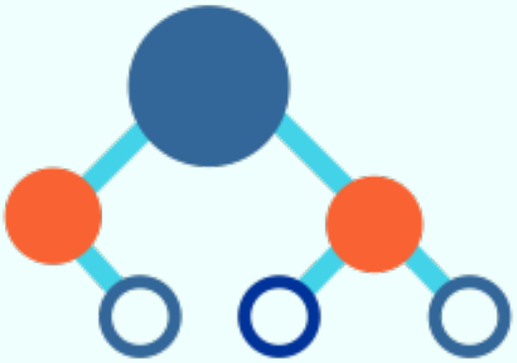
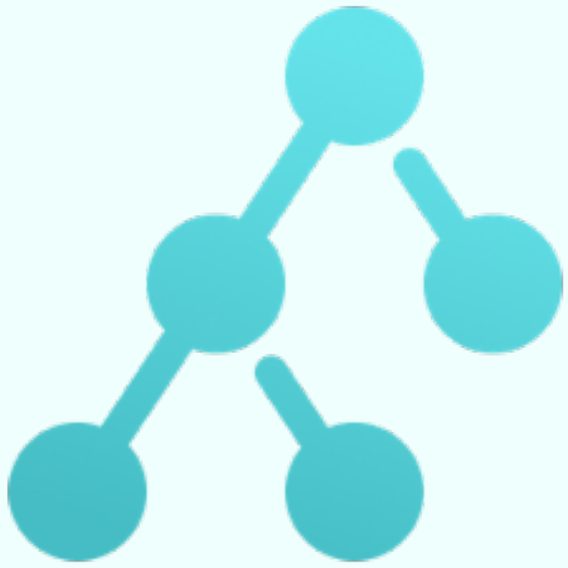
Data Structures

Exercise Session

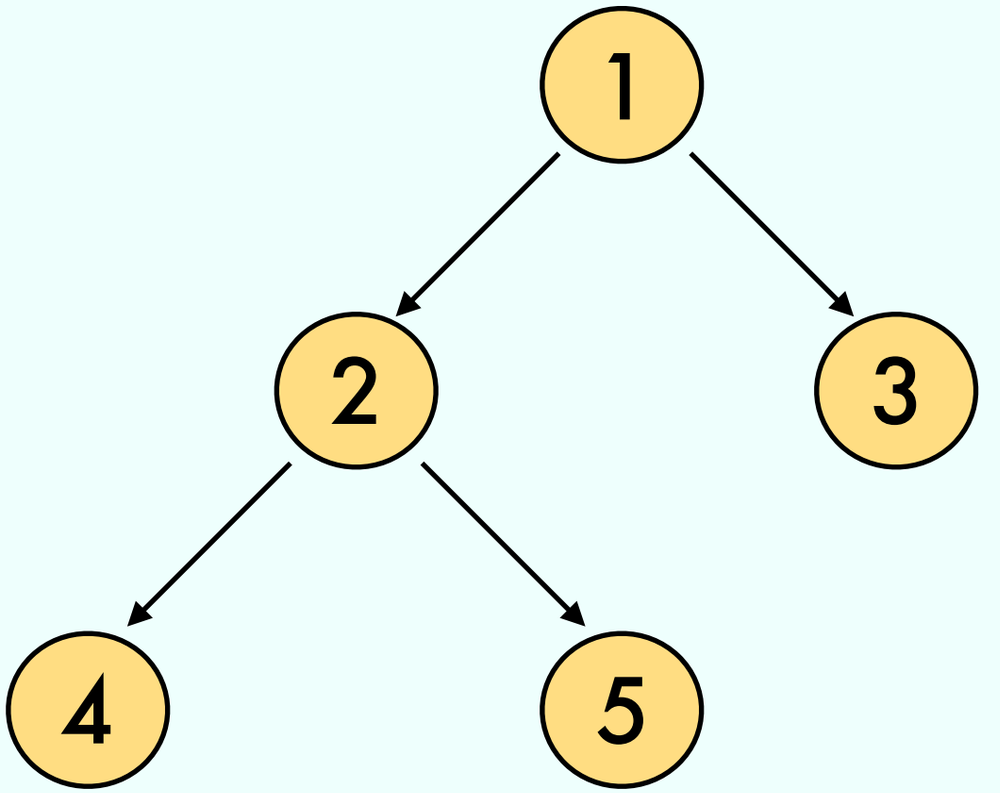


Marco Vassena

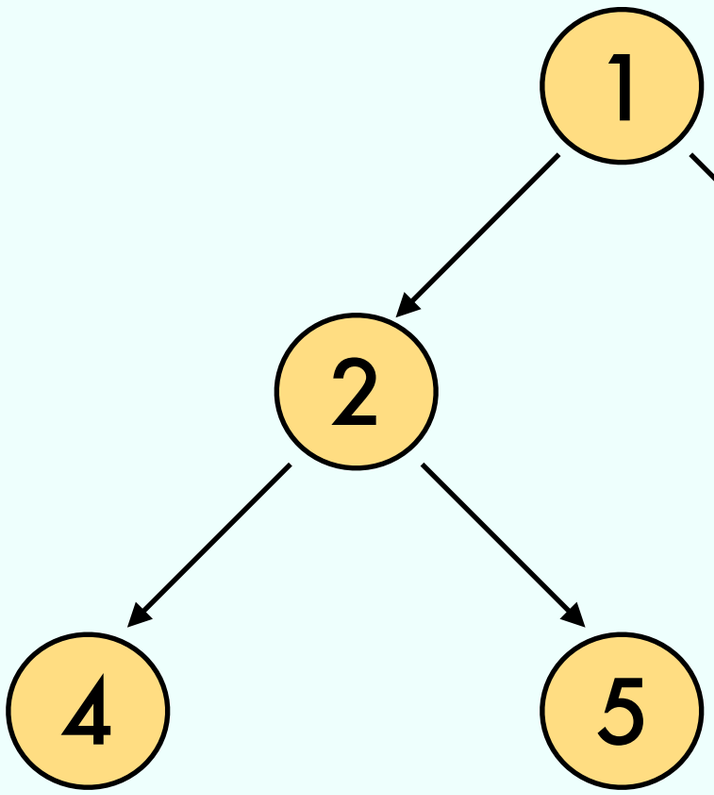


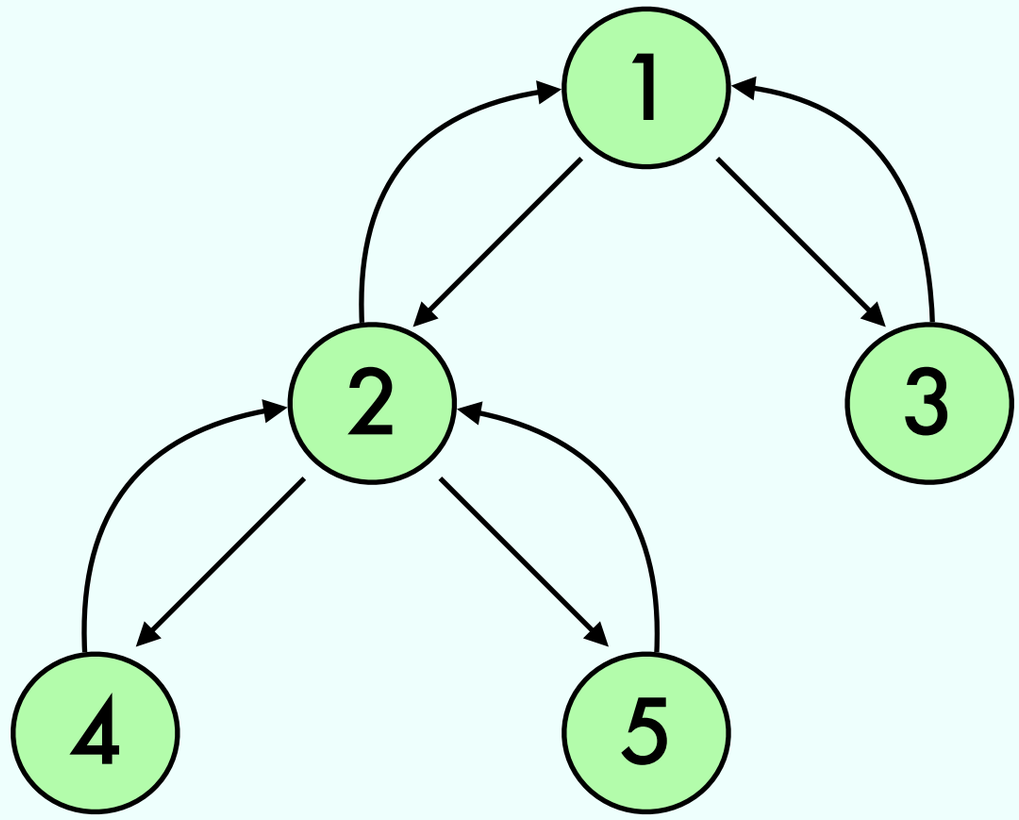
Exercise 2 from12/11

Exercise 2 from12/11



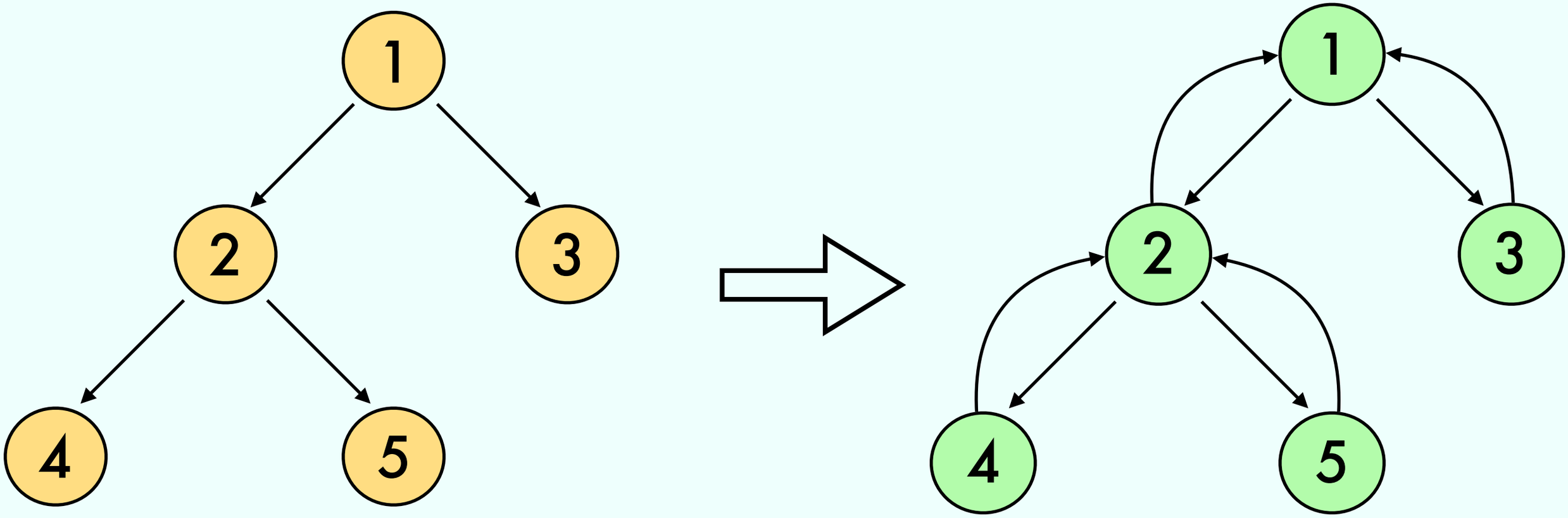
Exercise 2 from12/11





Add to each node a reference to the parent

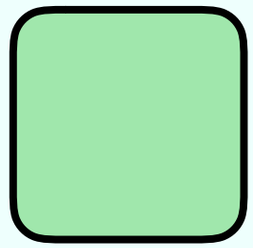
Exercise 2 from12/11



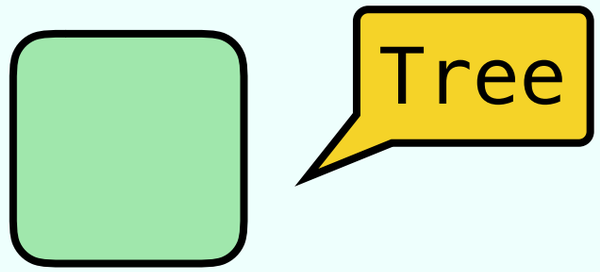
Add to each node a reference to the parent

Tree

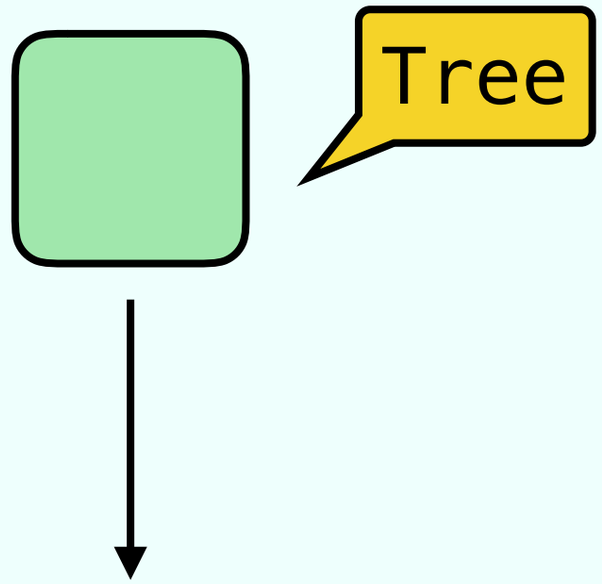
Tree



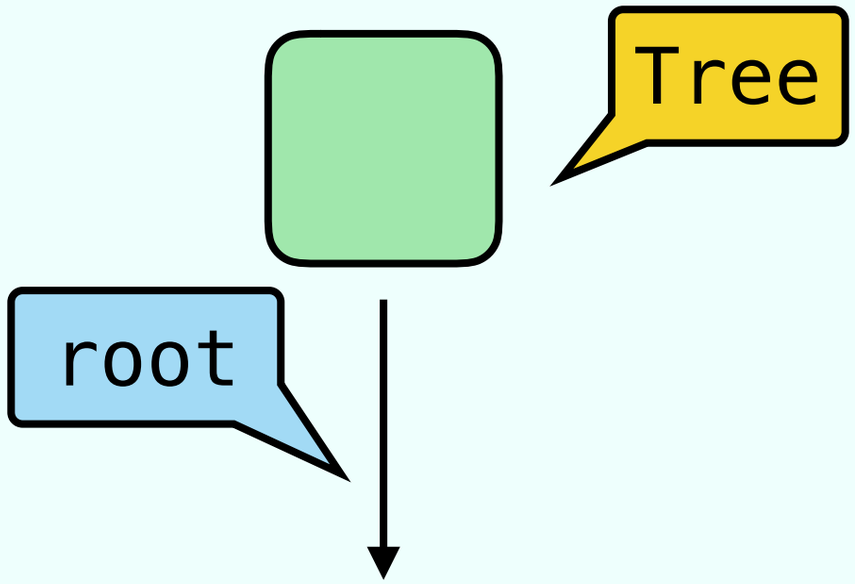
Tree



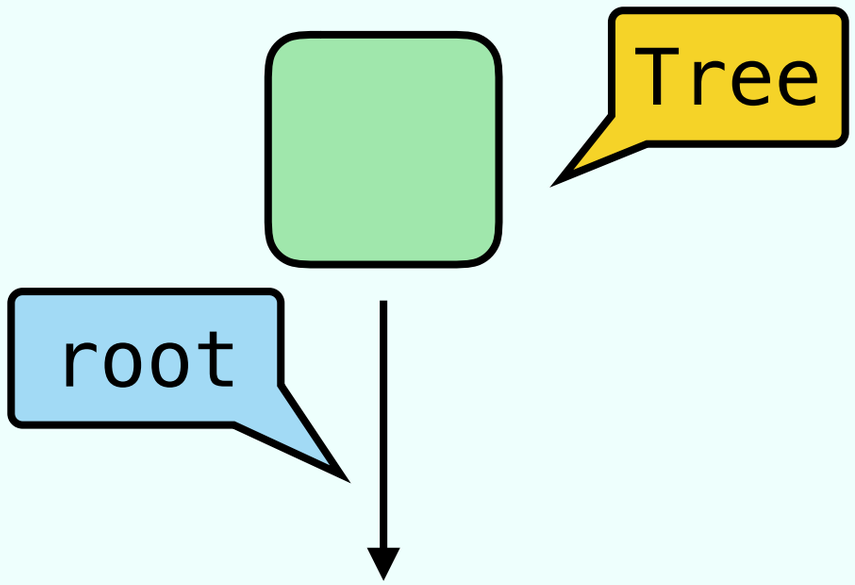
Tree



Tree

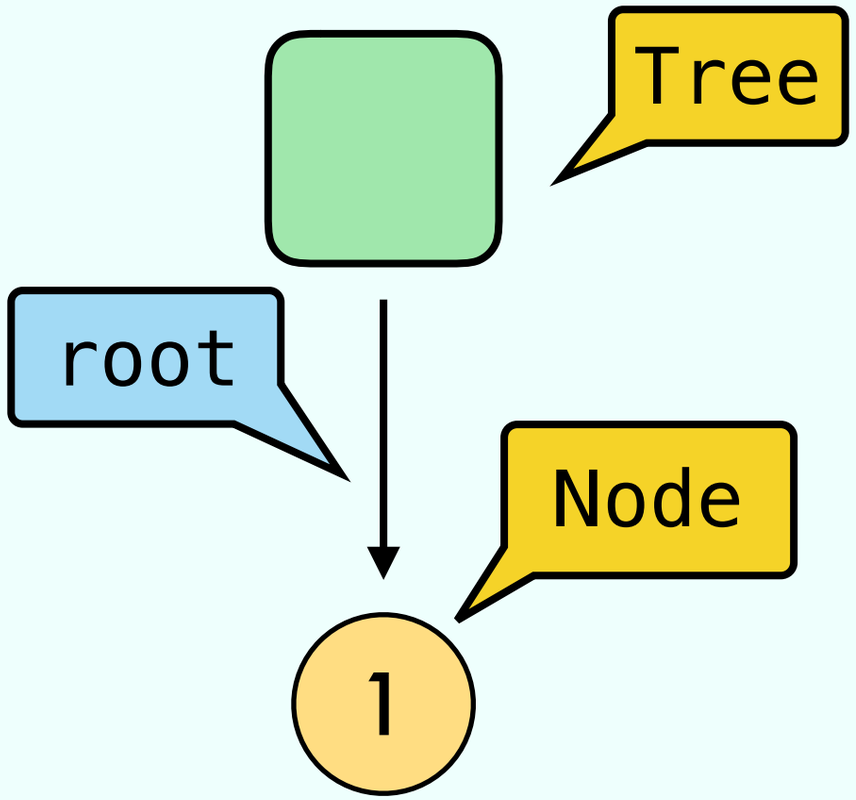


Tree

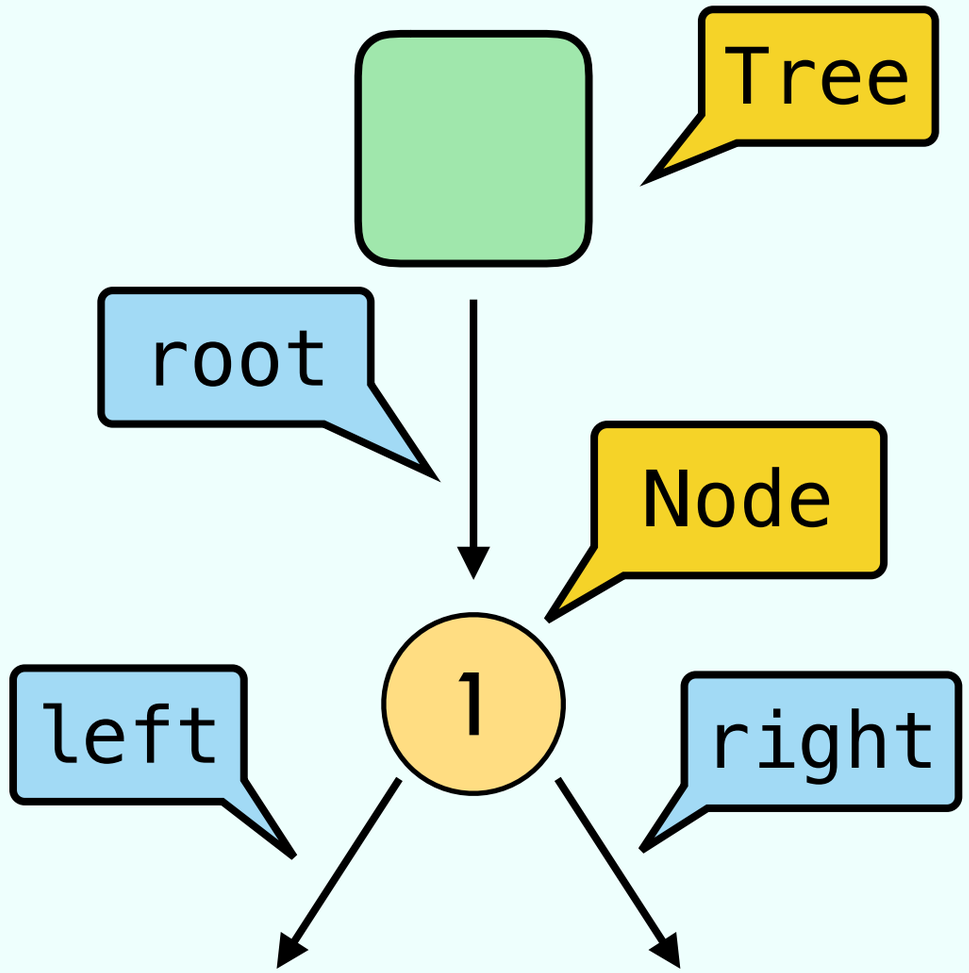


1

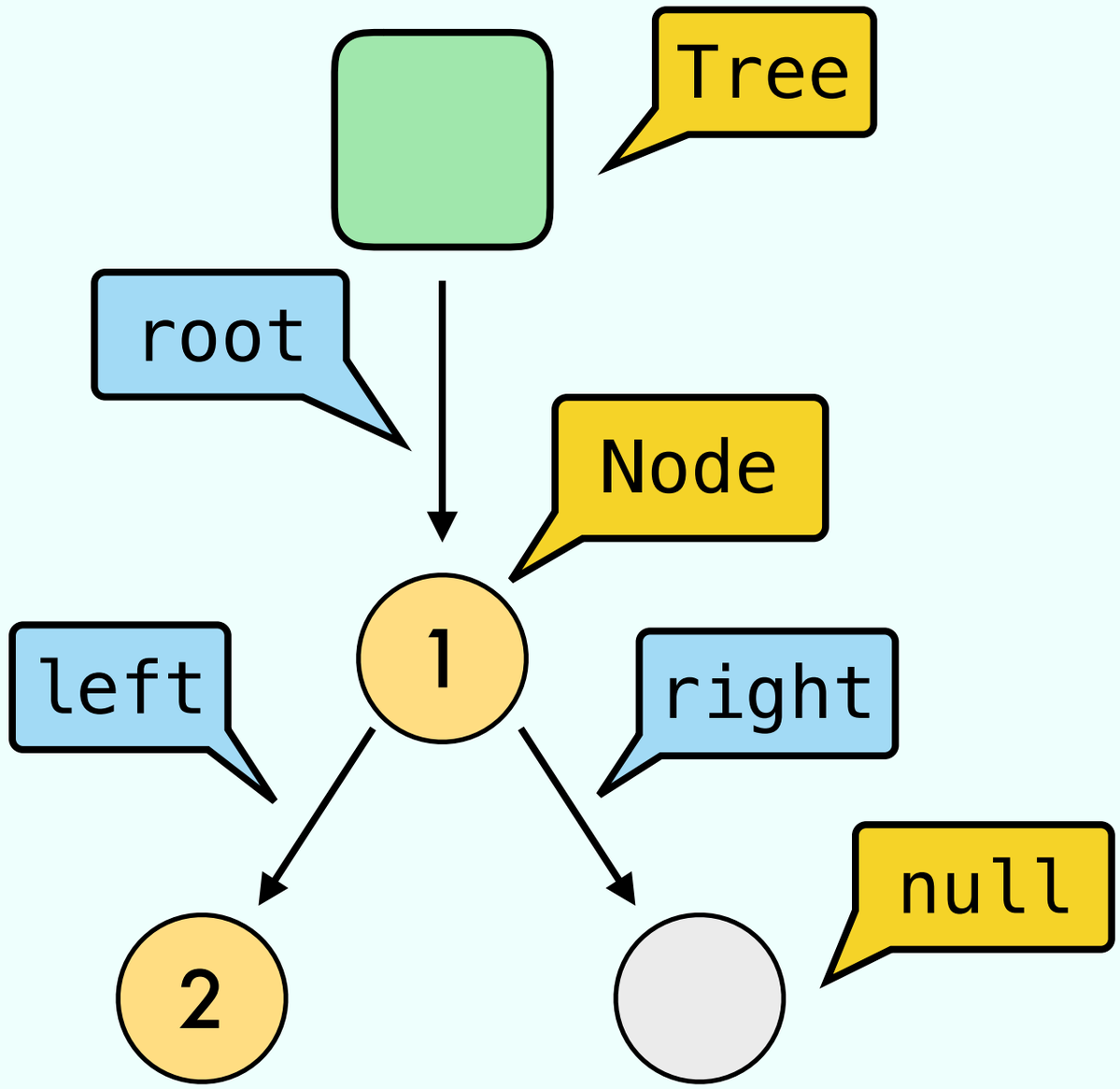
Tree



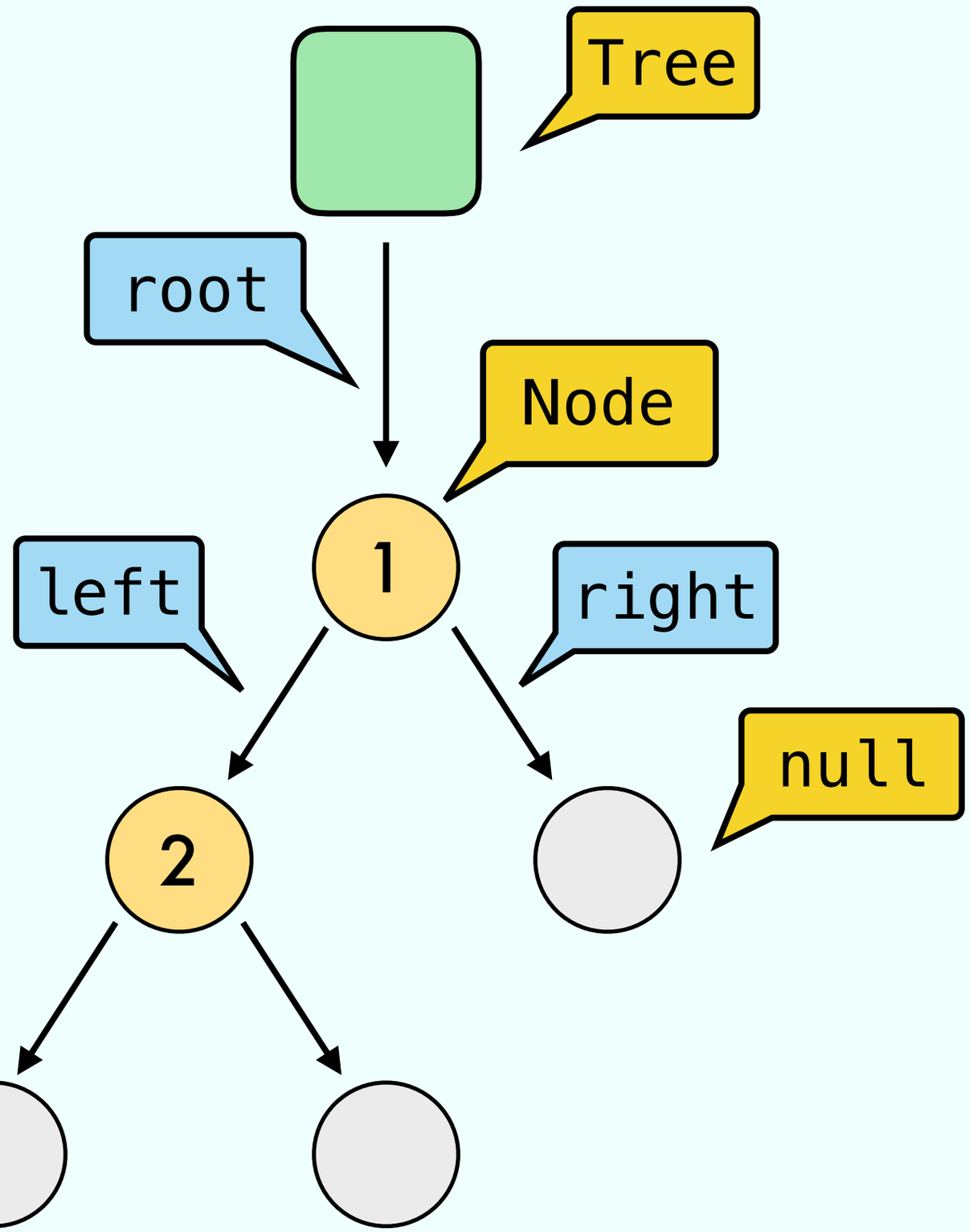
Tree



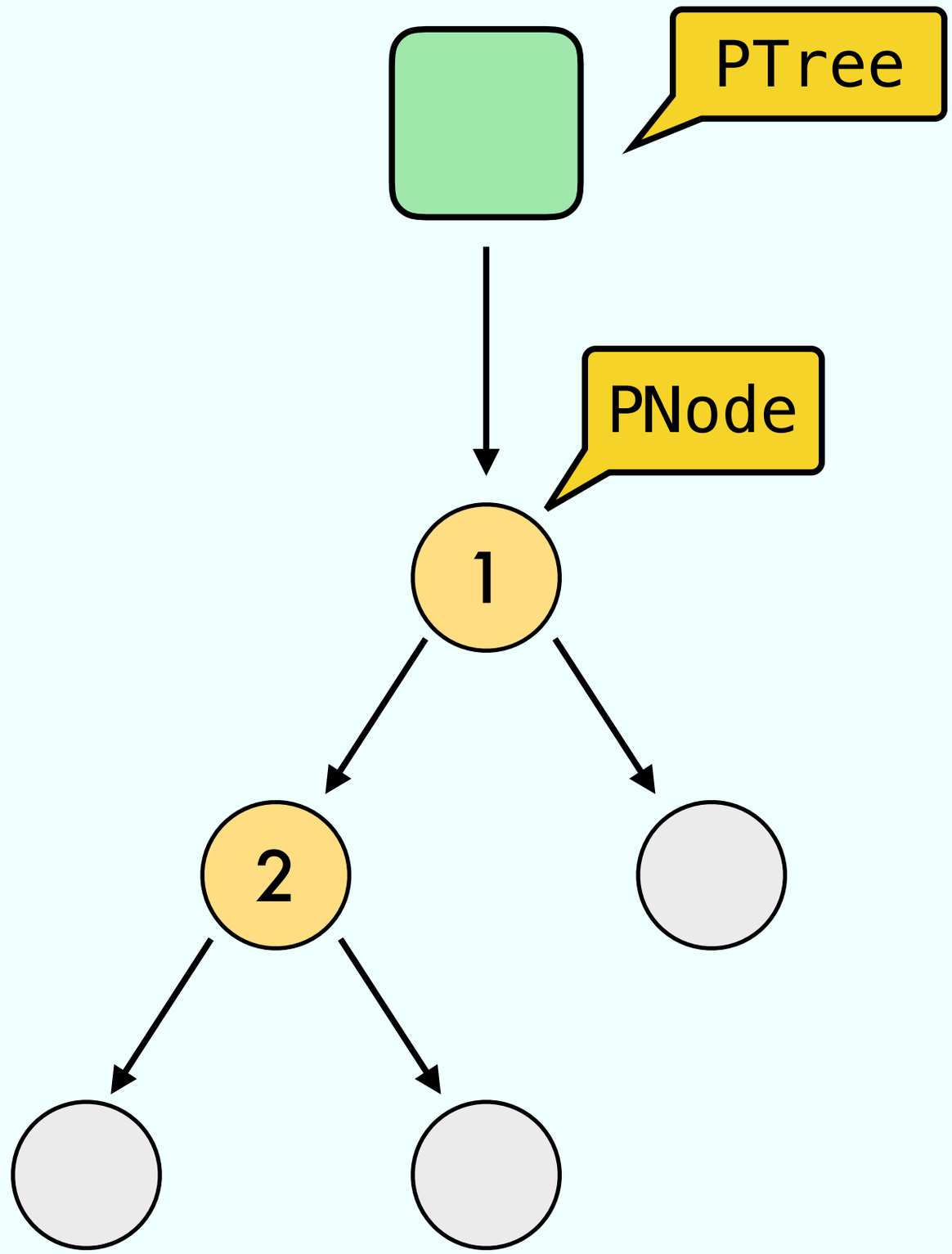
Tree



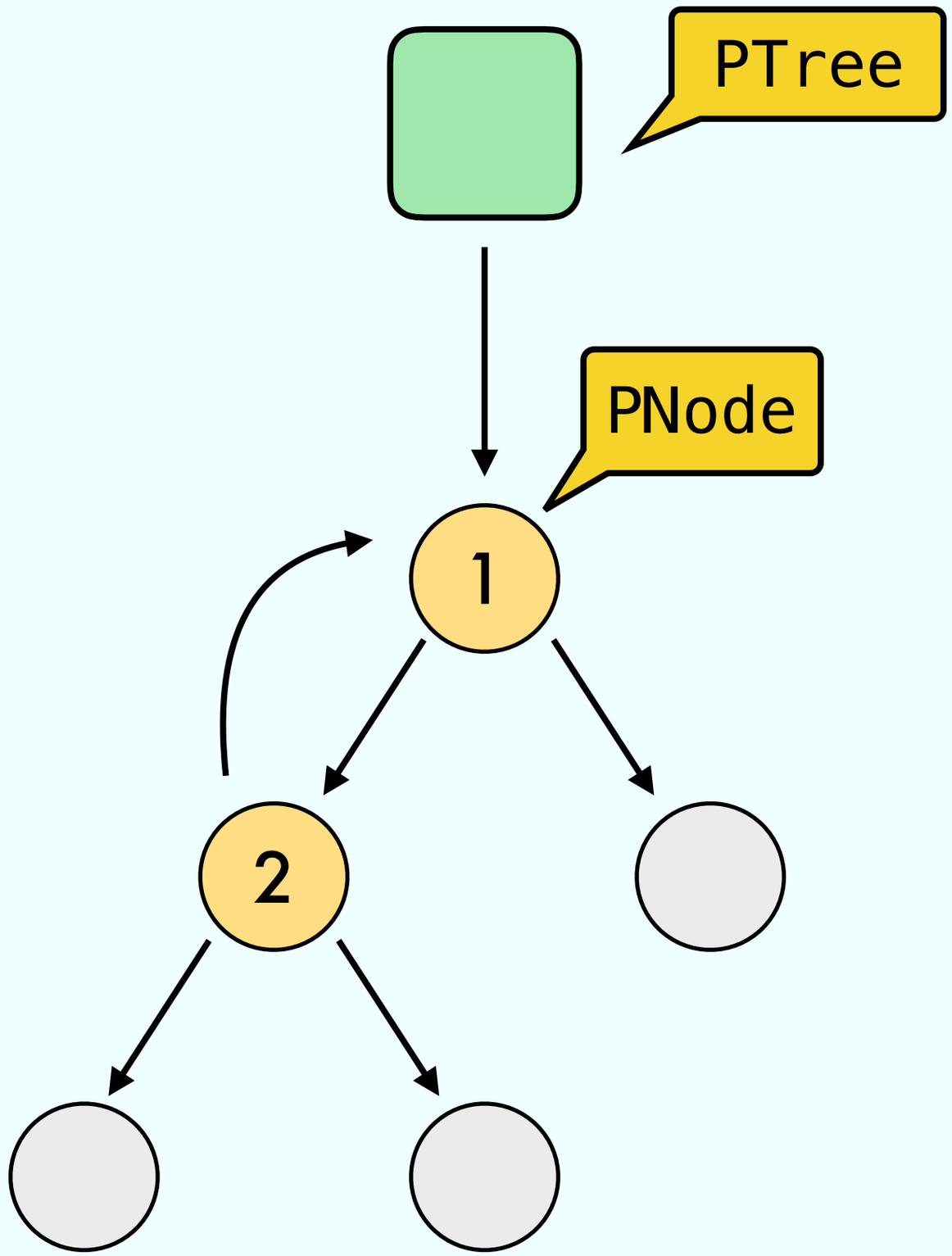
Tree



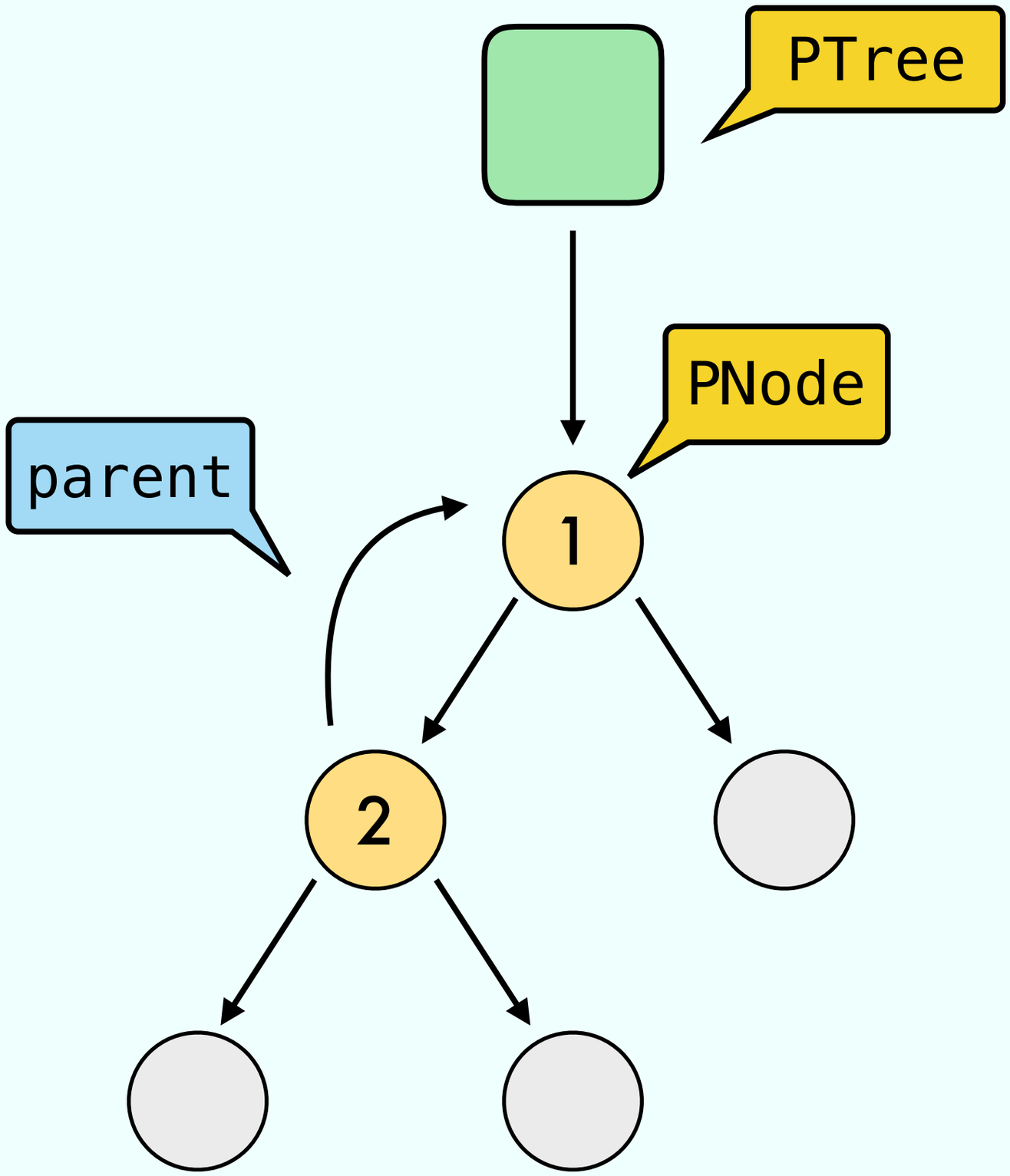
Parent Tree



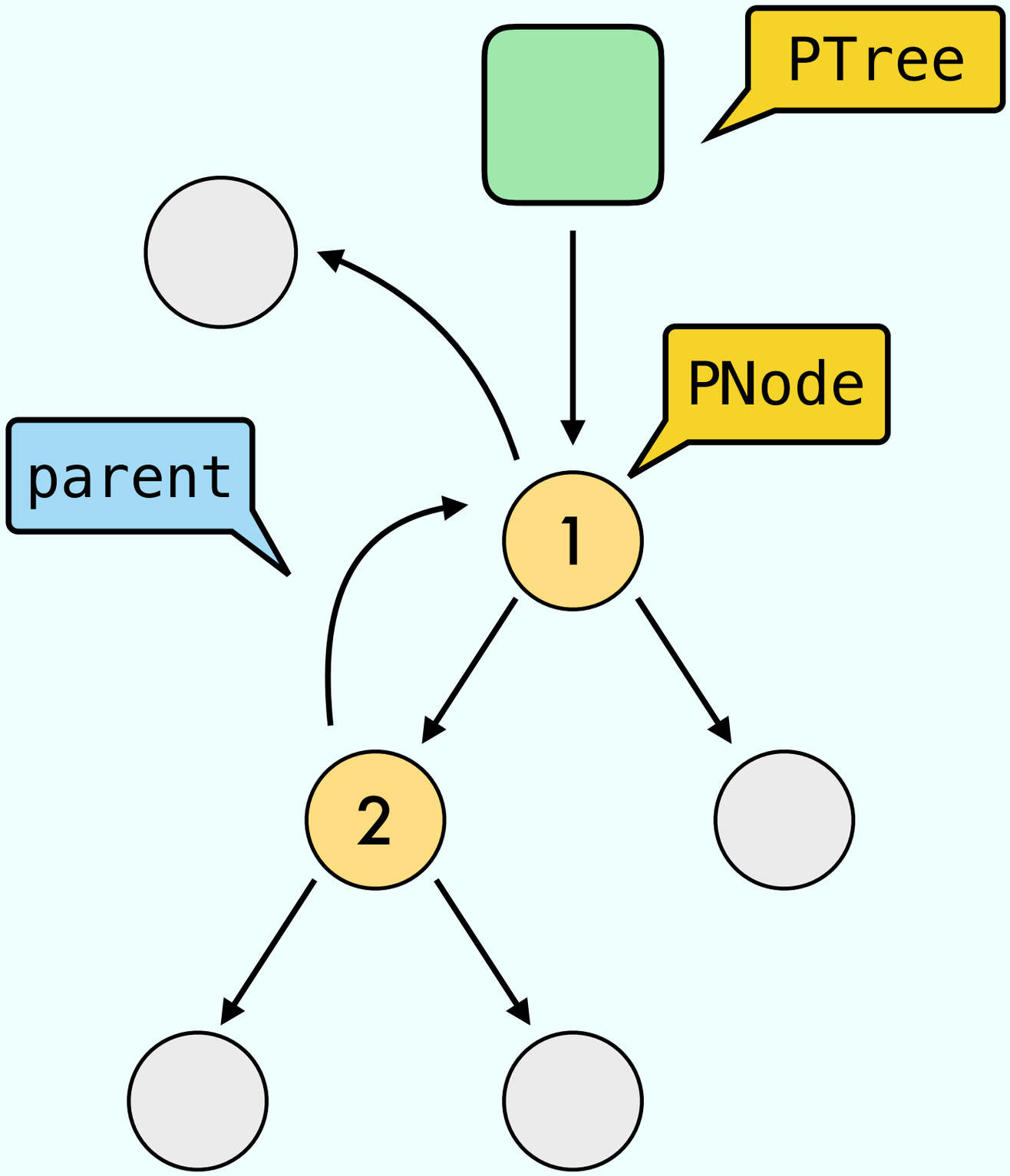
Parent Tree



Parent Tree



Parent Tree



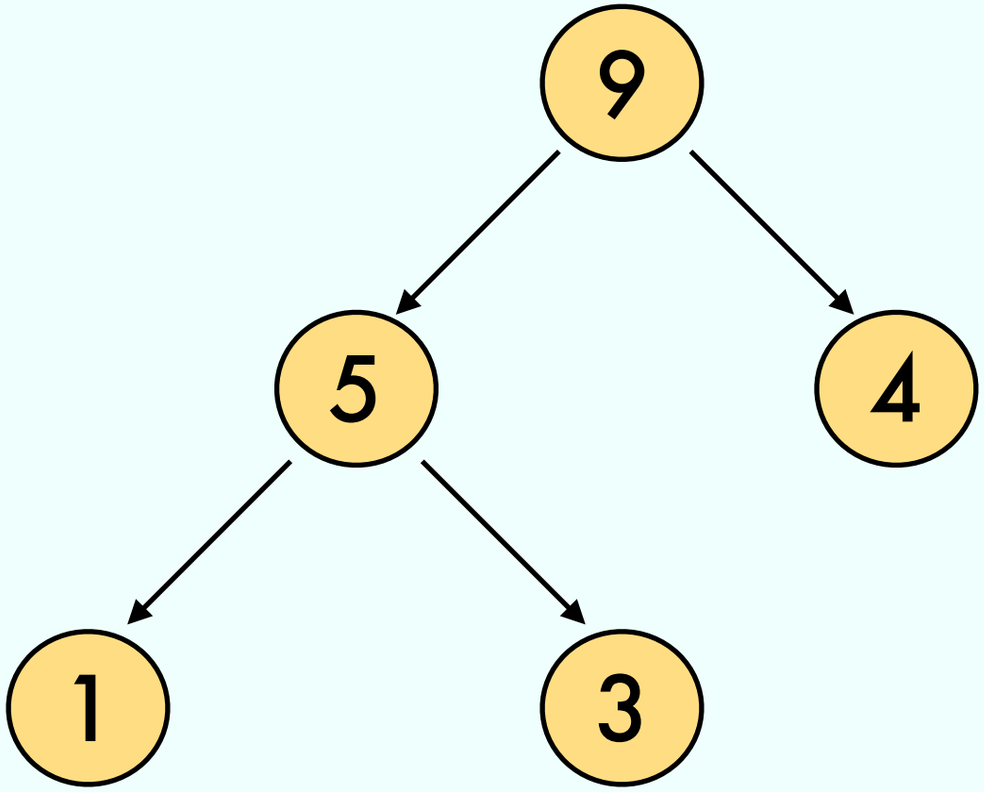
Exercise 1 from 12/08

Analyze the time complexity

```
// A : Dynamic Array
// PQ : Priority Queue, |PQ| = N^3
 for(int i = 0; i < N; i++)
    A.add(0 , PQ.deleteMin())
```

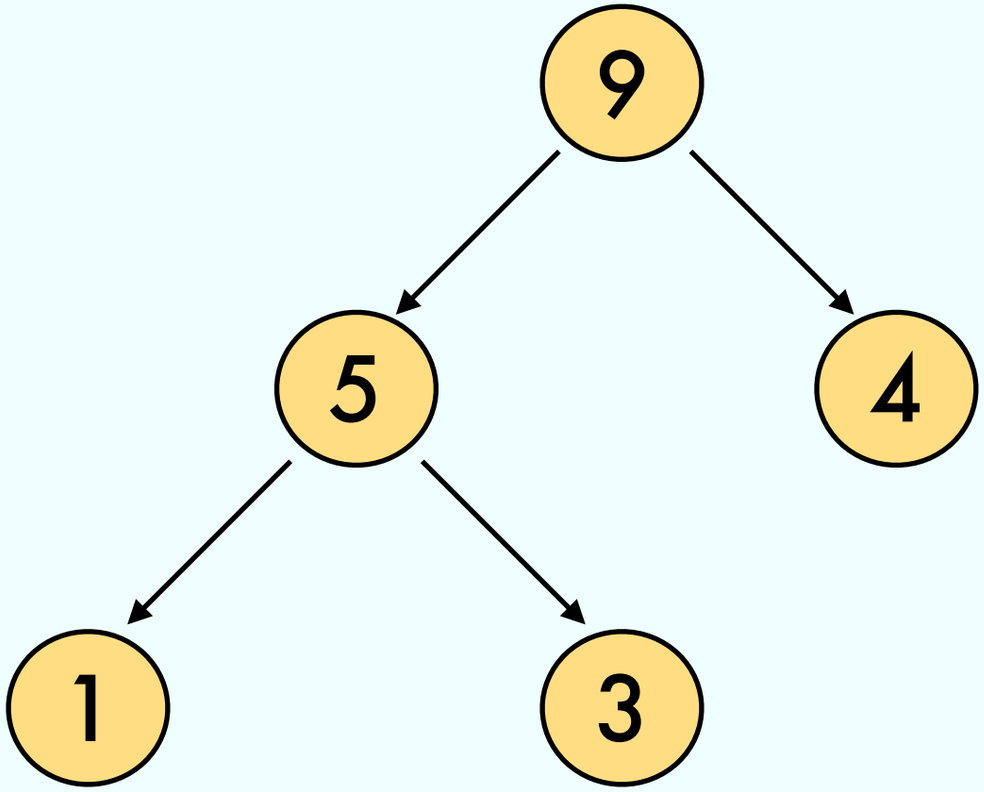
in terms of N

Heap Recap



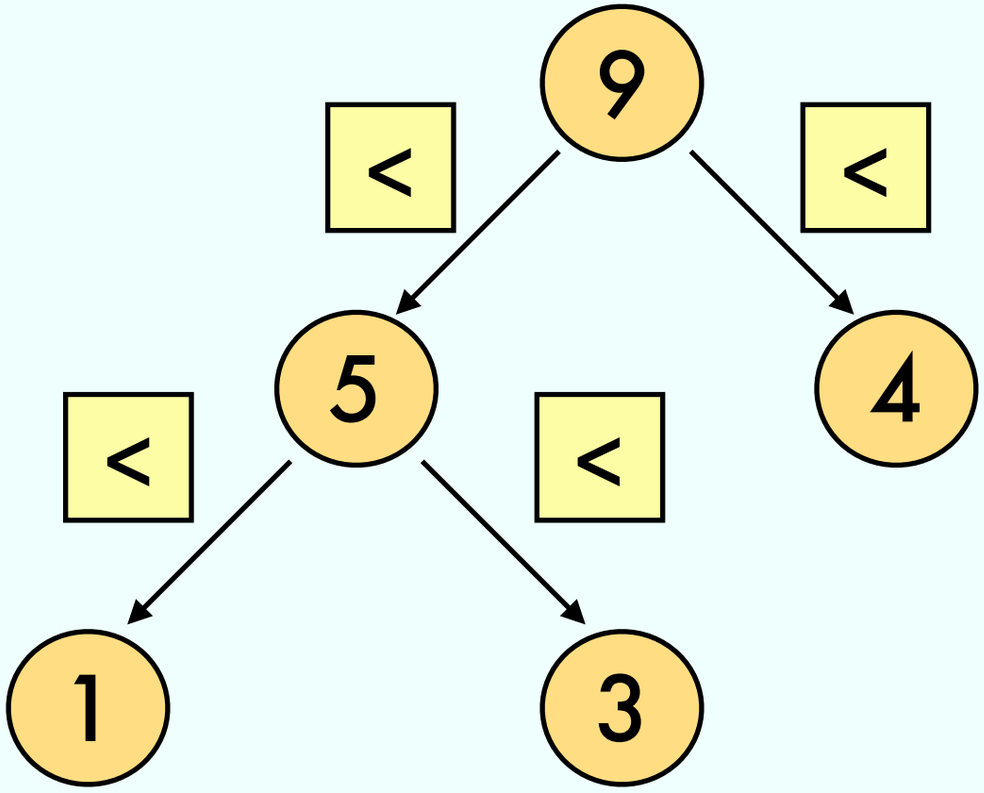
Heap Recap

Order Property



Heap Recap

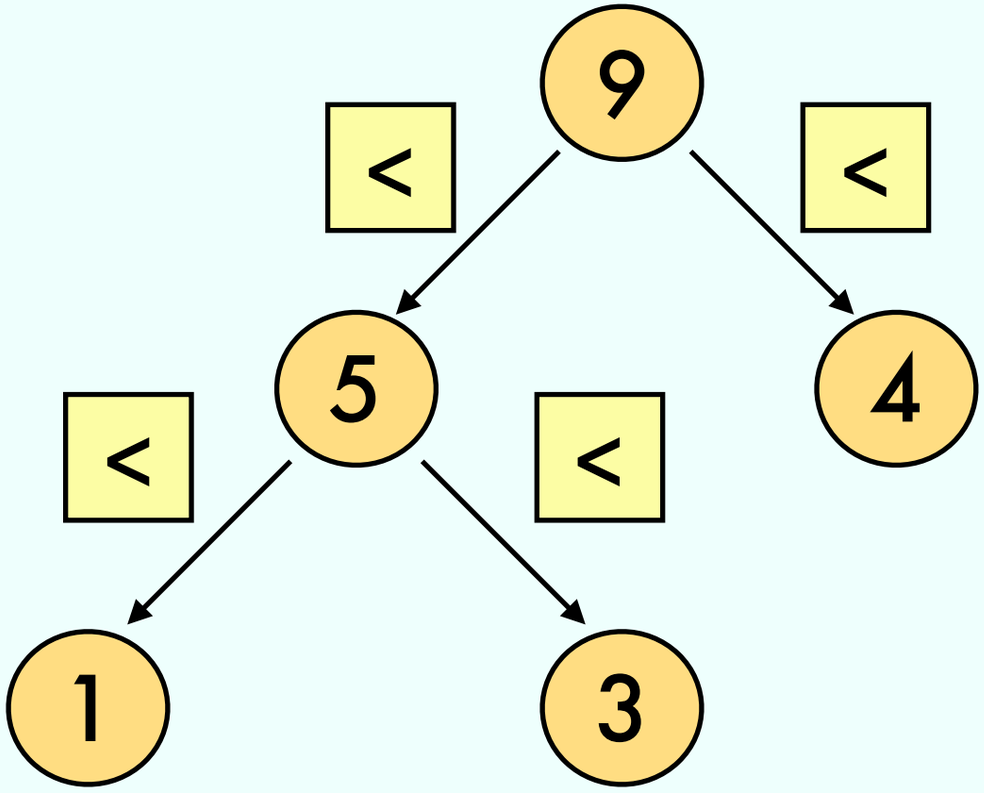
Order Property



Heap Recap

Order Property

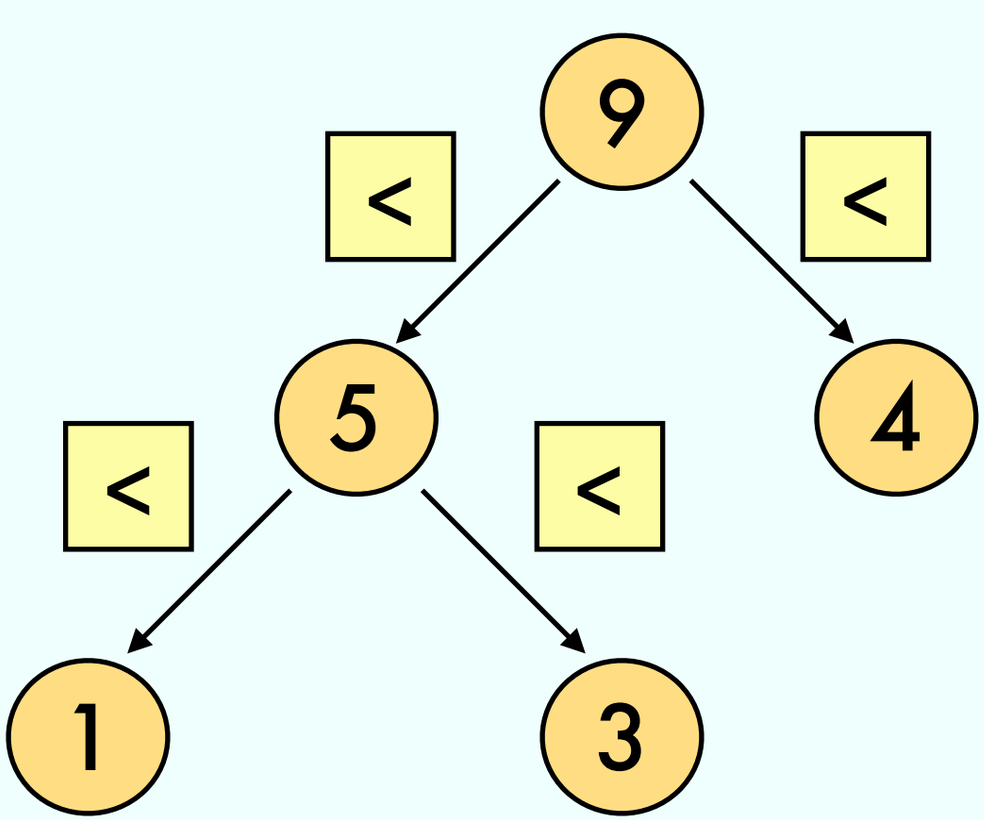
Structure Property



Heap Recap

Order Property

Structure Property



Complete Binary Tree

Except Last Level

Exercise 6.2, 6.3

Draw the heap after each operation

Exercise 6.2, 6.3

Draw the heap after each operation

1.
```
for i in [10,12,1,14,6,5,…]
  PQ.insert(i)
```

Exercise 6.2, 6.3

Draw the heap after each operation

1.
```
for i in [10,12,1,14,6,5,…]
  PQ.insert(i)
```

2.
```
PQ = buildHeap([10,12,1,14,6,…])
```

Exercise 6.2, 6.3

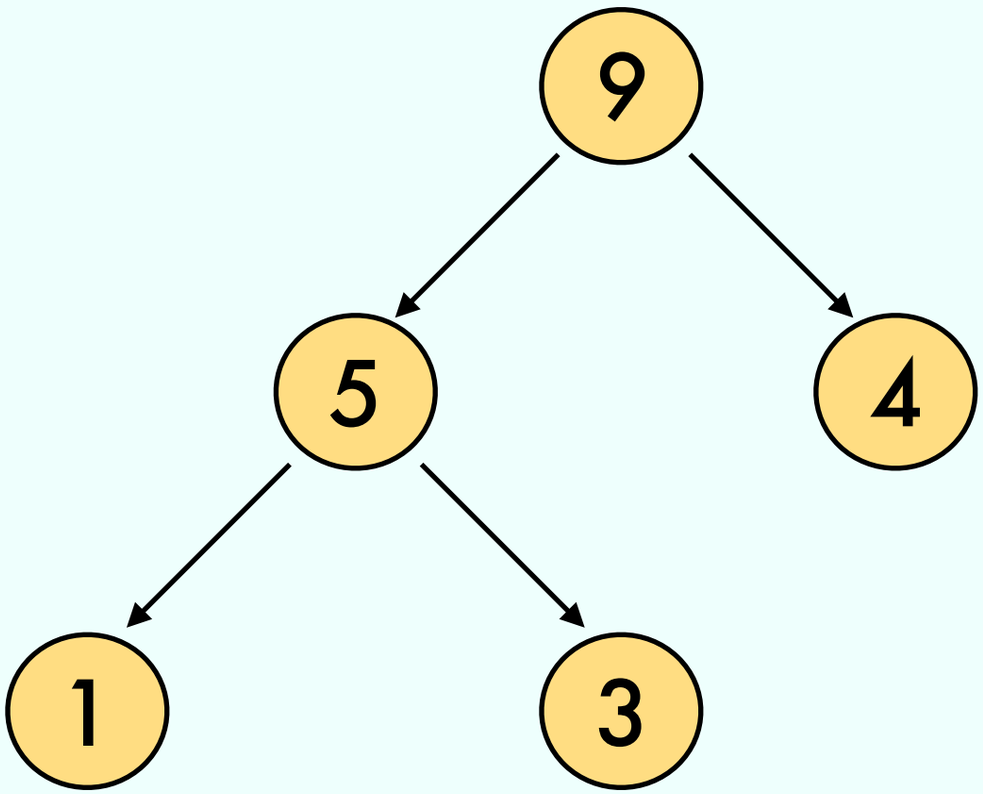
Draw the heap after each operation

1.
```
for i in [10,12,1,14,6,5,…]
  PQ.insert(i)
```

2.
```
PQ = buildHeap([10,12,1,14,6,…])
```

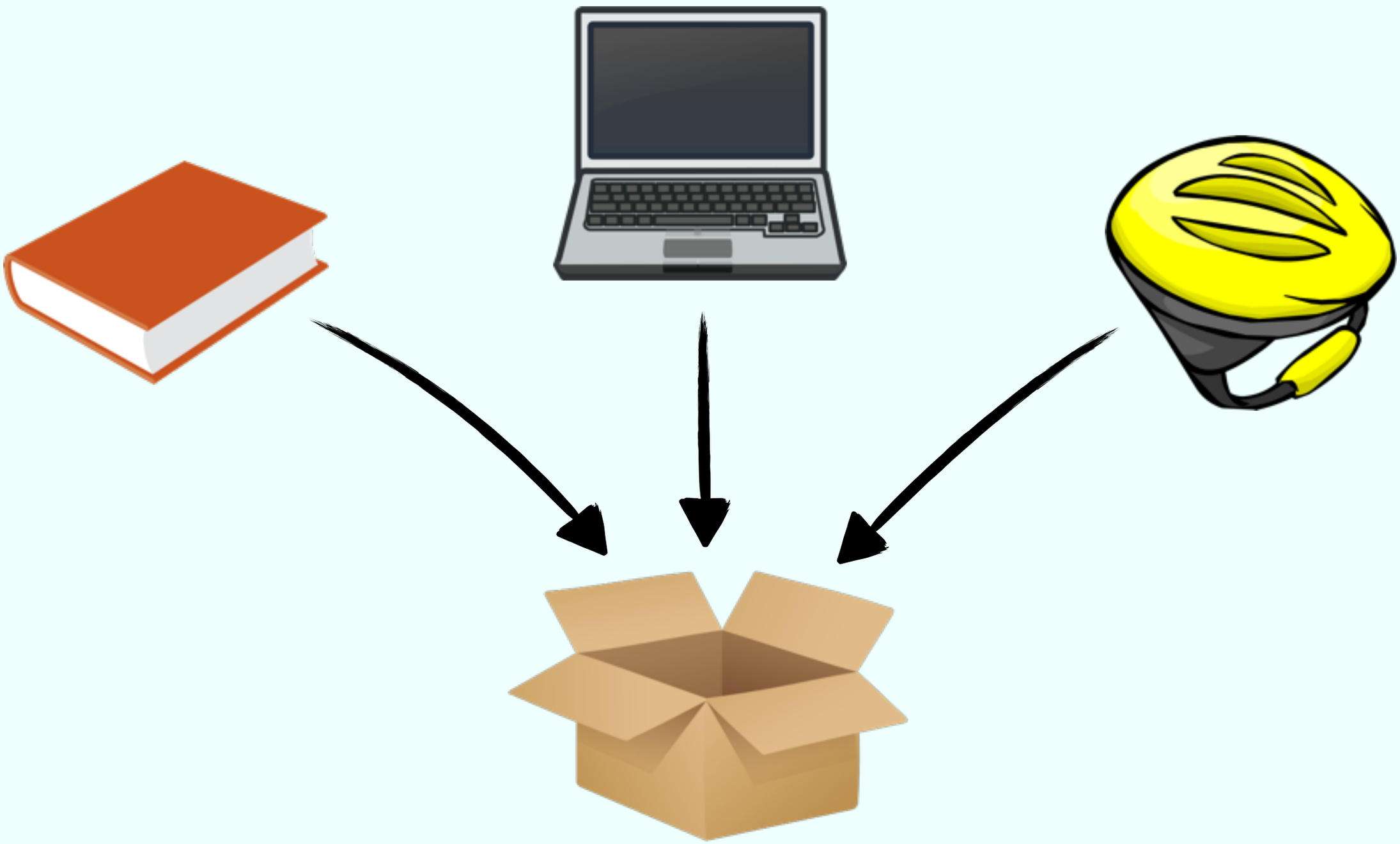
3.
```
PQ.deleteMin()
```

Heap Recap

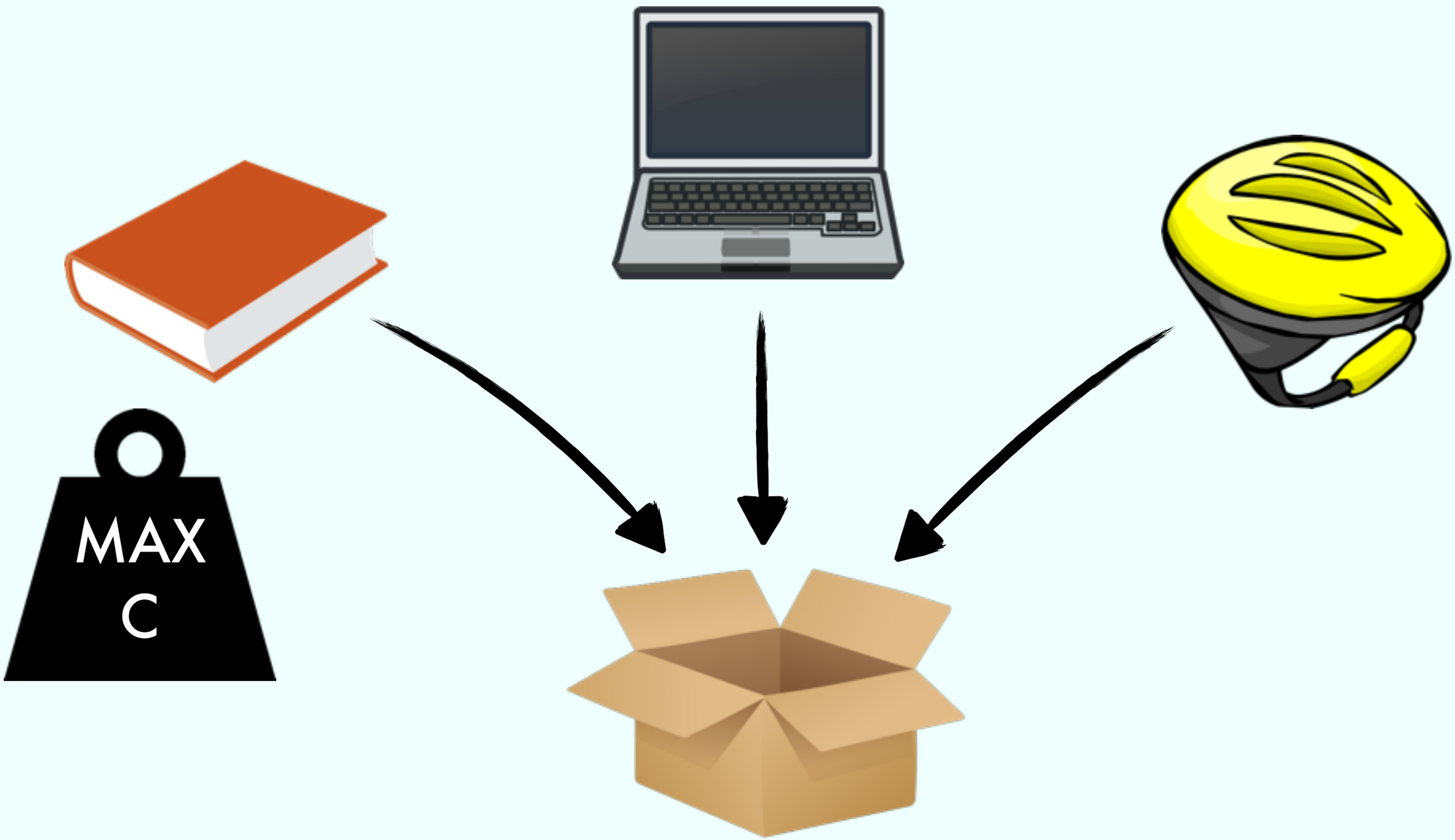


Operation	Time Complexity
`findMin()`	O(1)
`deleteMin()`	O(log N)
`insert(x)`	O(log N)

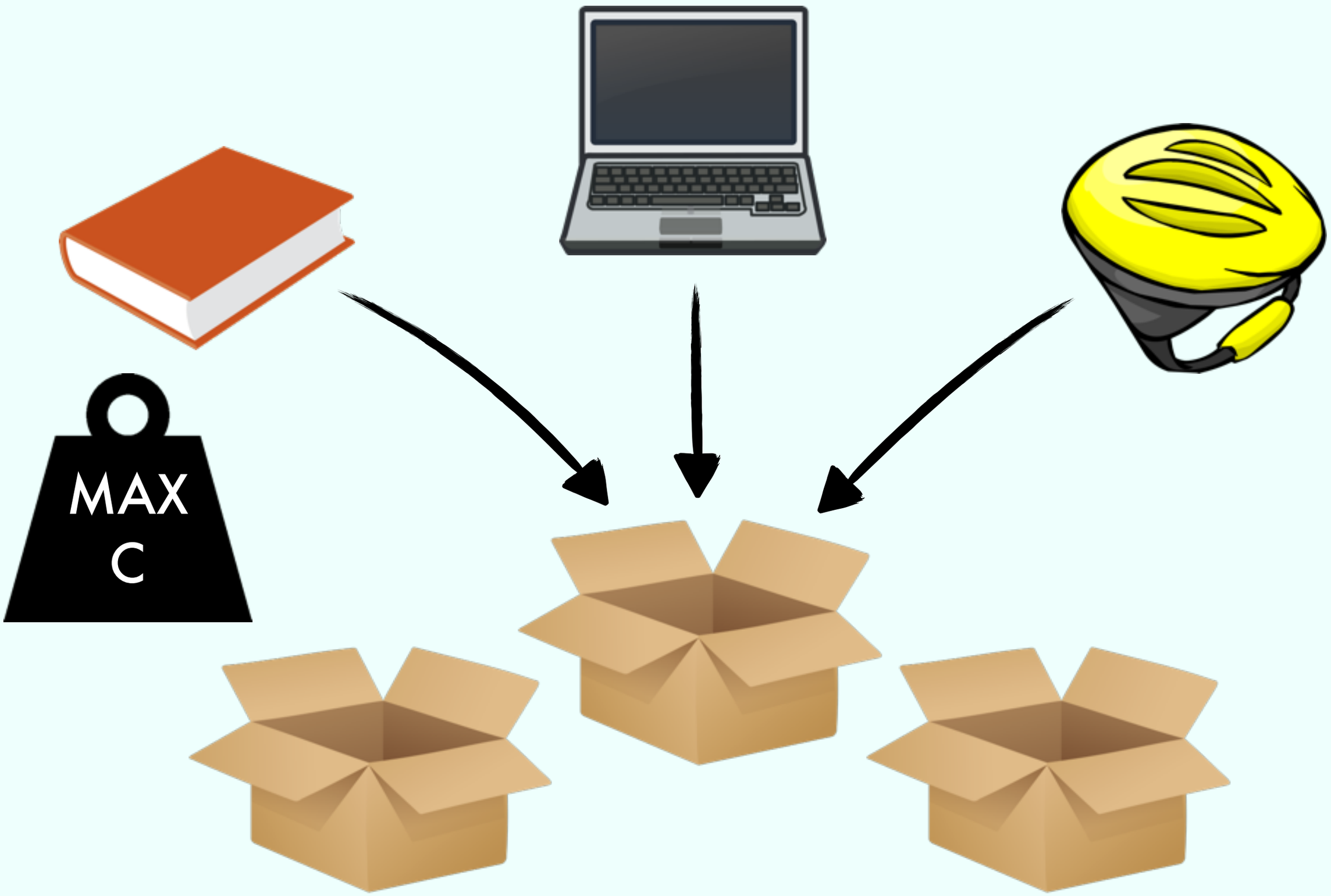
Exercise 6.2, 6.3



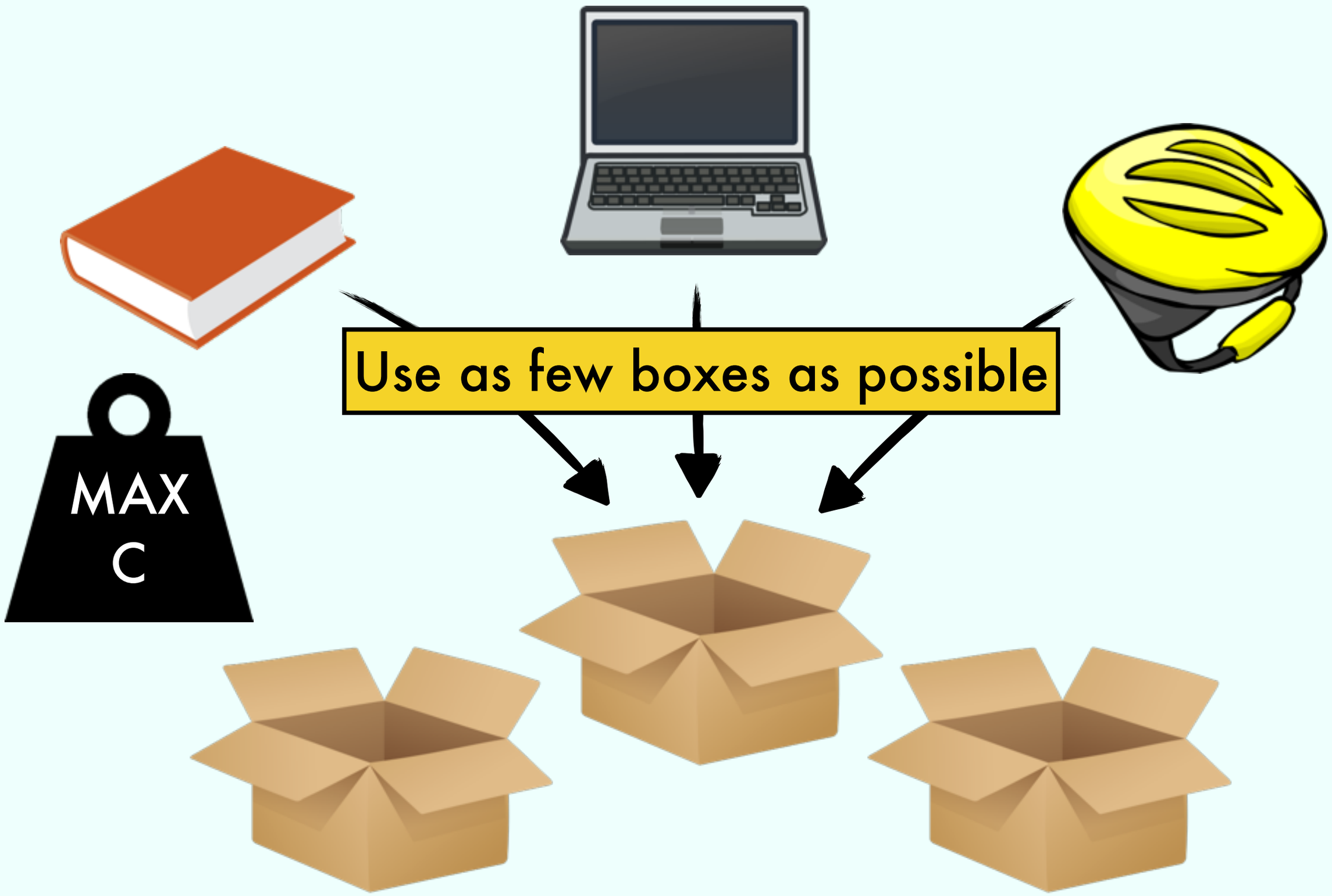
Exercise 6.2, 6.3



Exercise 6.2, 6.3



Exercise 6.2, 6.3



```
int pack (double C, double[] W)
```

- Use as few boxes as possible

- Put each weight in the box with most room for it

Capacity of boxes

```
int pack (double C, double[] W)
```

- Use as few boxes as possible

- Put each weight in the box with most room for it

Capacity of boxes

Weights of objects

```
int pack (double C, double[] W)
```

- Use as few boxes as possible

- Put each weight in the box with most room for it