Heaps and Priority Queues

Robert Horvick
SOFTWARE ENGINEER

@bubbafat www.roberthorvick.com

Overview

Heap Overview

- Min and Max Heaps

Tree as Array

Heap Operations

- Push
- Pop
- Top

Priority Queue

Heap

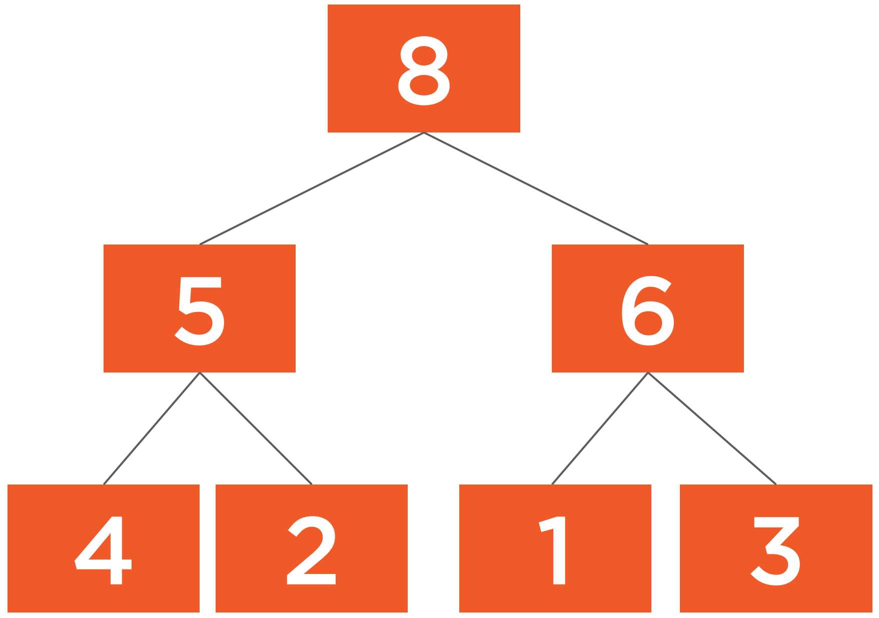
A tree-based container type that provides O(1) access to the minimum (min-heap) or maximum (max-heap) value while satisfying the heap property.

Heap Property

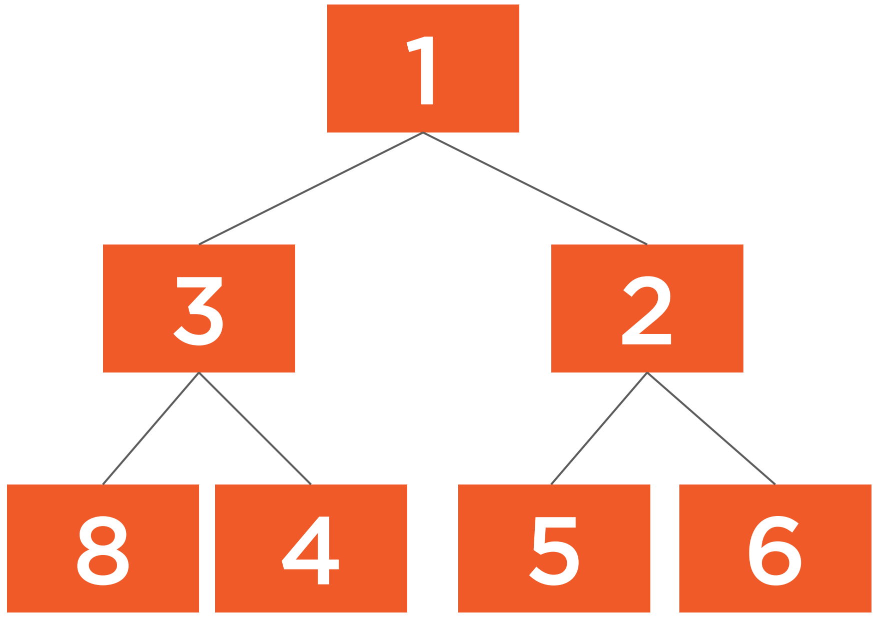
The value in the current tree node is greater than, or equal to, its children (max-heap).

Heaps

Max-heap

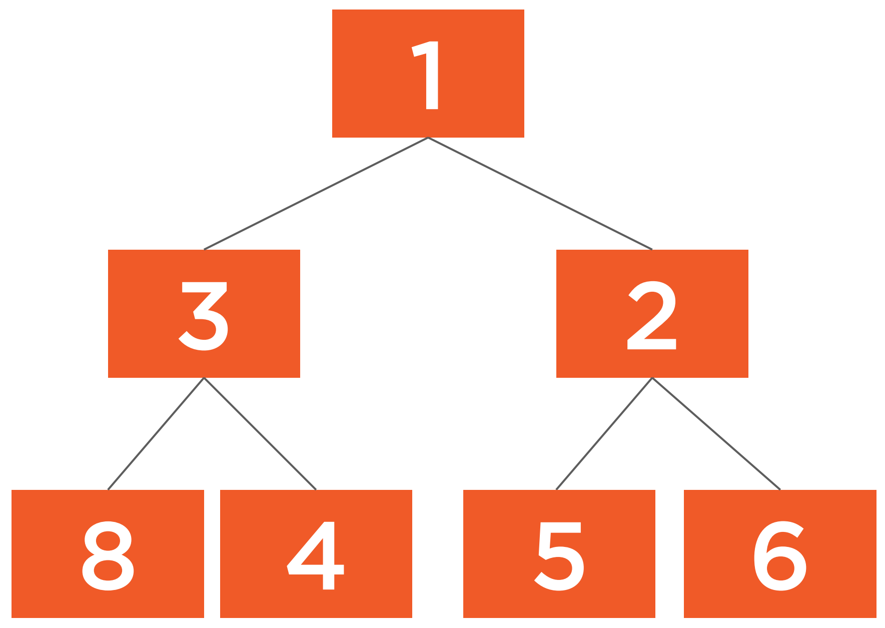


Min-heap



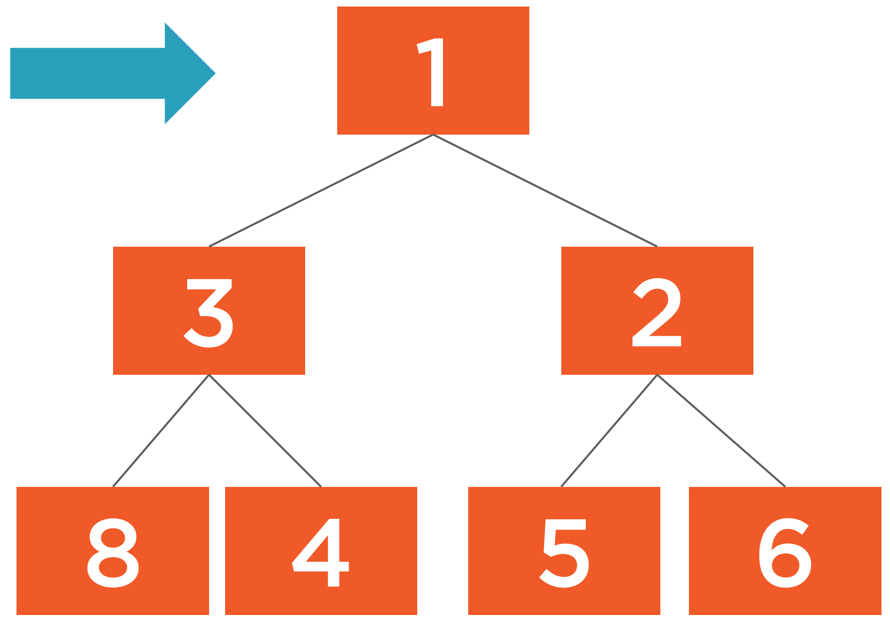
Satisfies the heap property

Satisfies the heap property

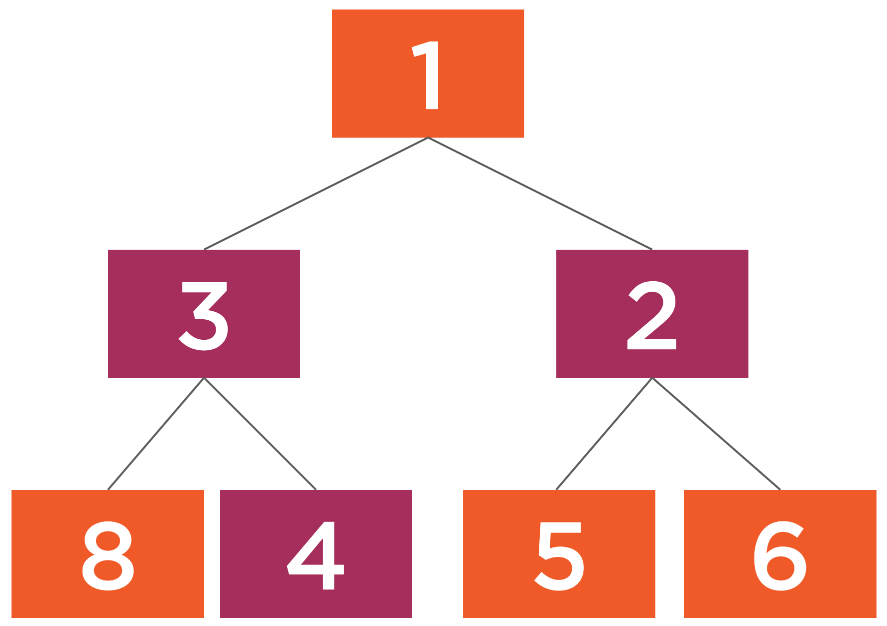


Min-heap

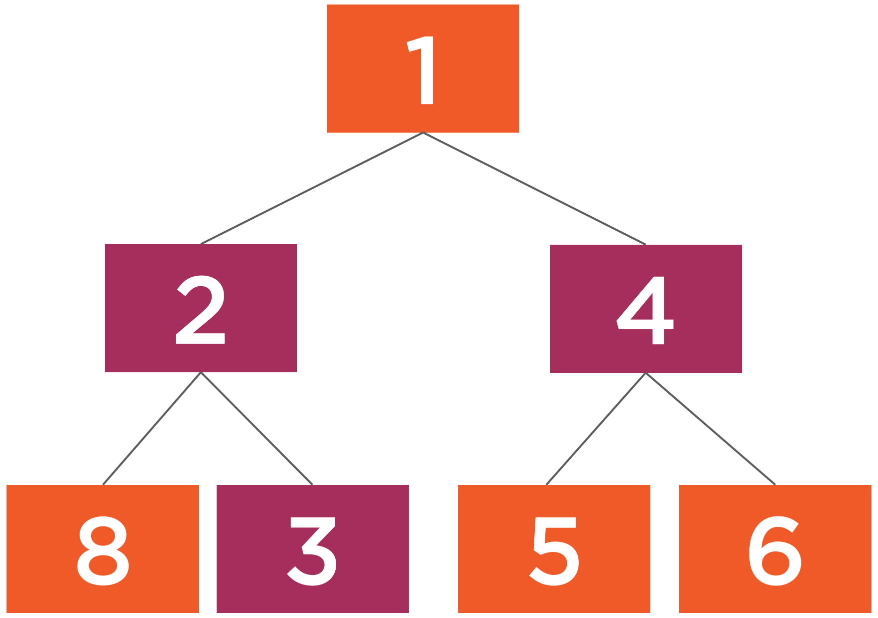
Satisfies the heap property



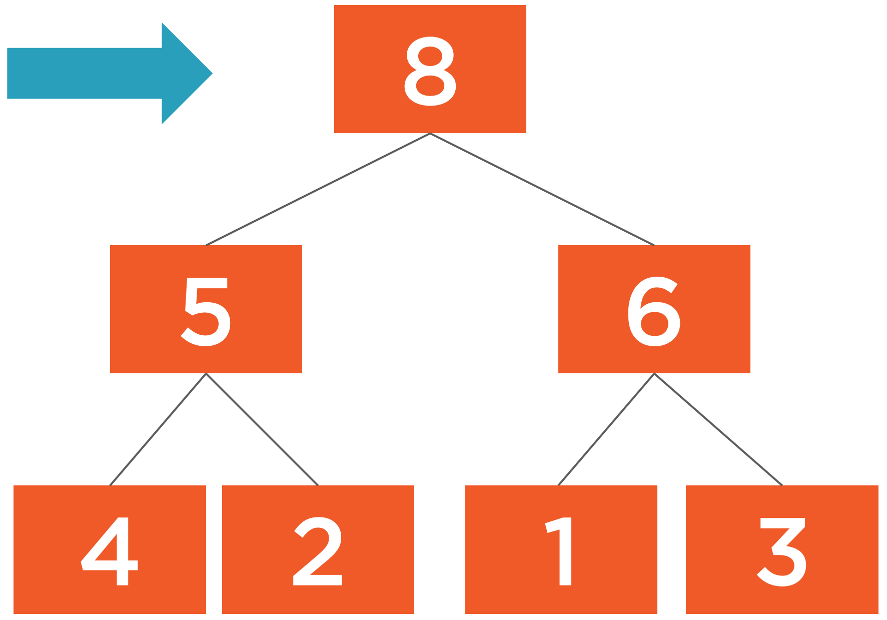
Satisfies the heap property



Satisfies the heap property



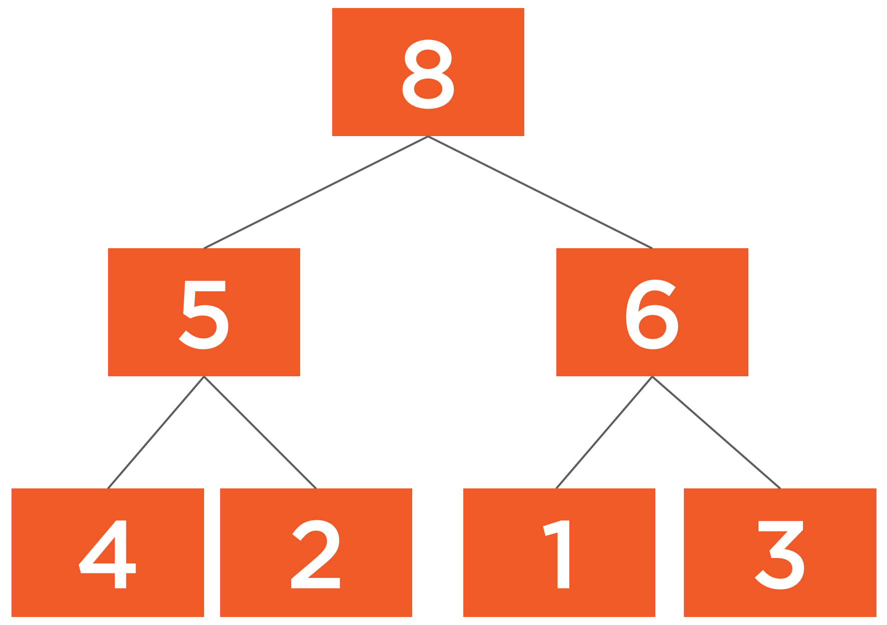
Satisfies the heap property



Max-heap

Satisfies the heap property

The tree is a complete tree



Complete Tree

A tree where every level is filled out from left-to-right be starting the next level.

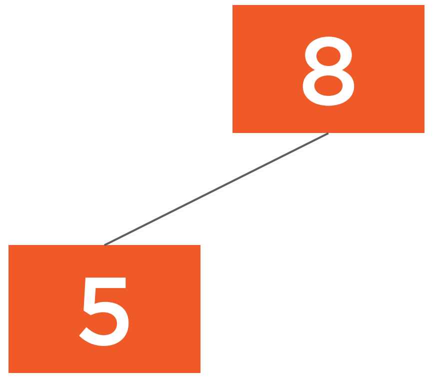
Satisfies the heap property

The tree is a complete tree

Complete tree

Satisfies the heap property

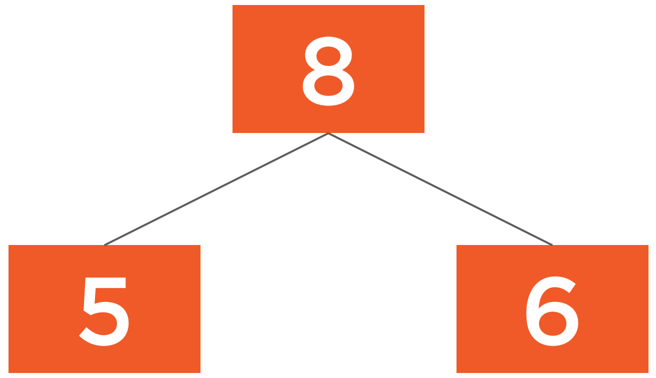
The tree is a complete tree



Complete tree

Satisfies the heap property

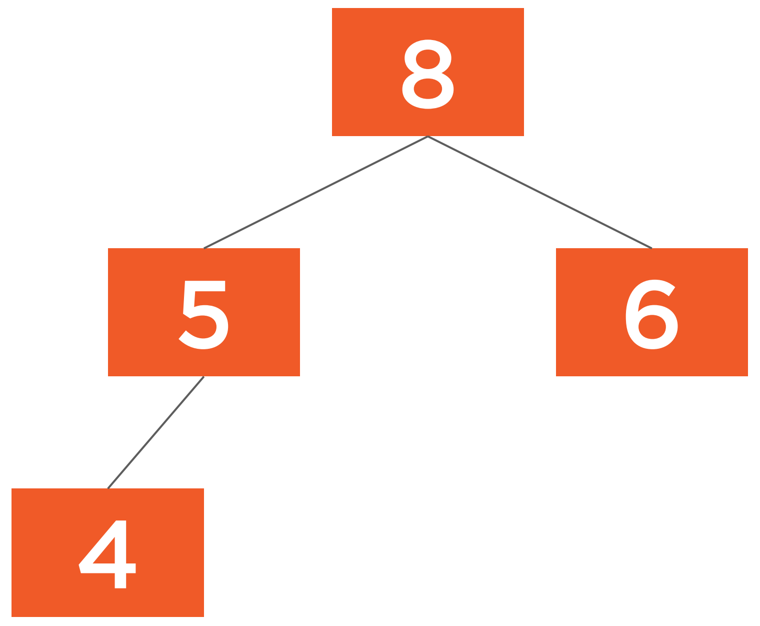
The tree is a complete tree



Complete tree

Satisfies the heap property

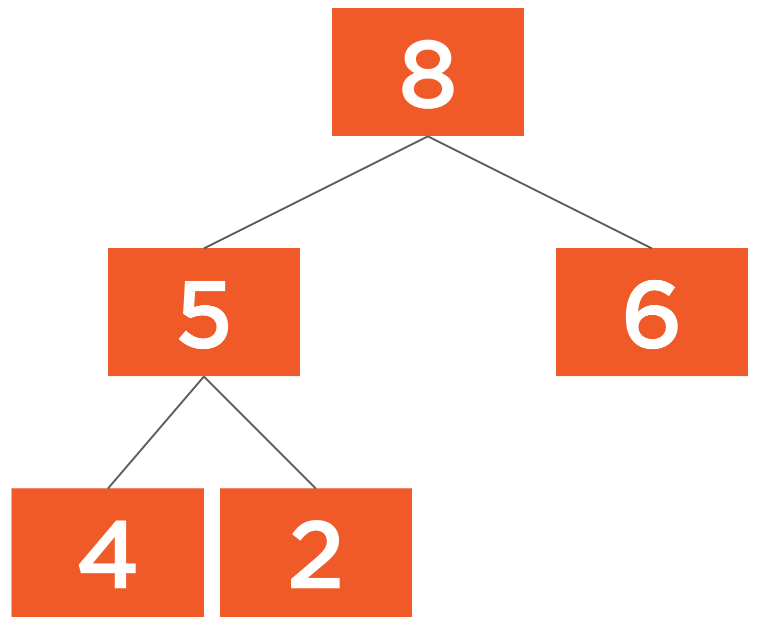
The tree is a complete tree



Complete tree

Satisfies the heap property

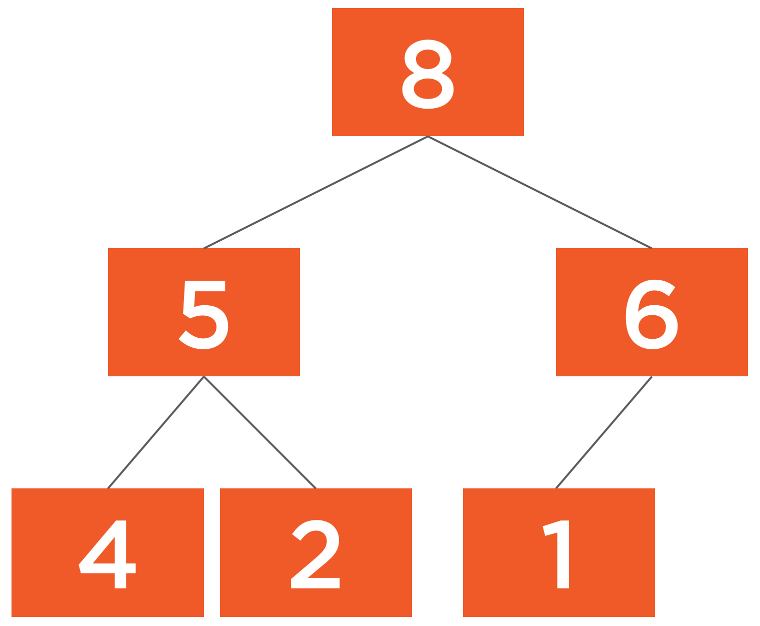
The tree is a complete tree



Complete tree

Satisfies the heap property

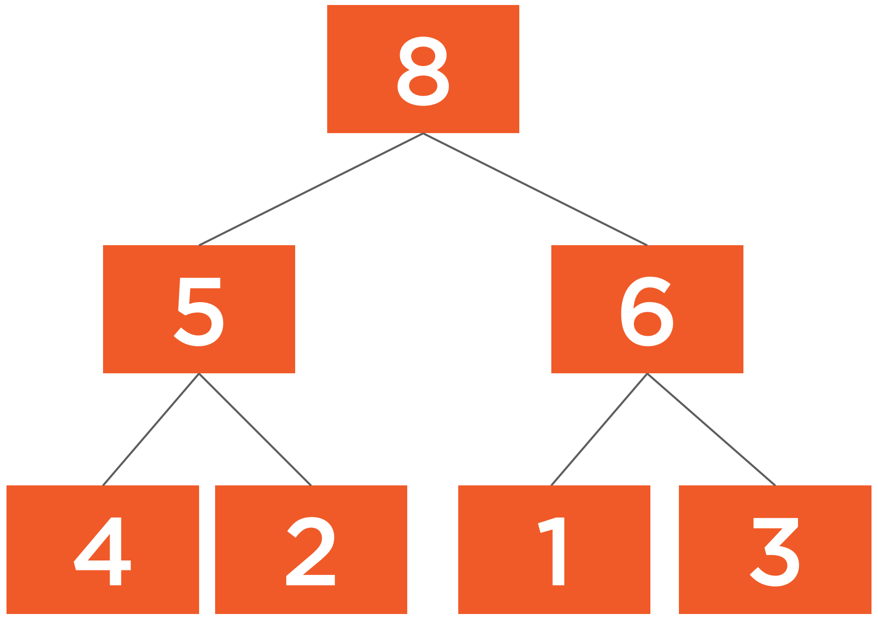
The tree is a complete tree



Complete tree

Satisfies the heap property

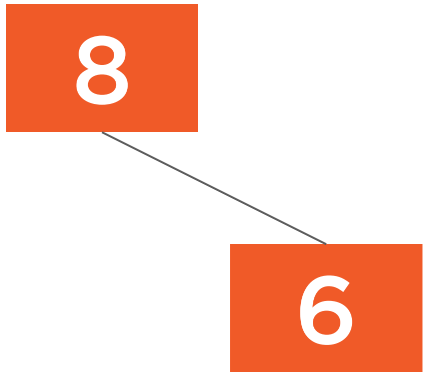
The tree is a complete tree



Complete tree

Satisfies the heap property

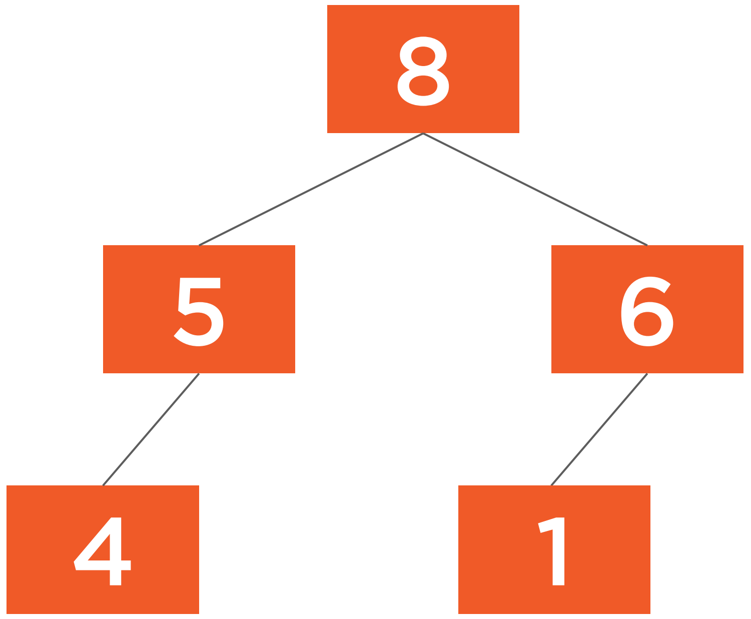
The tree is a complete tree



Incomplete tree

Satisfies the heap property

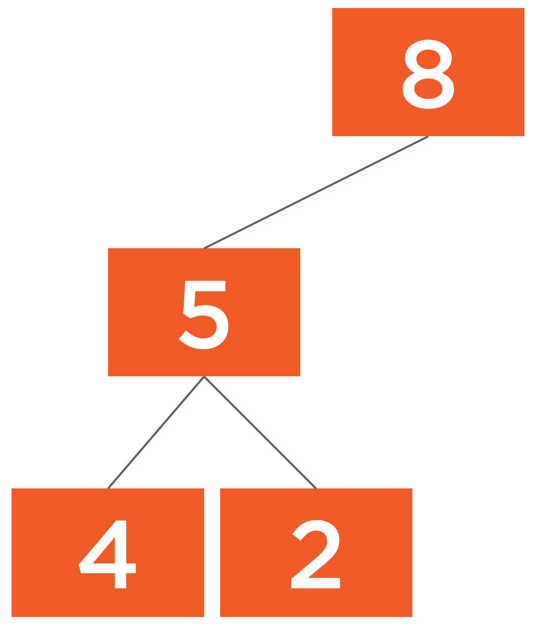
The tree is a complete tree



Incomplete tree

Satisfies the heap property

The tree is a complete tree

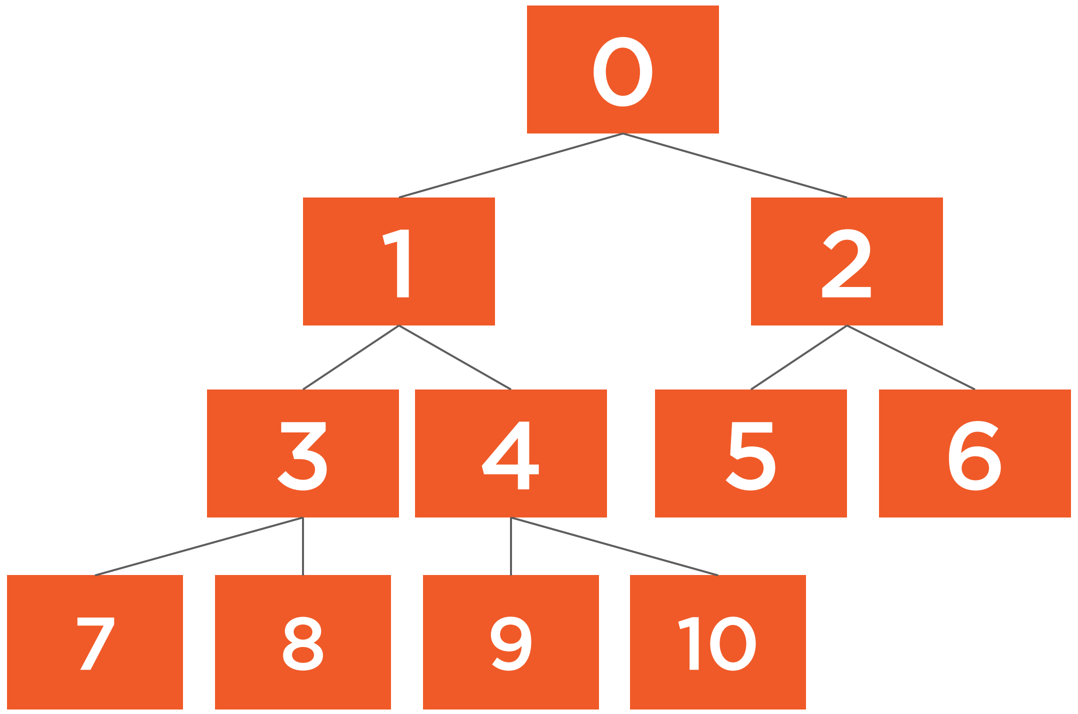


Incomplete tree

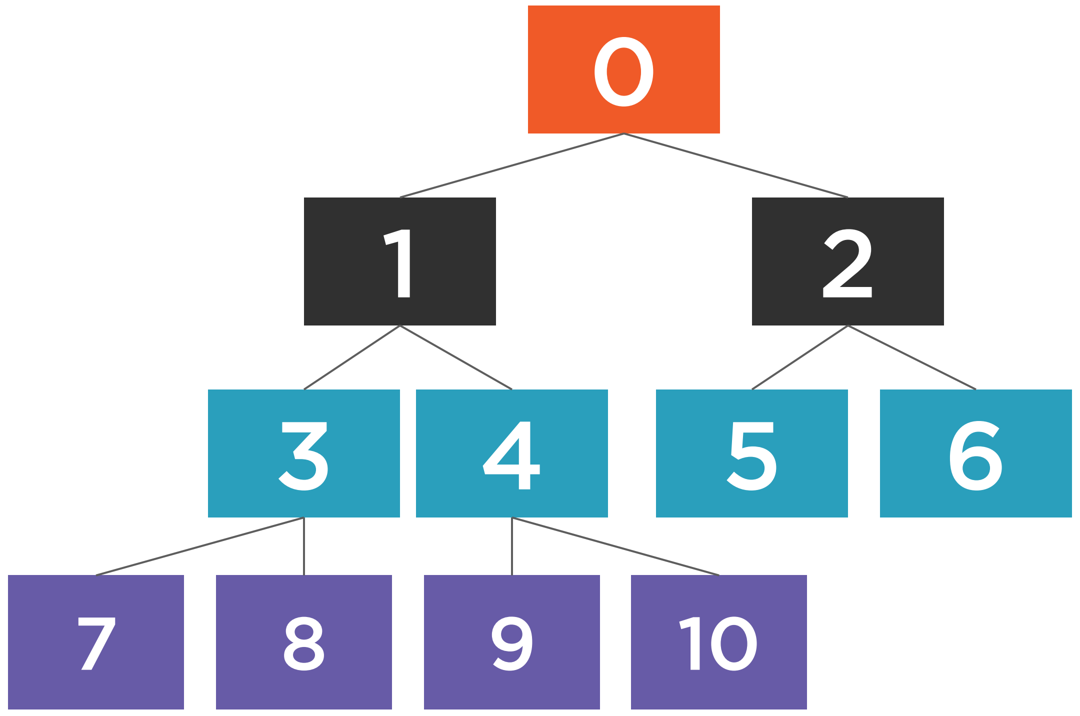
Trees as Arrays

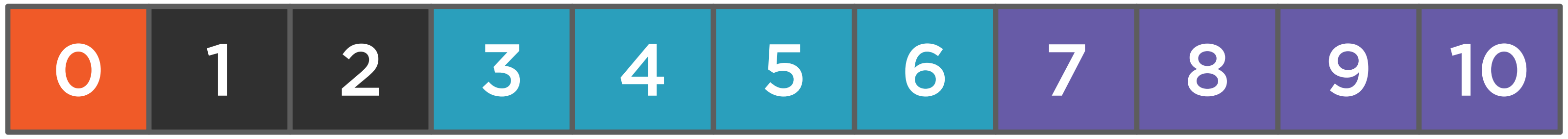
Complete binary trees can be compactly stored as arrays eliminating all structural overhead and providing O(1) data access.

Tree as Array



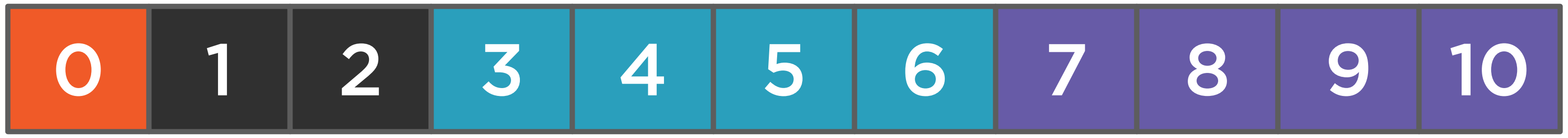
Tree as Array





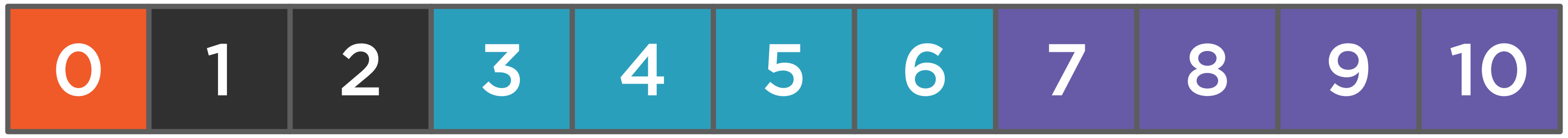
Tree as Array

Index	0	1	2	3	4	5	6	7	8	9	10
Parent	x	0	0	1	1	2	2	3	3	4	4
Left	1	3	5	7	9	11	13	15	17	19	21
Right	2	4	6	8	10	12	14	16	18	20	22



Tree as Array

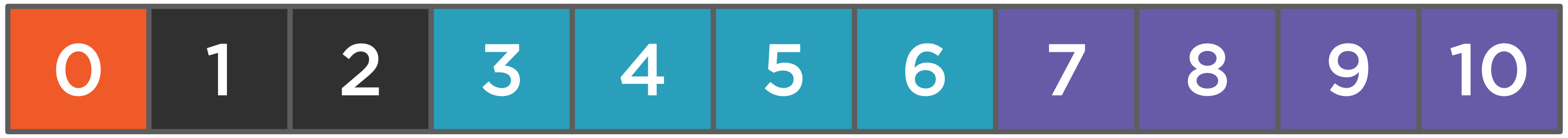
Index	0	1	2	3	4	5	6	7	8	9	10
Parent	x	0	0	1	1	2	2	3	3	4	4
Left	1	3	5	7	9	11	13	15	17	19	21
Right	2	4	6	8	10	12	14	16	18	20	22



Tree as Array

Index	0	1	2	3	4	5	6	7	8	9	10
Parent	x	0	0	1	1	2	2	3	3	4	4
Left	1	3	5	7	9	11	13	15	17	19	21
Right	2	4	6	8	10	12	14	16	18	20	22

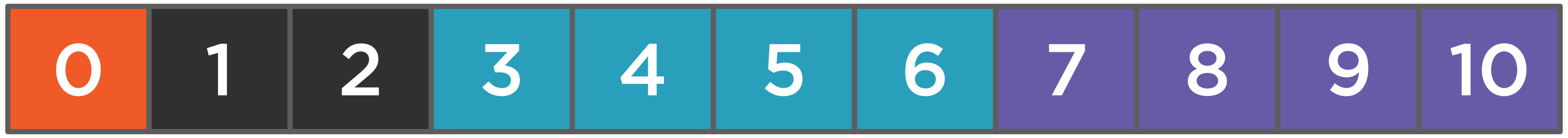
```
int parent(const int index) {
    return (index - 1) / 2;
}
```



Tree as Array

Index	0	1	2	3	4	5	6	7	8	9	10
Parent	x	0	0	1	1	2	2	3	3	4	4
Left	1	3	5	7	9	11	13	15	17	19	21
Right	2	4	6	8	10	12	14	16	18	20	22

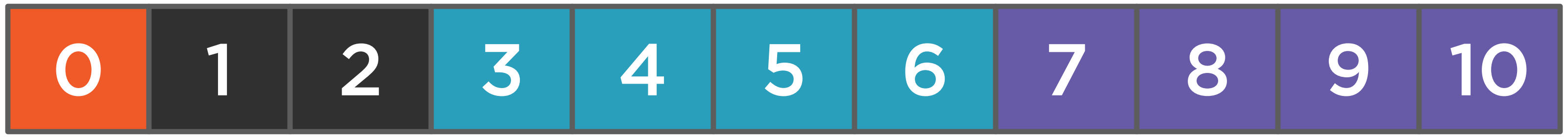
```
int left(const int index) {
    return 2 * index + 1;
}
```



Tree as Array

Index	0	1	2	3	4	5	6	7	8	9	10
Parent	x	0	0	1	1	2	2	3	3	4	4
Left	1	3	5	7	9	11	13	15	17	19	21
Right	2	4	6	8	10	12	14	16	18	20	22

```
int right(const int index) {
    return 2 * index + 2;
}
```



Heap Operations

Heap Operations

Adding values (push)

Retrieving min or max value (top)

Removing min or max value (pop)

Push

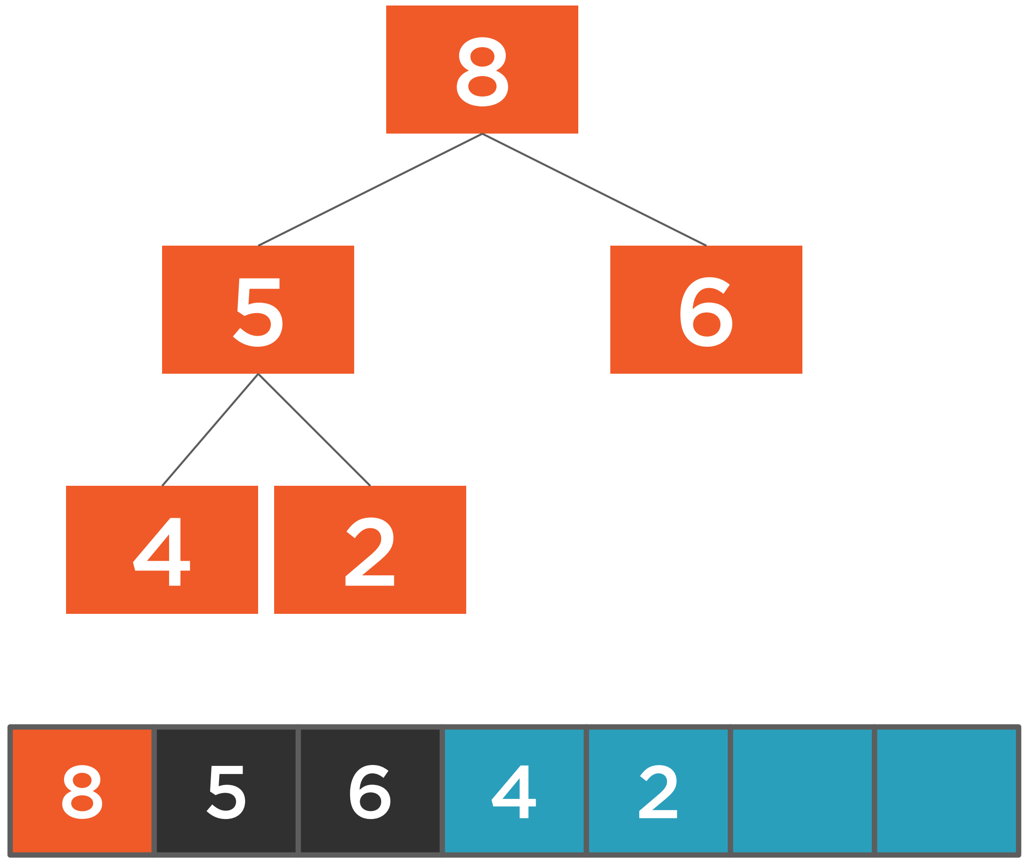
Adds an item to the heap, placing it in the first valid position that retains the tree rules.

Push Operations

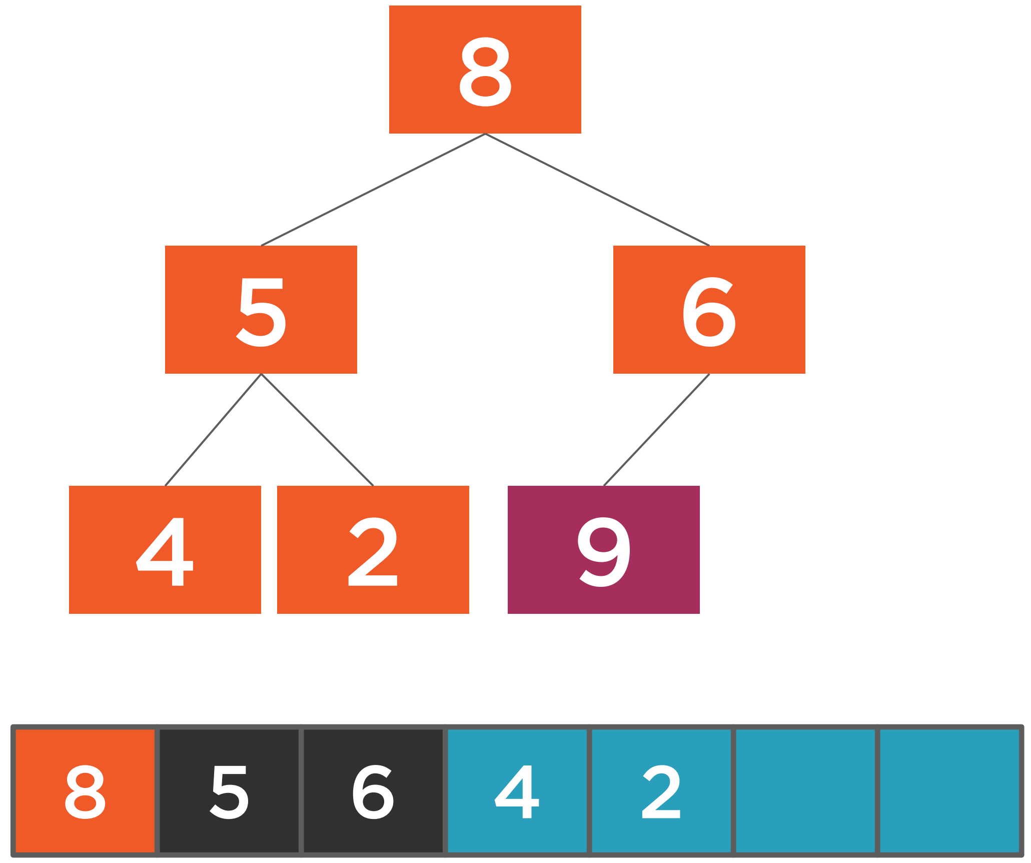
Add the new value to the end of the heap array

While the heap property is not satisfied, swap the new item with its parent

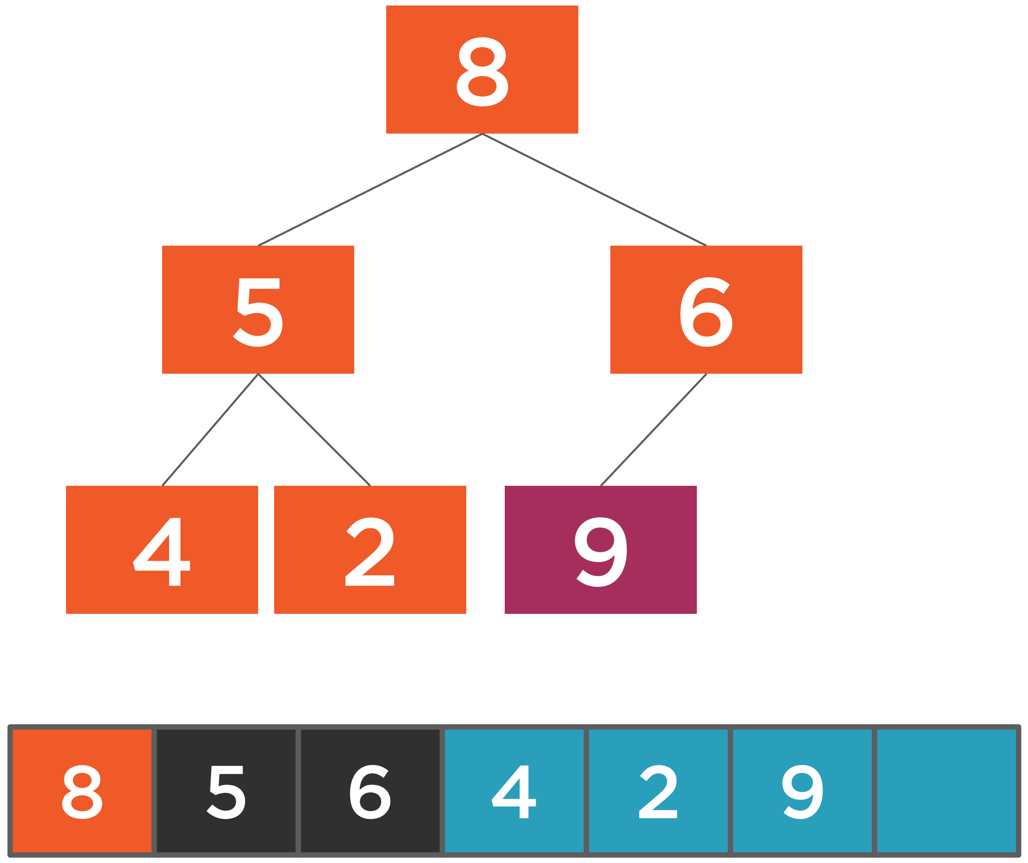
Add new value to end of heap



Add new value to end of heap

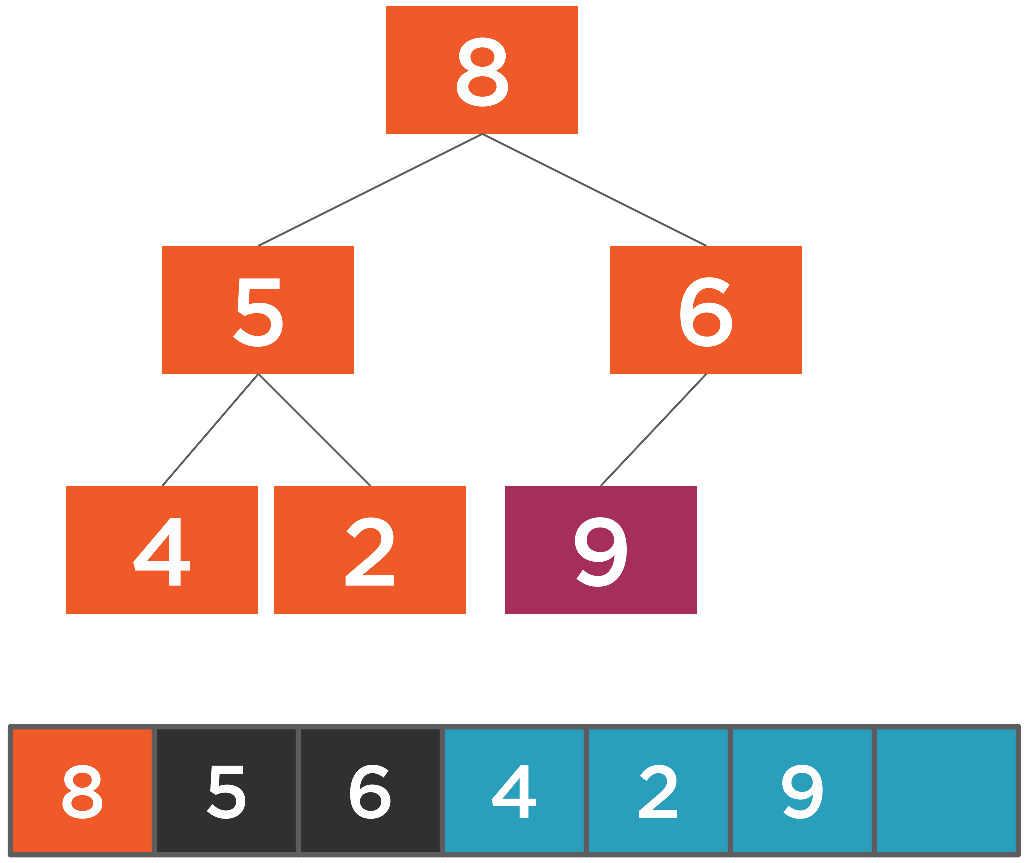


Add new value to end of heap



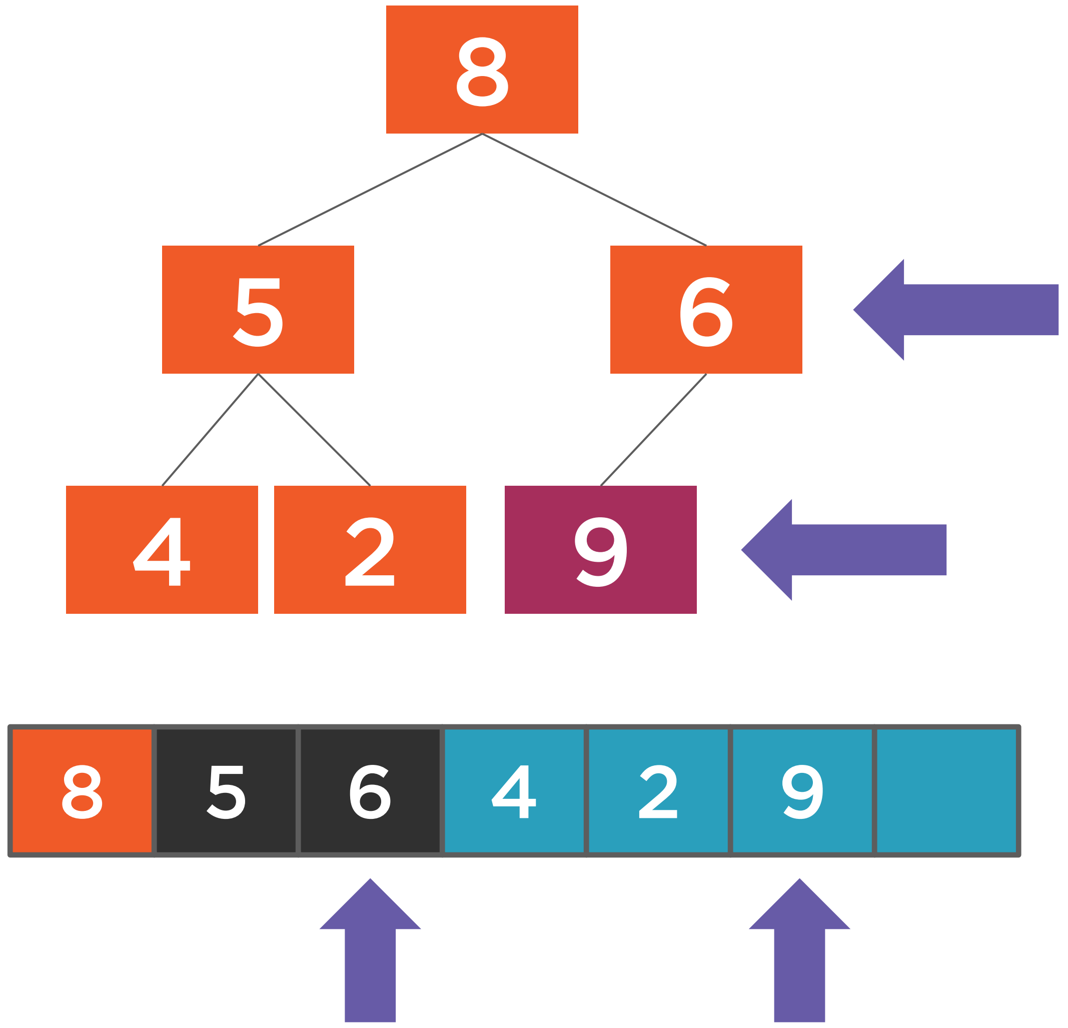
Add new value to end of heap

While the heap property is not satisfied, swap with parent



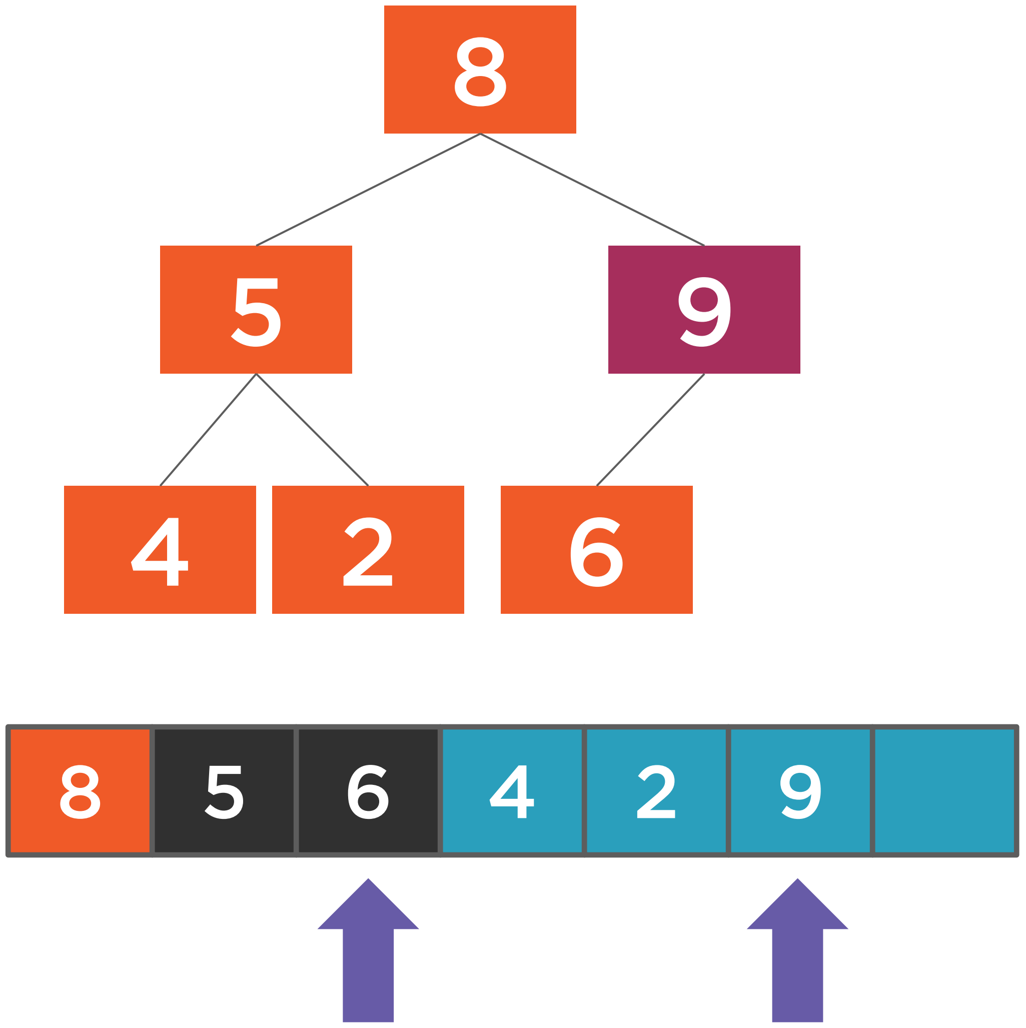
Add new value to end of heap

While the heap property is not satisfied, swap with parent



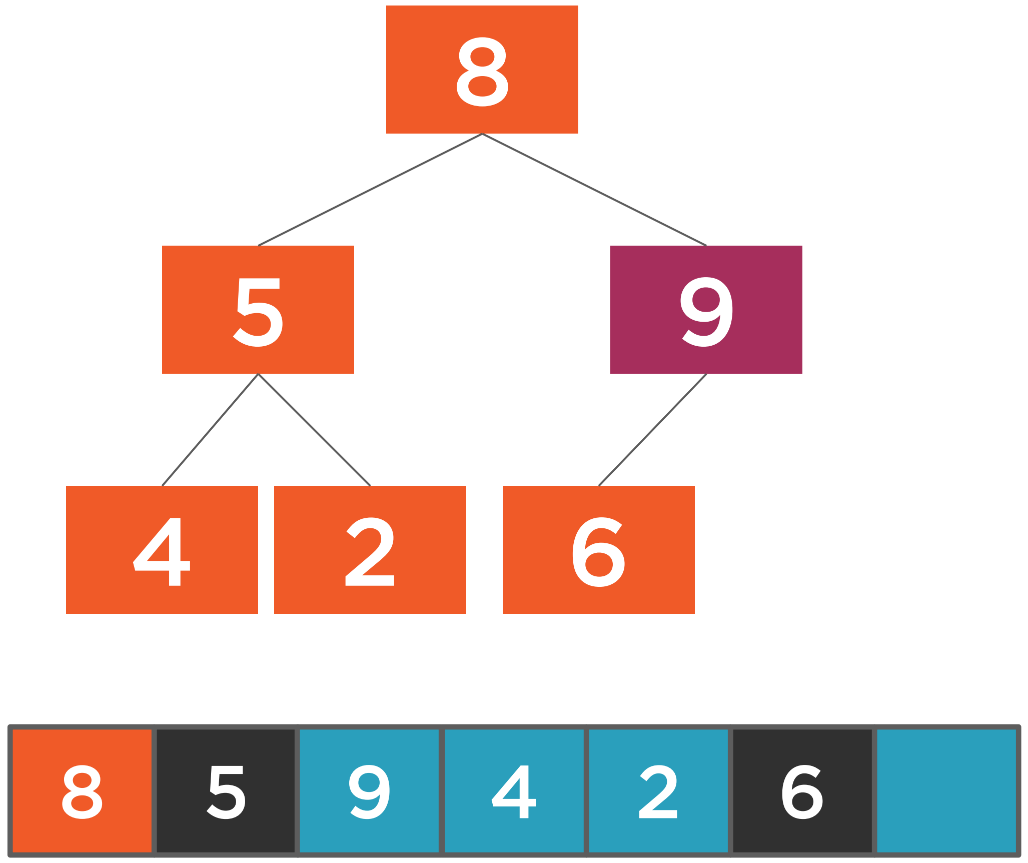
Add new value to end of heap

While the heap property is not satisfied, swap with parent



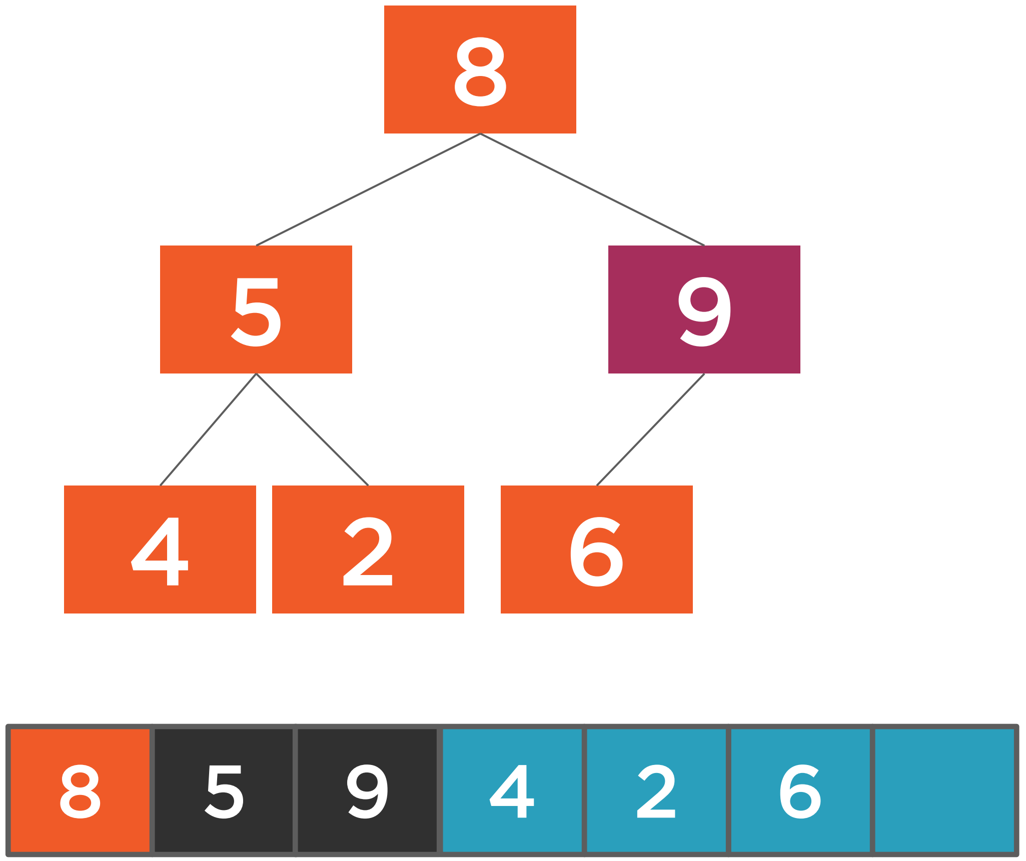
Add new value to end of heap

While the heap property is not satisfied, swap with parent



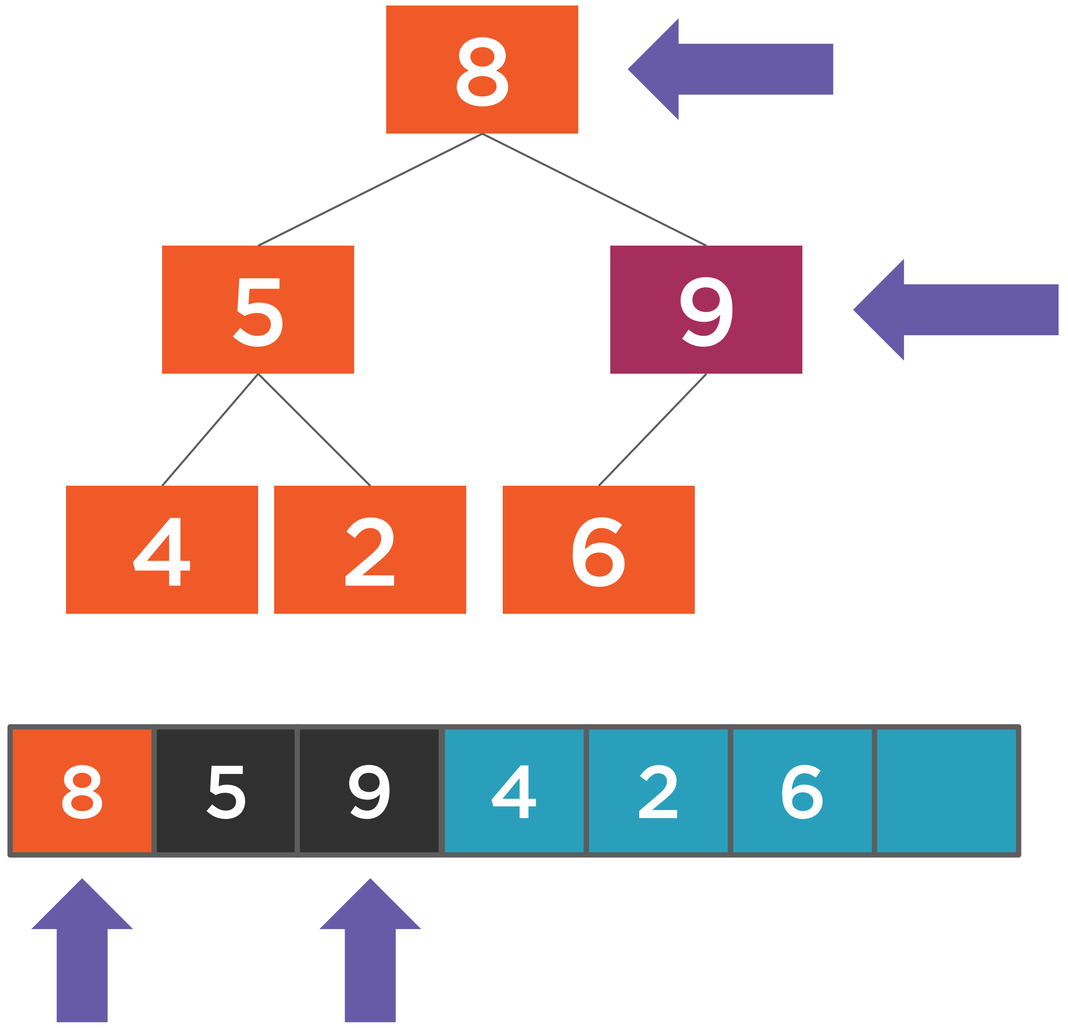
Add new value to end of heap

While the heap property is not satisfied, swap with parent



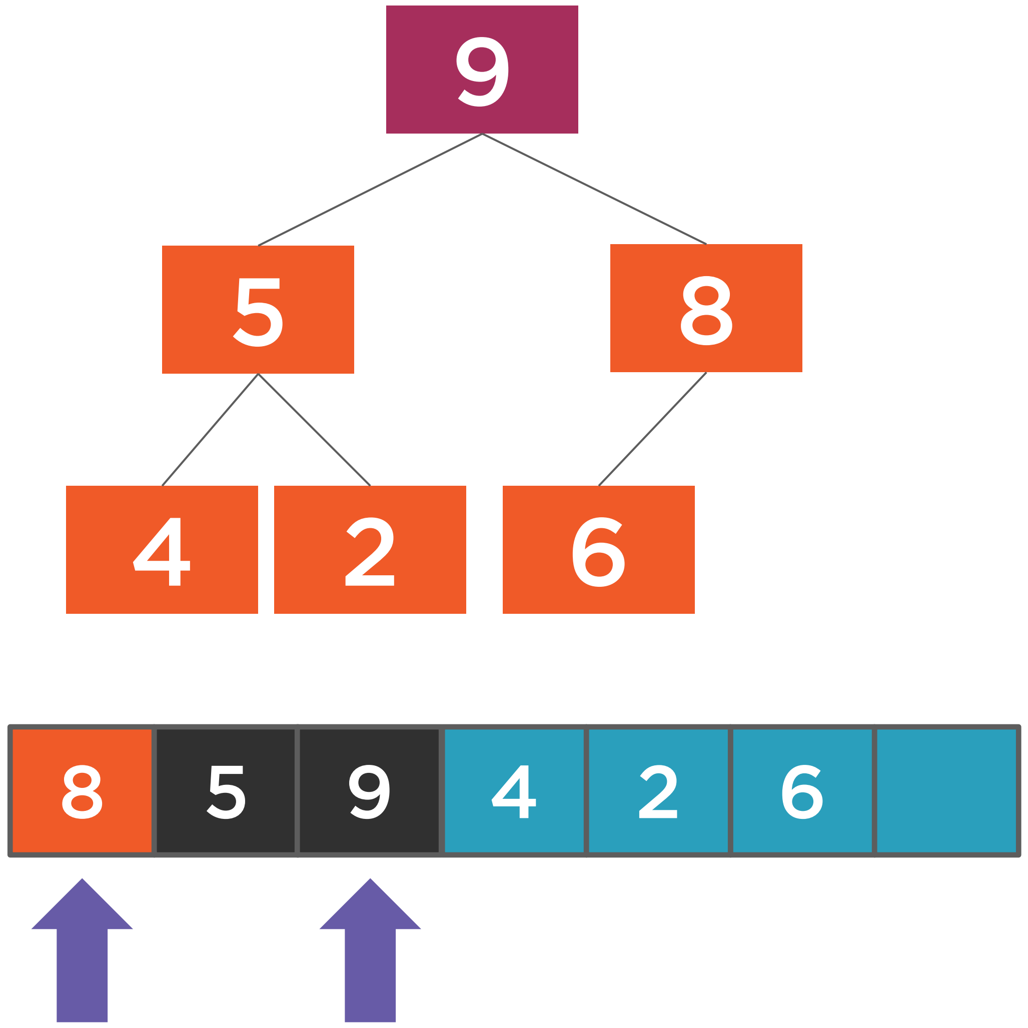
Add new value to end of heap

While the heap property is not satisfied, swap with parent



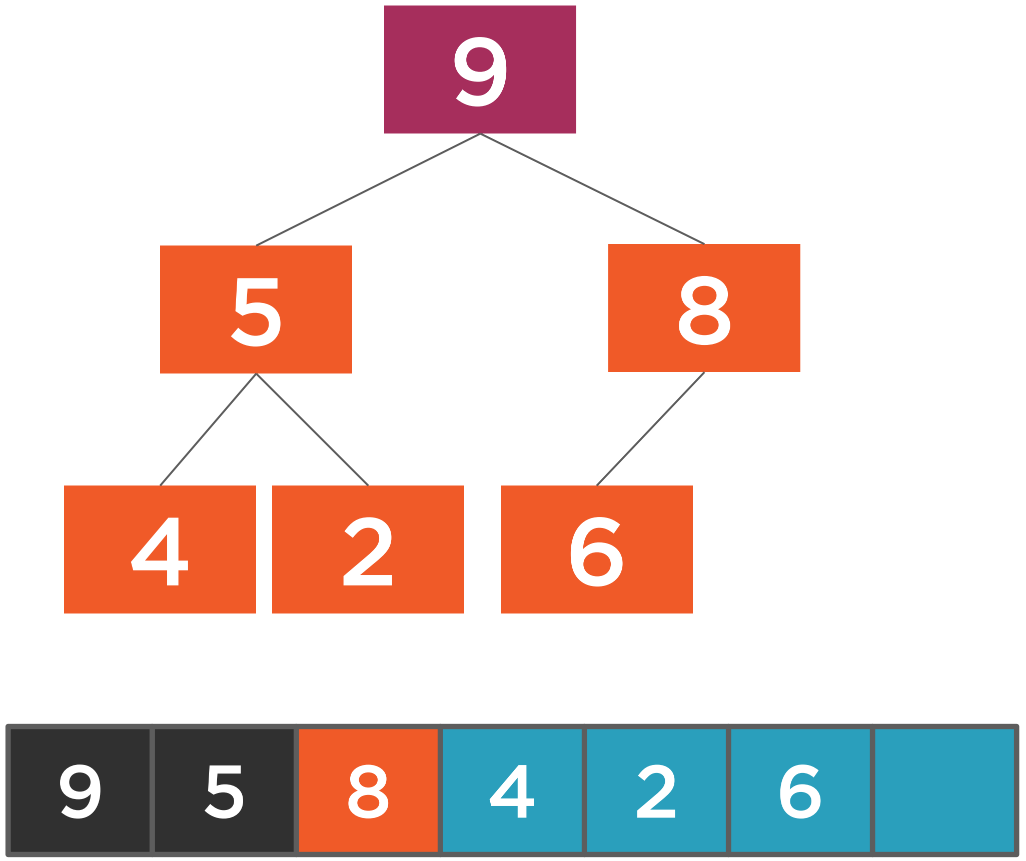
Add new value to end of heap

While the heap property is not satisfied, swap with parent



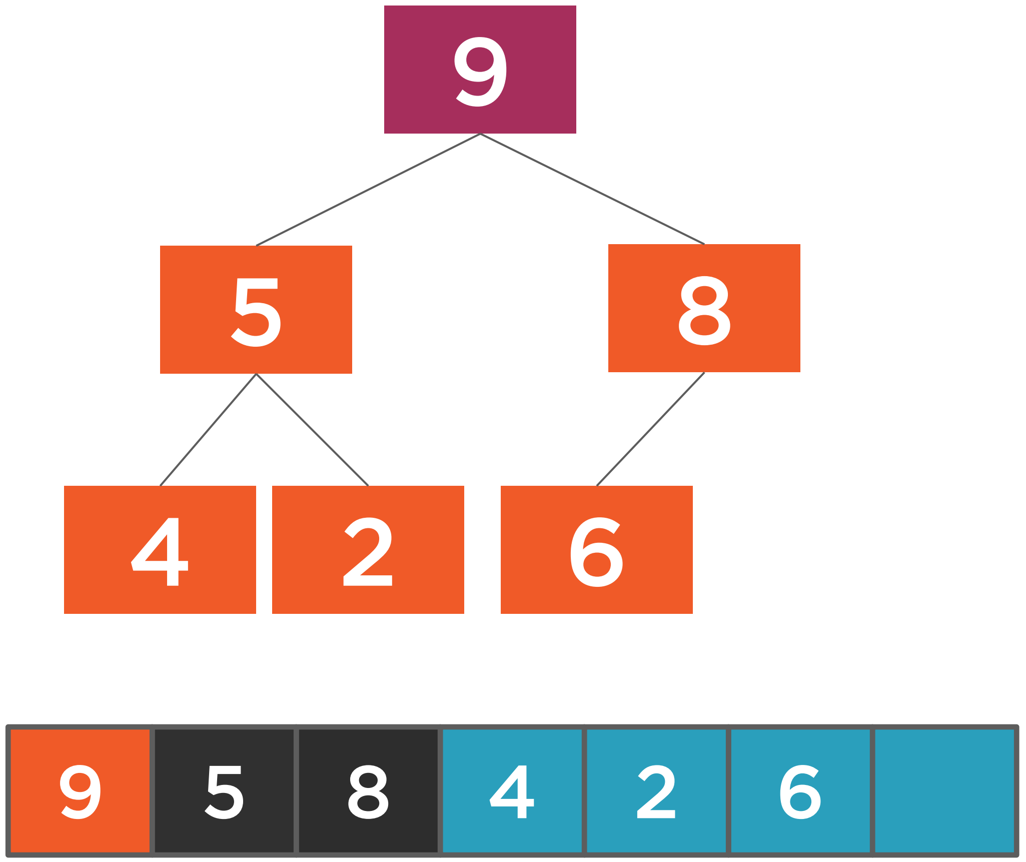
Add new value to end of heap

While the heap property is not satisfied, swap with parent



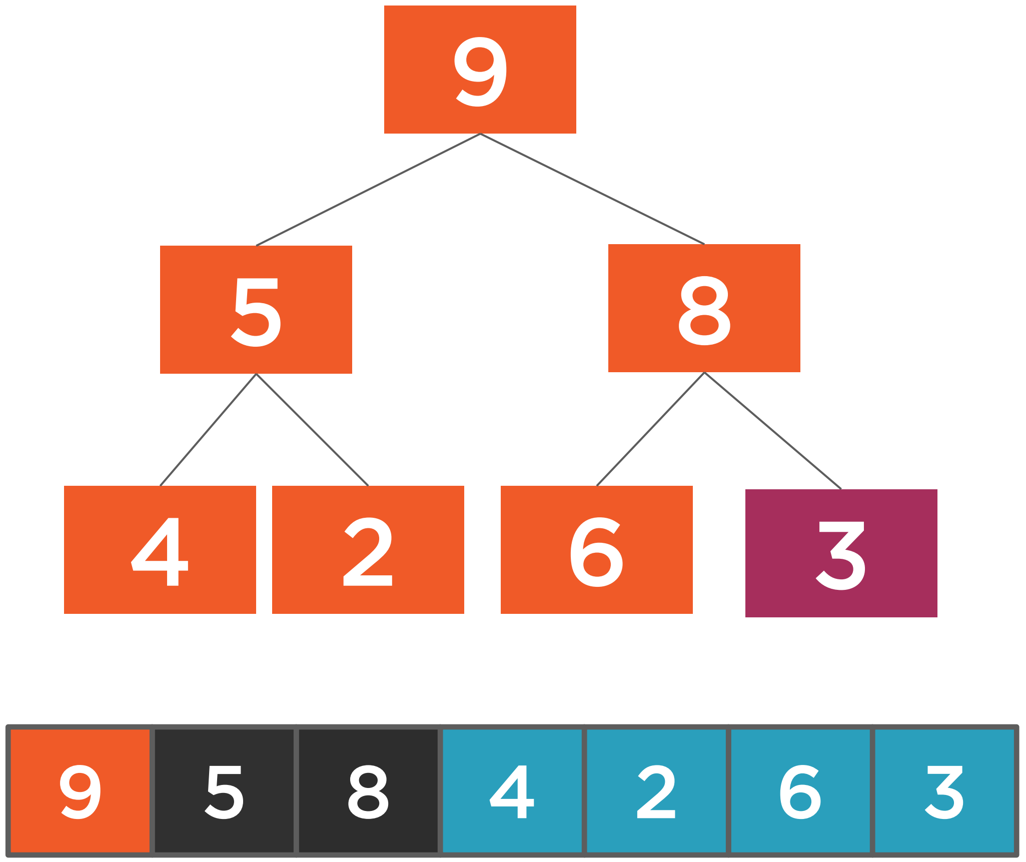
Add new value to end of heap

While the heap property is not satisfied, swap with parent



Add new value to end of heap

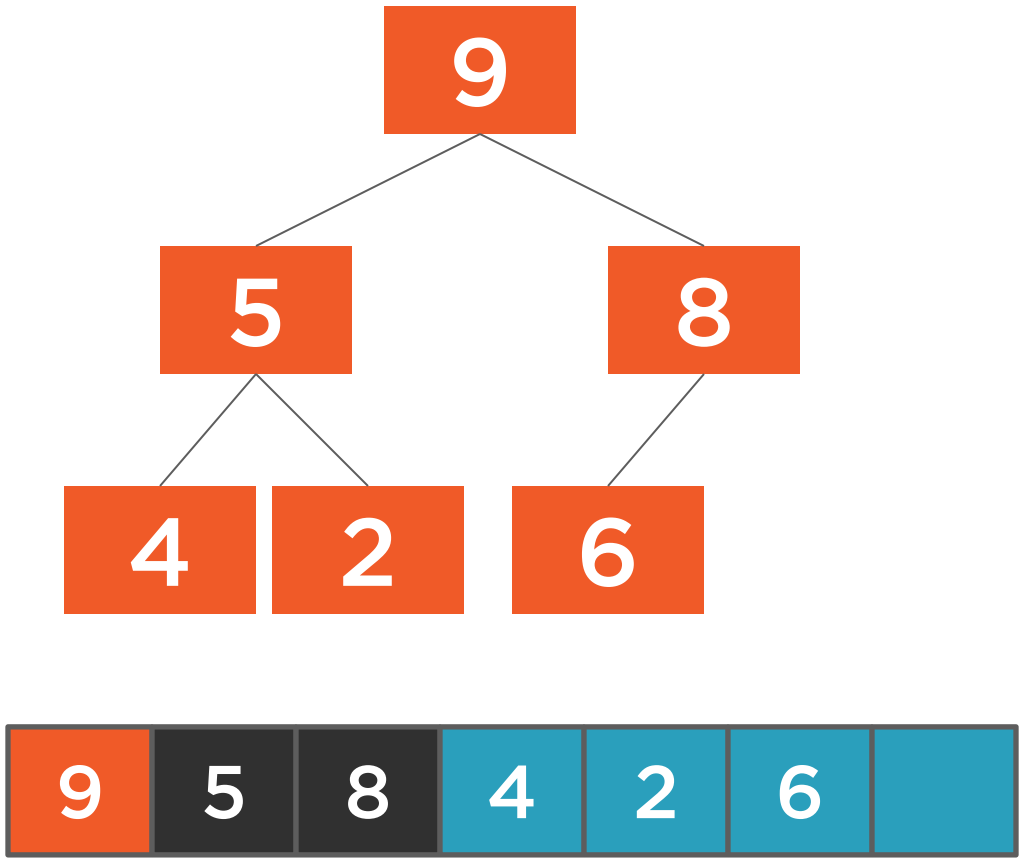
While the heap property is not satisfied, swap with parent



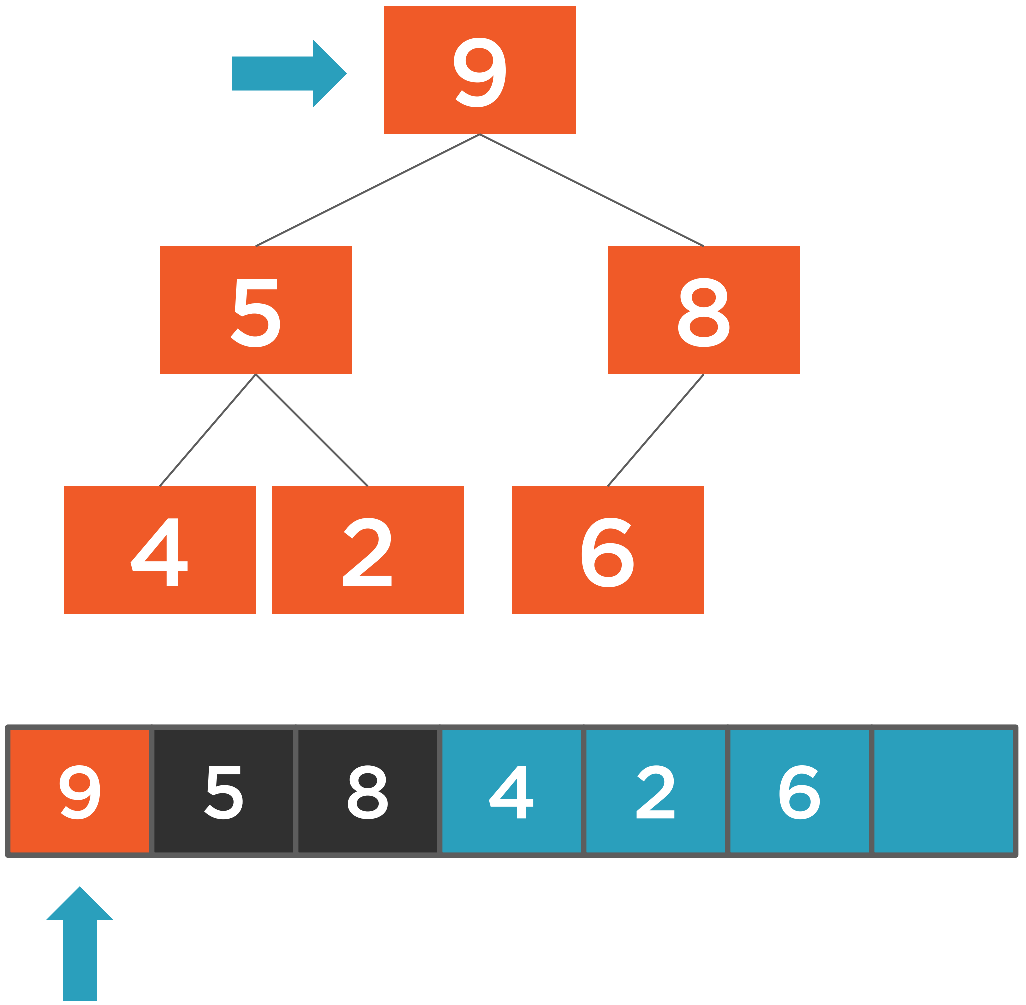
Top

Returns the first item (min or max) in the heap.

Top



Top



```
public T Top() {
    if (Count > 0) {
        return data[0];
    }
    throw new Exception("Top called on empty heap");
}
```

Top

Return the first item in the heap

Pop

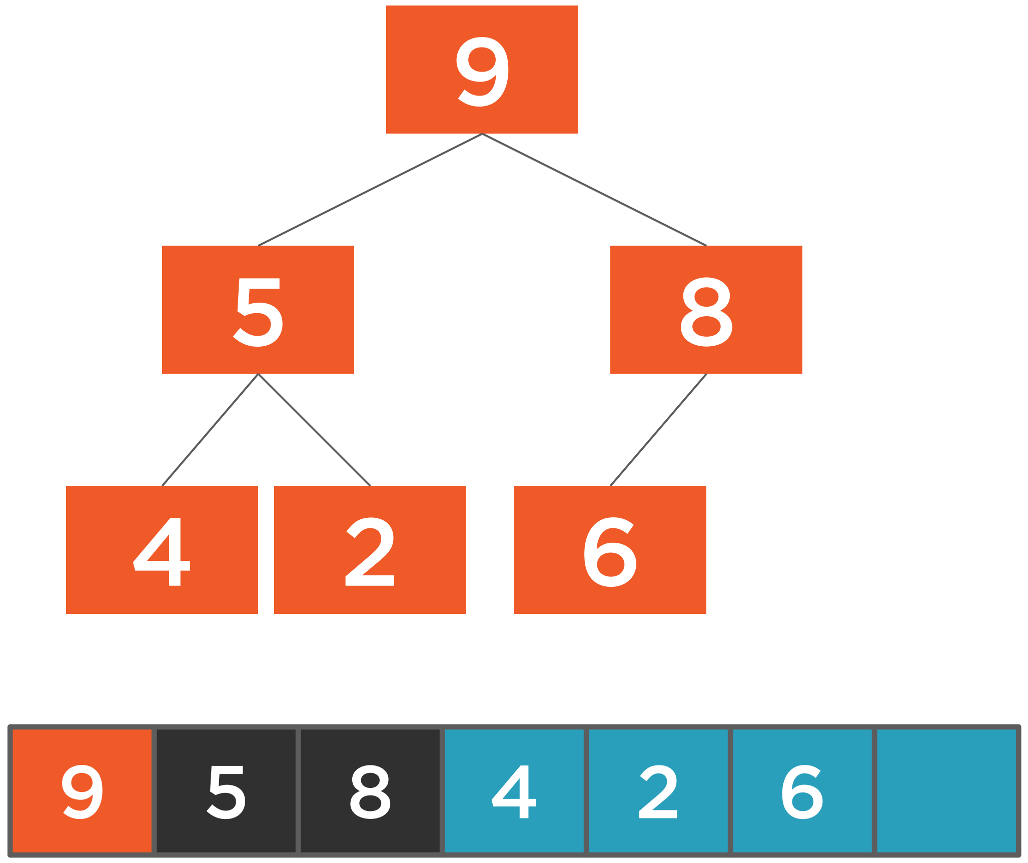
Removes the top item from the heap, moving the replacement item into the first valid position in the heap tree.

Pop Operations

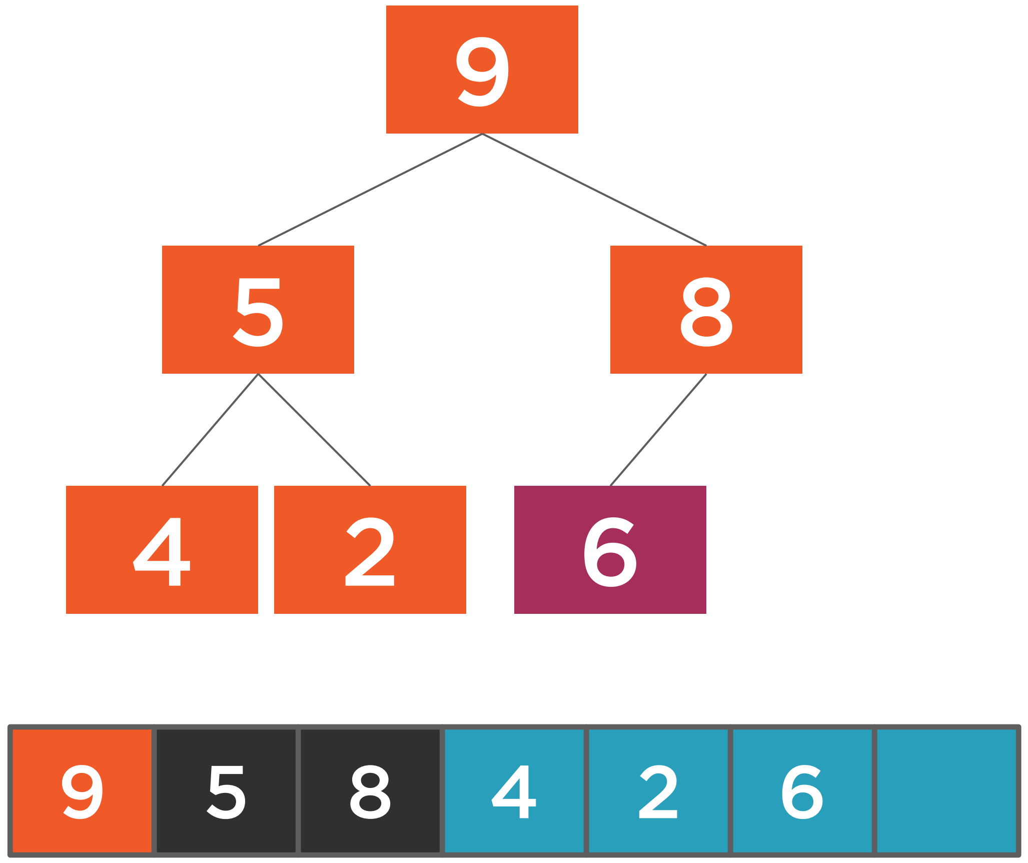
Move the last item in the heap array into the zero (root) index

While the heap property is not satisfied, swap the item with one of its children

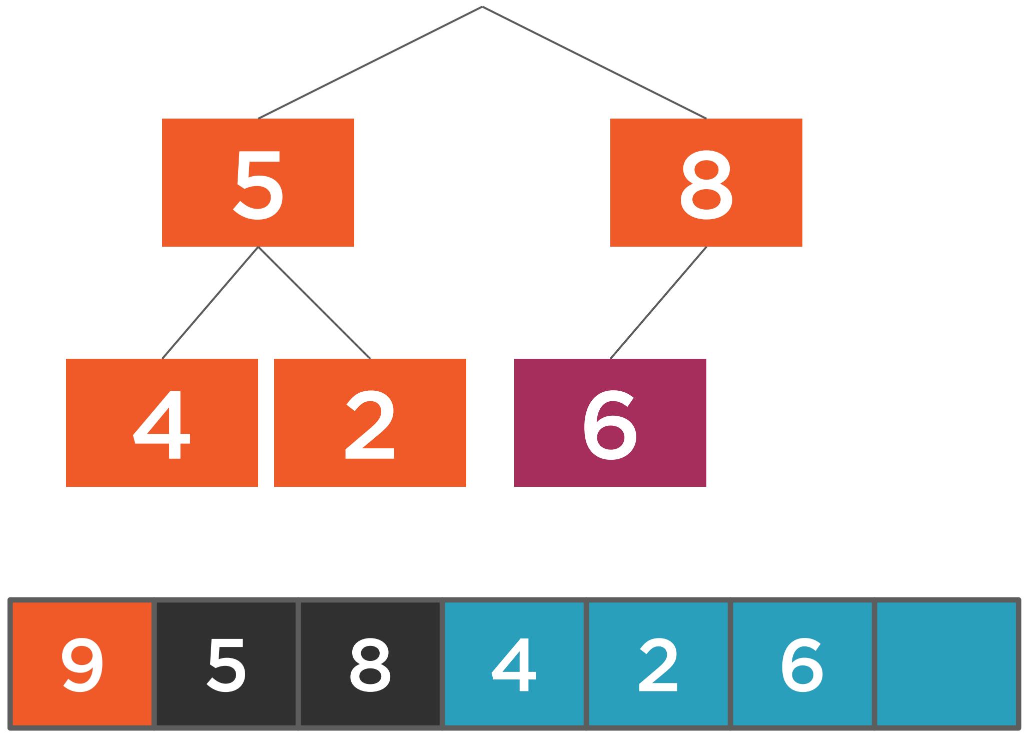
Replace top value with right-most child



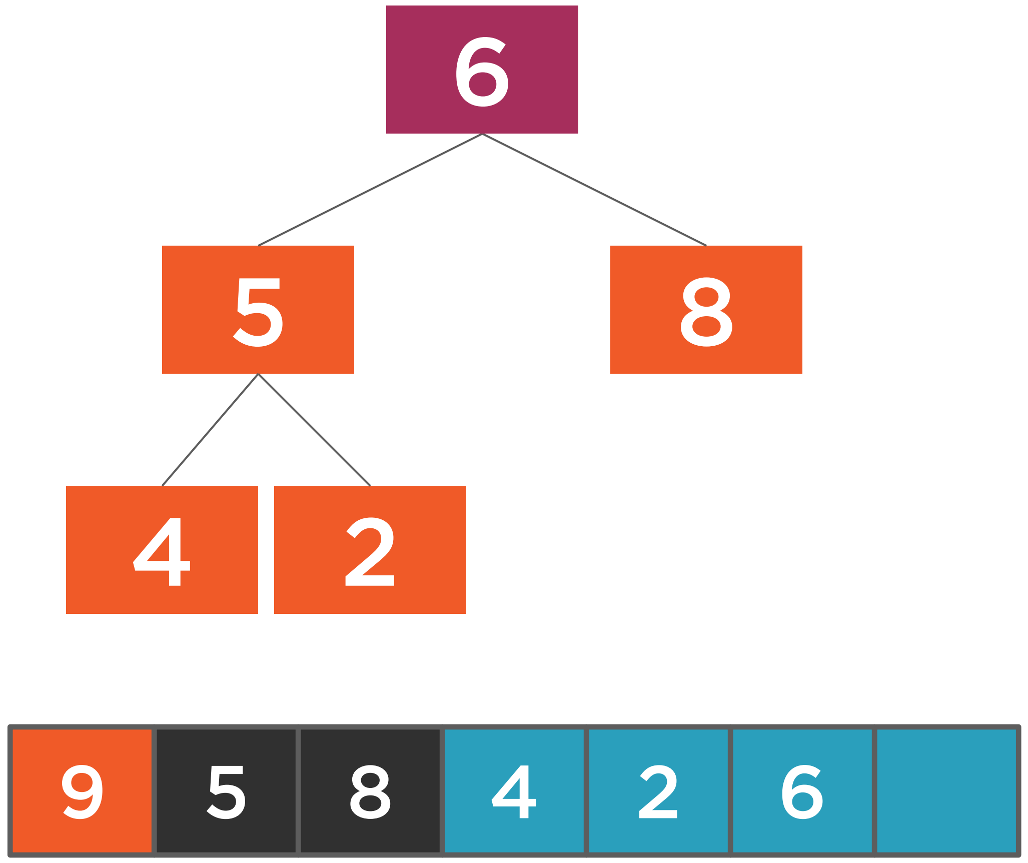
Replace top value with right-most child



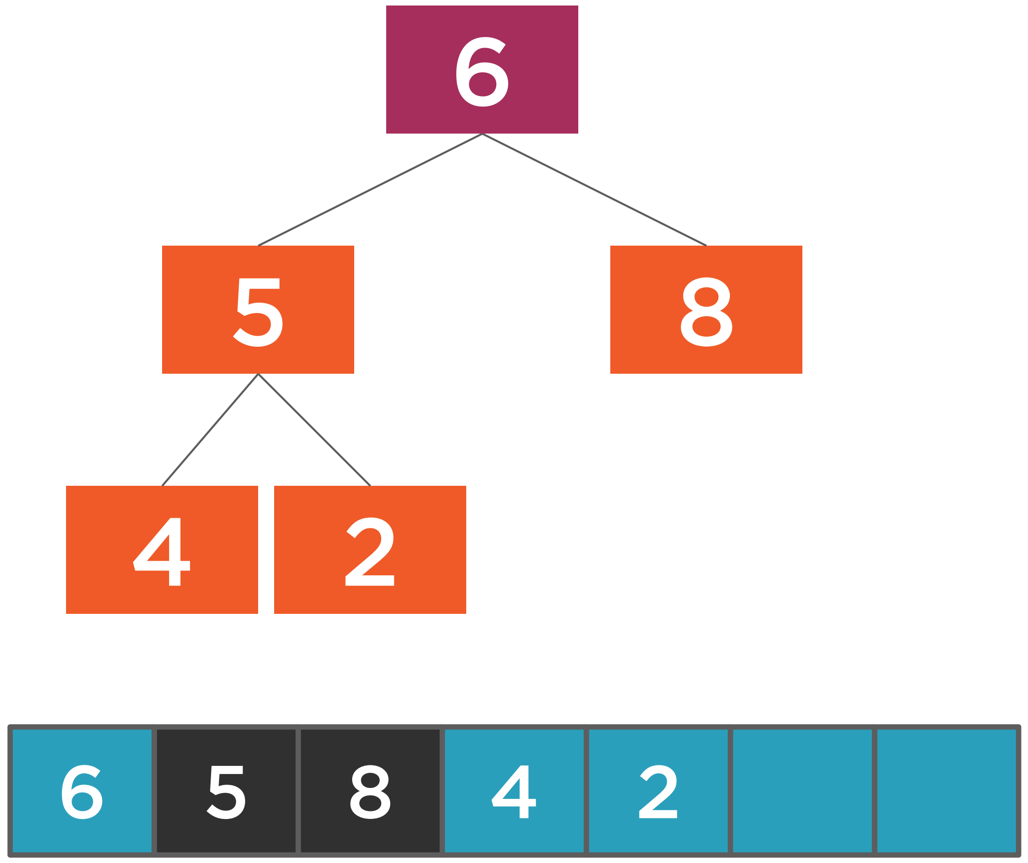
Replace top value with right-most child



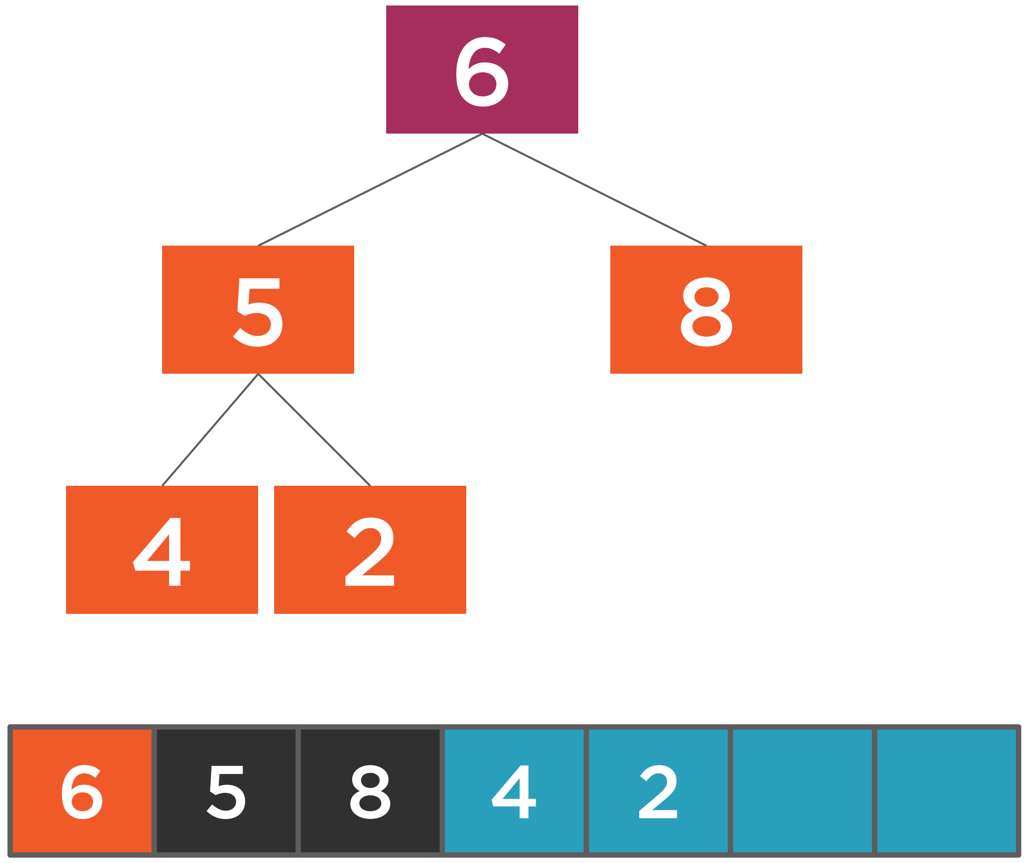
Replace top value with right-most child



Replace top value with right-most child

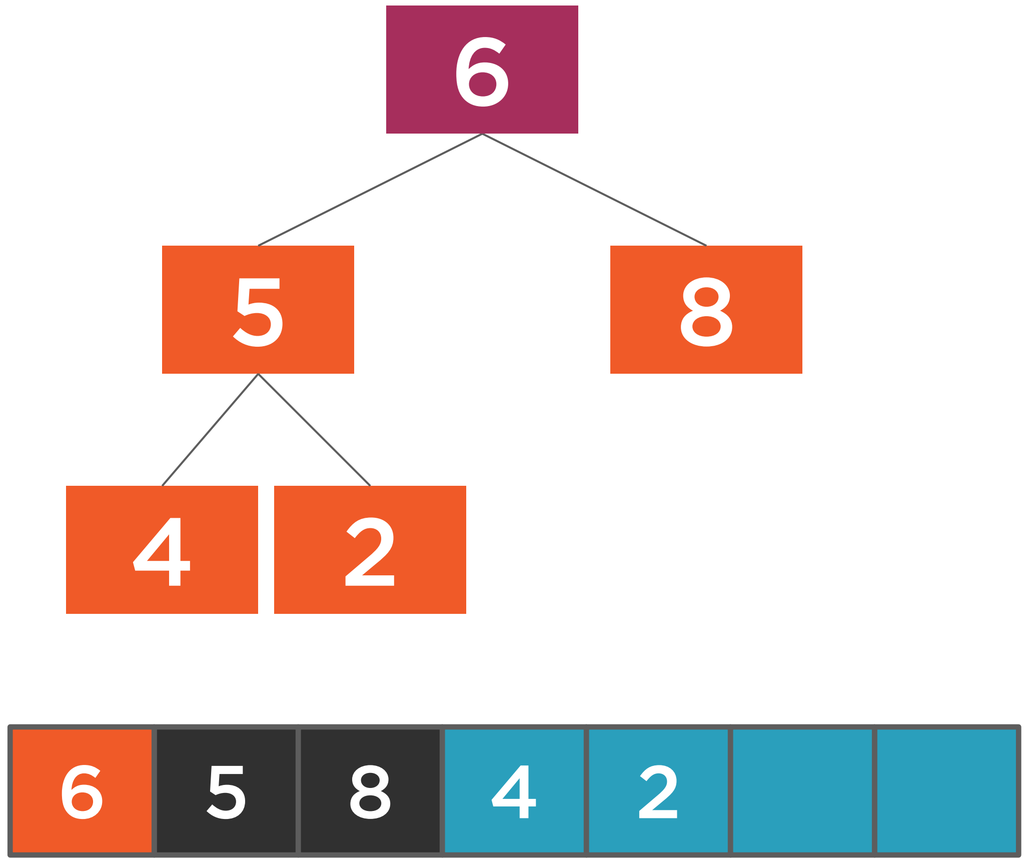


Replace top value with right-most child



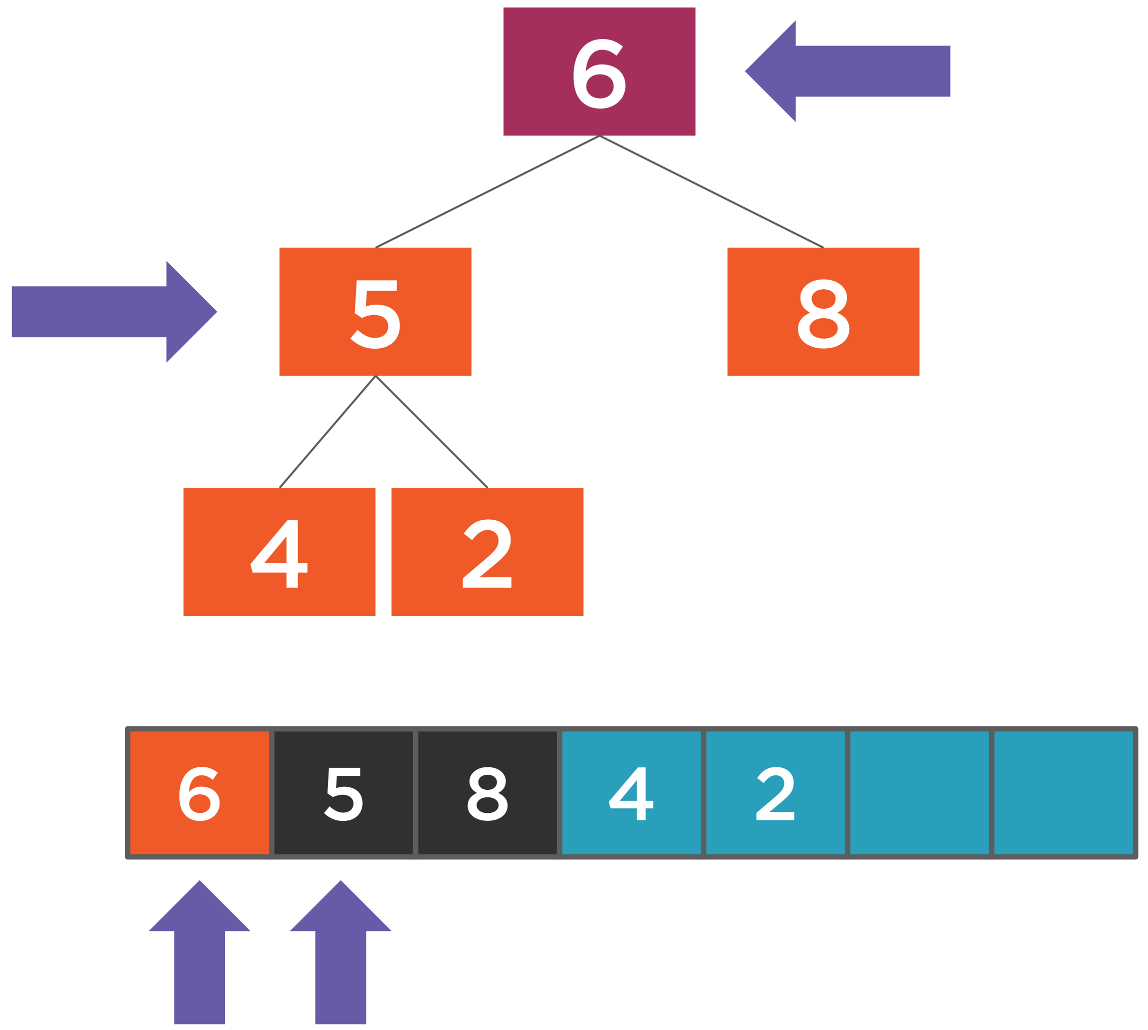
Replace top value with right-most child

Swap new top with children until heap property is satisfied



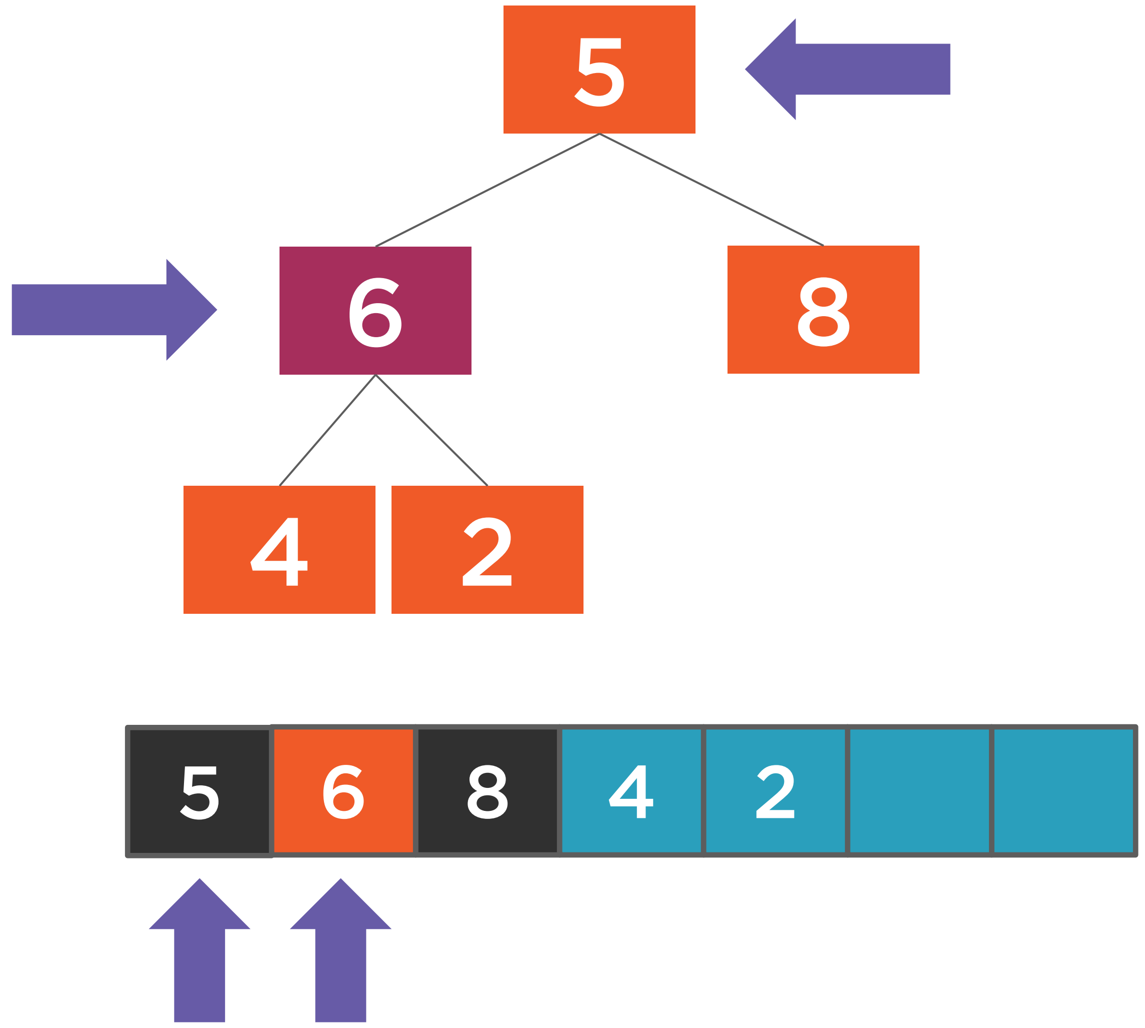
Replace top value with right-most child

Swap new top with children until heap property is satisfied



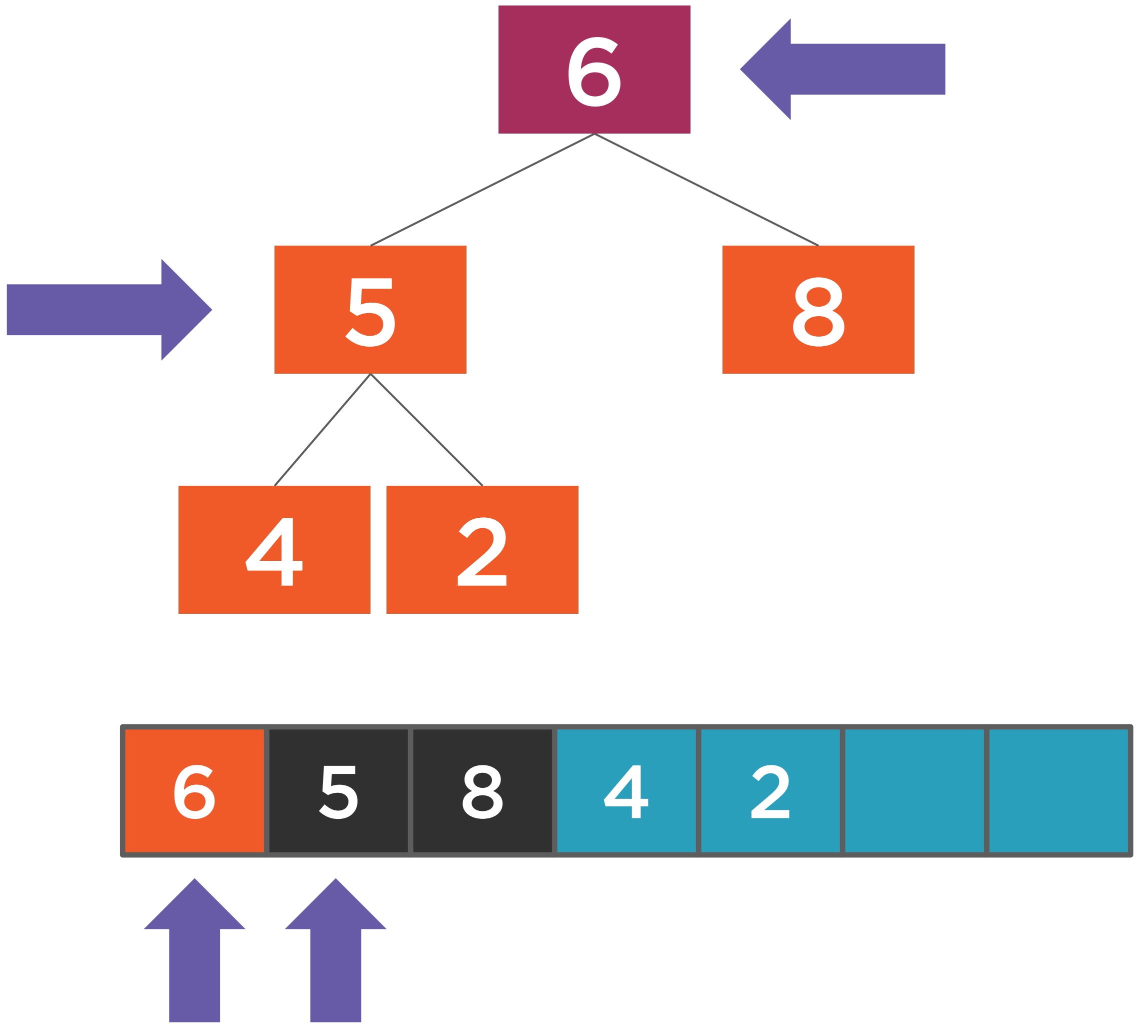
Replace top value with right-most child

Swap new top with children until heap property is satisfied



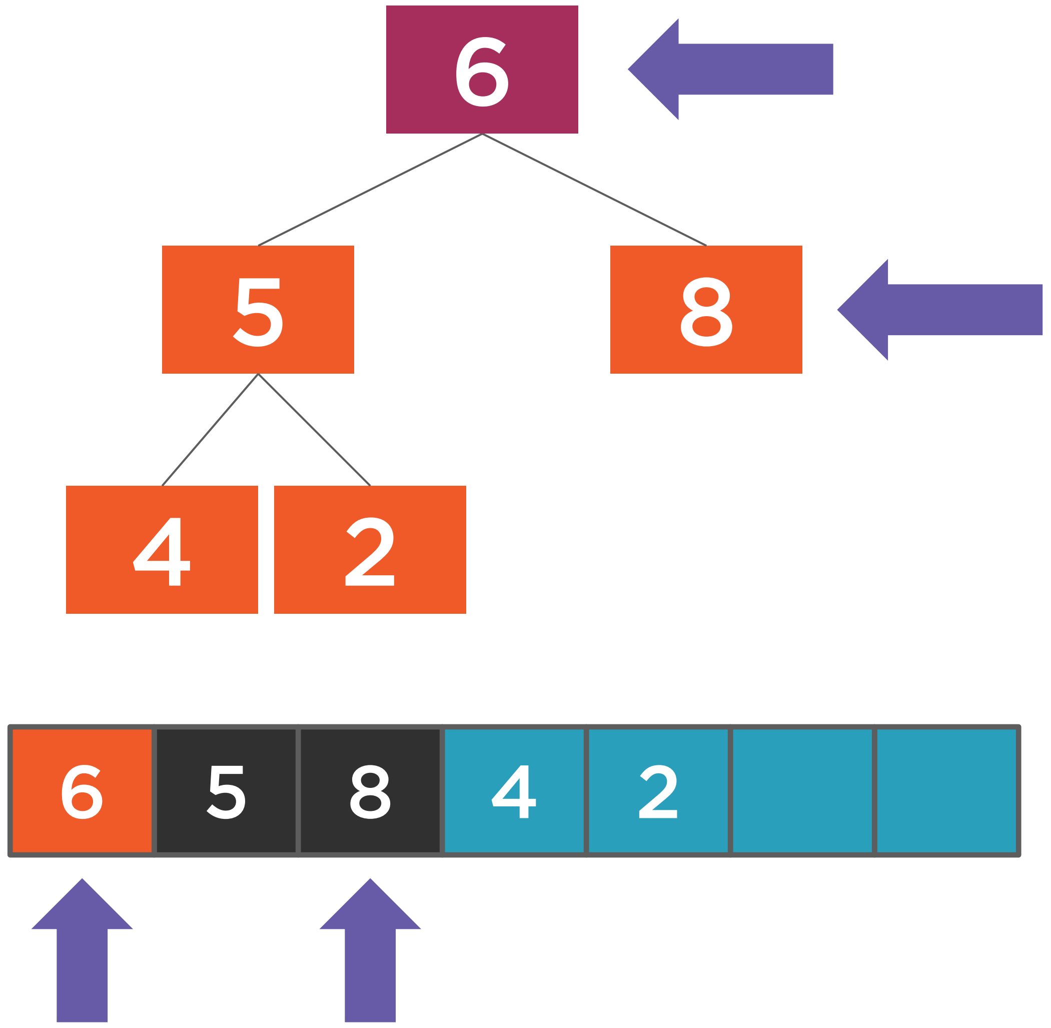
Replace top value with right-most child

Swap new top with children until heap property is satisfied



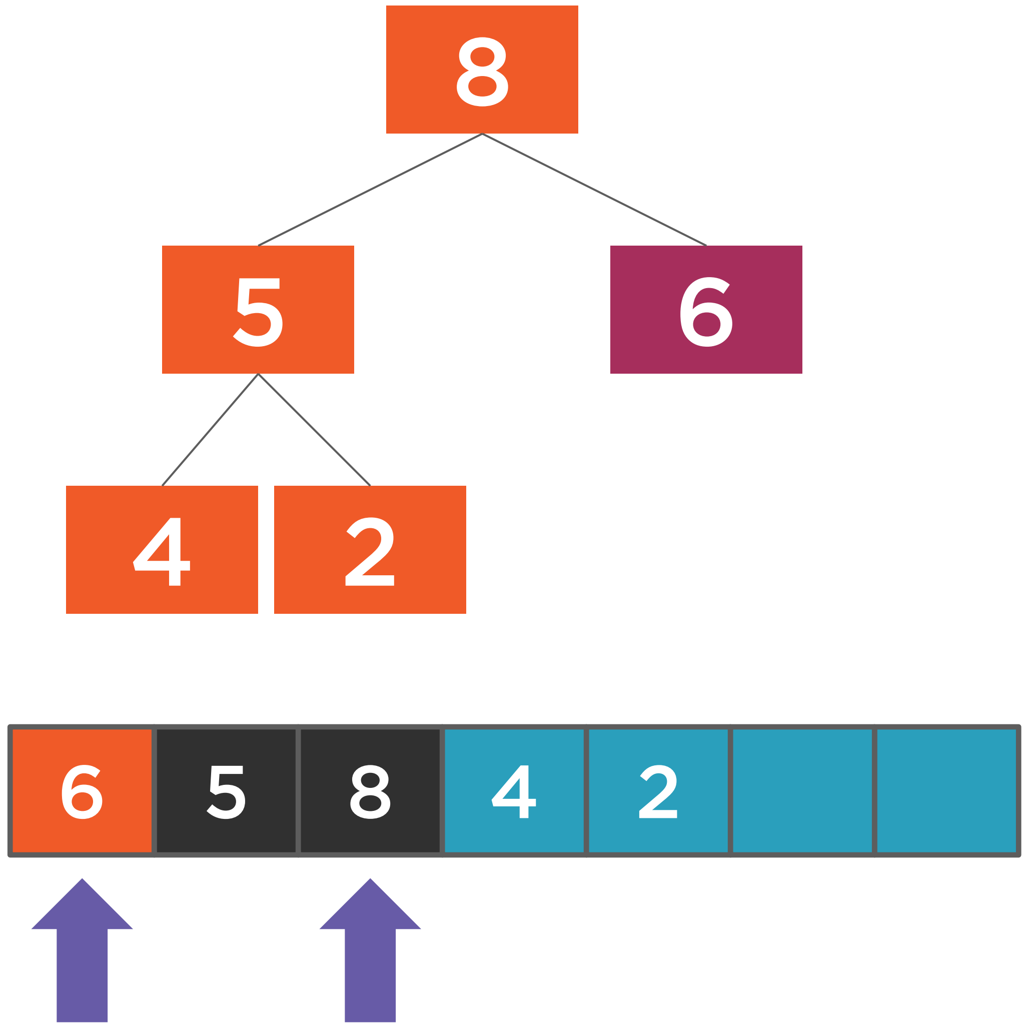
Replace top value with right-most child

Swap new top with children until heap property is satisfied



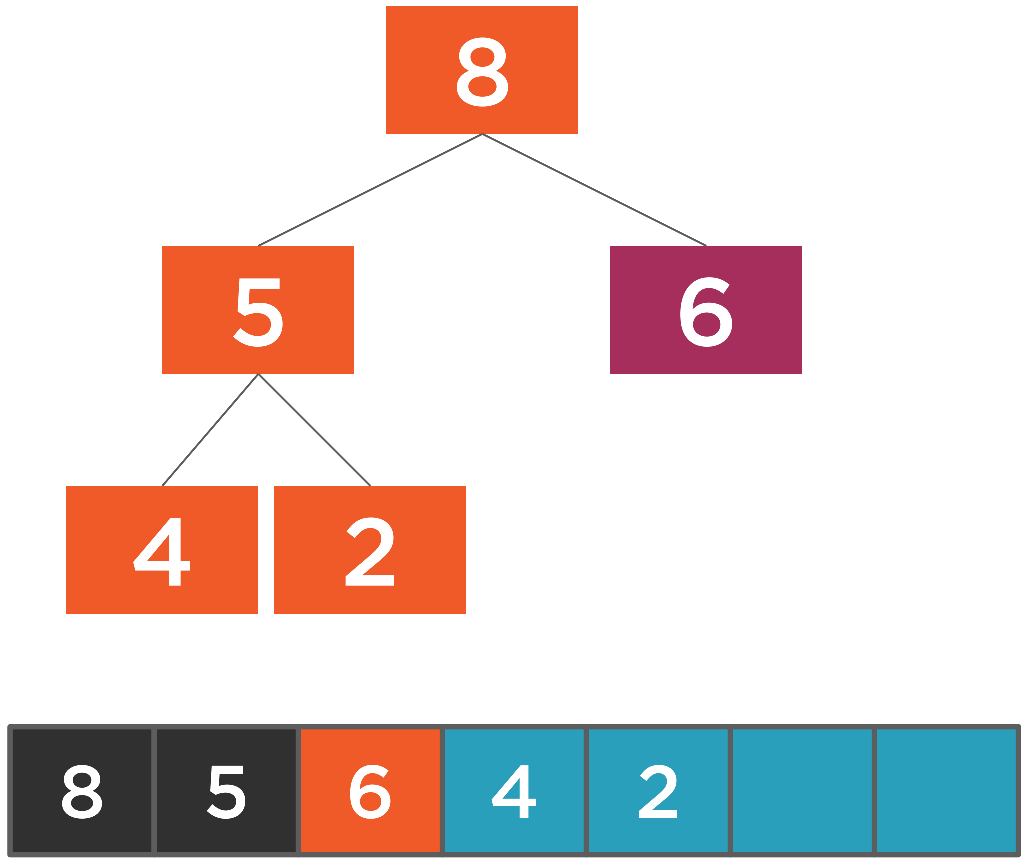
Replace top value with right-most child

Swap new top with children until heap property is satisfied



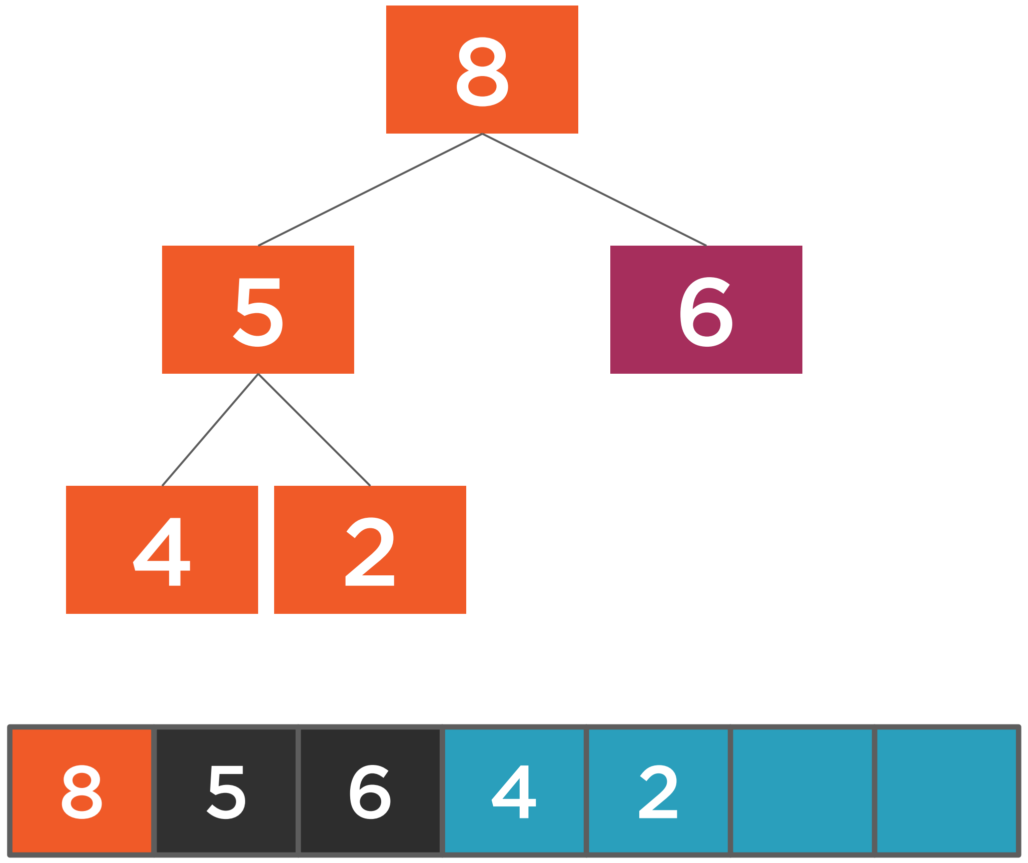
Replace top value with right-most child

Swap new top with children until heap property is satisfied

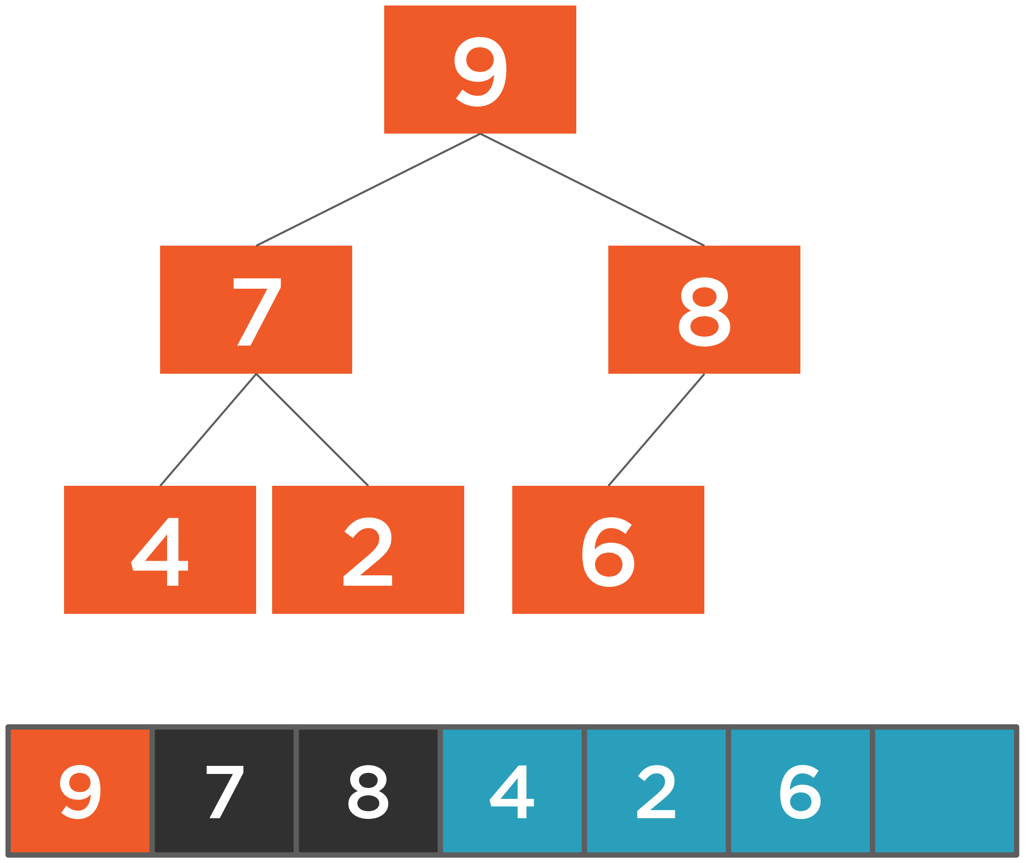


Replace top value with right-most child

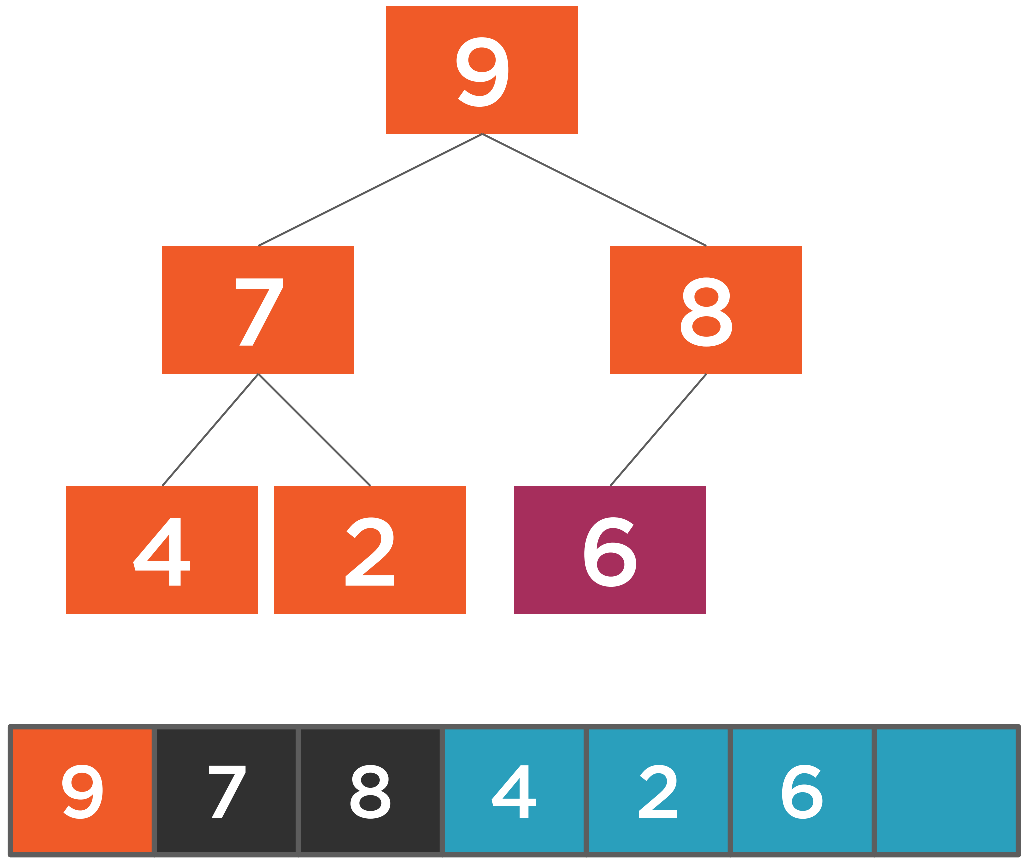
Swap new top with children until heap property is satisfied



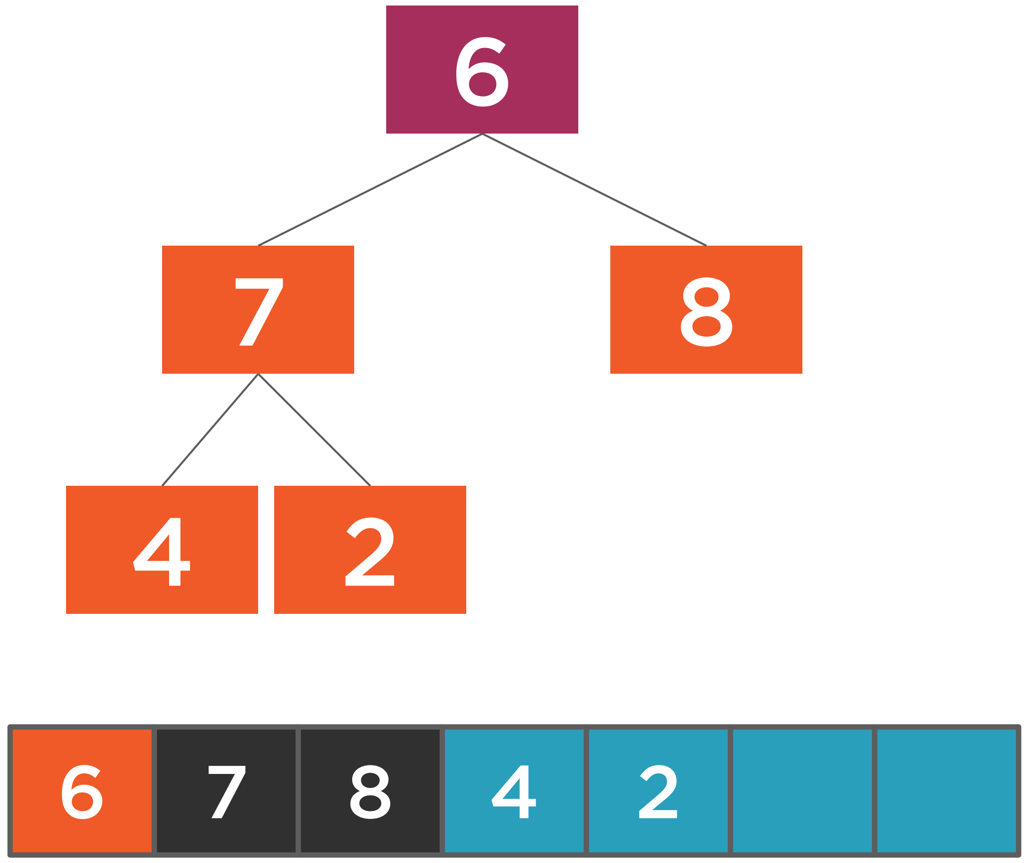
Replace top value with right-most child



Replace top value with right-most child

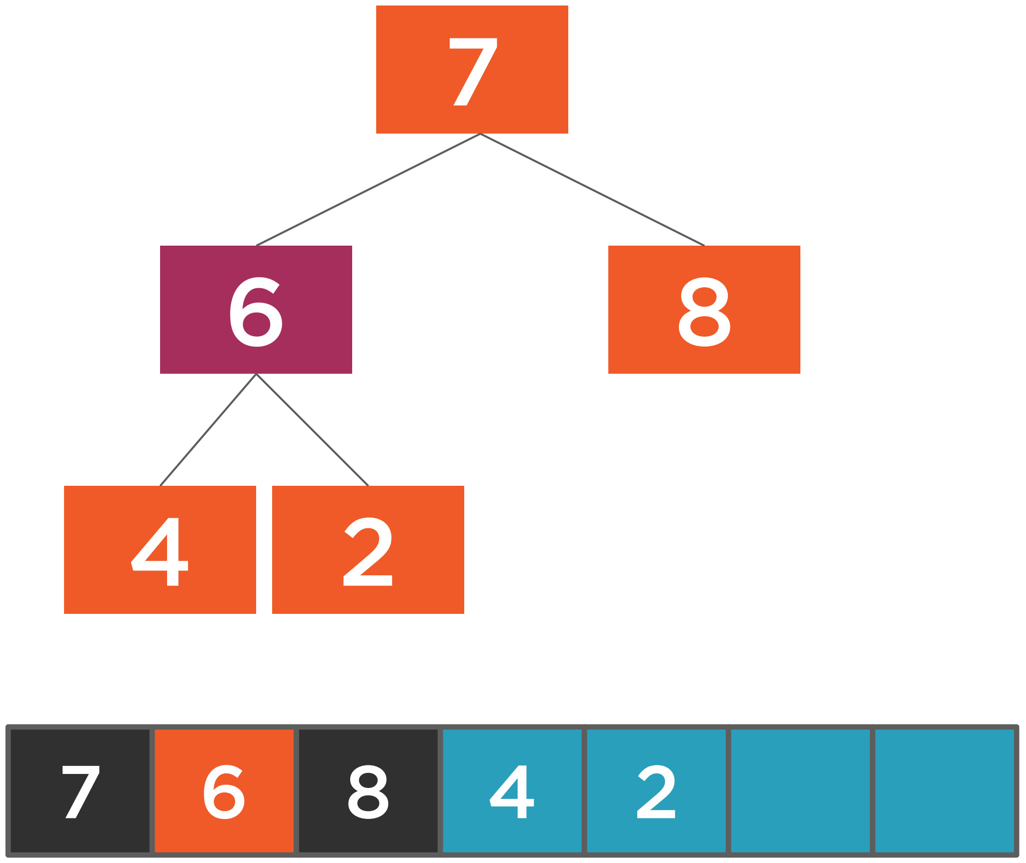


Replace top value with right-most child



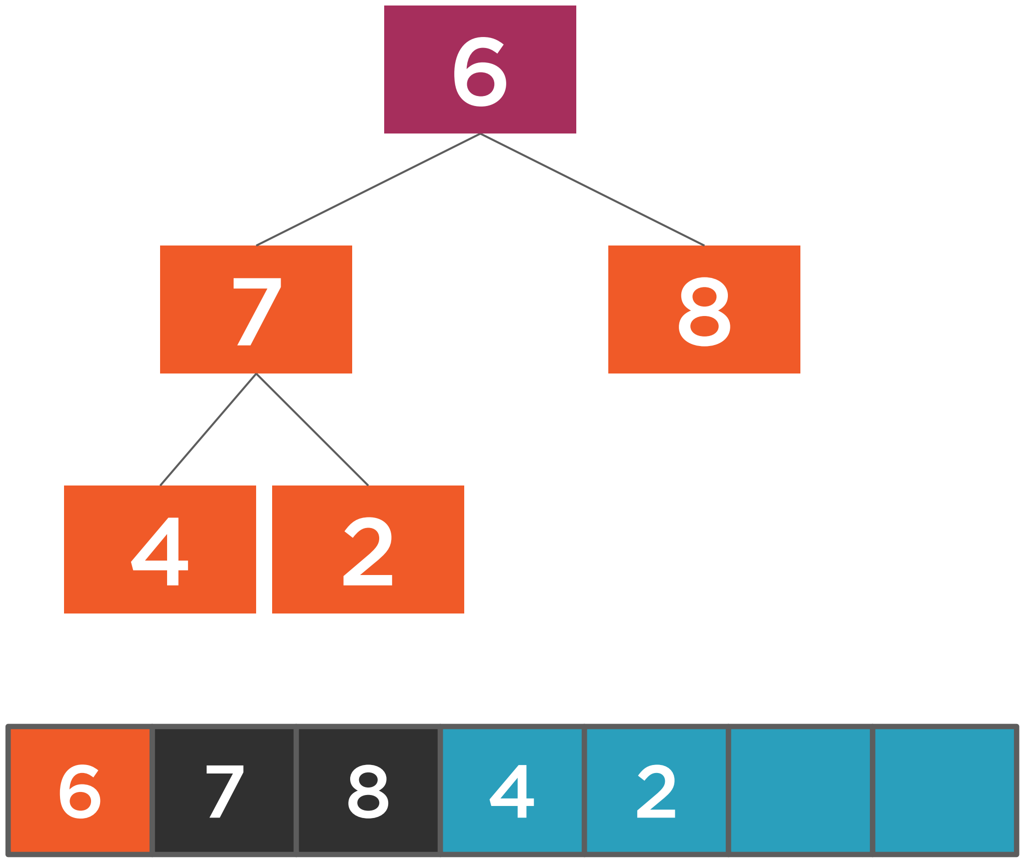
Replace top value with right-most child

Swap new top with children until heap property is satisfied



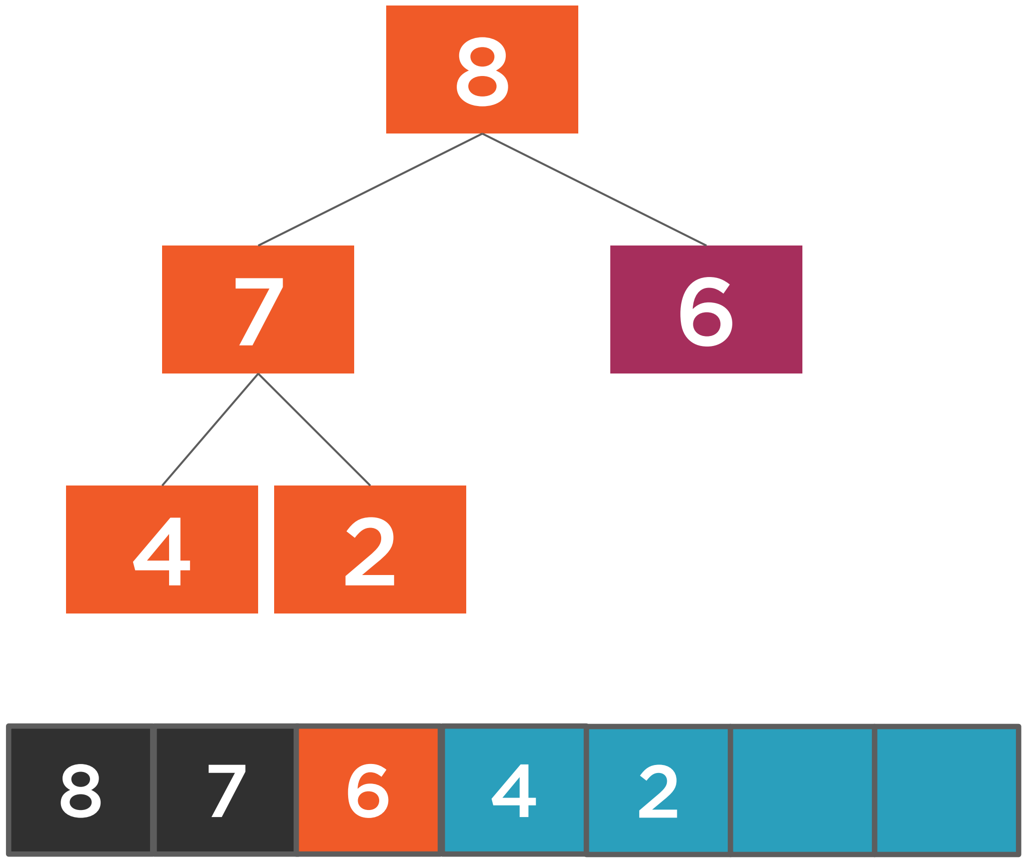
Replace top value with right-most child

Swap new top with children until heap property is satisfied



Replace top value with right-most child

Swap new top with children until heap property is satisfied

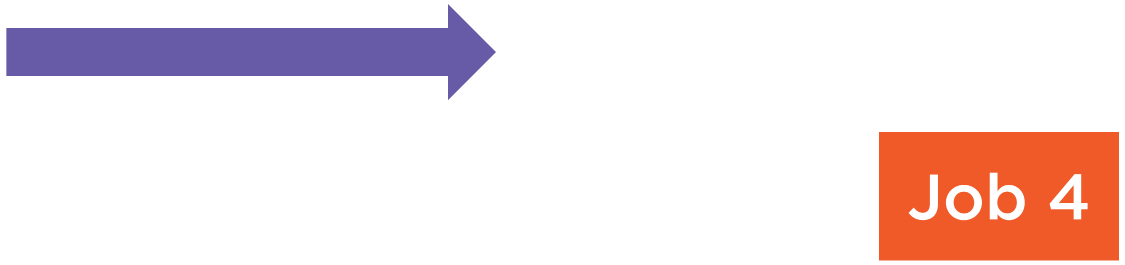


Priority Queue

A queue that pops item in priority, not FIFO, order.

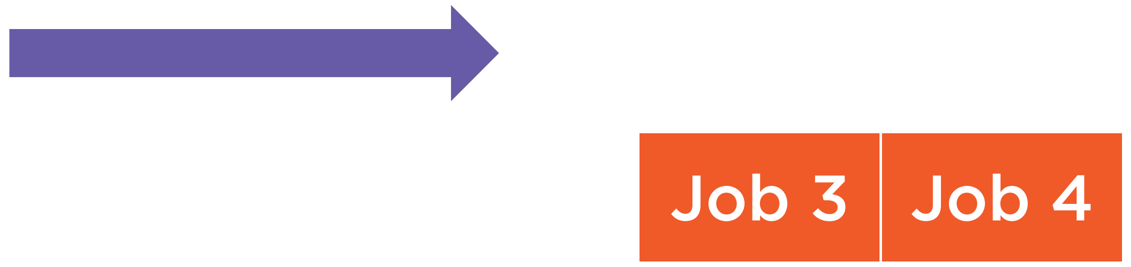
Priority Queue

Priority Queue

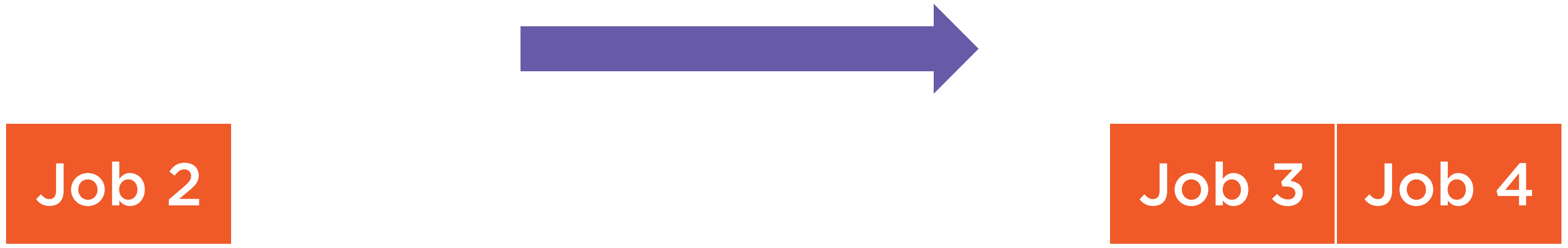


Priority Queue

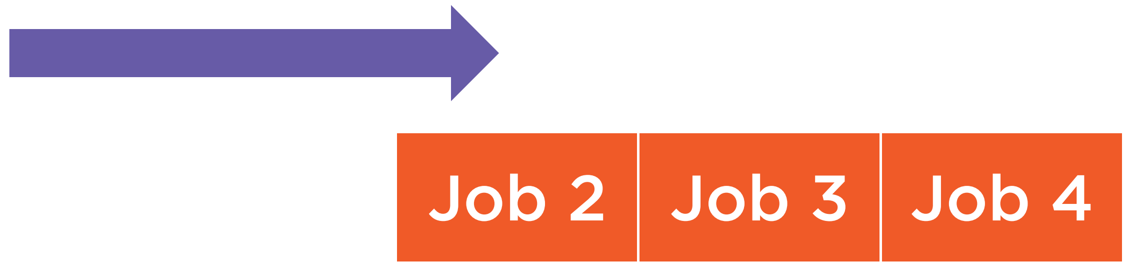
Priority Queue



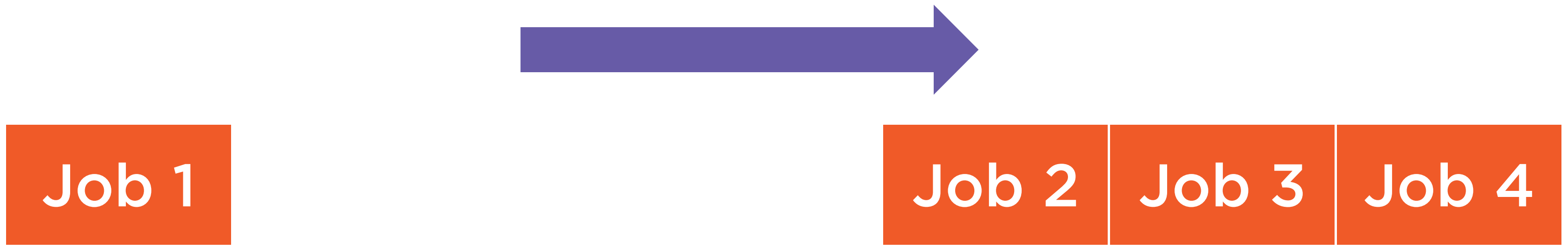
Priority Queue



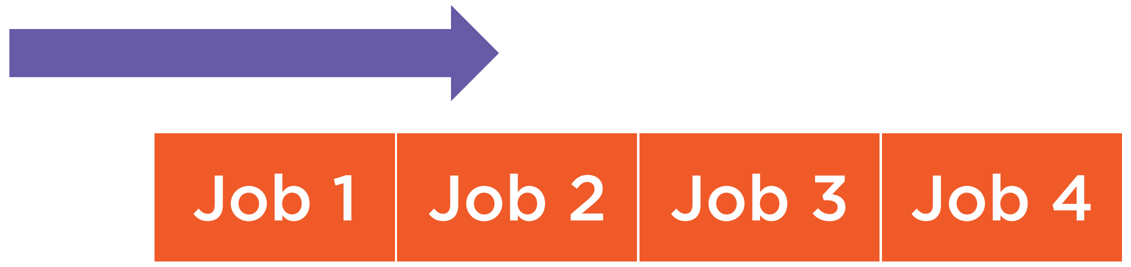
Priority Queue



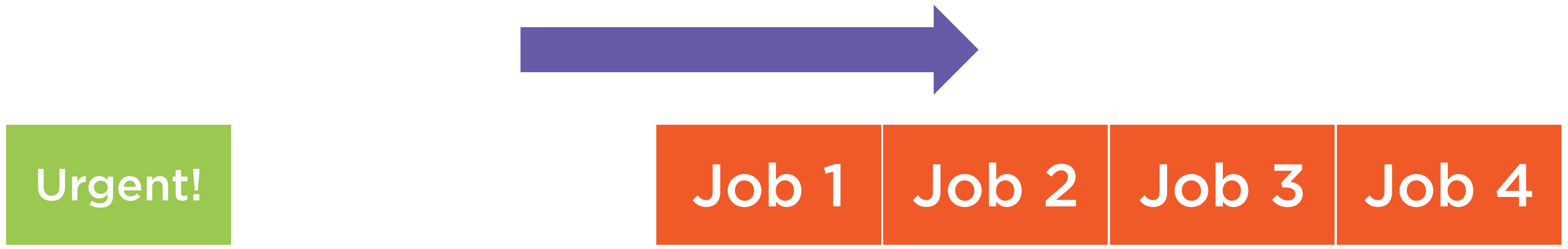
Priority Queue



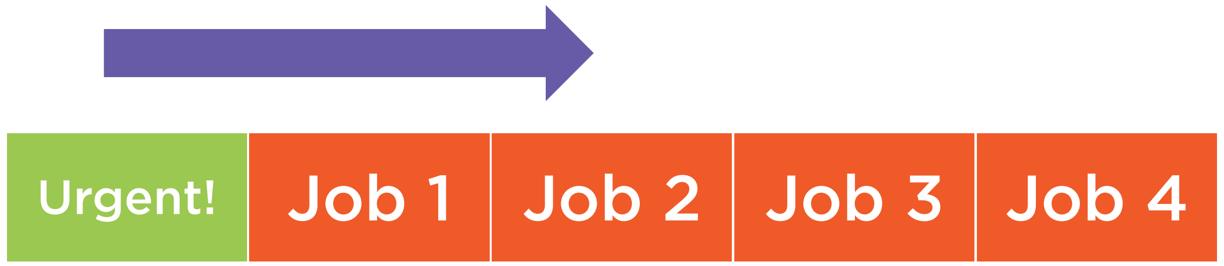
Priority Queue



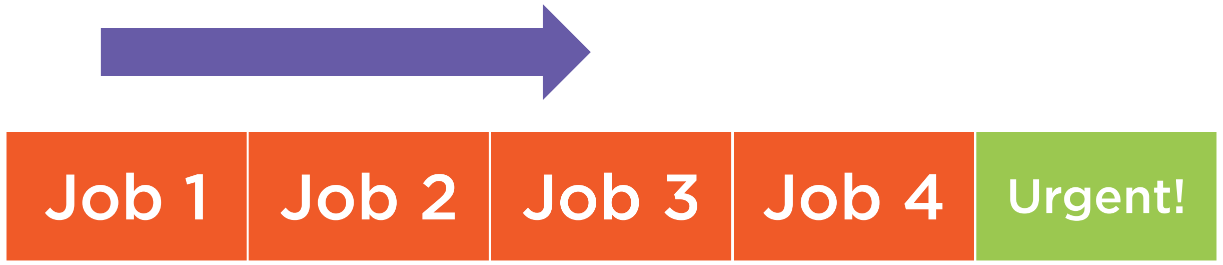
Priority Queue



Priority Queue



Priority Queue



```
public class PriorityQueue<T> {

    readonly Heap<T> heap = new Heap<T>();

    public void Enqueue(T value) {

        heap.Push(value);

    }

    public T Deque() {

        T value = heap.Top();

        heap.Pop();


        return value;

    }

}
```

- ◀ Generic class
- ◀ Data is stored in a heap
- ◀ Enqueue pushes a value onto the heap
- ◀ Deque pops a value from the heap

Demo

Review heap code

Review priority queue

Example: job queue