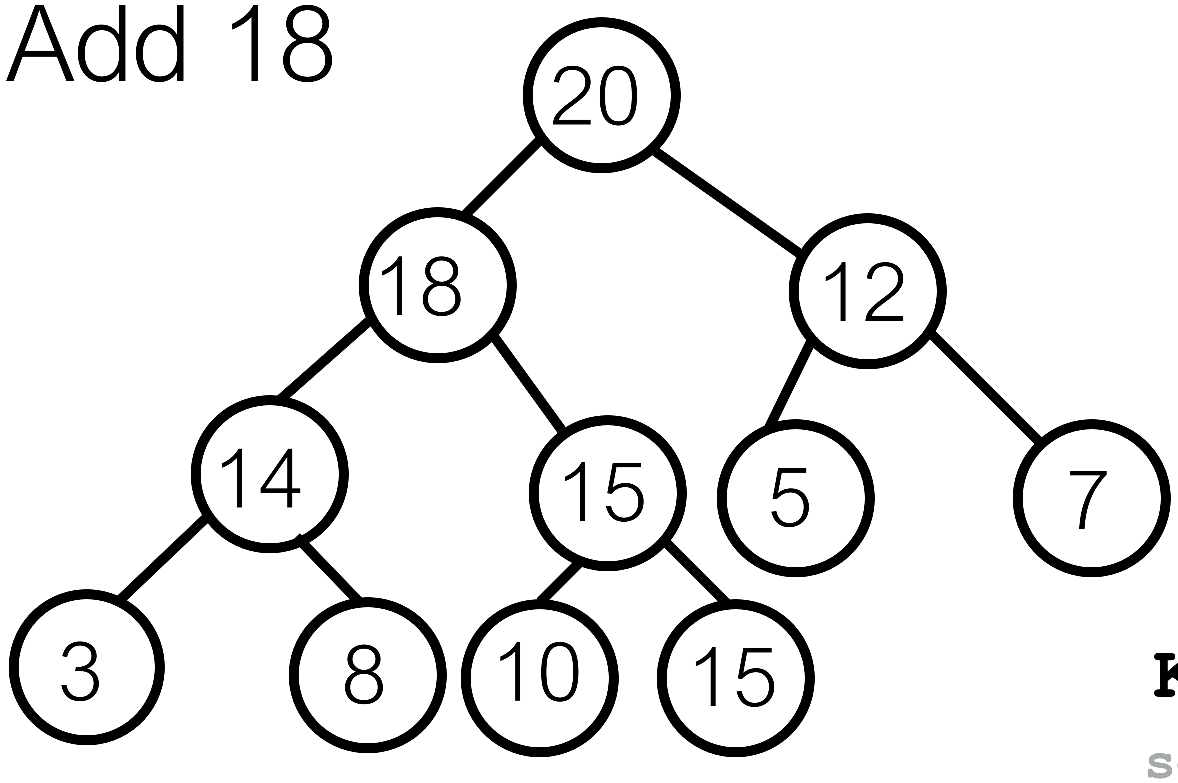
self.array

	20	18	12	14	15	5	7	3	8	10	15
0	1	2	3	4	5	6	7	8	9	10	11

```
def rise(self, k):
    while k > 1 and self.array[k//2][0] < self.array[k][0]:
        self.swap(k, k//2)
        k //= 2

def add(self, key, value):
    item = (key, value)
    if self.count + 1 < len(self.array):
        self.array[self.count+1] = item
    else:
        self._resize()
        self.array[self.count+1] = item
    self.count += 1
    self.rise(self.count)
```

Add 18



Key = 18 **self.count = 11**

self.array

0	1	2	3	4	5	6	7	8	9	10	11
	20	18	12	14	15	5	7	3	8	10	15

```python
def rise(self, k):
    while k > 1 and self.array[k//2][0] < self.array[k][0]:
        self.swap(k, k//2)
        k //= 2

def add(self, key, value):
    item = (key, value)
    if self.count + 1 < len(self.array):
        self.array[self.count+1] = item
    else:
        self._resize()
        self.array[self.count+1] = item
    self.count += 1
    self.rise(self.count)
```

best case: O(1)

worst case: O(log N)

(may need to consider comparison operations)

Complexity of add

- Loop in **rise** can iterate at most depth times ≈ log(N)
  (after depth iterations, the new item is at the root)

- **Best case: O(1)***OCompare when the item is smaller or equal than its parent.

- **Worst case: O(log N)***OCompare when the item rises all the way to the top.

Operations

add:
- put at the bottom
- while order is broken, rise.

get_max:
- swap root with last item
- remove last item
- while order is broken, sink.

Summary

- A simple Heap implementation
  - rise
  - sink
  - largest_child
- Heap Sort