

Heaps



Tiziana Ligorio
Hunter College of The City University of New York

Heap

A **Heap** is a complete binary tree that is either

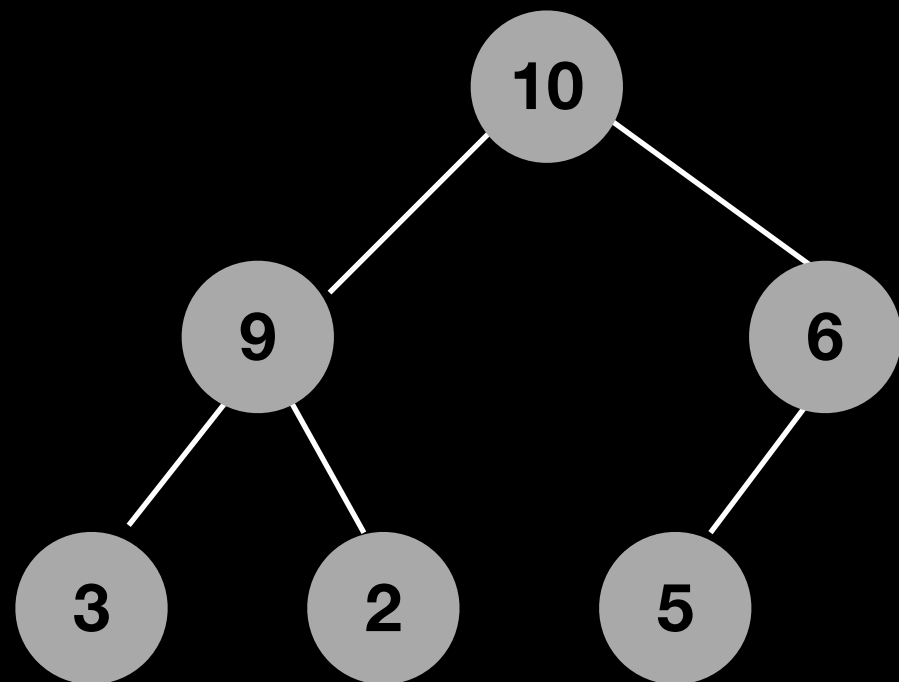
- Empty or
- Its **root** contains a value \geq (or \leq) both of its **children** and has **heaps as subtrees**

Heap

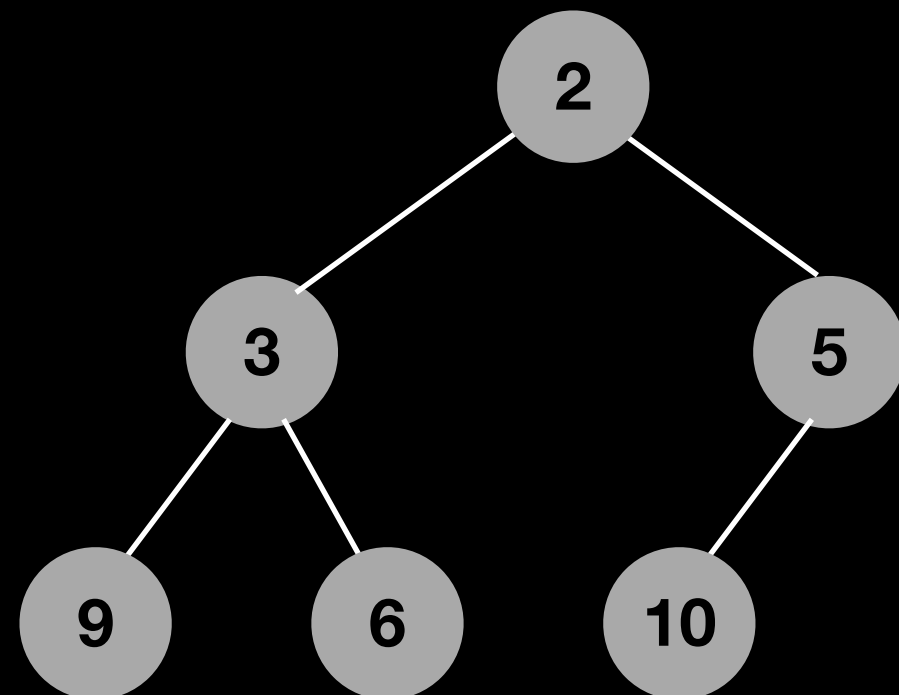
A special binary tree:

- **Ordered** in a weaker sense
- Always a **complete** binary tree

MaxHeap



MinHeap



Implementation

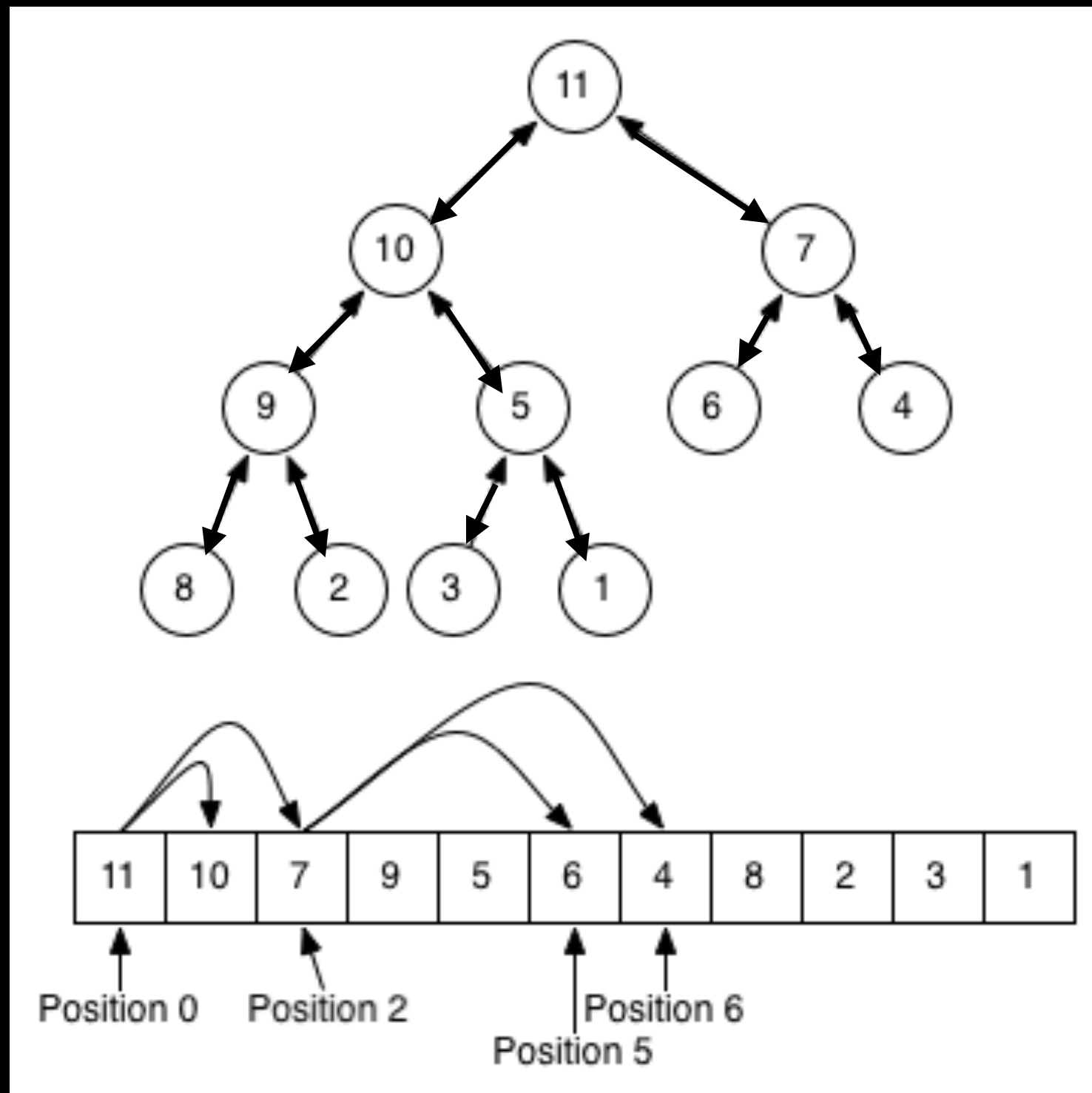
How would you implement it???

Implementation

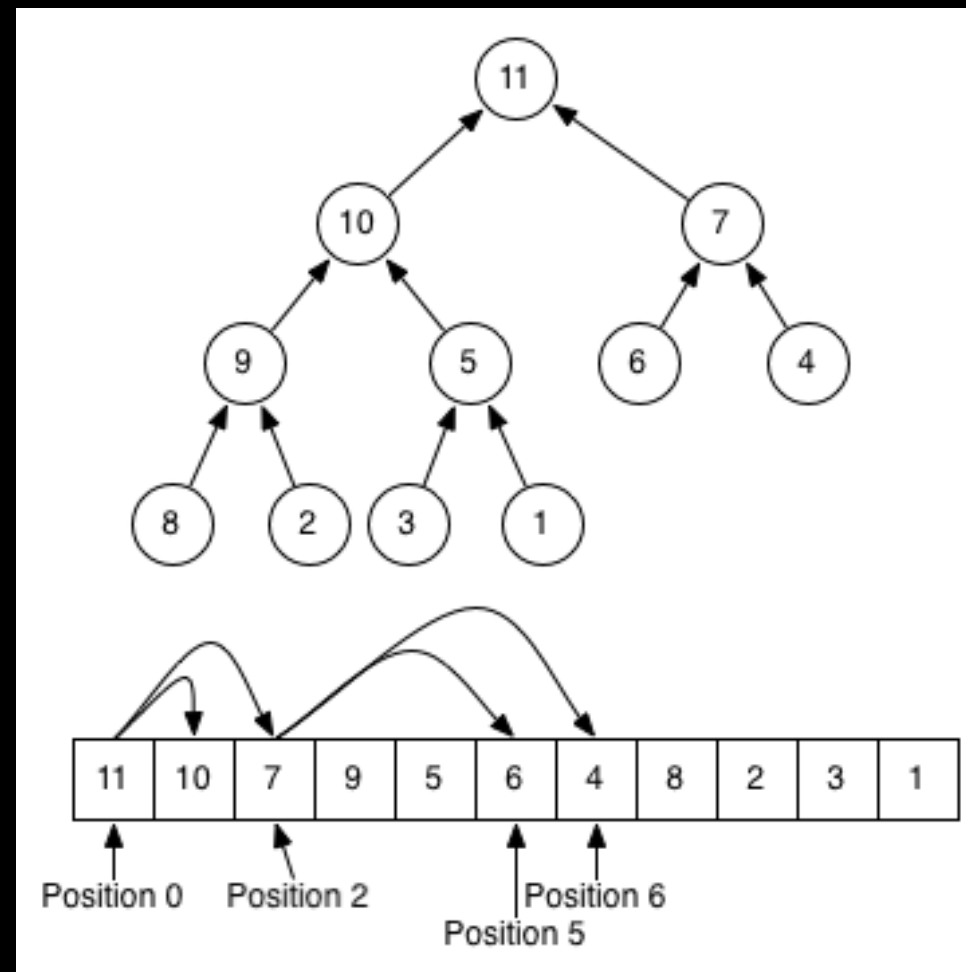
How would you implement it???

Insight: it is **always complete**

Implementation

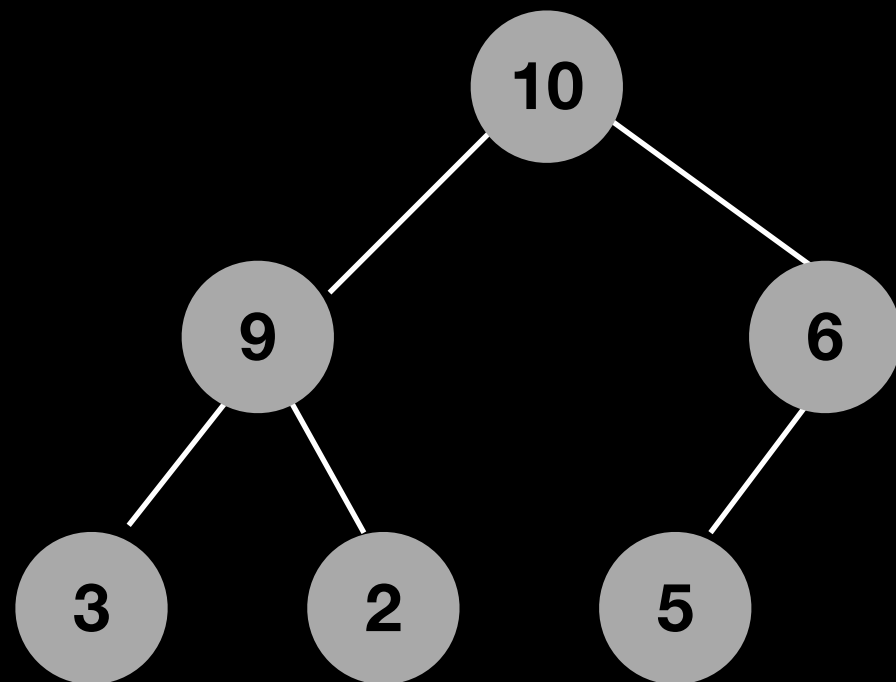


Implementation



```
root_ = items_[0]
items_[i] left_child = items_[2 * i + 1]
items_[i] right_child = items_[ 2 * i + 2]
items_[i] parent = items_[(i-1)//2]
```


MaxHeap



Priority Queue

10

?

?

5

Retrieve

Can only retrieve max/min item

Stored at root

$O(1)$

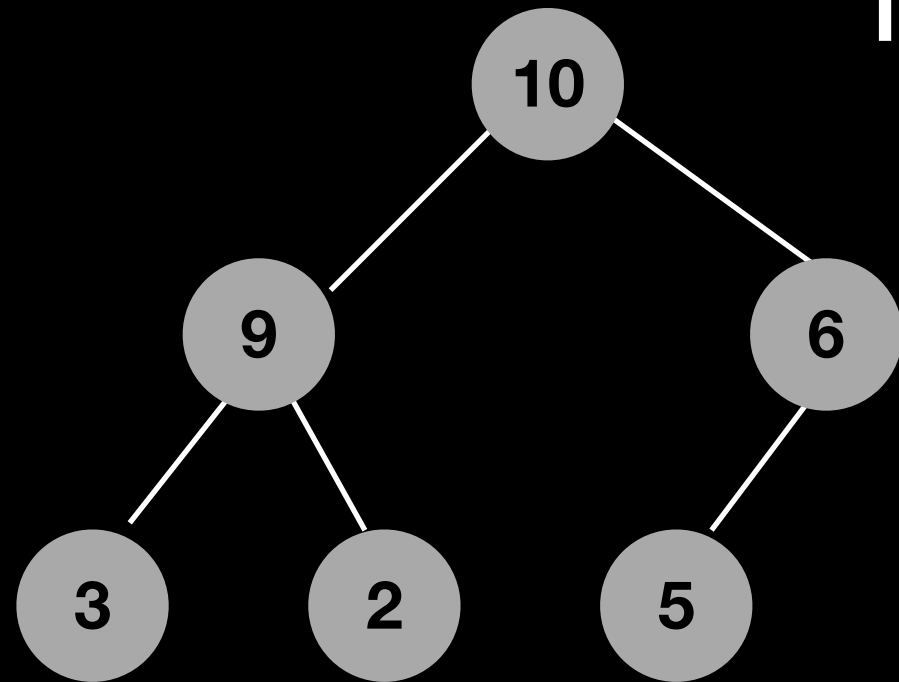
Remove

Remove max/min item (the root)

Must retain Heap

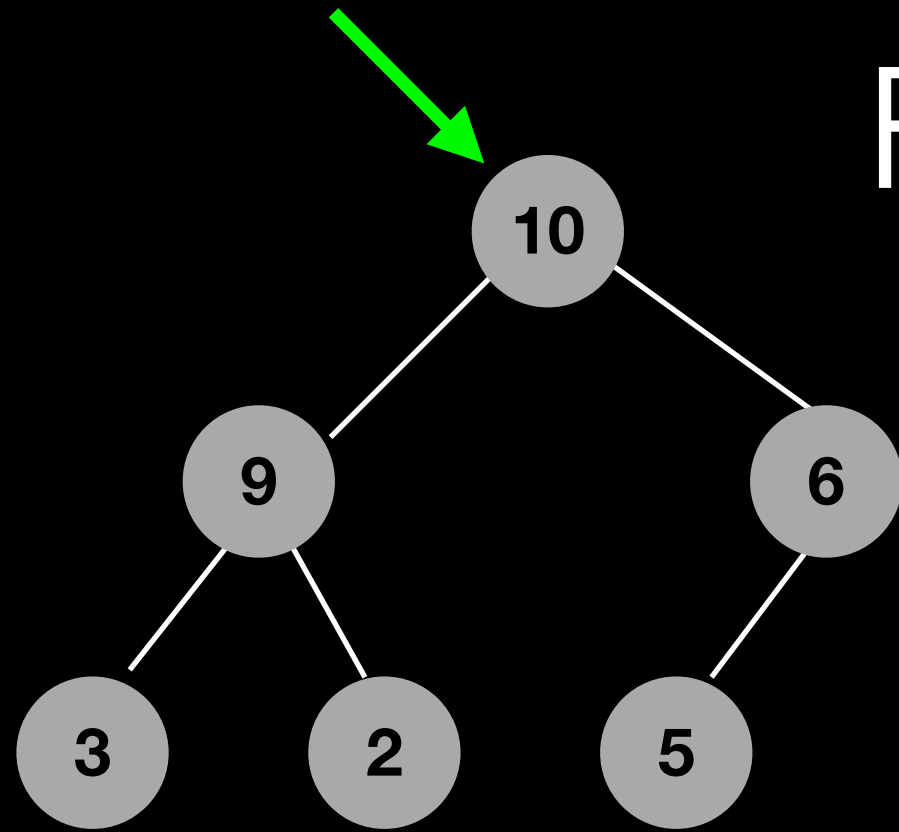
- Heap ordering property
- Complete

Remove



What element do we remove?

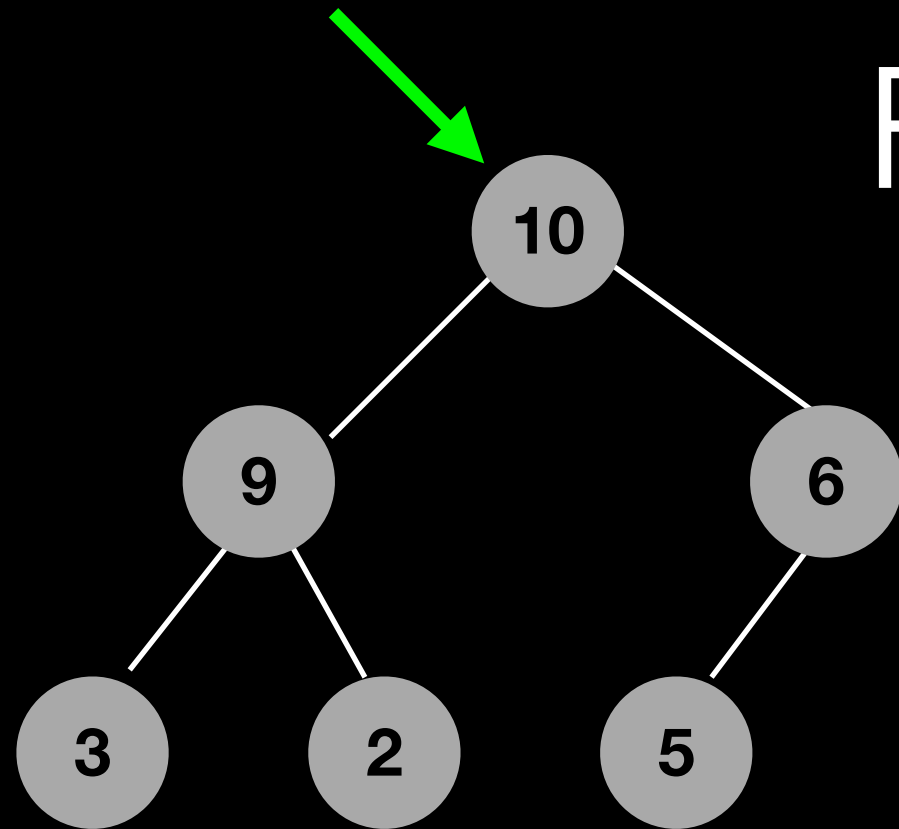
Remove



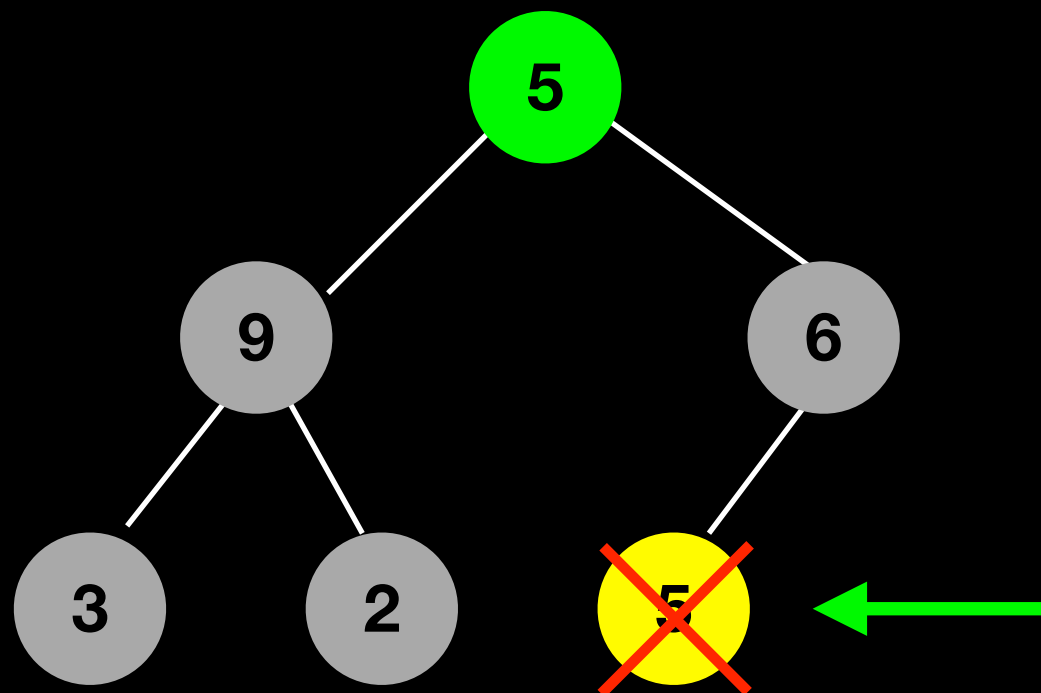
What element do we remove?

What node do we remove?

Remove

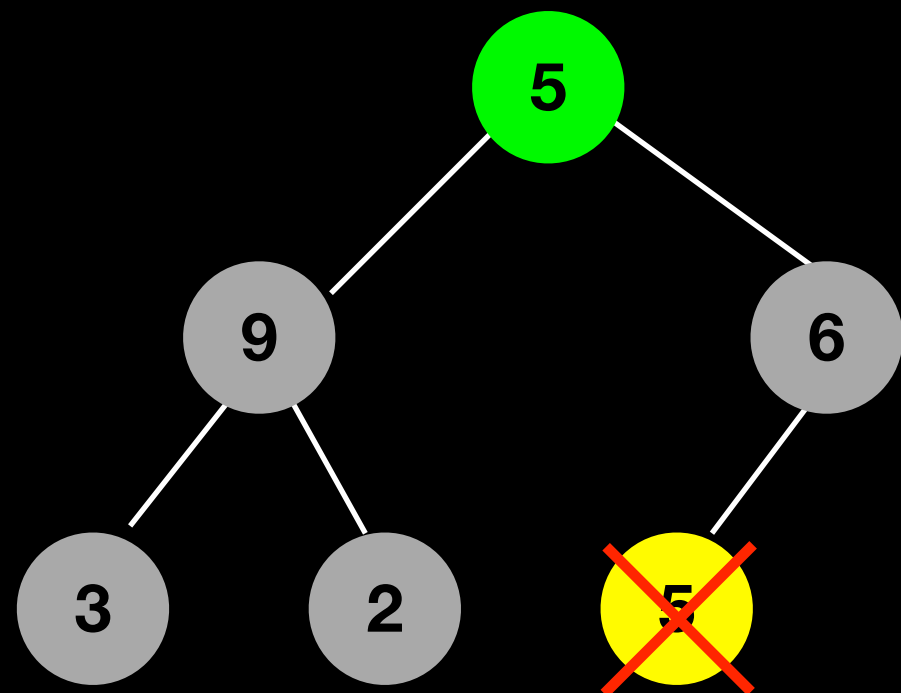
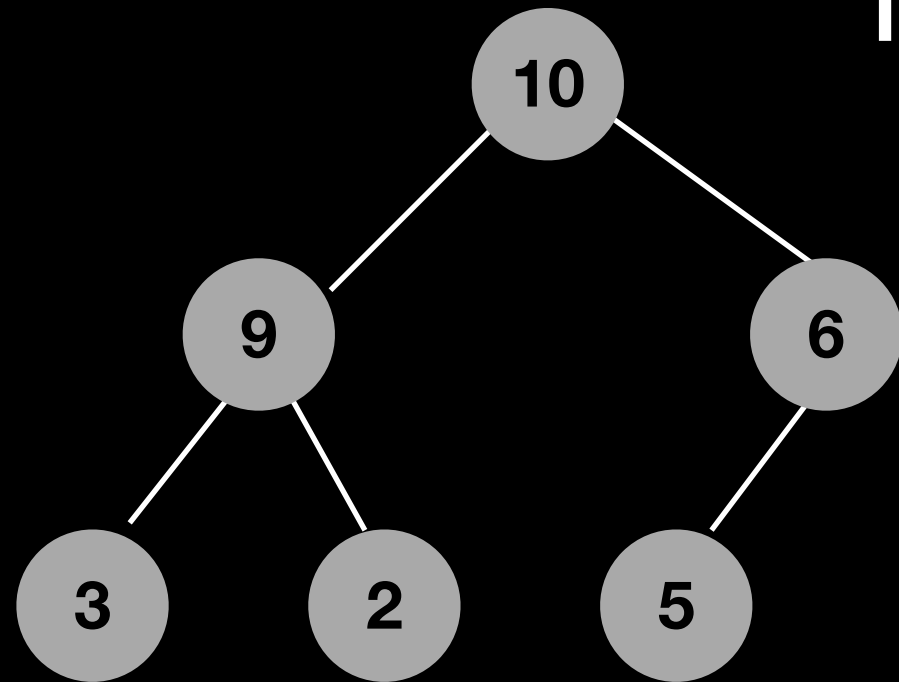


What element do we remove?

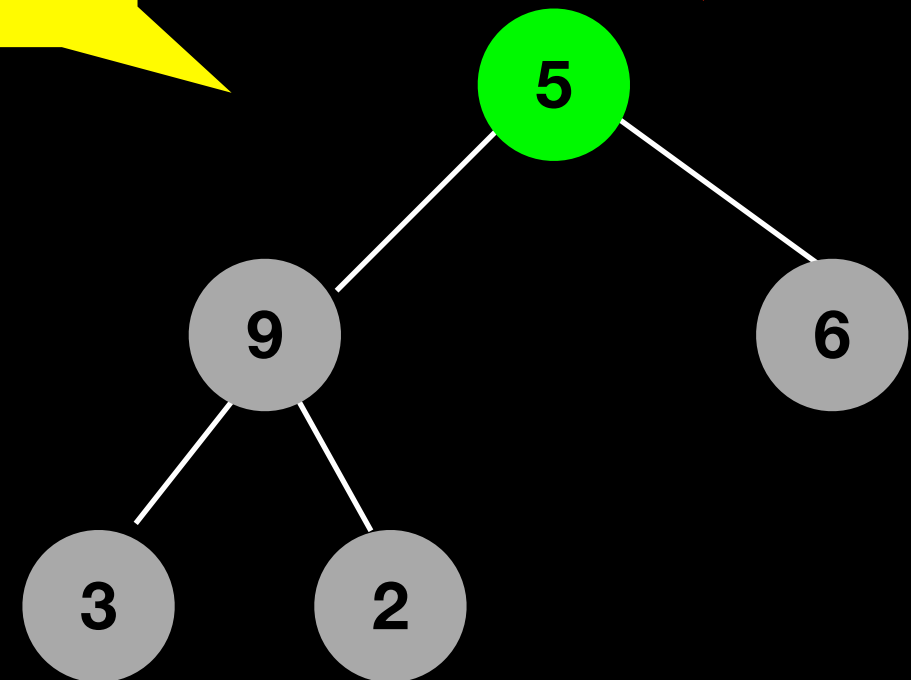


Remove this node form complete tree

Remove



Complete

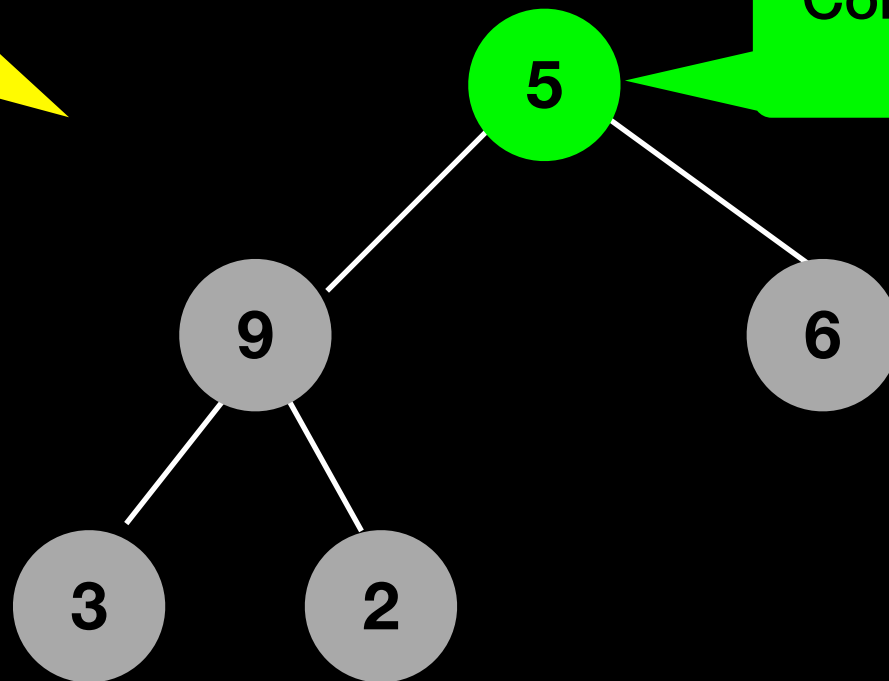


Not Heap

heapRebuild

Remove

Complete

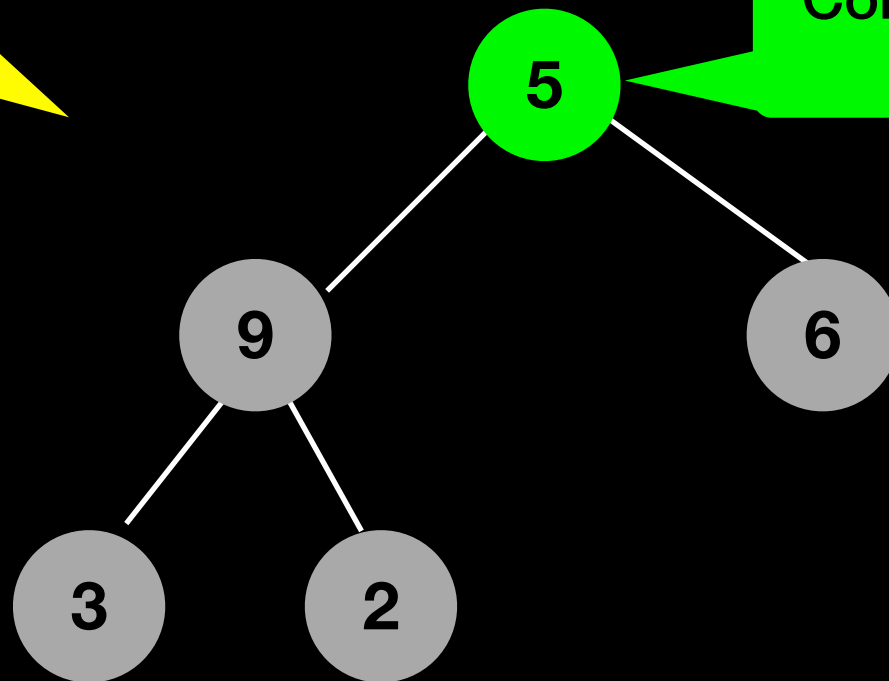


Trickle down by
swapping
Compare left and
right

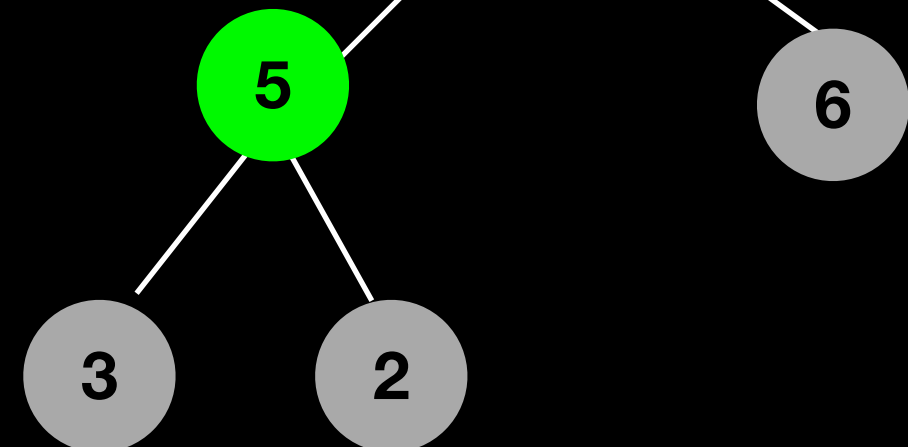
heapRebuild

Remove

Complete



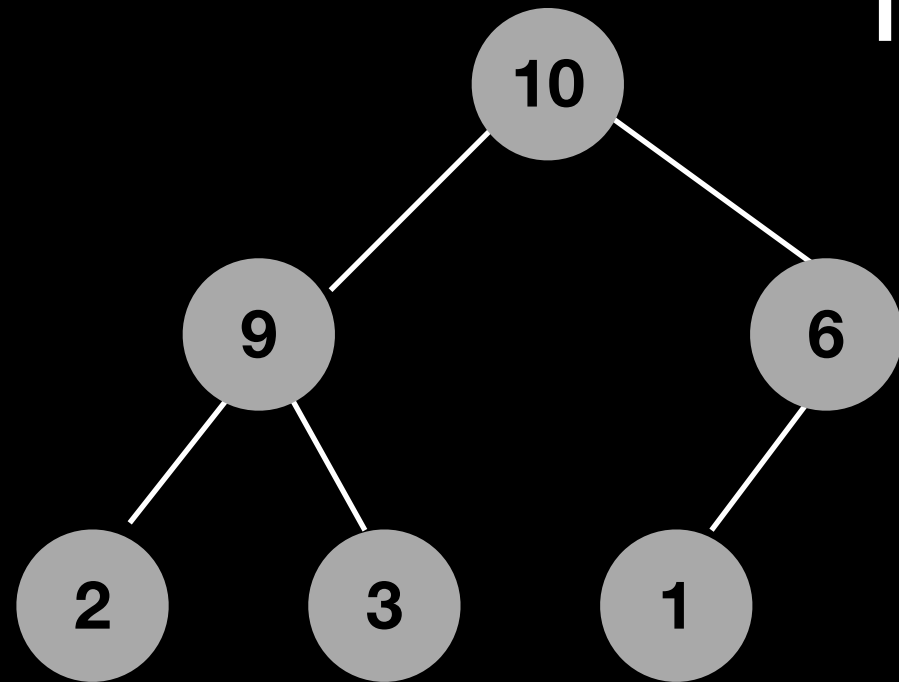
Trickle down by
swapping
Compare left and
right



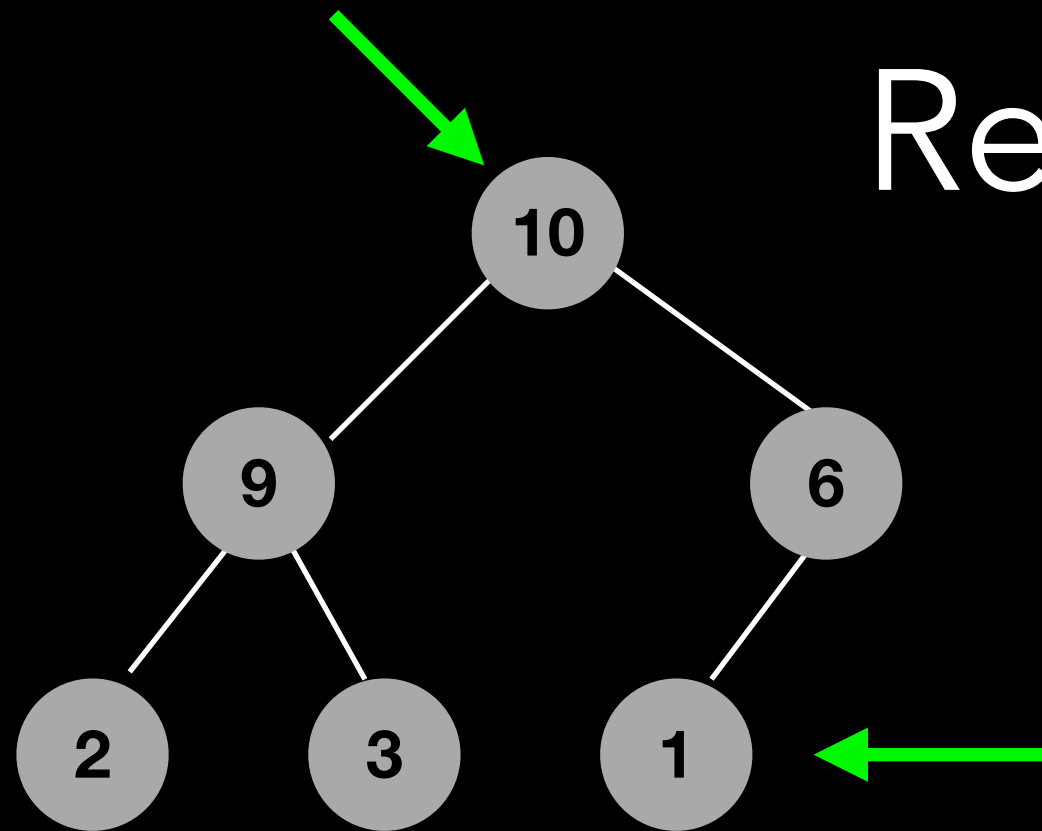
Heap!



Remove

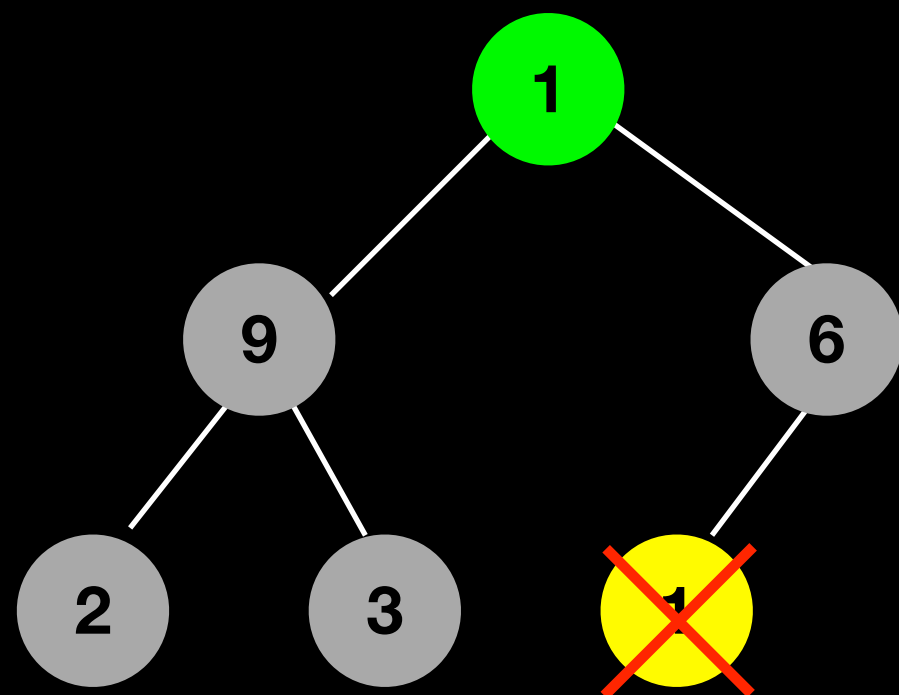
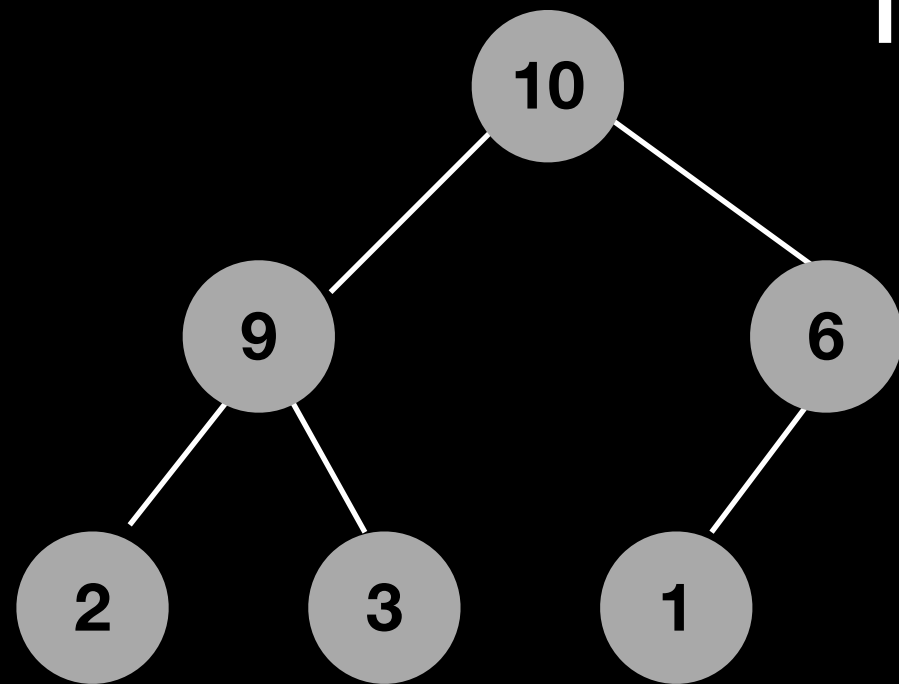


Remove

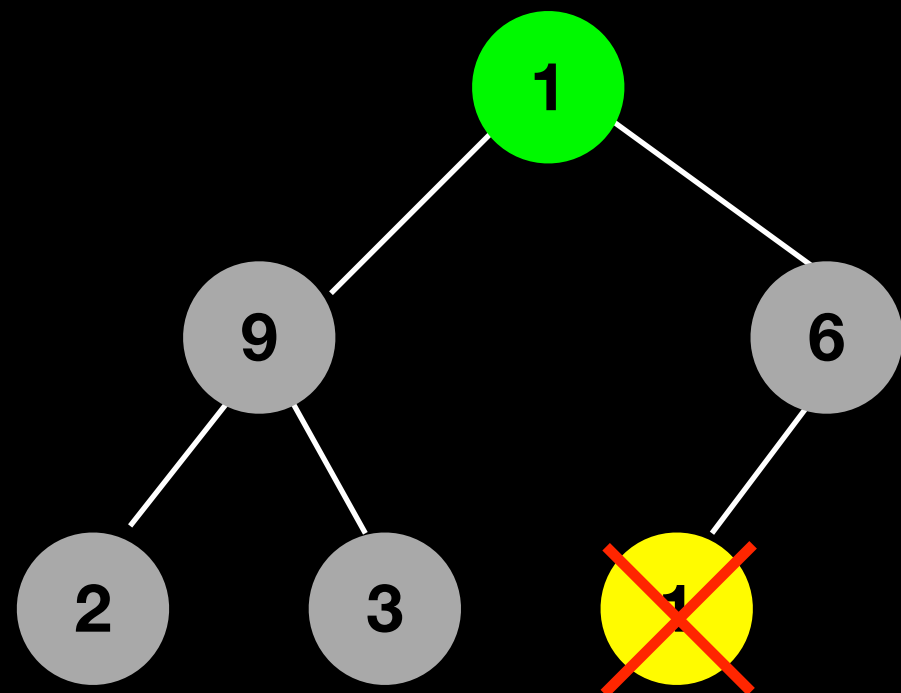
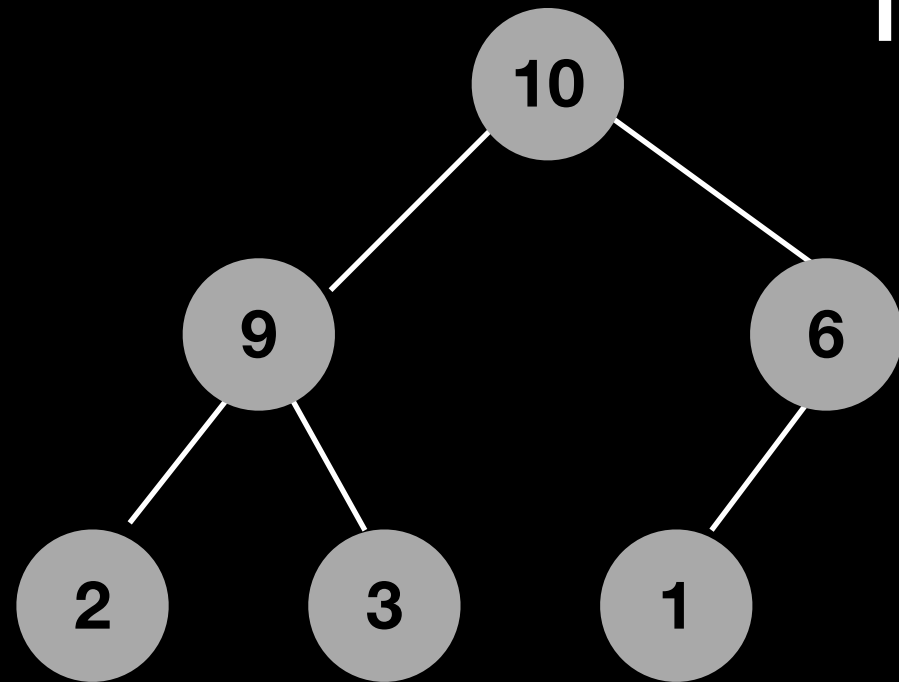


Remove this node form complete tree

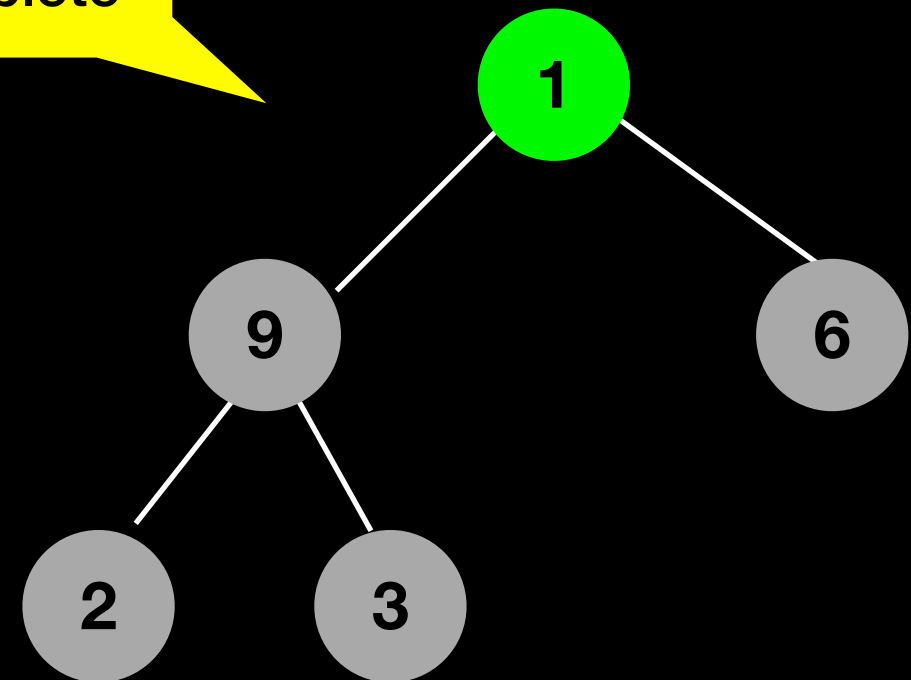
Remove



Remove



Complete



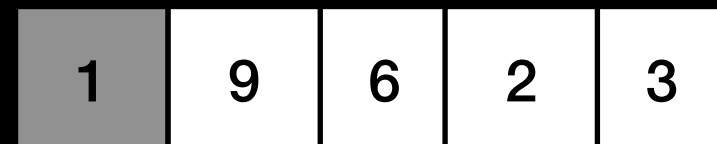
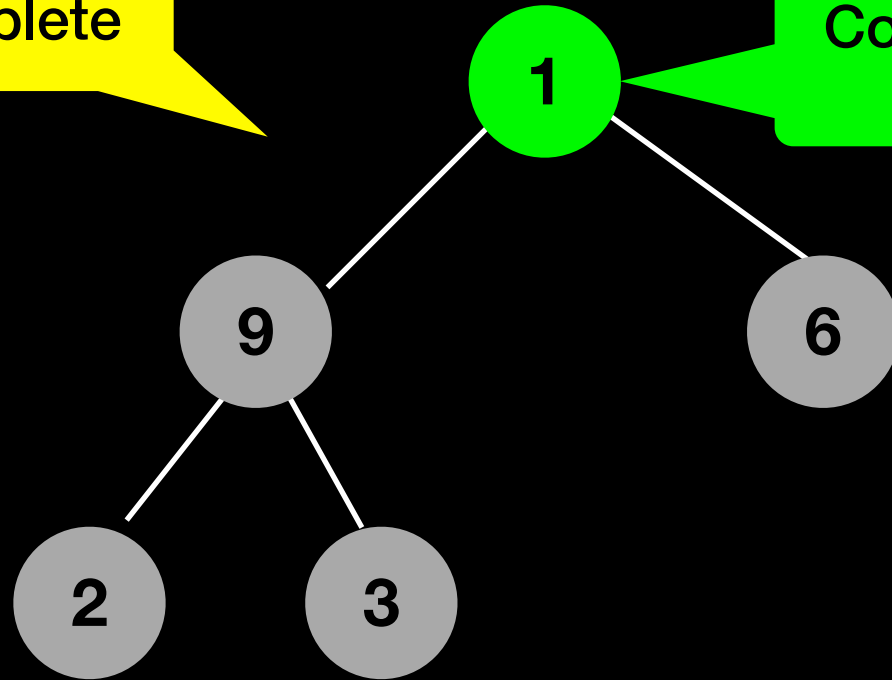
Not Heap

heapRebuild

Remove

Complete

Trickle down by
swapping
Compare left and
right

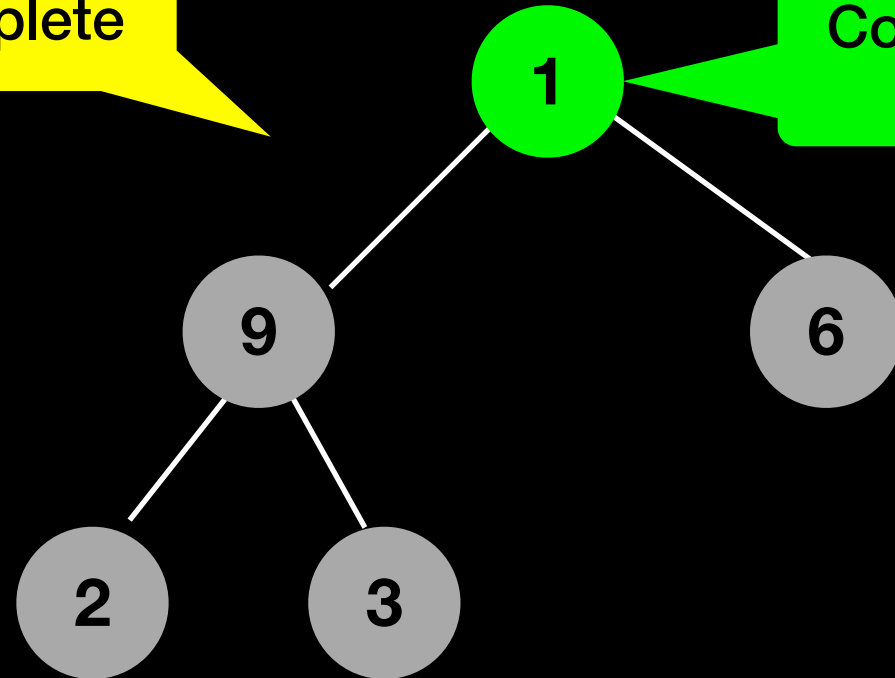


```
items[i] left_child = items[2 * i + 1]
items[i] right_child = items[2 * i + 2]
```

heapRebuild

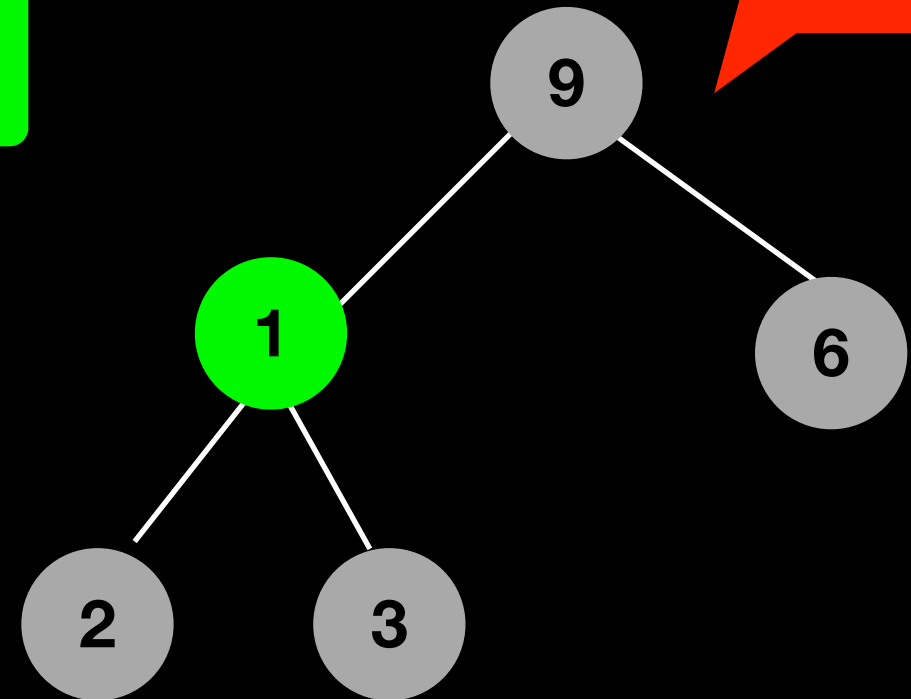
Remove

Complete



Trickle down by
swapping
Compare left and
right

Not Heap



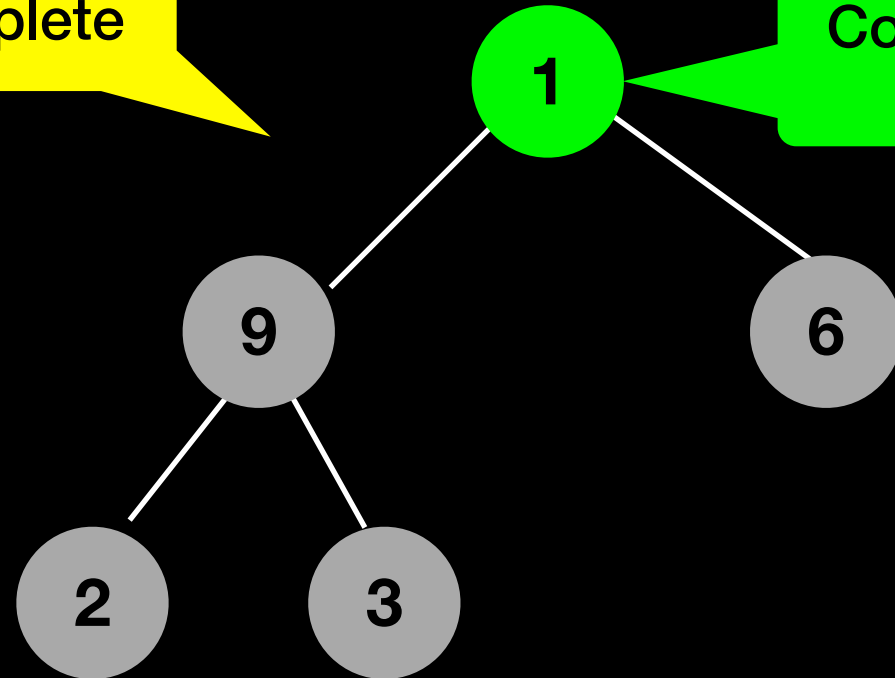
1	9	6	2	3
1	9	6	2	3

```
items[i] left_child = items[2 * i + 1]
items[i] right_child = items[2 * i + 2]
```

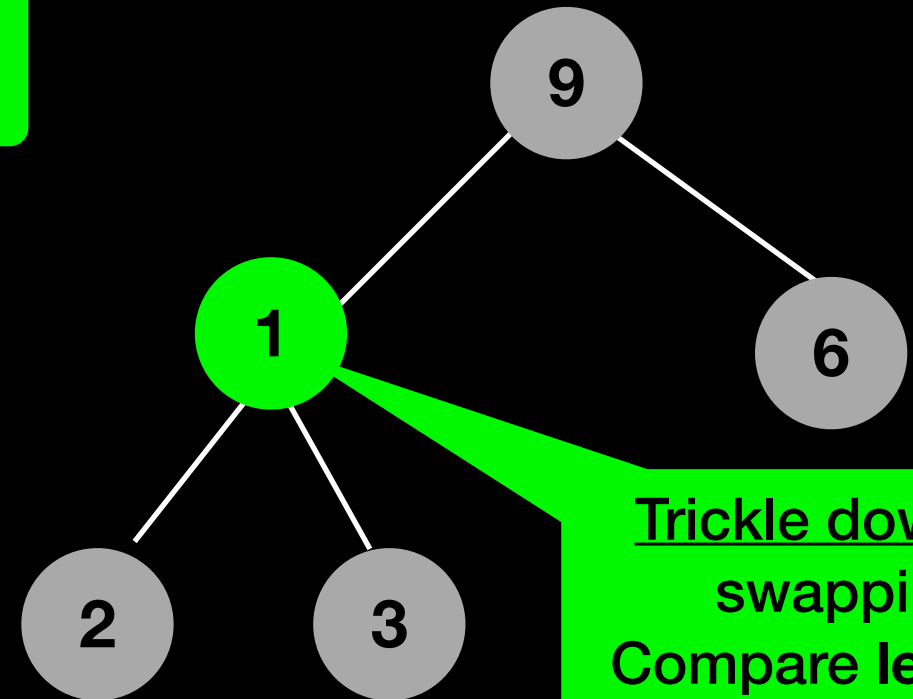
heapRebuild

Remove

Complete



Trickle down by
swapping
Compare left and
right



Trickle down by
swapping
Compare left and
right

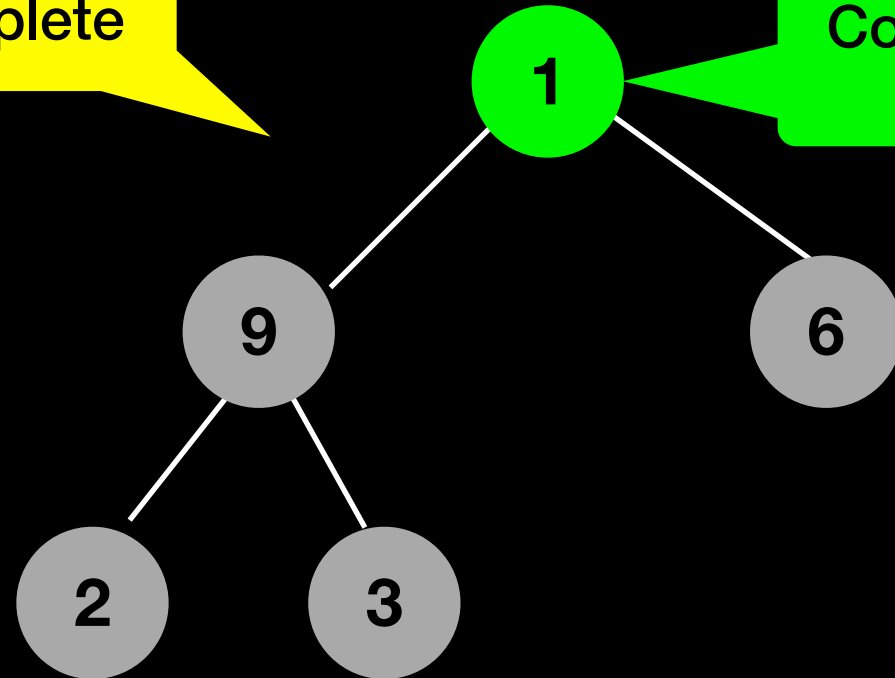
1	9	6	2	3
1	9	6	2	3

```
items[i] left_child = items[2 * i + 1]
items[i] right_child = items[2 * i + 2]
```

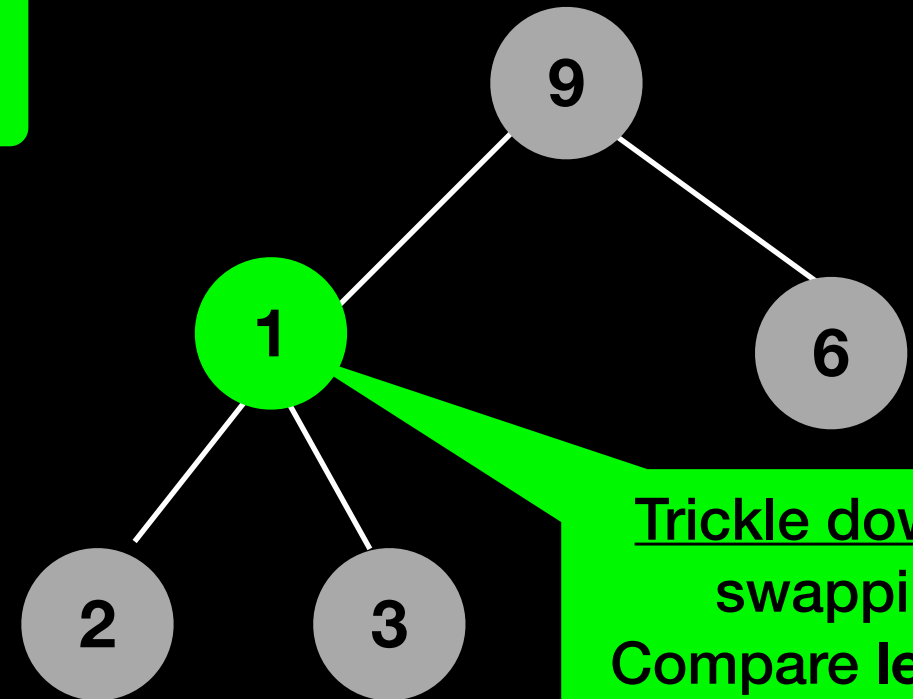

heapRebuild

Remove

Complete



Trickle down by
swapping
Compare left and
right



Trickle down by
swapping
Compare left and
right

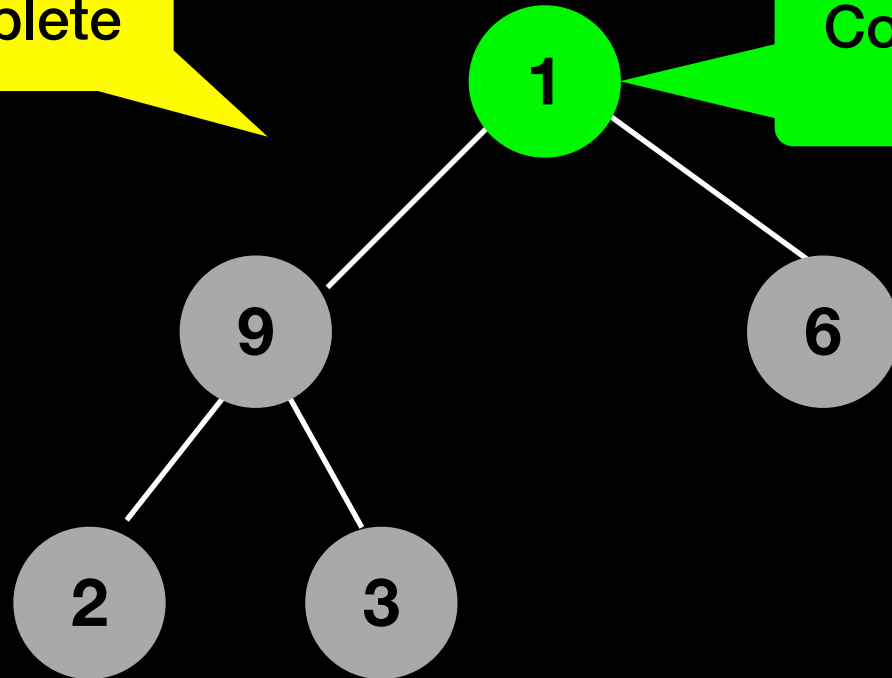
1	9	6	2	3
1	9	6	2	3
9	1	6	2	3

```
items[i] left_child = items[2 * i + 1]
items[i] right_child = items[2 * i + 2]
```

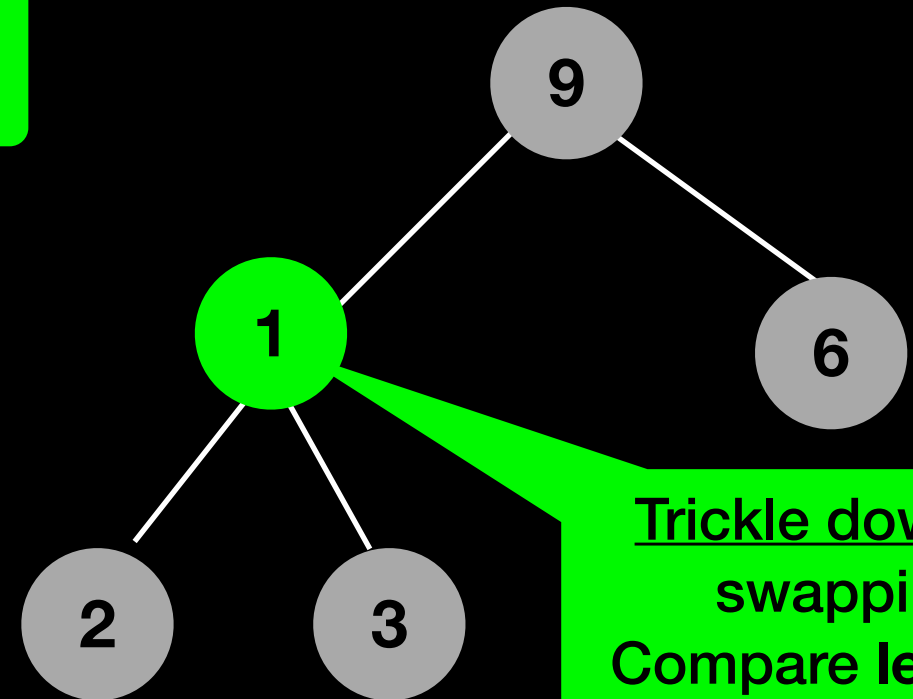
heapRebuild

Remove

Complete

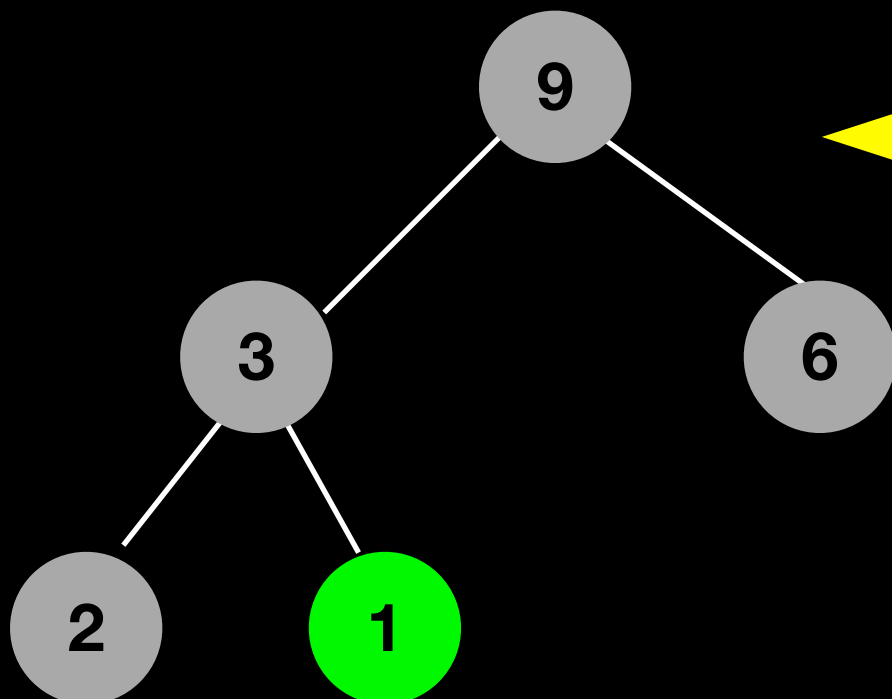


Trickle down by
swapping
Compare left and
right



Trickle down by
swapping
Compare left and
right

Heap!

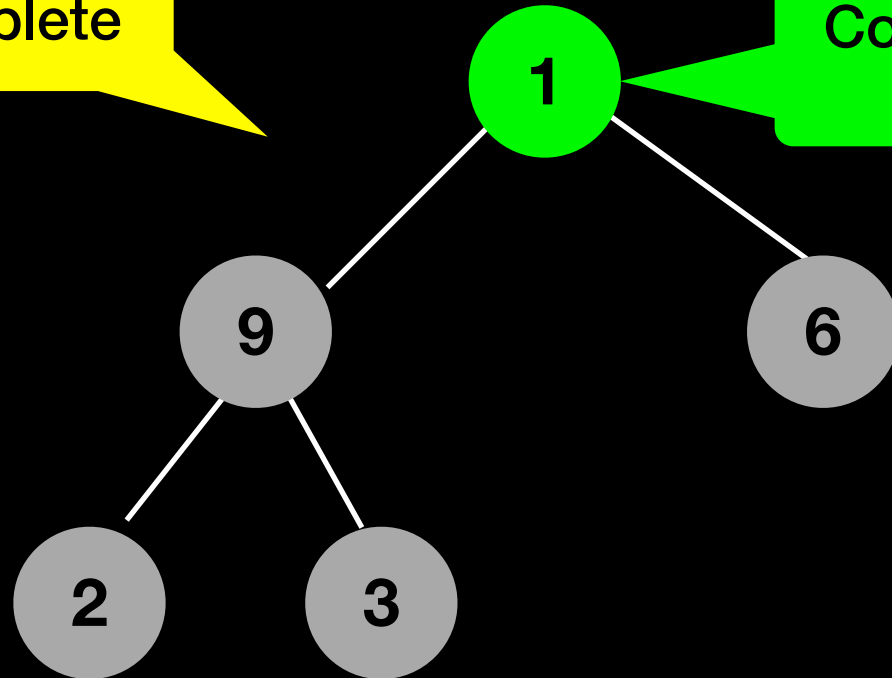


1	9	6	2	3
1	9	6	2	3
9	1	6	2	3
9	3	6	2	1

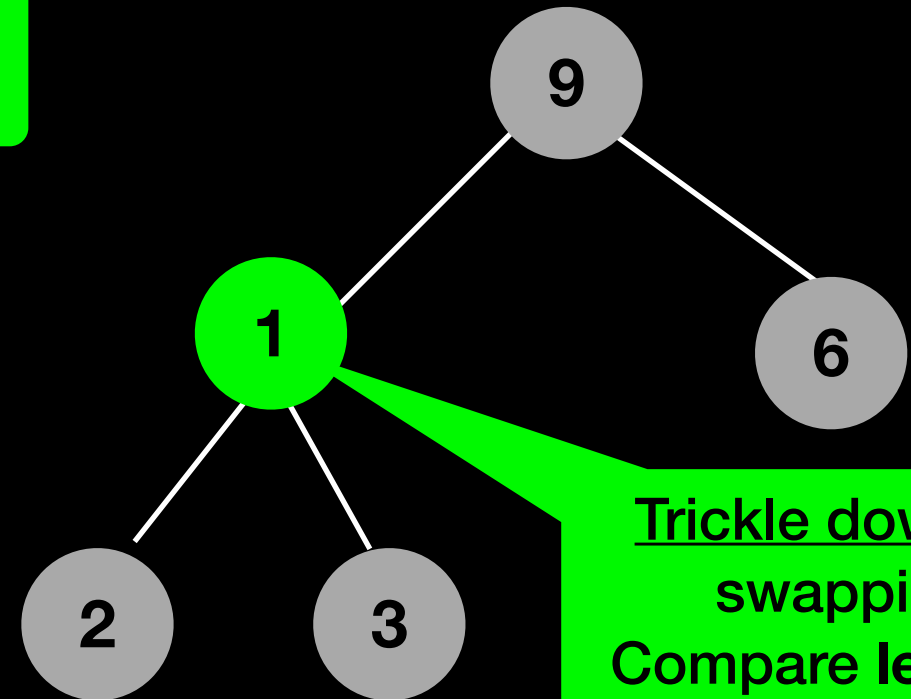
heapRebuild

Remove

Complete

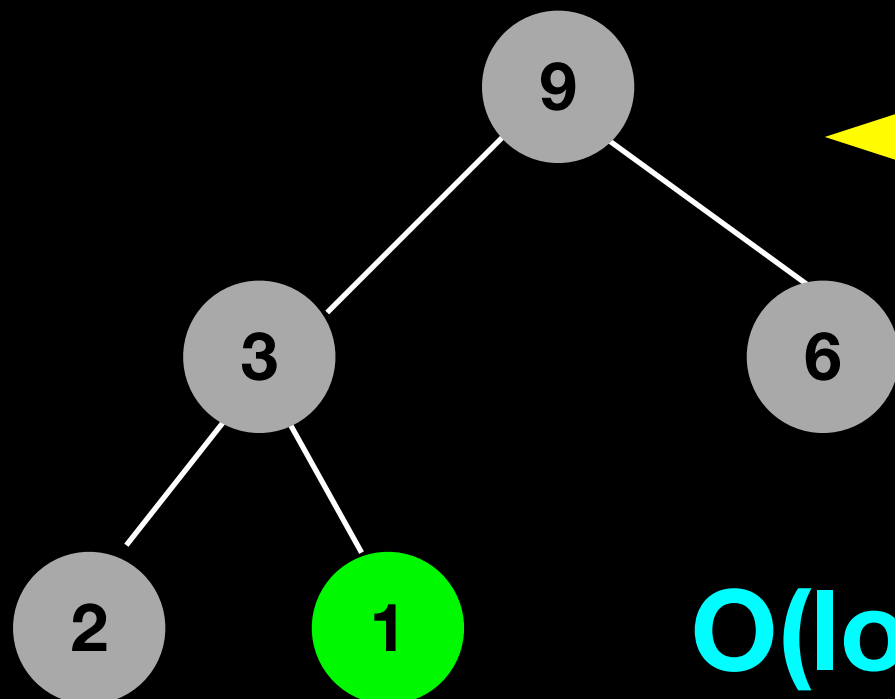


Trickle down by
swapping
Compare left and
right



Trickle down by
swapping
Compare left and
right

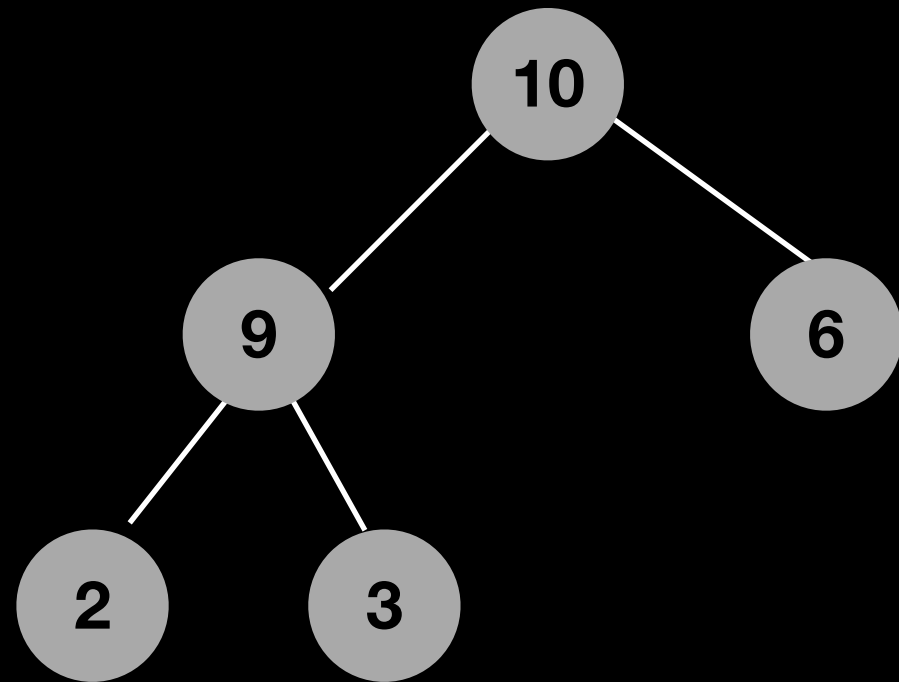
Heap!



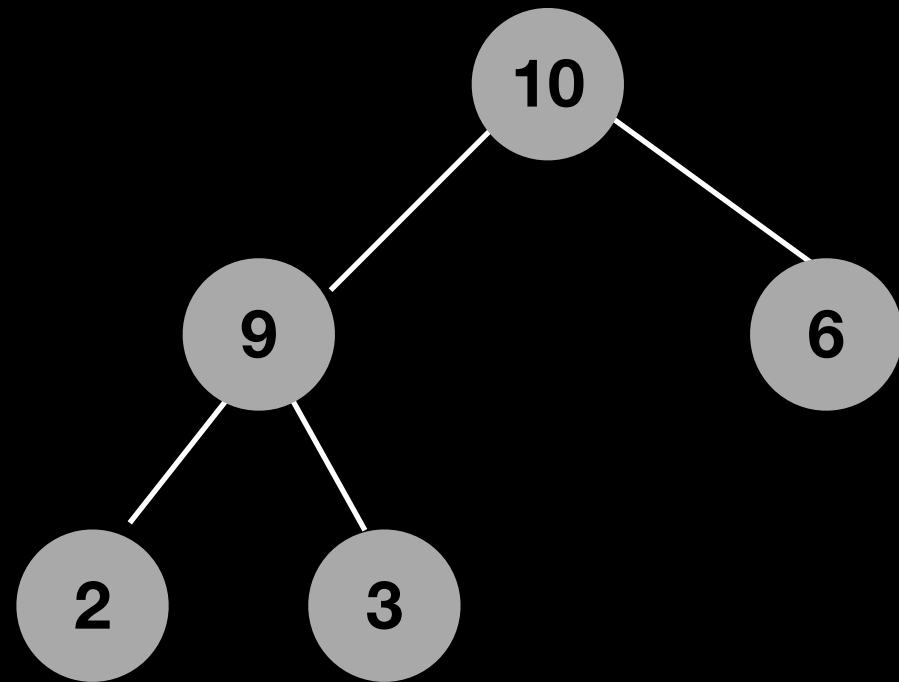
$O(\log n)$

1	9	6	2	3
1	9	6	2	3
9	1	6	2	3
9	3	6	2	1

Add



Add

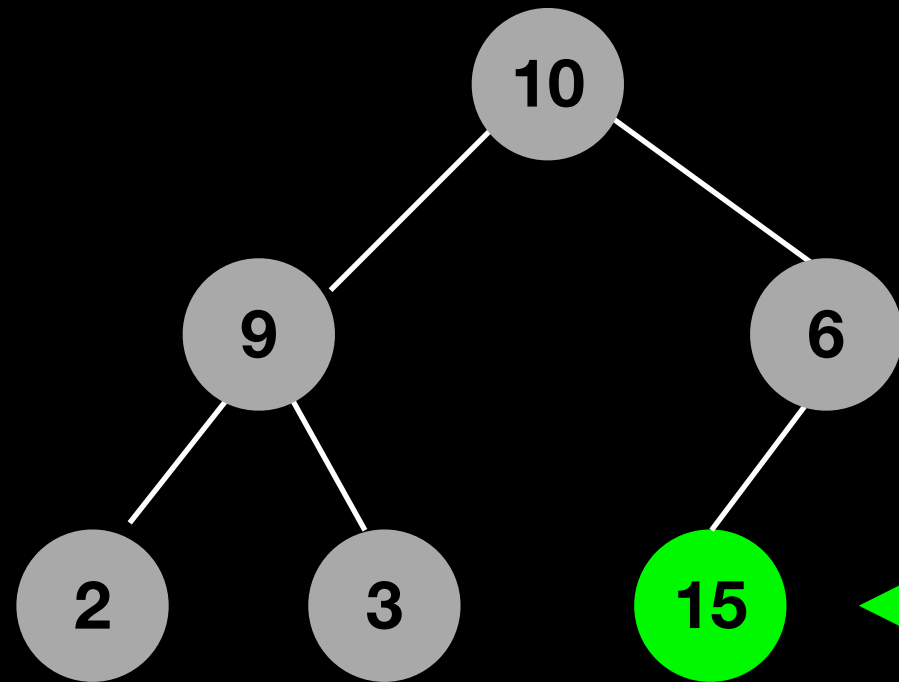


15

Where do we add?

10	9	6	2	3
----	---	---	---	---

Add

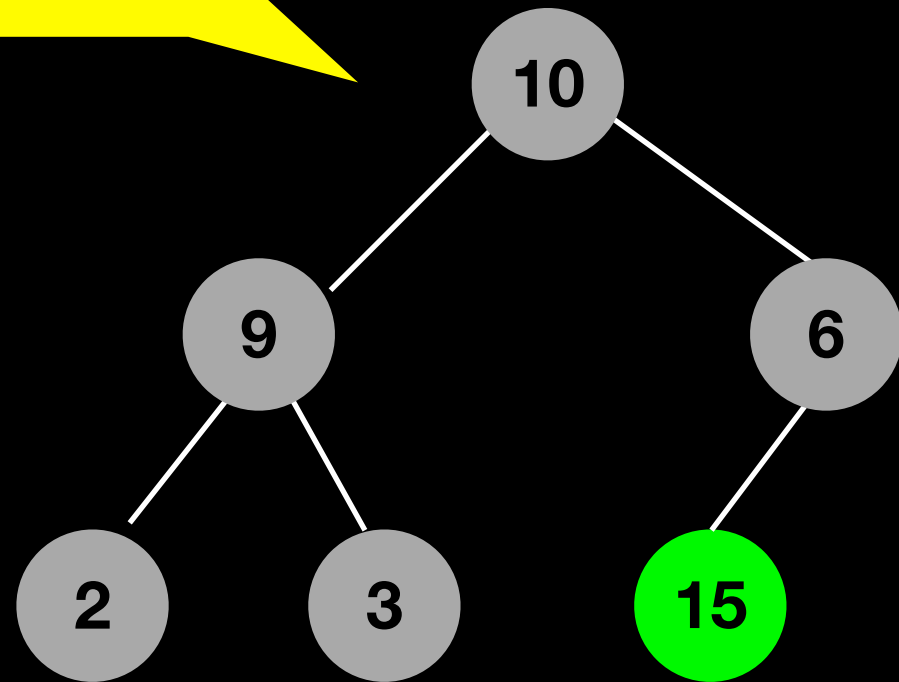


Add here for complete tree

10	9	6	2	3	15
----	---	---	---	---	----

```
items[i] left_child = items[2 * i + 1] 30
```

Complete

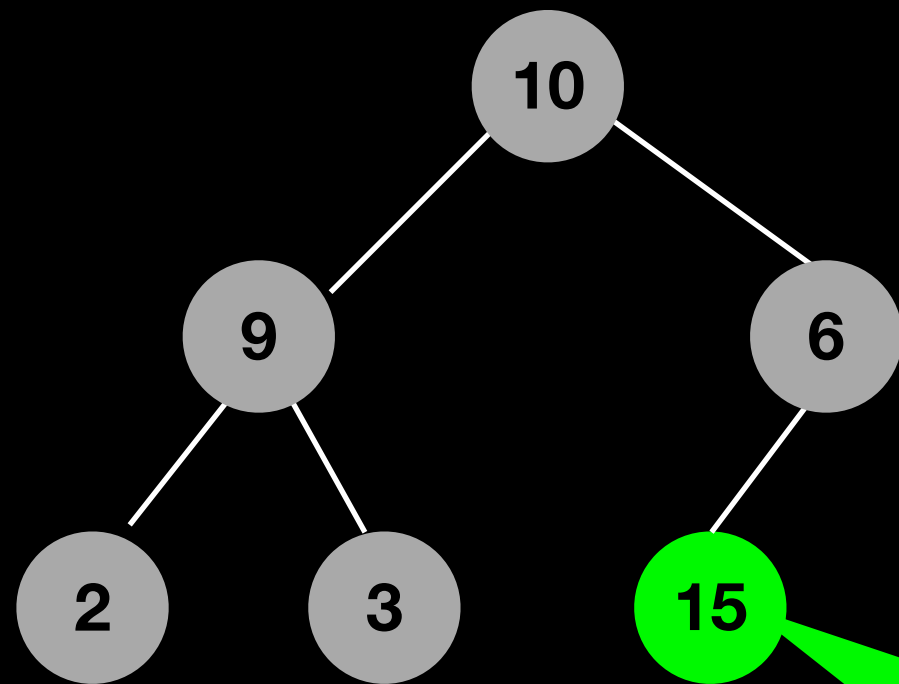


Add

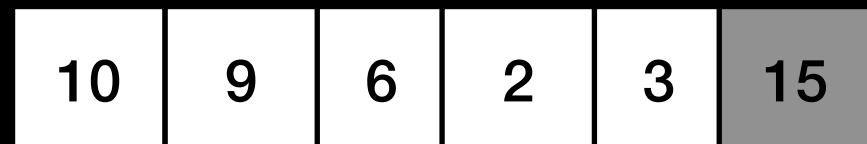
Not Heap

10	9	6	2	3	15
----	---	---	---	---	----

Add

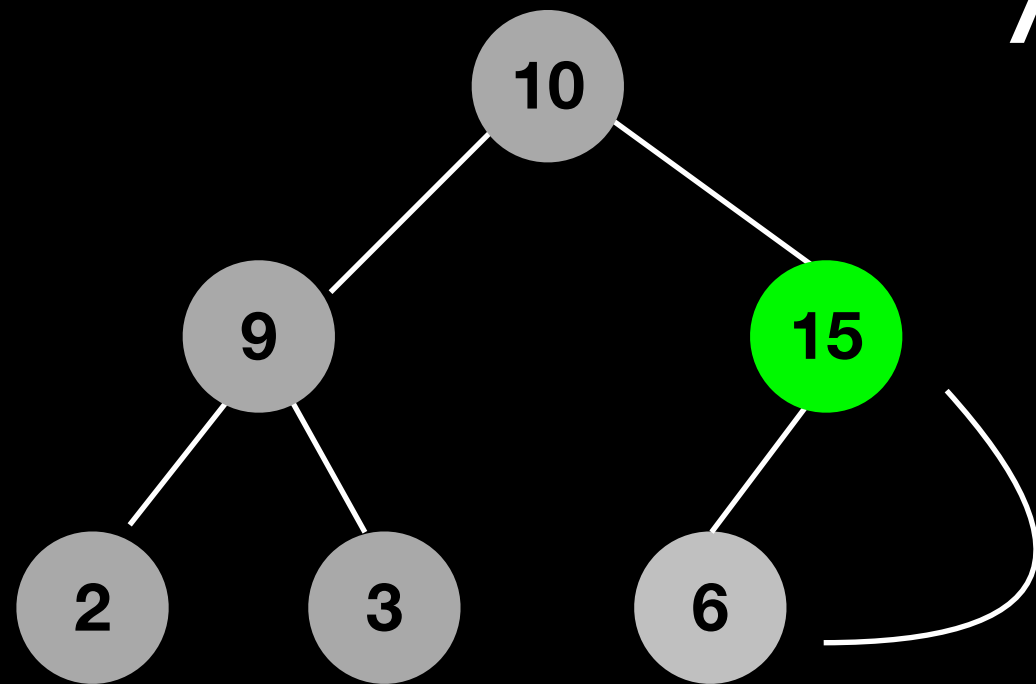


Bubble-up by
swapping
with parent



```
items[i] parent = items[(i-1)//2]
```

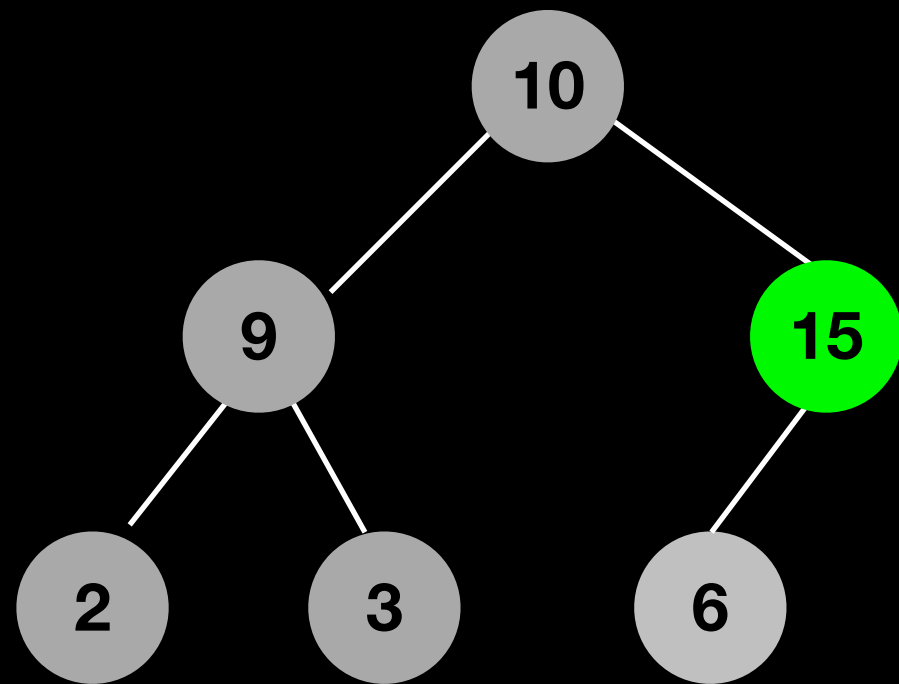

Add



10	9	6	2	3	15
10	9	15	2	3	6

```
items[i] parent = items_[(i-1)//2]
```

Add

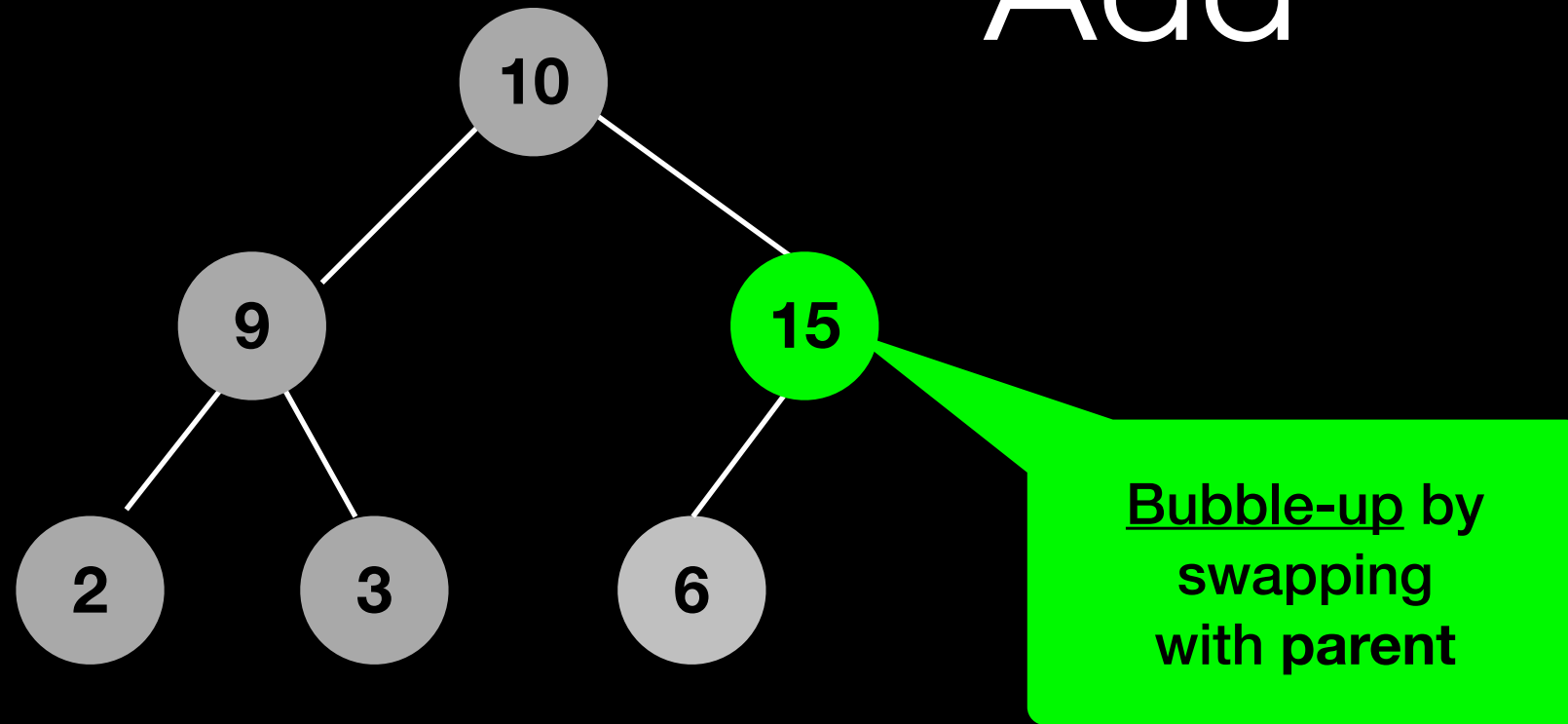


Not Heap

10	9	6	2	3	15
10	9	15	2	3	6

```
items[i] parent = items_[(i-1)//2]
```

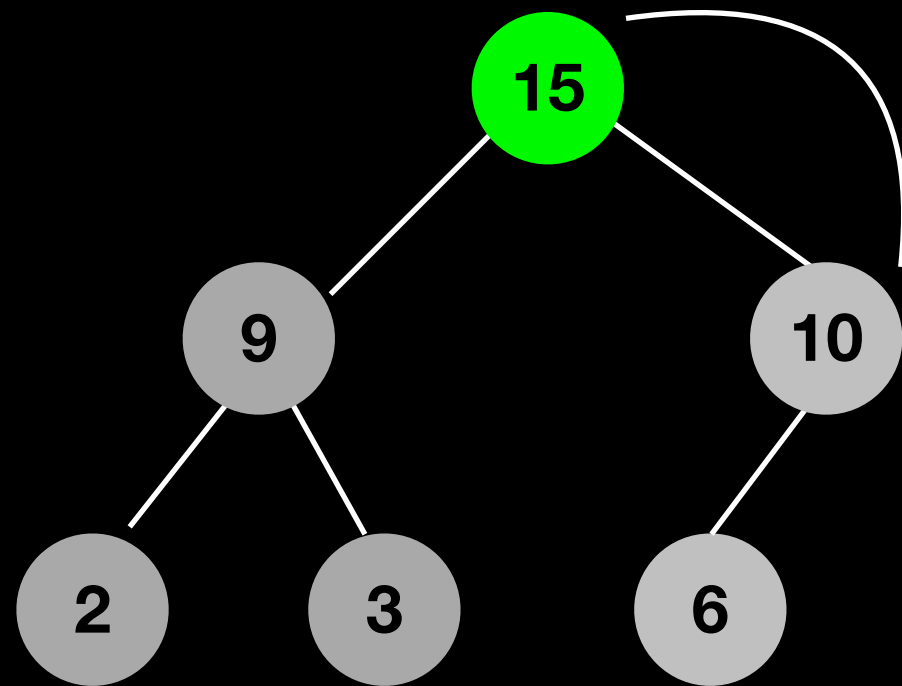
Add



10	9	6	2	3	15
10	9	15	2	3	6

```
items[i] parent = items_[(i-1)//2]
```

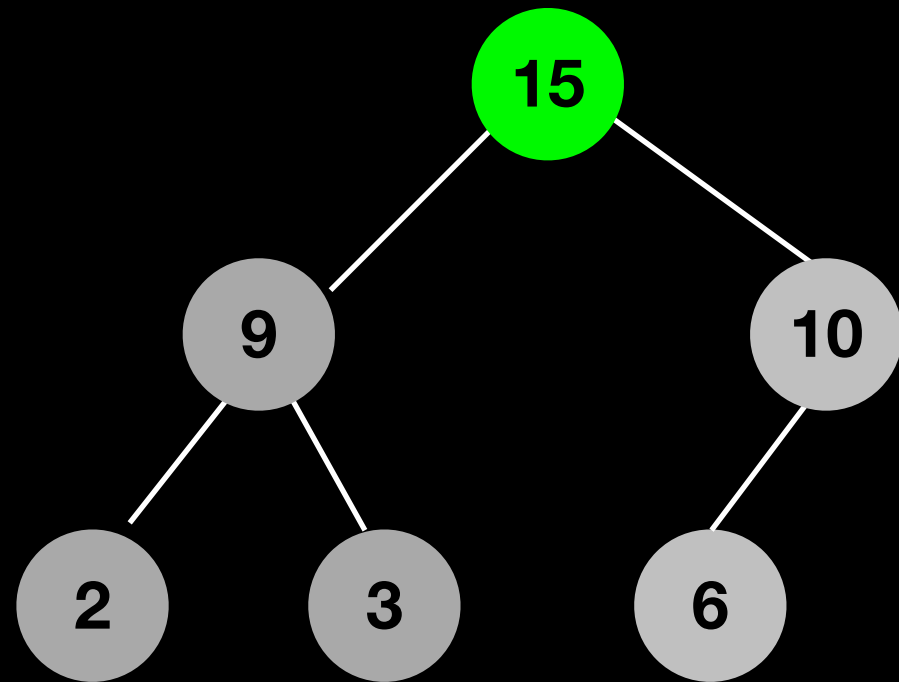
Add



10	9	6	2	3	15
10	9	15	2	3	6
15	9	10	2	3	6

```
items[i] parent = items_[(i-1)//2]
```

Add



Heap!



$O(\log n)$

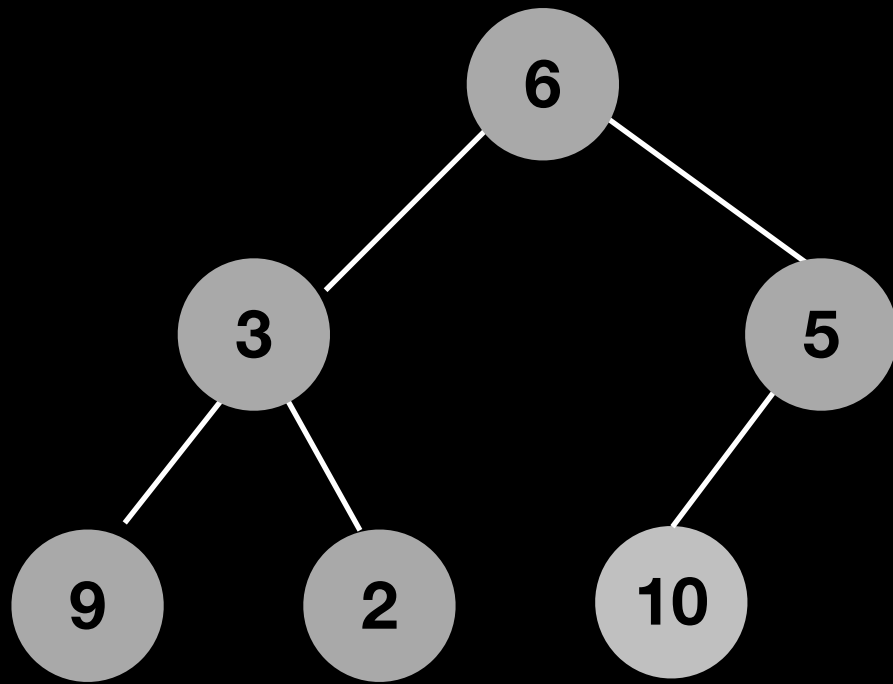
10	9	6	2	3	15
10	9	15	2	3	6
15	9	10	2	3	6

```
items[i] parent = items_[(i-1)//2]
```

heapCreate

6	3	5	9	2	10
---	---	---	---	---	----

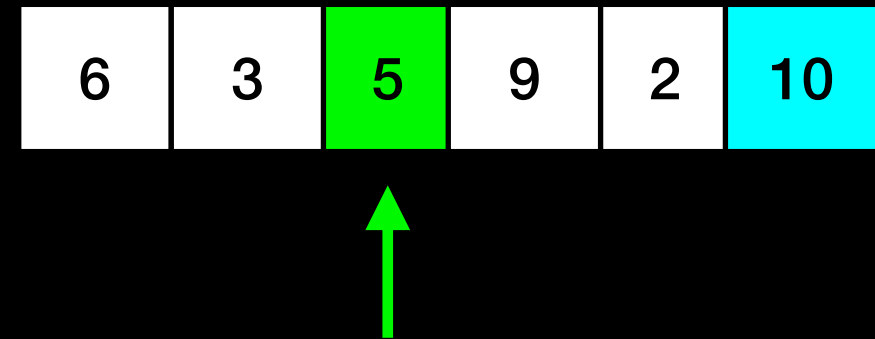
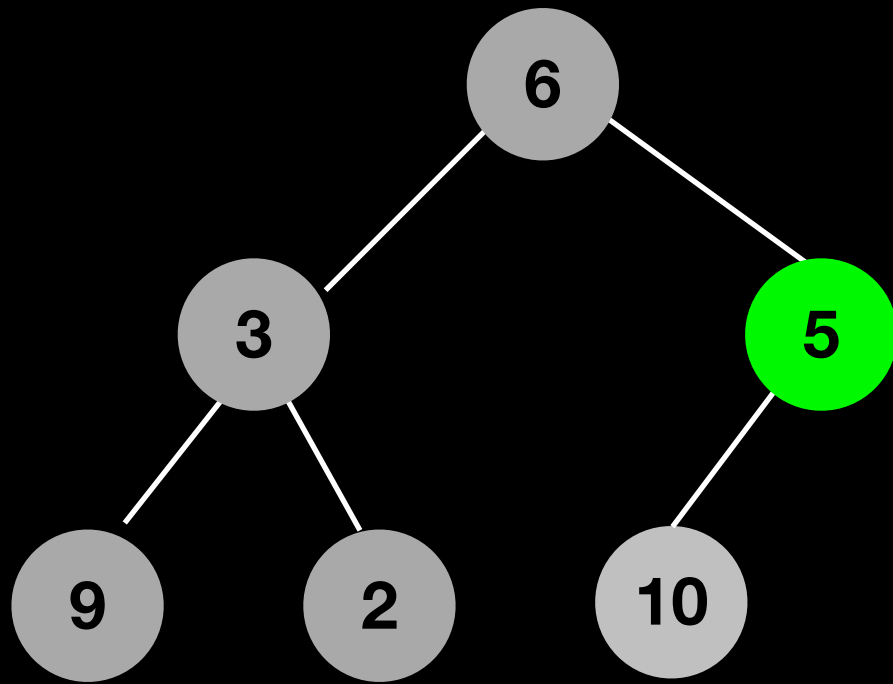
heapCreate



6	3	5	9	2	10
---	---	---	---	---	----

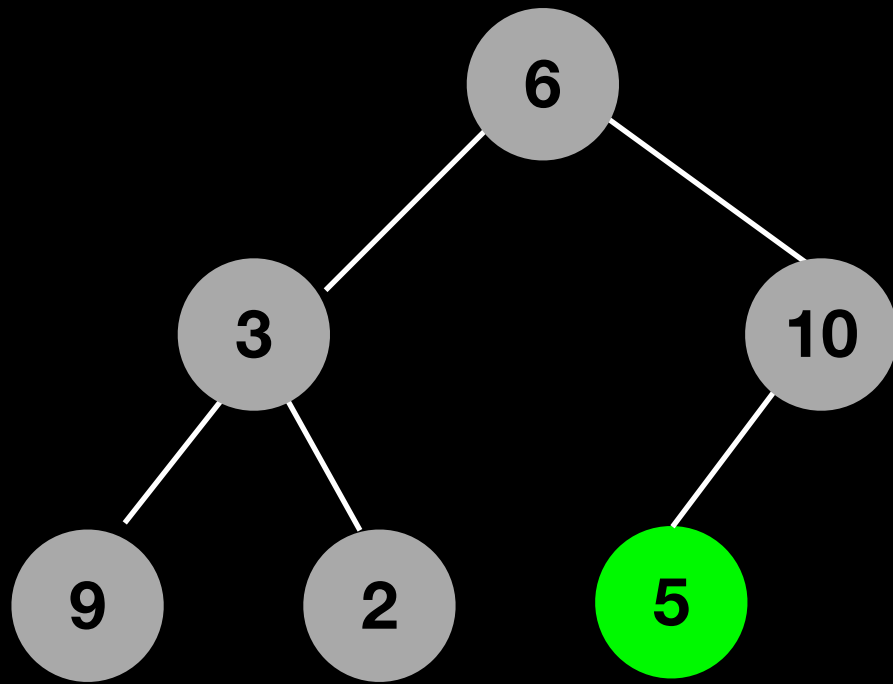
```
for(int i=(itemCount/2)-1; i >=0; i--)  
{  
    heapRebuild(i);  
}
```

heapCreate

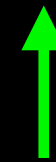


```
for(int i=(itemCount/2)-1; i >=0; i--)  
{  
    heapRebuild(i);  
}
```


heapCreate

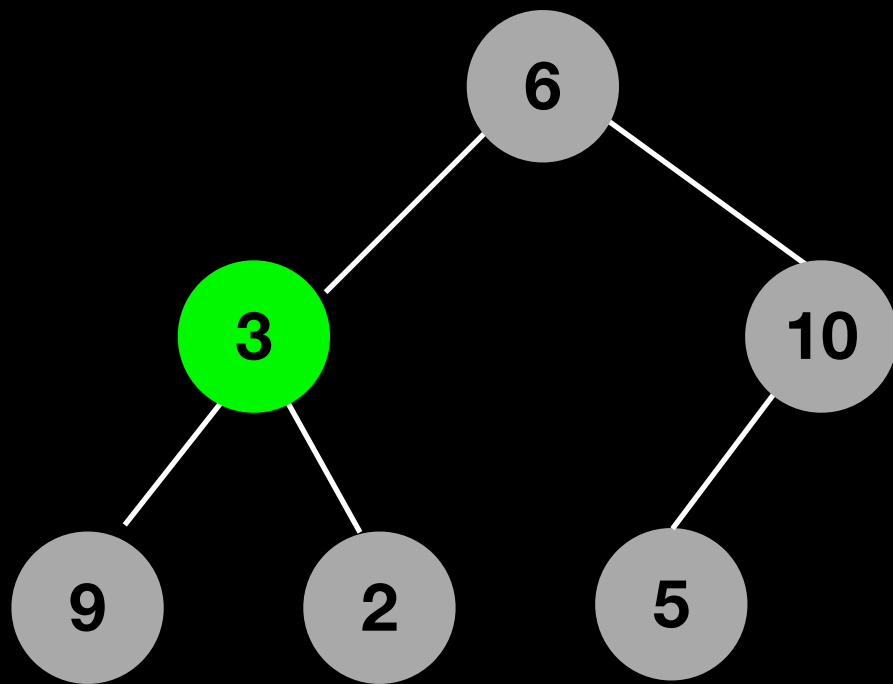


6	3	5	9	2	10
6	3	10	9	2	5



```
for(int i=(itemCount/2)-1; i >=0; i--)  
{  
    heapRebuild(i);  
}
```

heapCreate

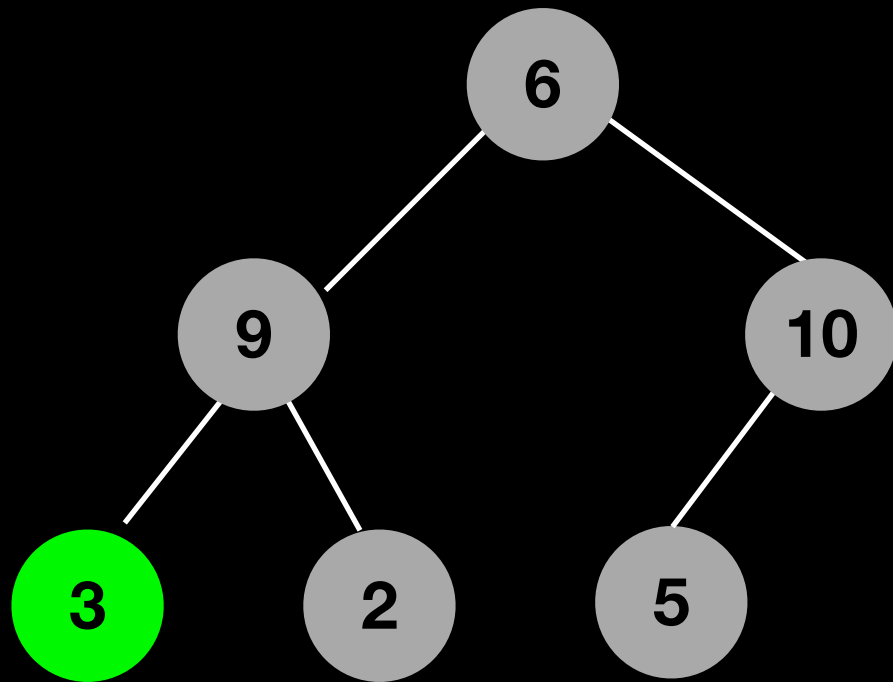


6	3	5	9	2	10
6	3	10	9	2	5

↑

```
for(int i=(itemCount/2)-1; i >=0; i--)  
{  
    heapRebuild(i);  
}
```

heapCreate

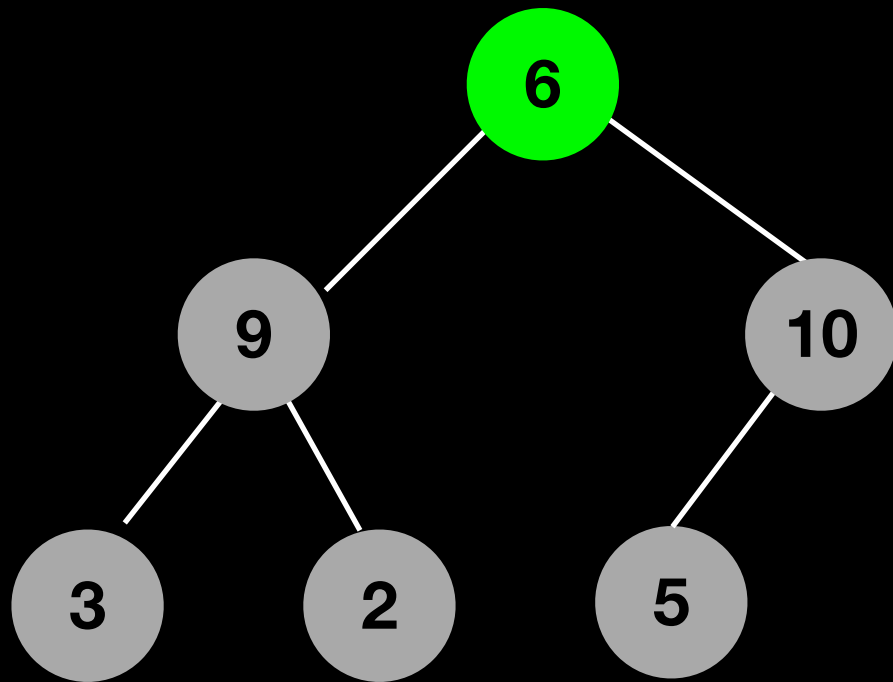


6	3	5	9	2	10
6	3	10	9	2	5
6	9	10	3	2	5

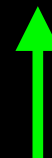


```
for(int i=(itemCount/2)-1; i >=0; i--)  
{  
    heapRebuild(i);  
}
```

heapCreate

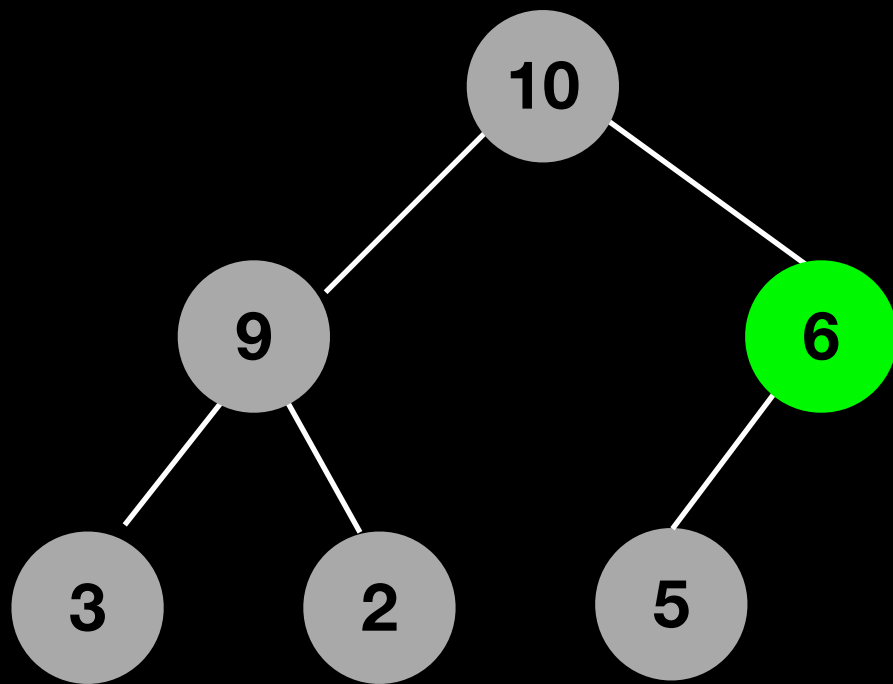


6	3	5	9	2	10
6	3	10	9	2	5
6	9	10	3	2	5

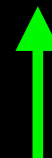


```
for(int i=(itemCount/2)-1; i >=0; i--)  
{  
    heapRebuild(i);  
}
```

heapCreate

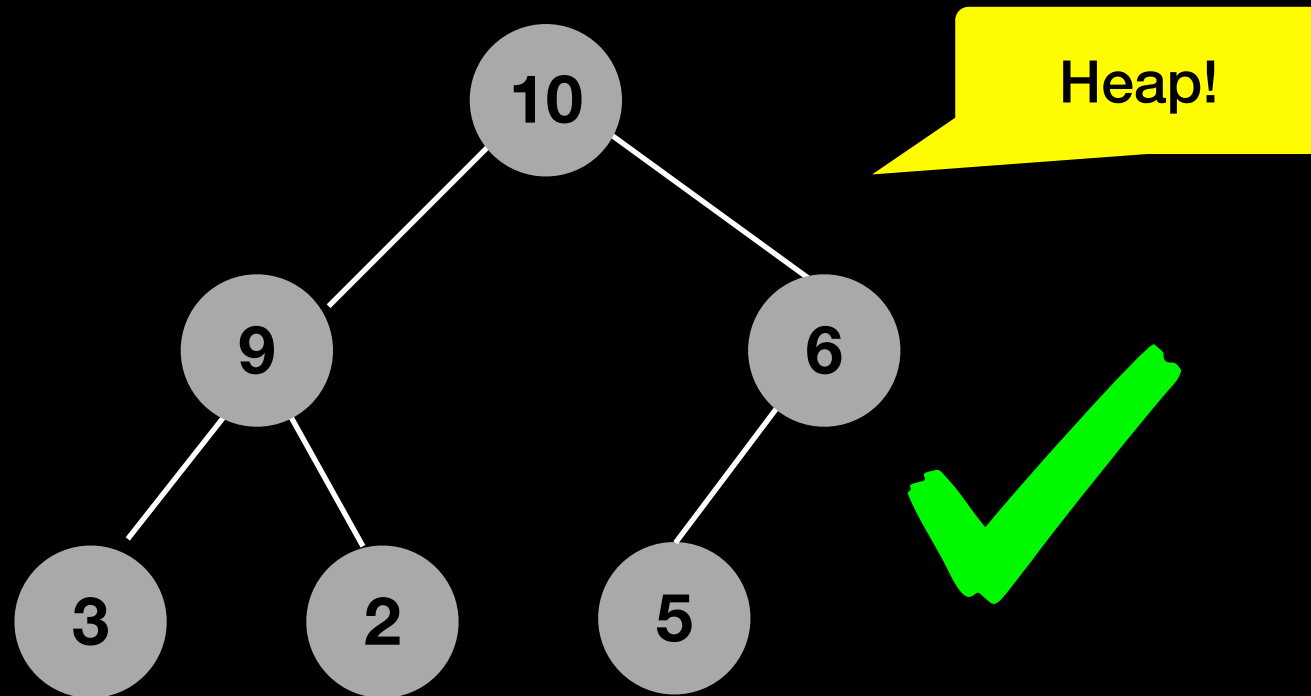


6	3	5	9	2	10
6	3	10	9	2	5
10	9	6	3	2	5



```
for(int i=(itemCount/2)-1; i >=0; i--)  
{  
    heapRebuild(i);  
}
```

heapCreate



6	3	5	9	2	10
6	3	10	9	2	5
10	9	6	3	2	5

$n/2$ heapRebuild

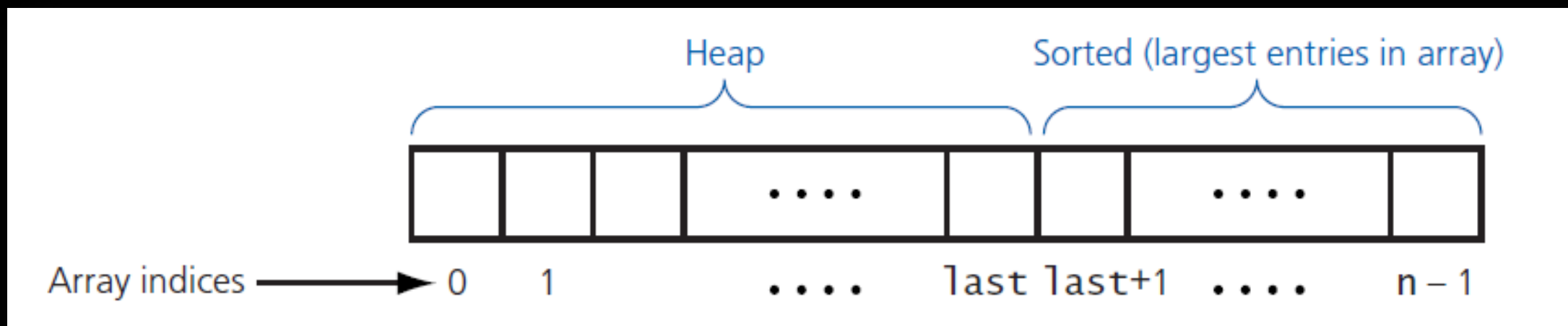
```
for(int i=(itemCount/2)-1; i >=0; i--)  
{  
    heapRebuild(i);  
}
```

Heapsort

Heapsort

Given an unsorted array:

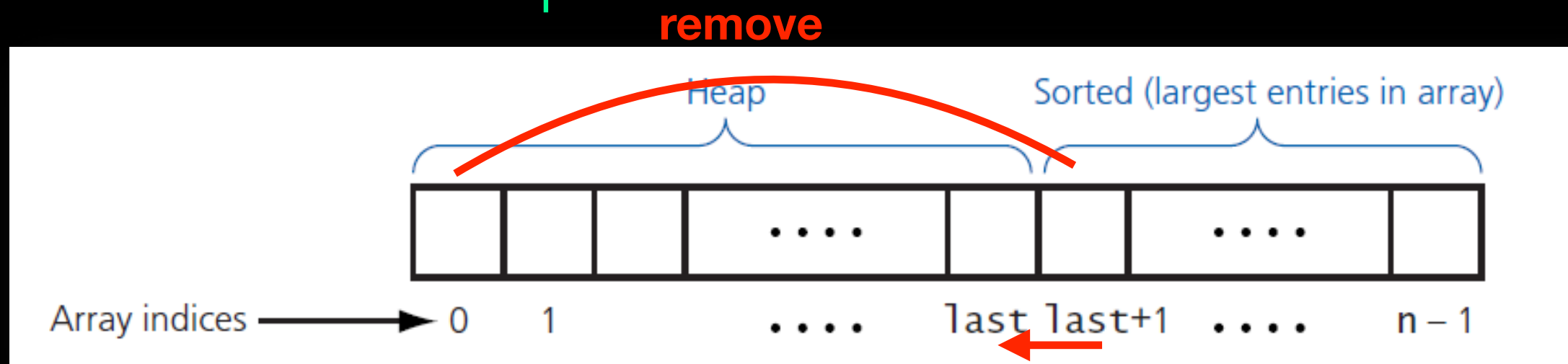
- **heapCreate**
- $\text{last} = n - 2$
- repeat:
 - swap $\text{items}[0]$ with $\text{items}[\text{last}+1]$
 - $\text{last}--$
 - **rebuildHeap**



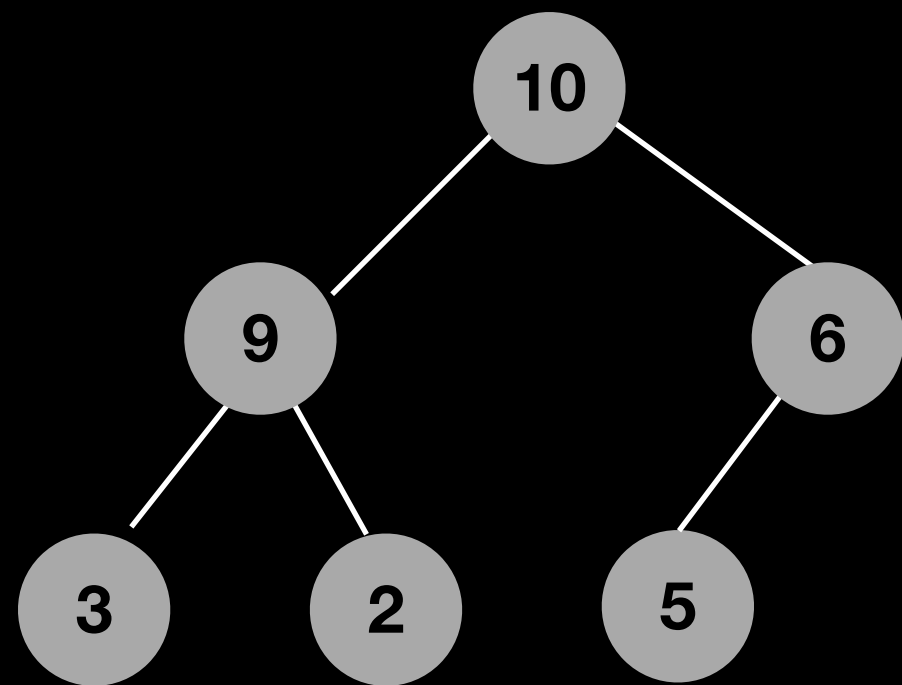
Heapsort

Given an unsorted array:

- **heapCreate**
- $\text{last} = n - 2$
- repeat:
 - swap items[0] with items[last+1]
 - last--
 - **rebuildHeap**

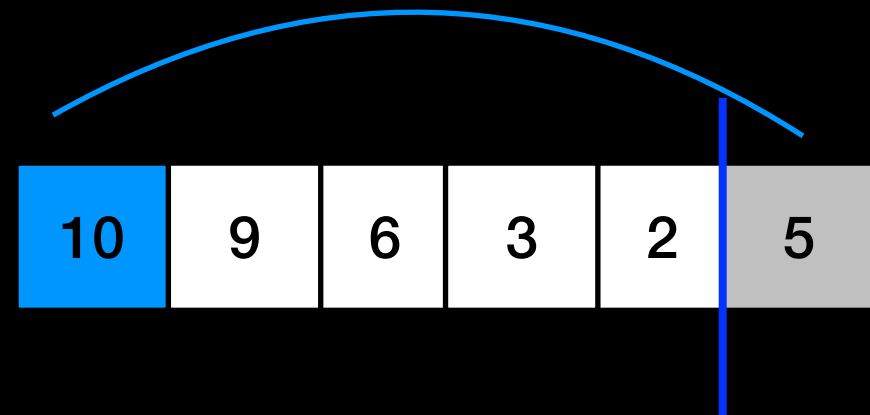
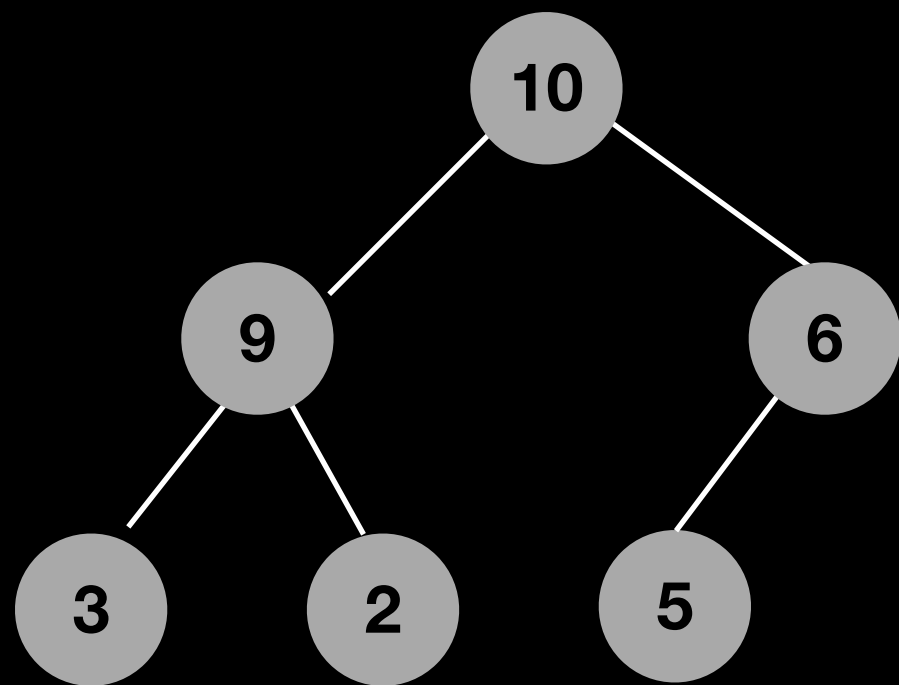


Heapsort

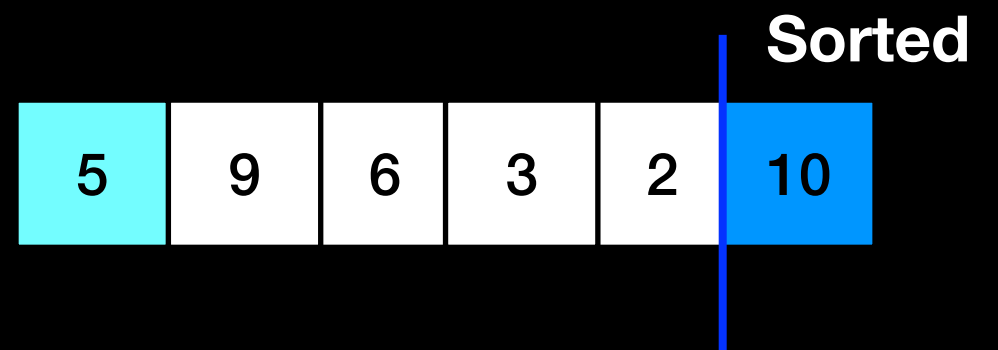
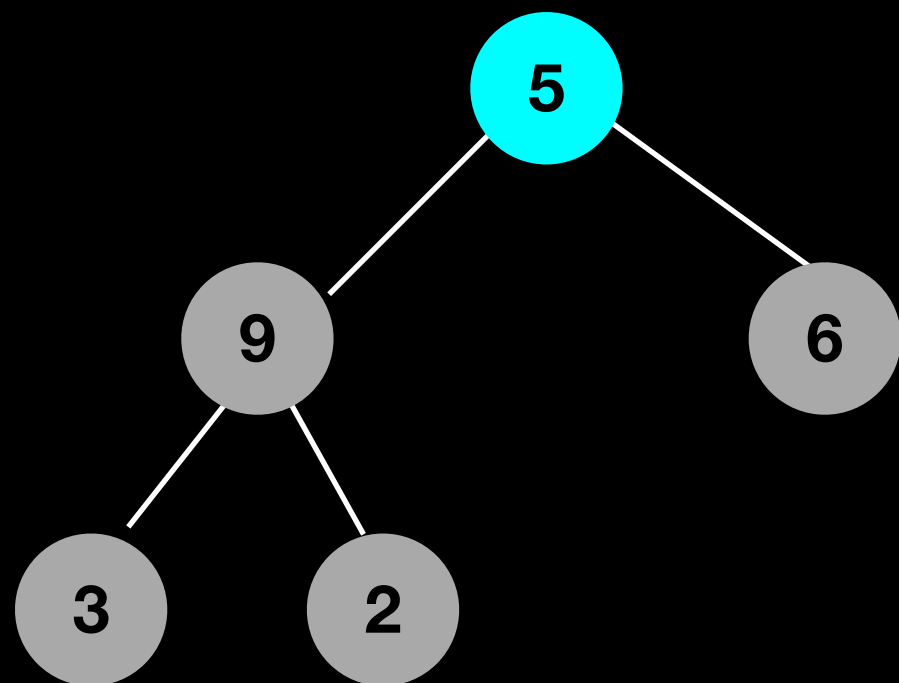


10	9	6	3	2	5
----	---	---	---	---	---

Heapsort

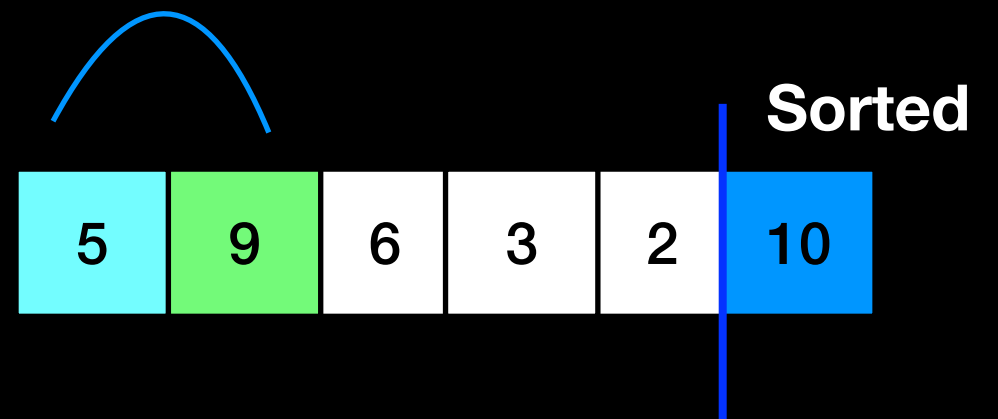
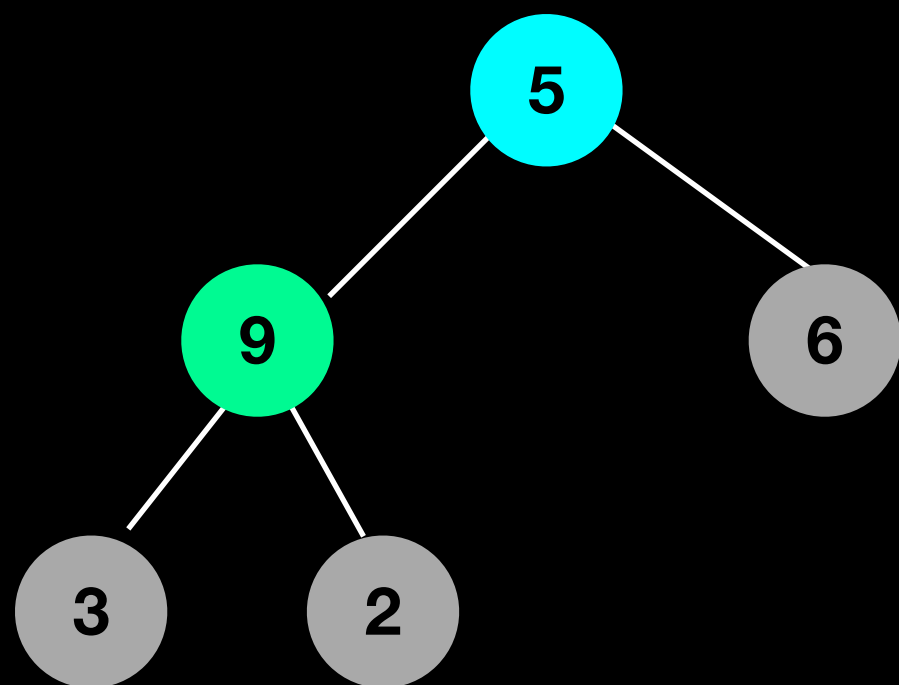


Heapsort

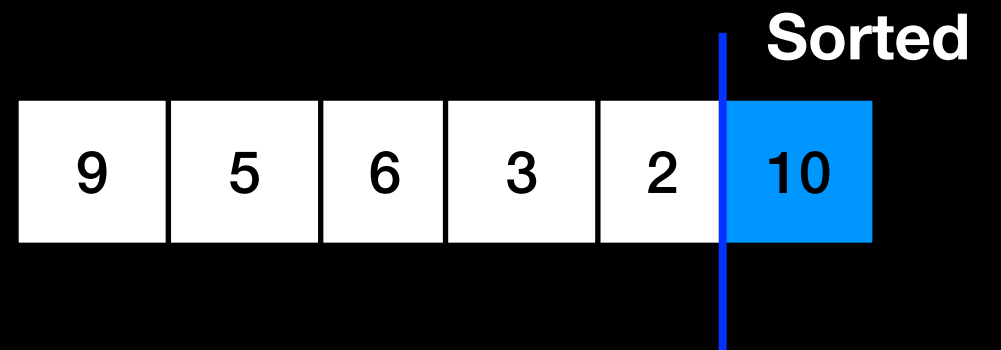
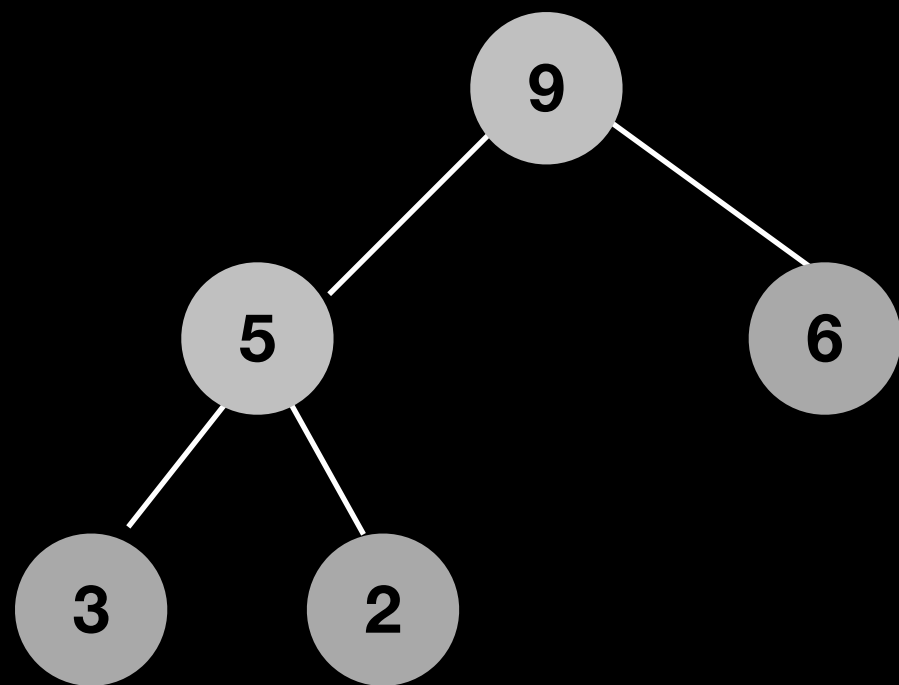


heapRebuild

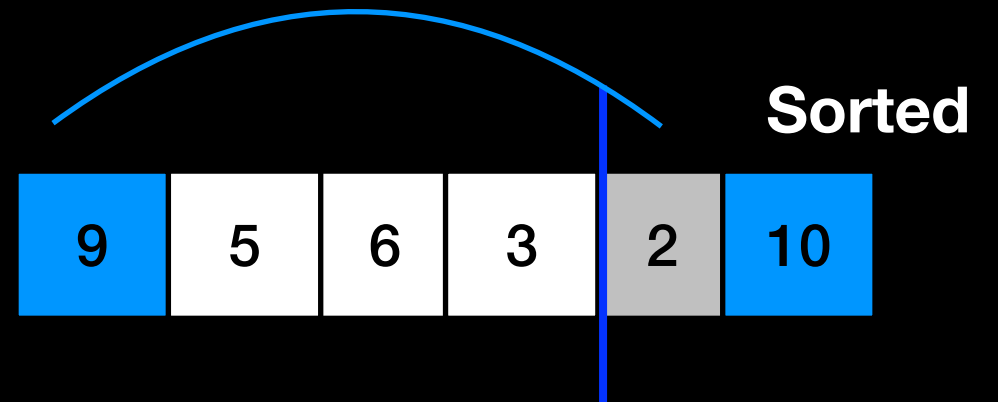
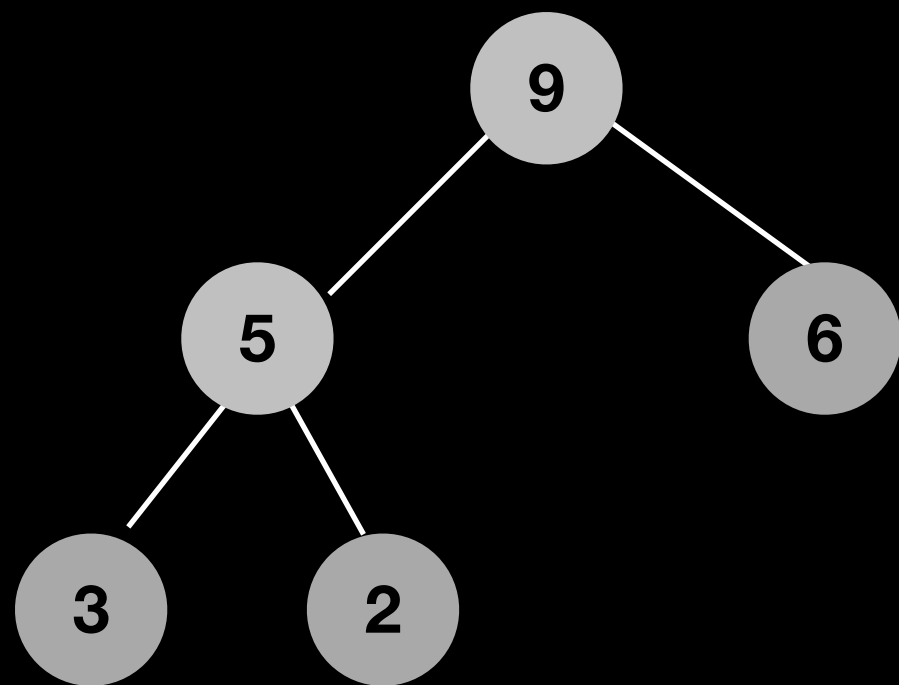
Heapsort



Heapsort

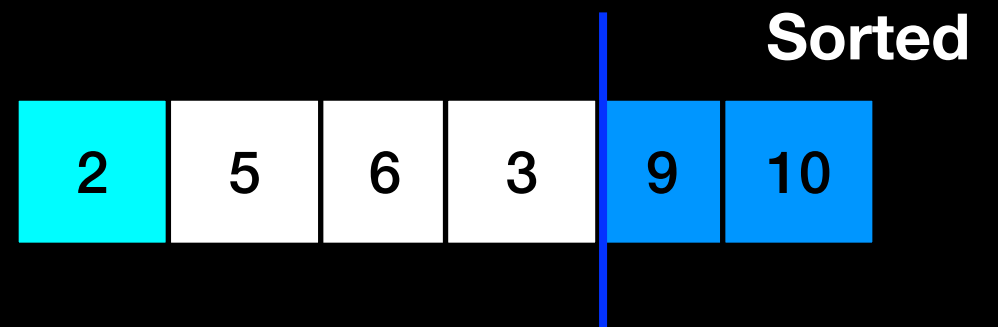
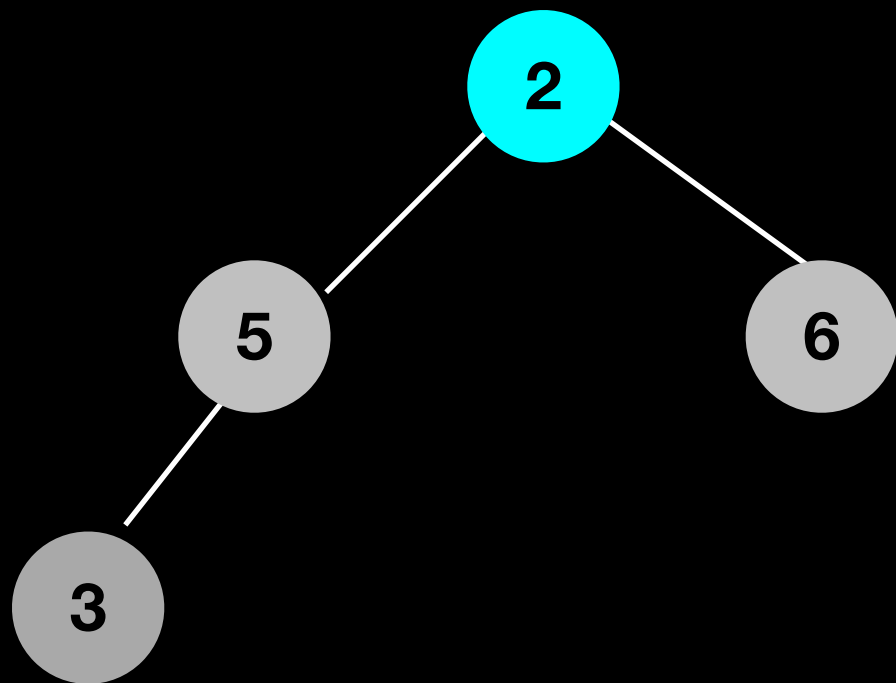


Heapsort



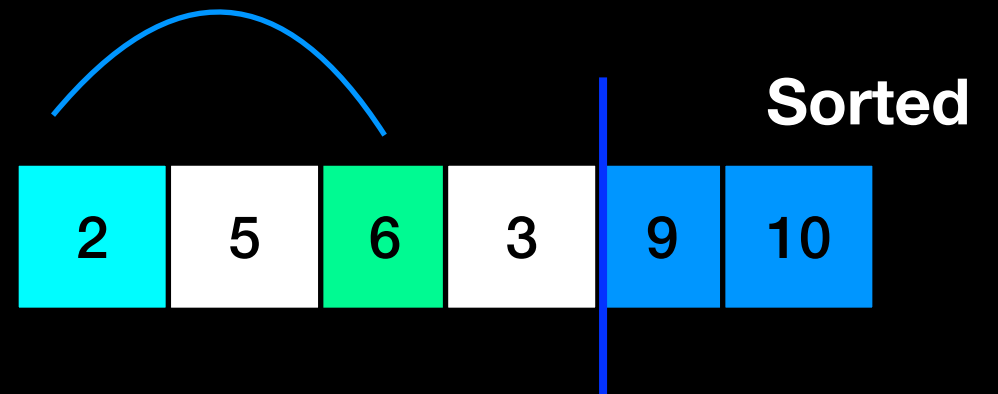
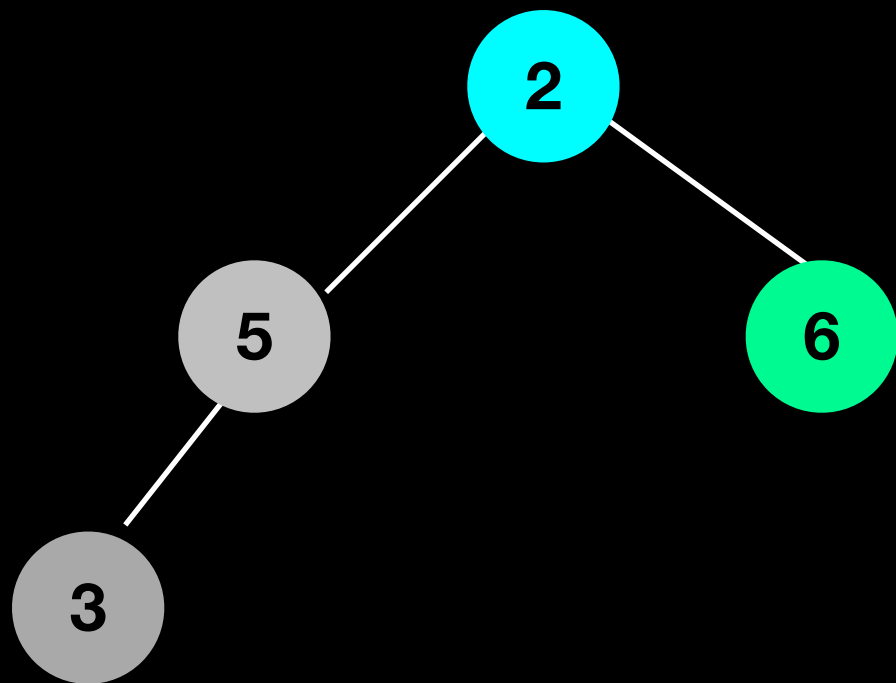
heapRebuild

Heapsort

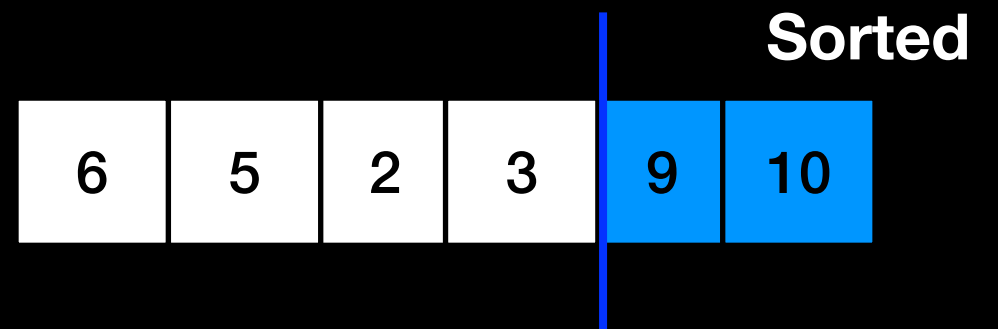
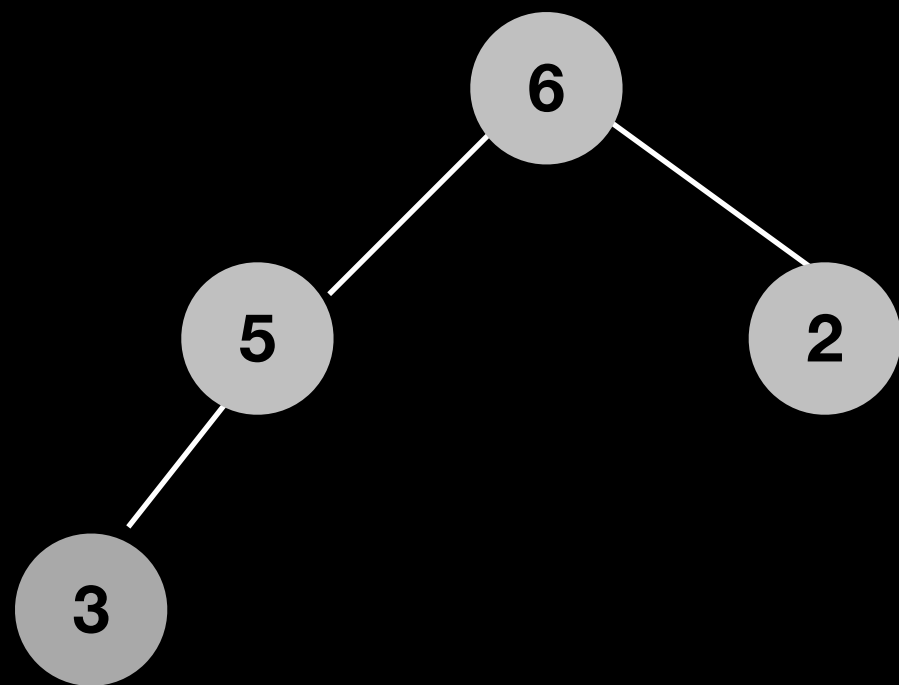


heapRebuild

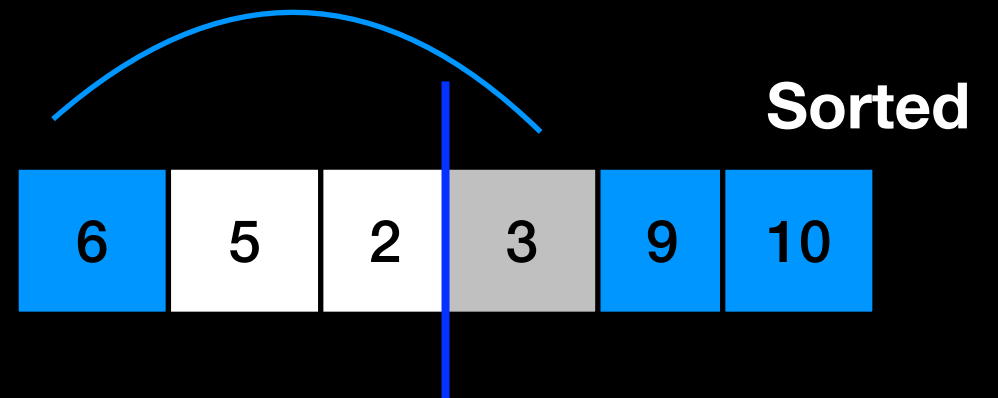
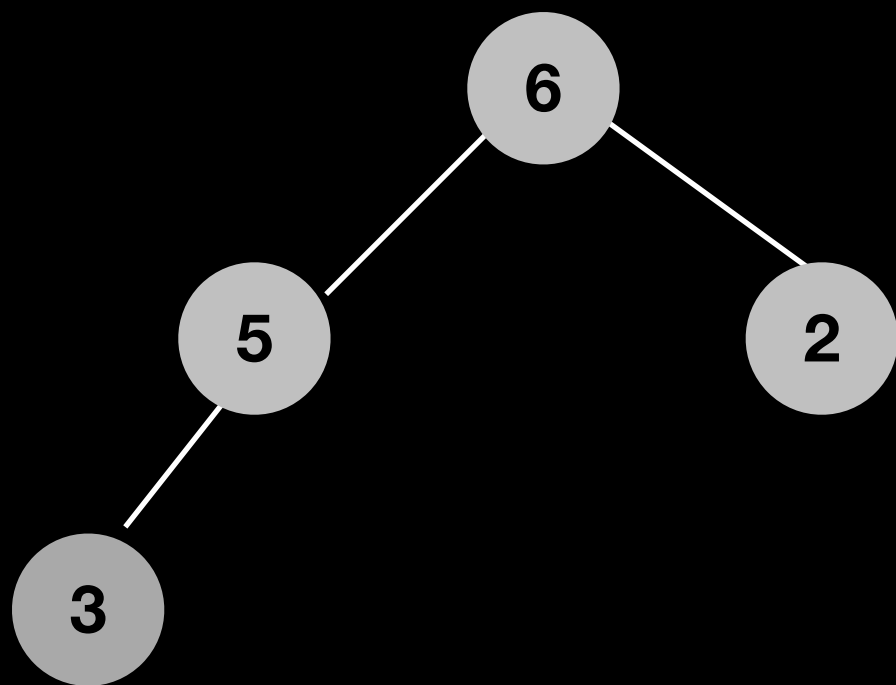
Heapsort



Heapsort

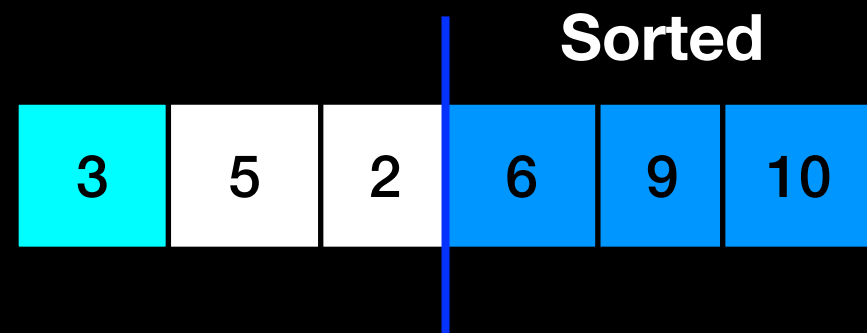
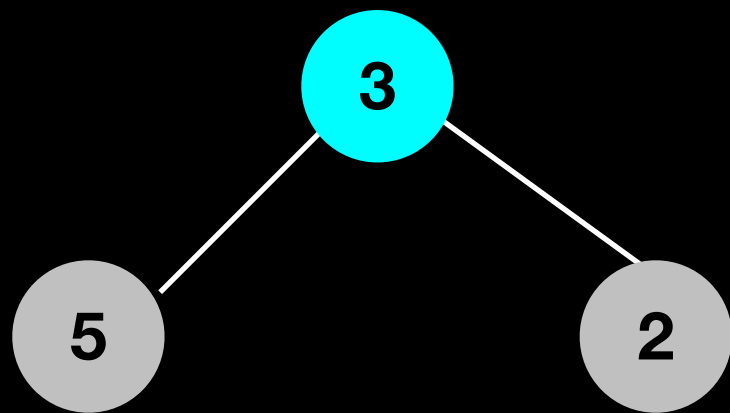


Heapsort



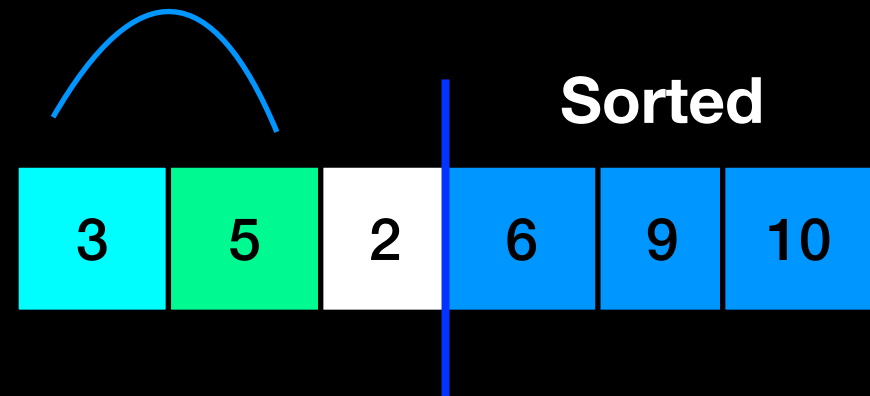
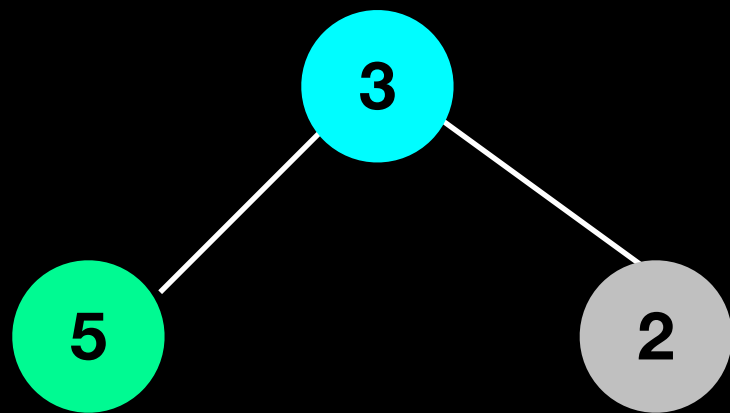
heapRebuild

Heapsort

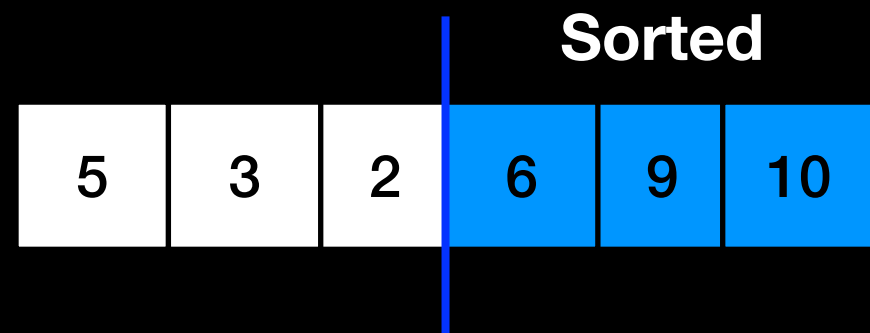
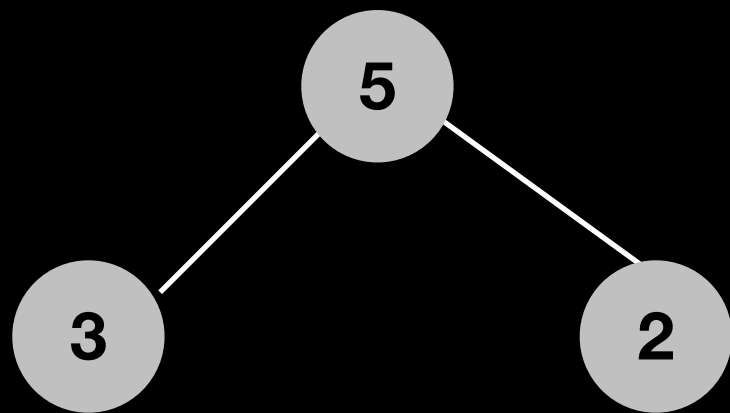


heapRebuild

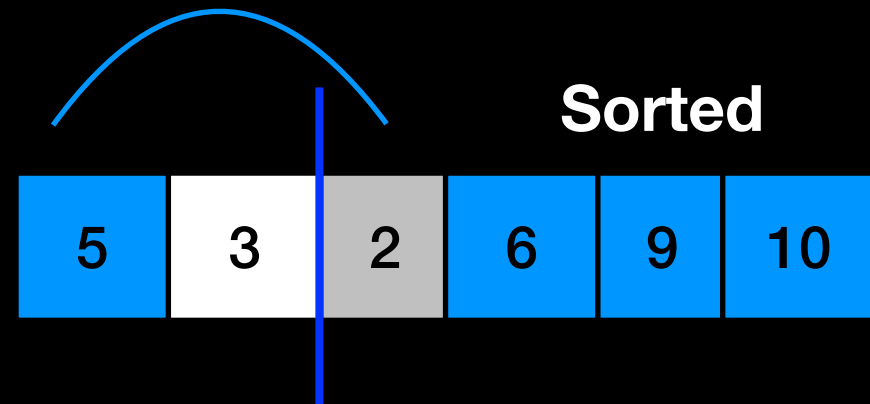
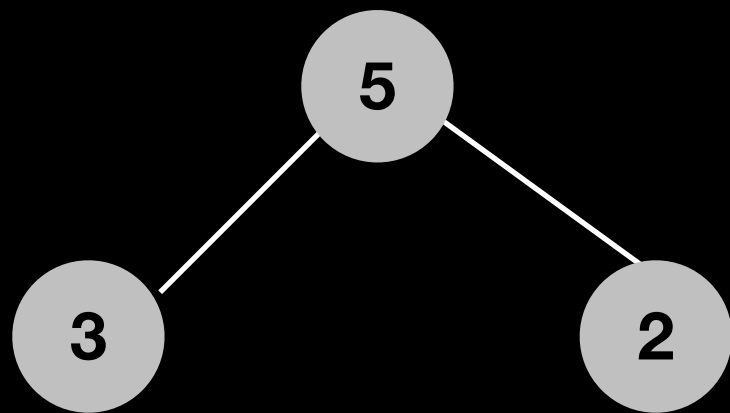
Heapsort



Heapsort

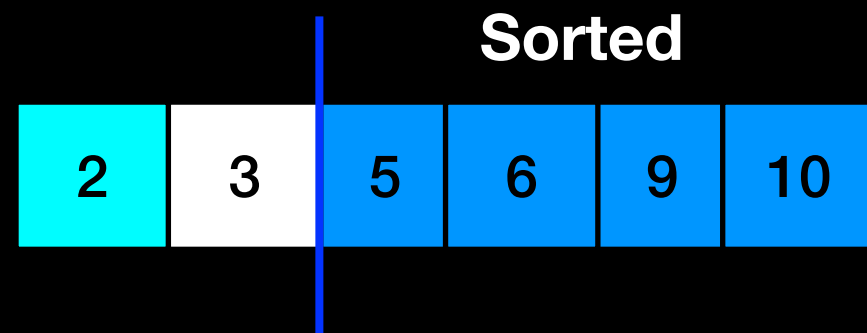
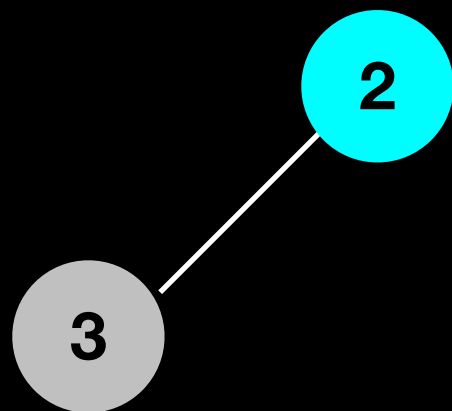


Heapsort



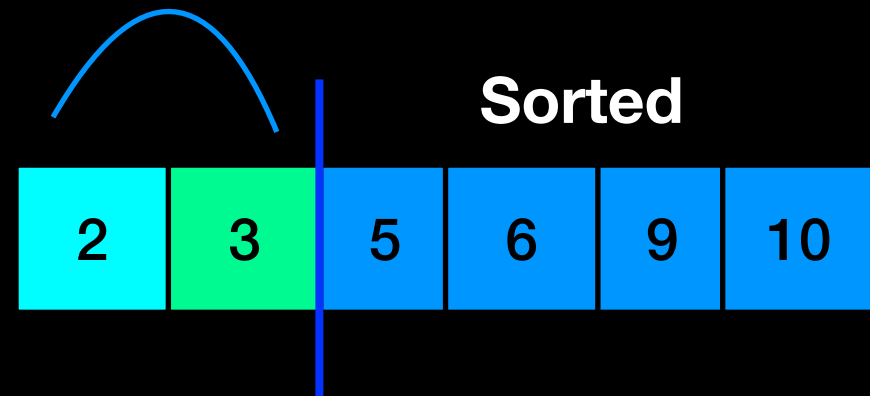
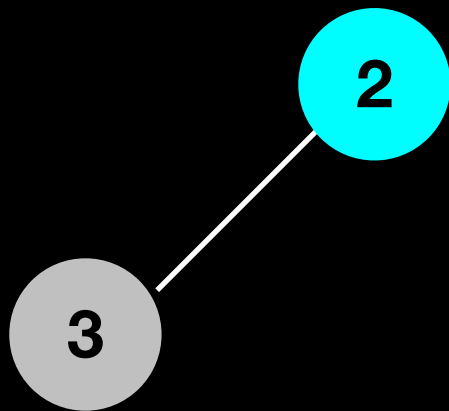
heapRebuild

Heapsort

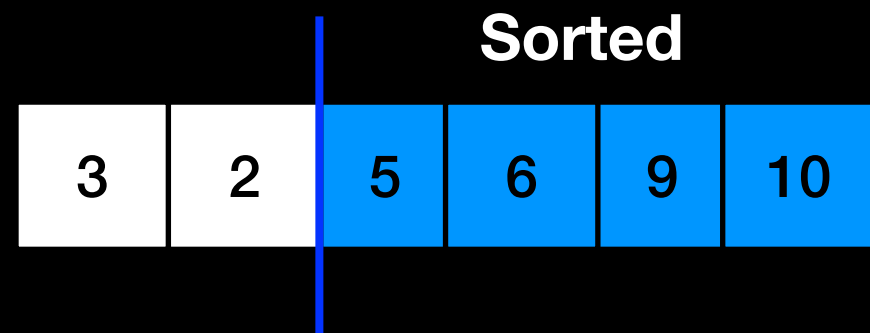
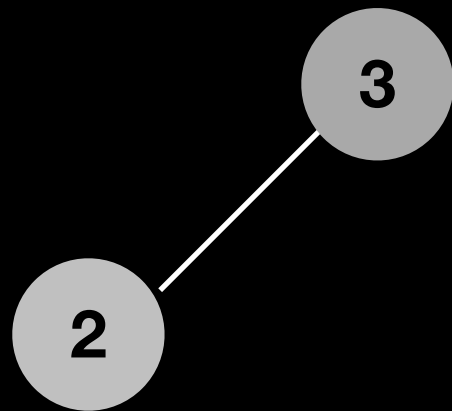


heapRebuild

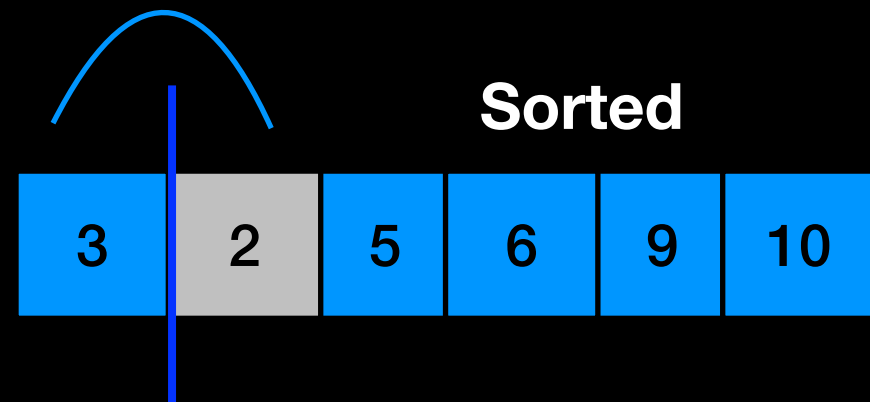
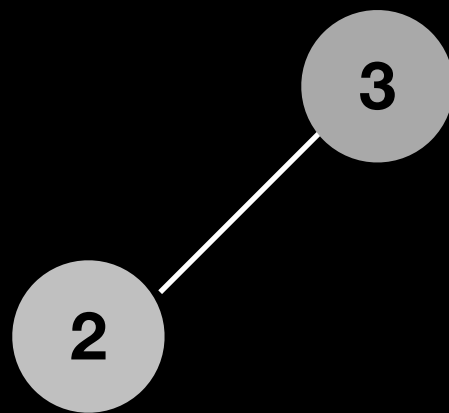
Heapsort



Heapsort

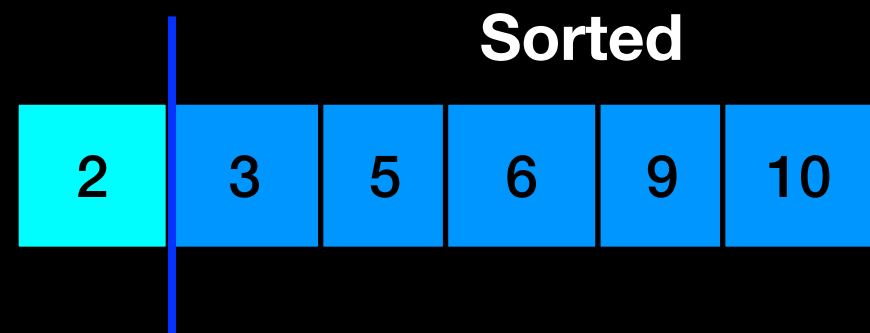


Heapsort



Heapsort

2



Heapsort

Sorted

2	3	5	6	9	10
---	---	---	---	---	----



Heapsort Analysis

1. heapCreate -> $O(n \log n)$

2. heapRebuild -> $O(\log n)$ repeated for each of the n sorted items

$$O(n \log n) + O(n \log n) = O(n \log n)$$

Like MergeSort but no extra space needed!