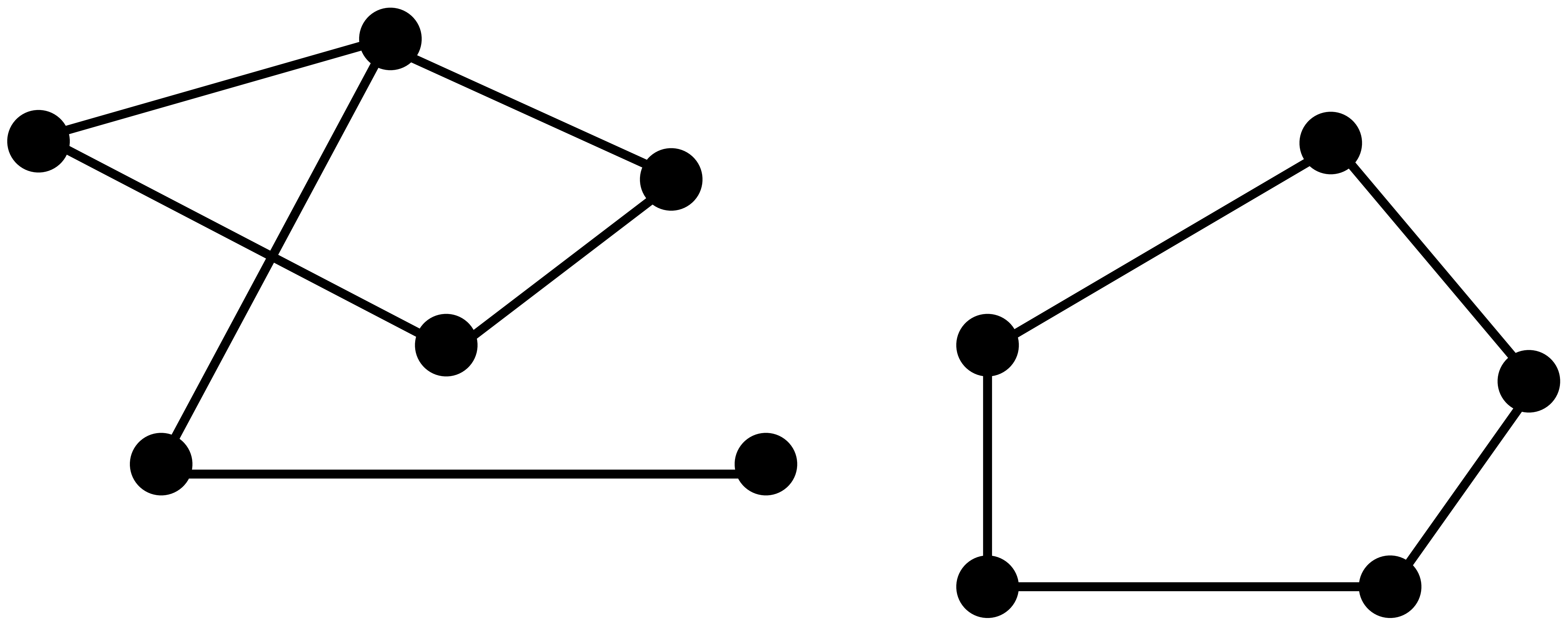
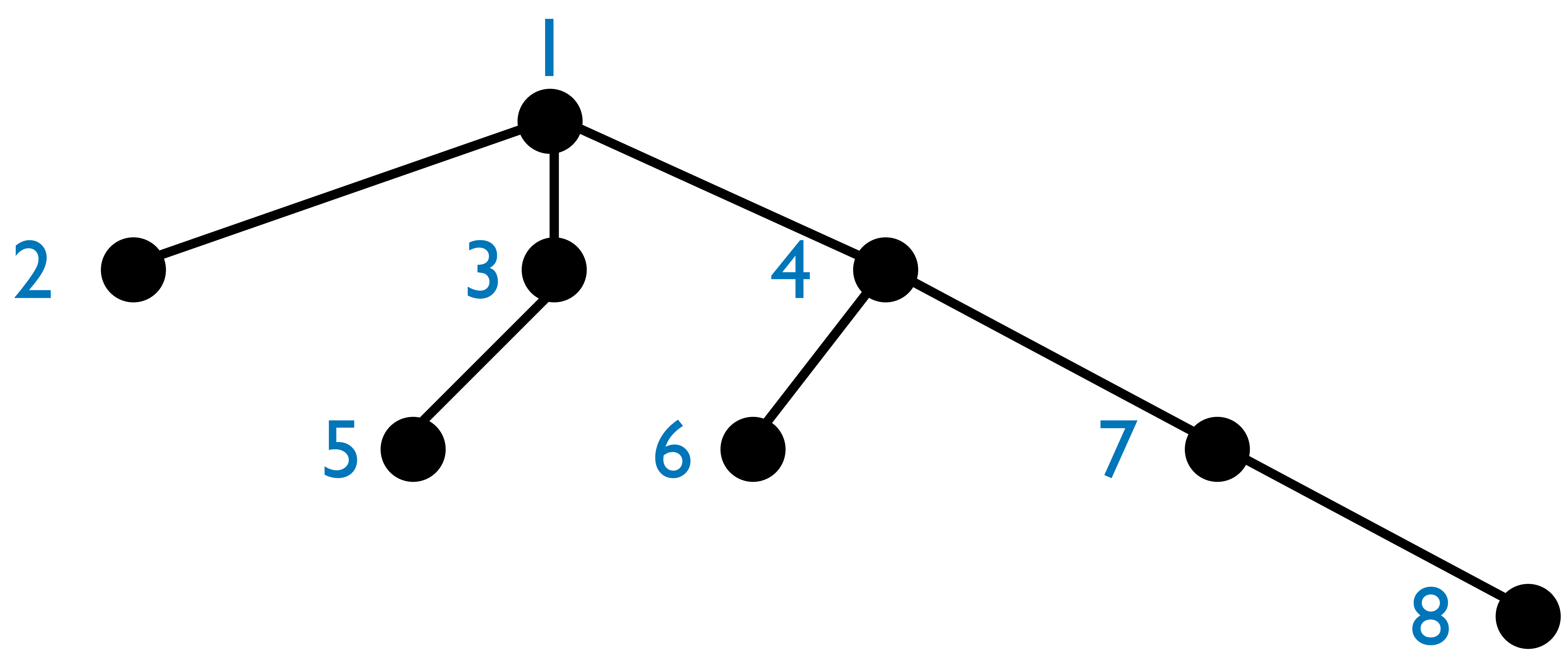
Graphs

Graphs



Anatomy of a Tree



Priority Queue

What is a priority queue?

Priority Queue

What is a priority queue?

In a priority queue elements are organised by priority.

Priority Queue

What is a priority queue?

In a priority queue elements are organised by priority.

Pop on a (max) priority queue removes the element with highest priority.

Priority Queue

What is a priority queue?

In a priority queue elements are organised by priority.

Pop on a (max) priority queue removes the element with highest priority.

Think about a to-do list. A queue or stack is not the best model for a to-do list.

Priority Queue

What is a priority queue?

In a priority queue elements are organised by priority.

Pop on a (max) priority queue removes the element with highest priority.

Think about a to-do list. A queue or stack is not the best model for a to-do list.

We want our to do list to tell us the next task that is due, regardless of when this task was entered into the to do list.

Heap

We can implement a priority queue via a binary heap.

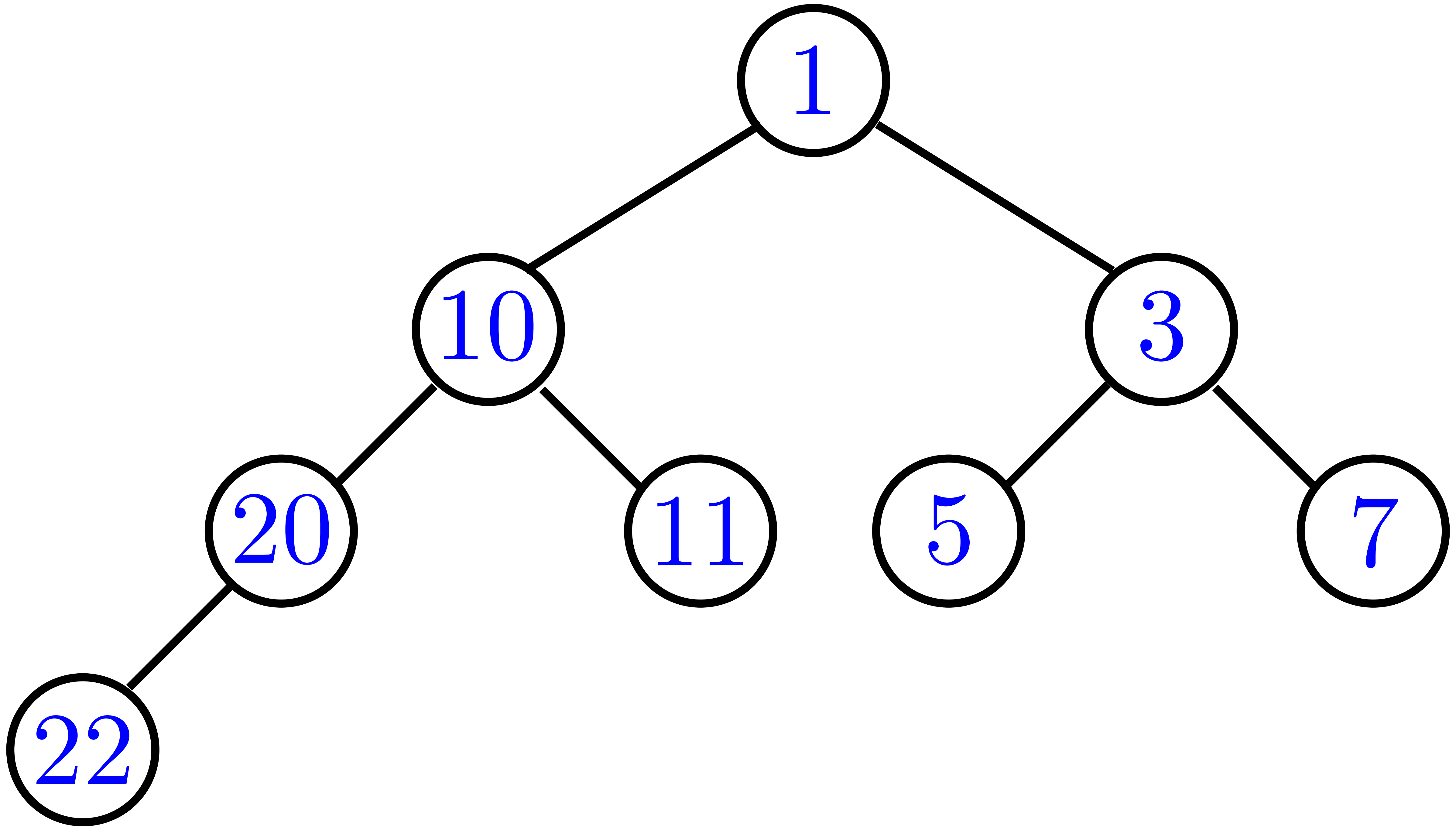
What is a binary heap?

Heap

We can implement a priority queue via a binary heap.

What is a binary heap?

Min binary heap example:



Invariants

Invariants are properties that are always satisfied by a data structure.

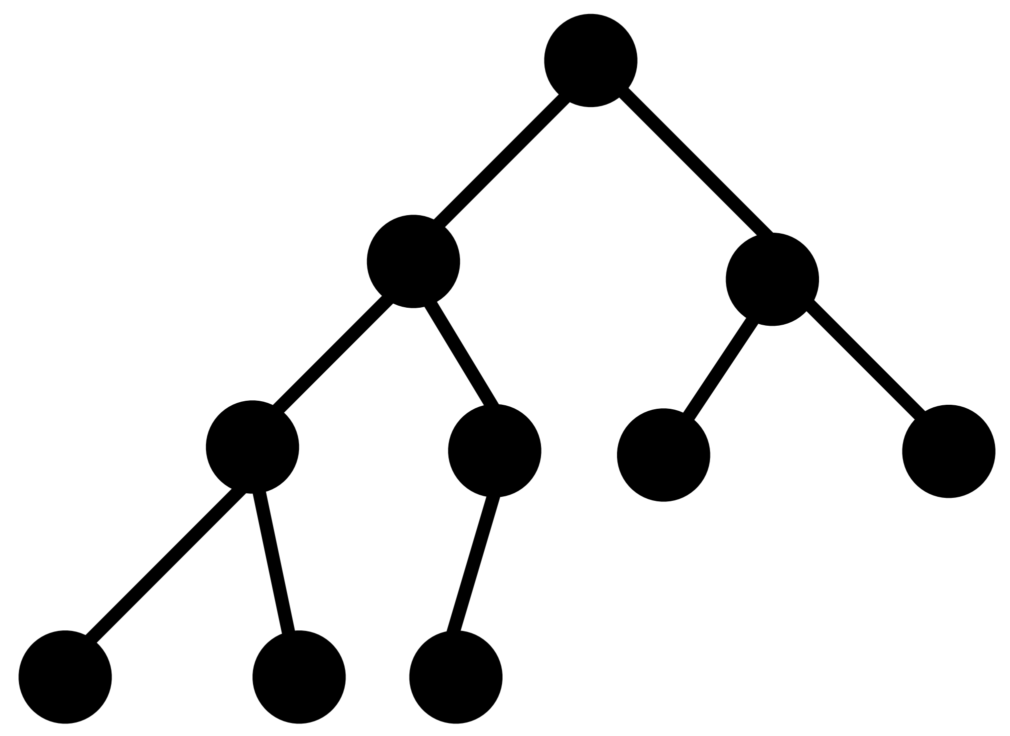
Proper operation of the data structure depends on the invariants holding.

These invariants have to be maintained in modifying (e.g. inserting or removing elements) from the data structure.

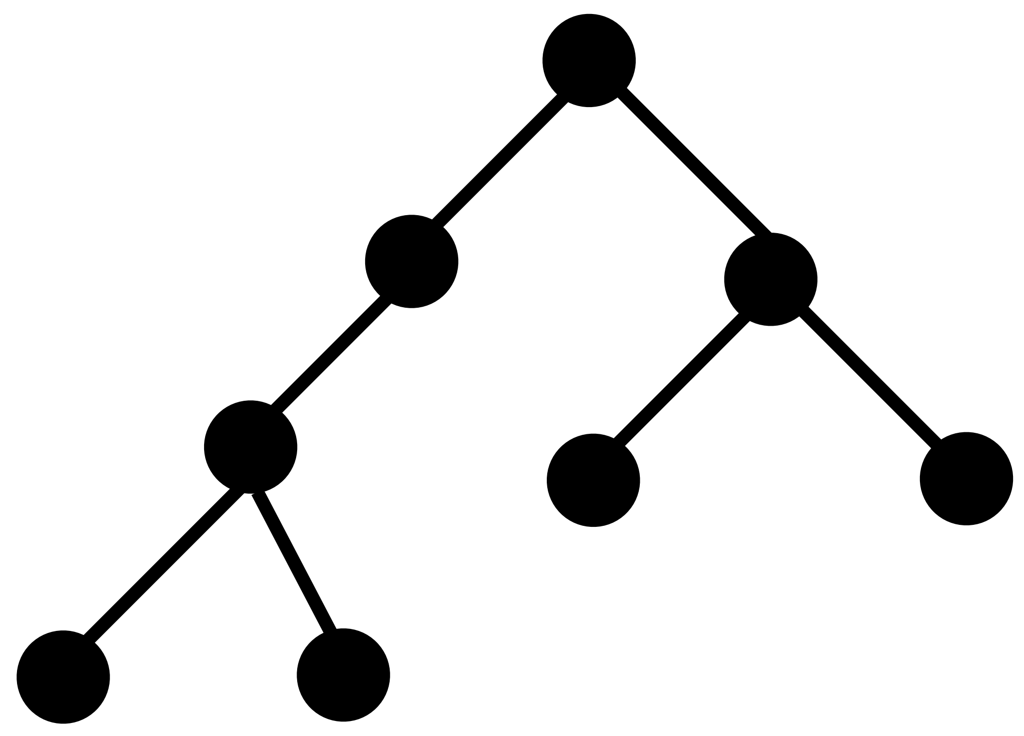
What invariants does a min heap satisfy?

Complete binary tree

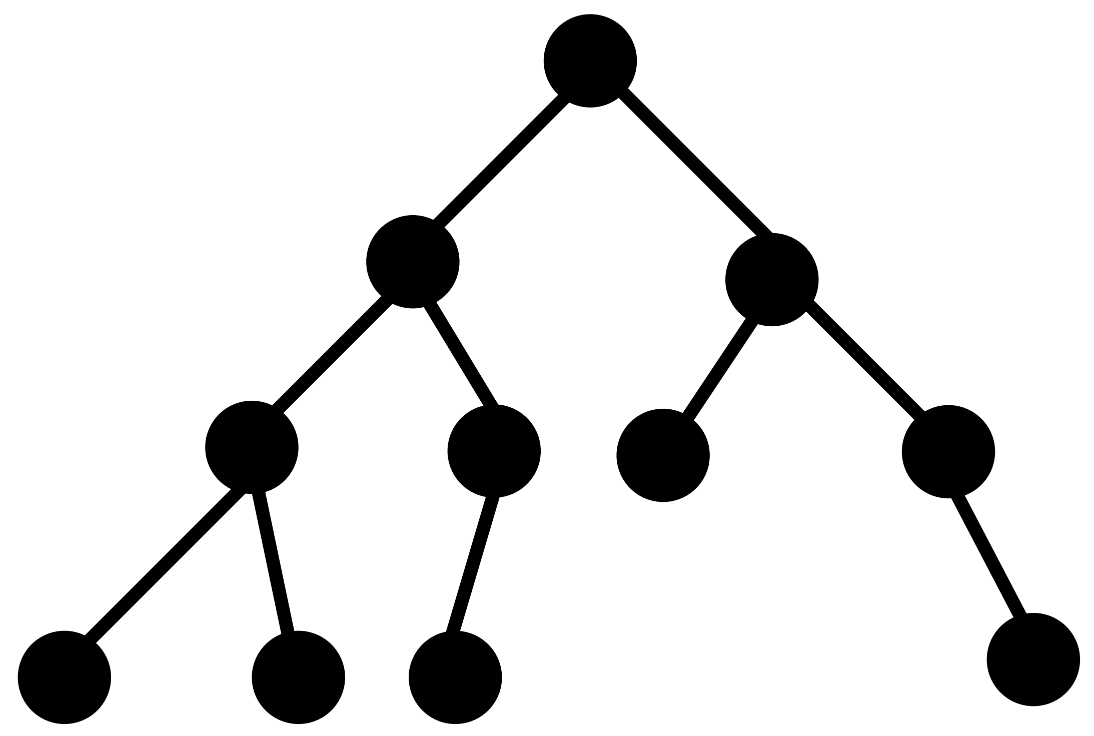
In a complete binary tree, every layer is totally filled except possibly the bottom one, which is filled from the left.



complete



not complete



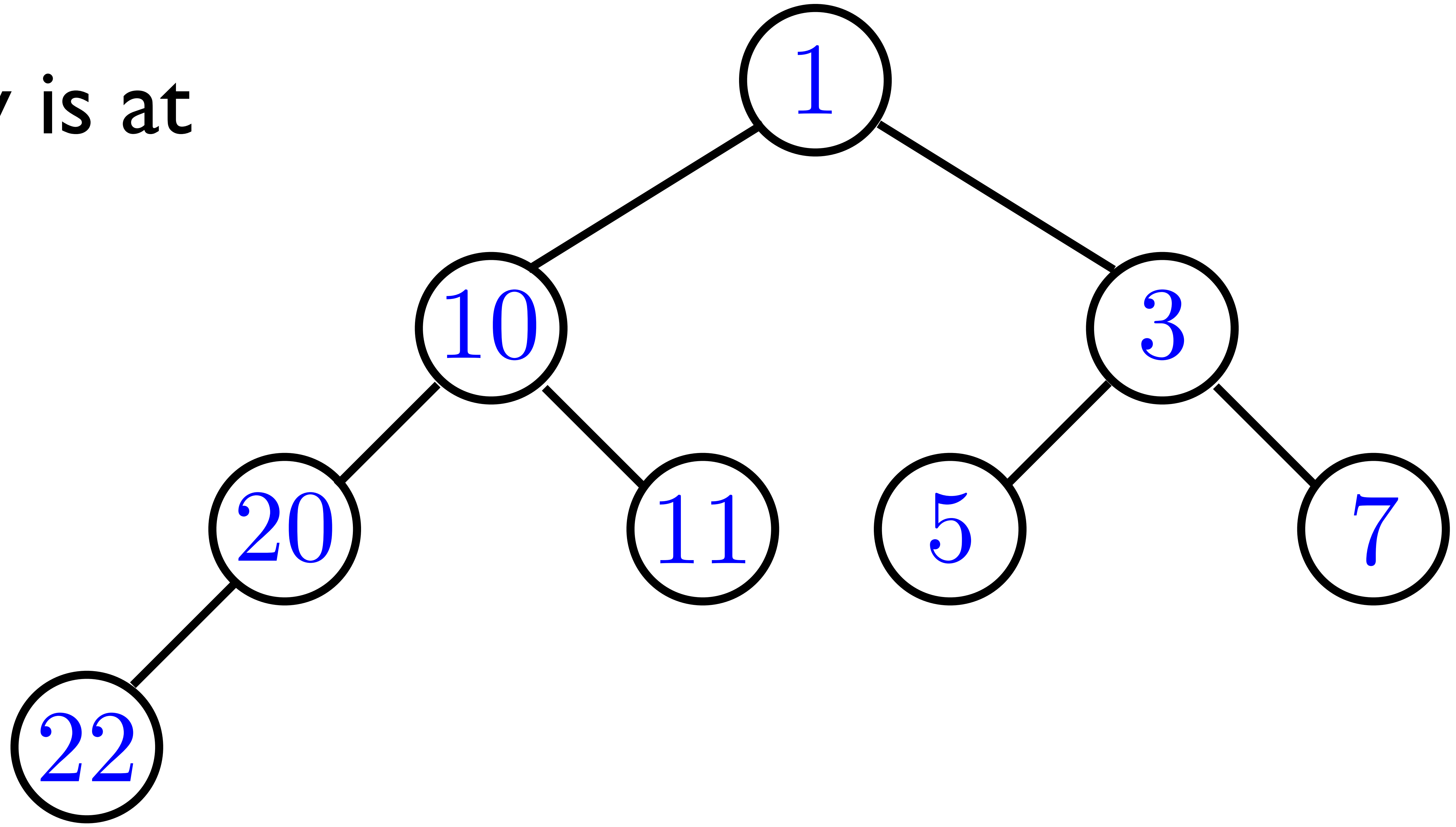
not complete

A complete binary tree with n nodes has height $\lfloor \log n \rfloor$.

Min Heap Property

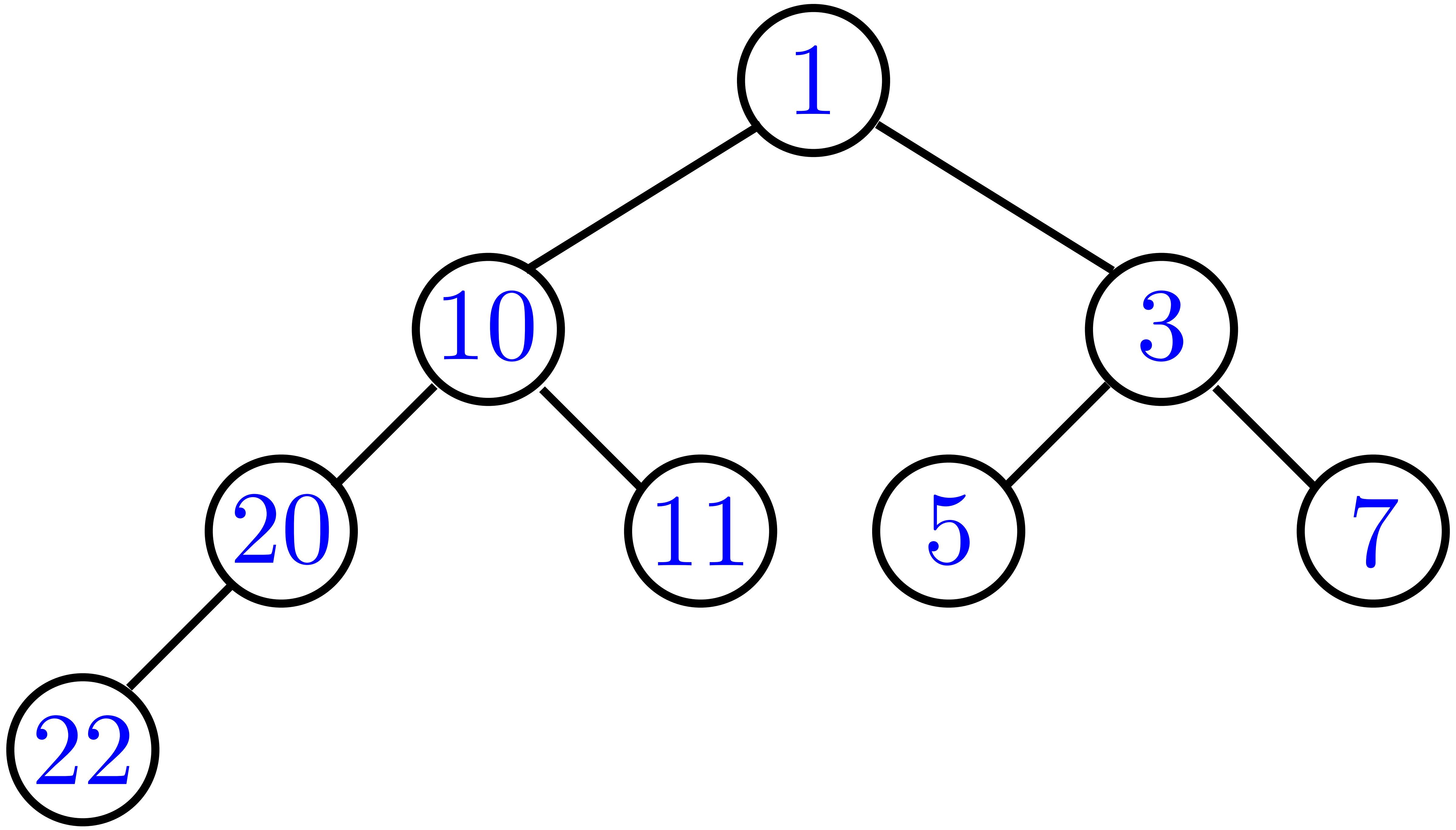
Min Heap Property: The key stored at each node is at most the keys of its children.

This guarantees that the minimum key is at the root.



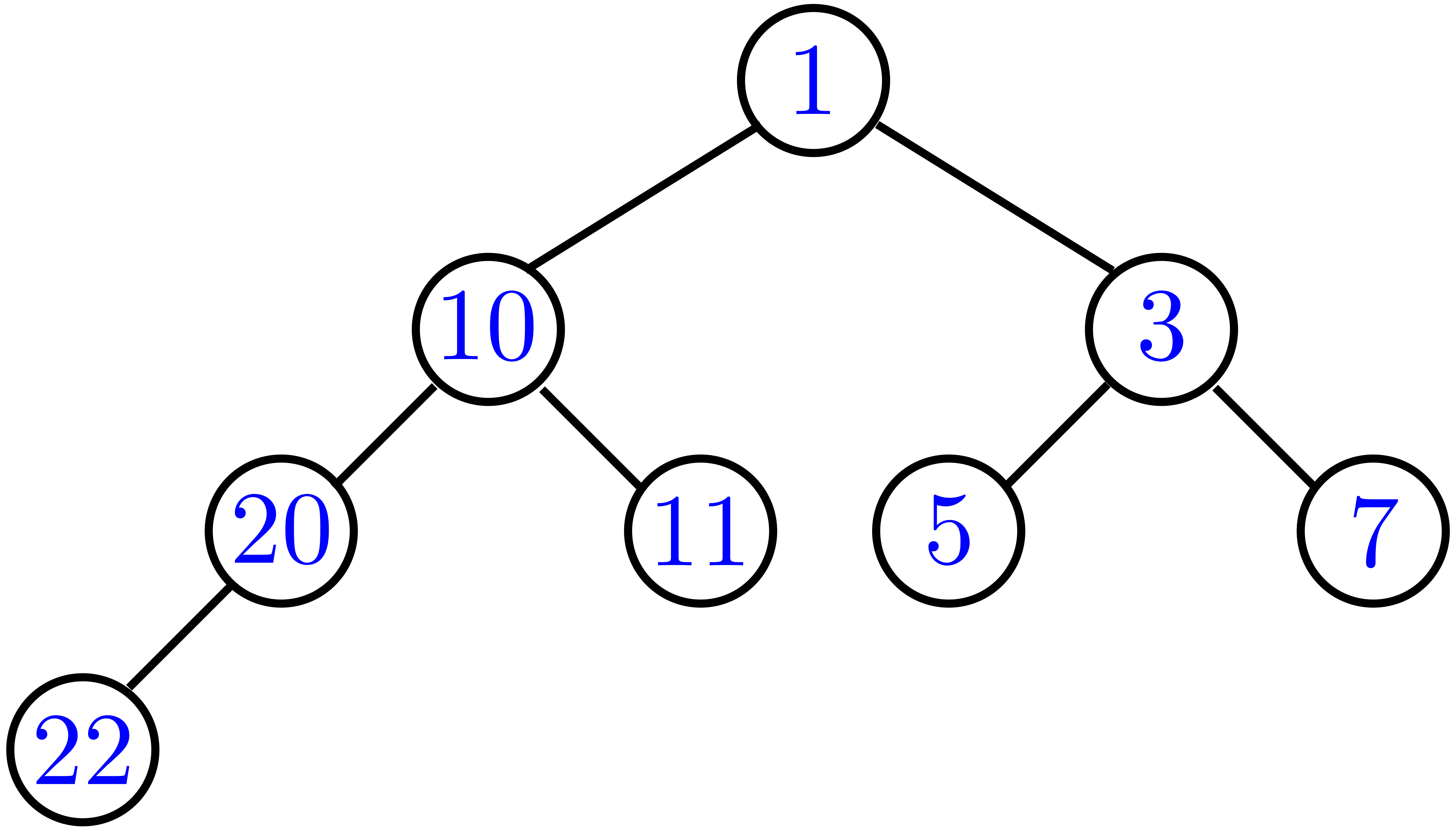
Operations

What operations does a min heap support?



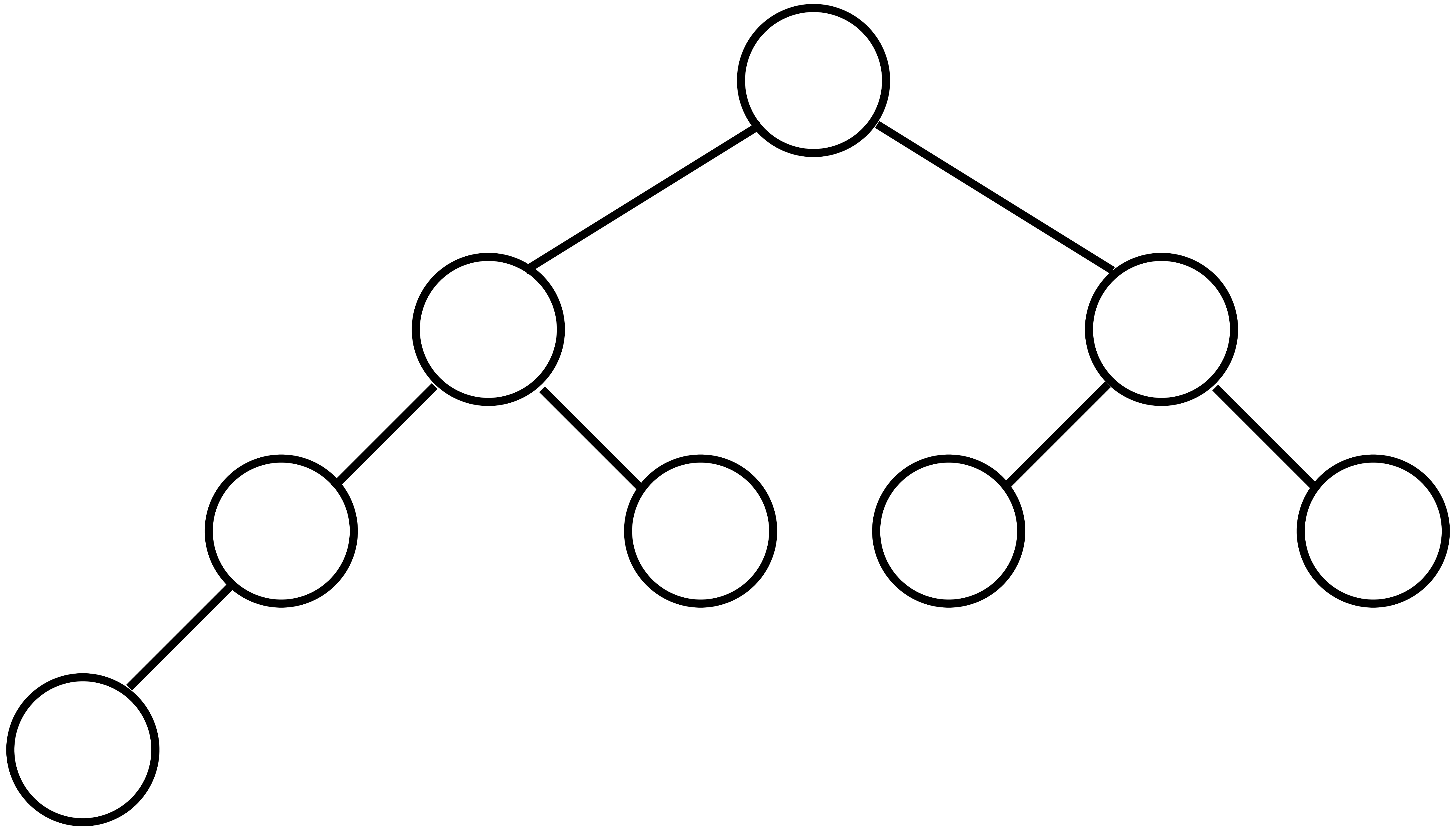
Peek (Top)

Return the minimum element.



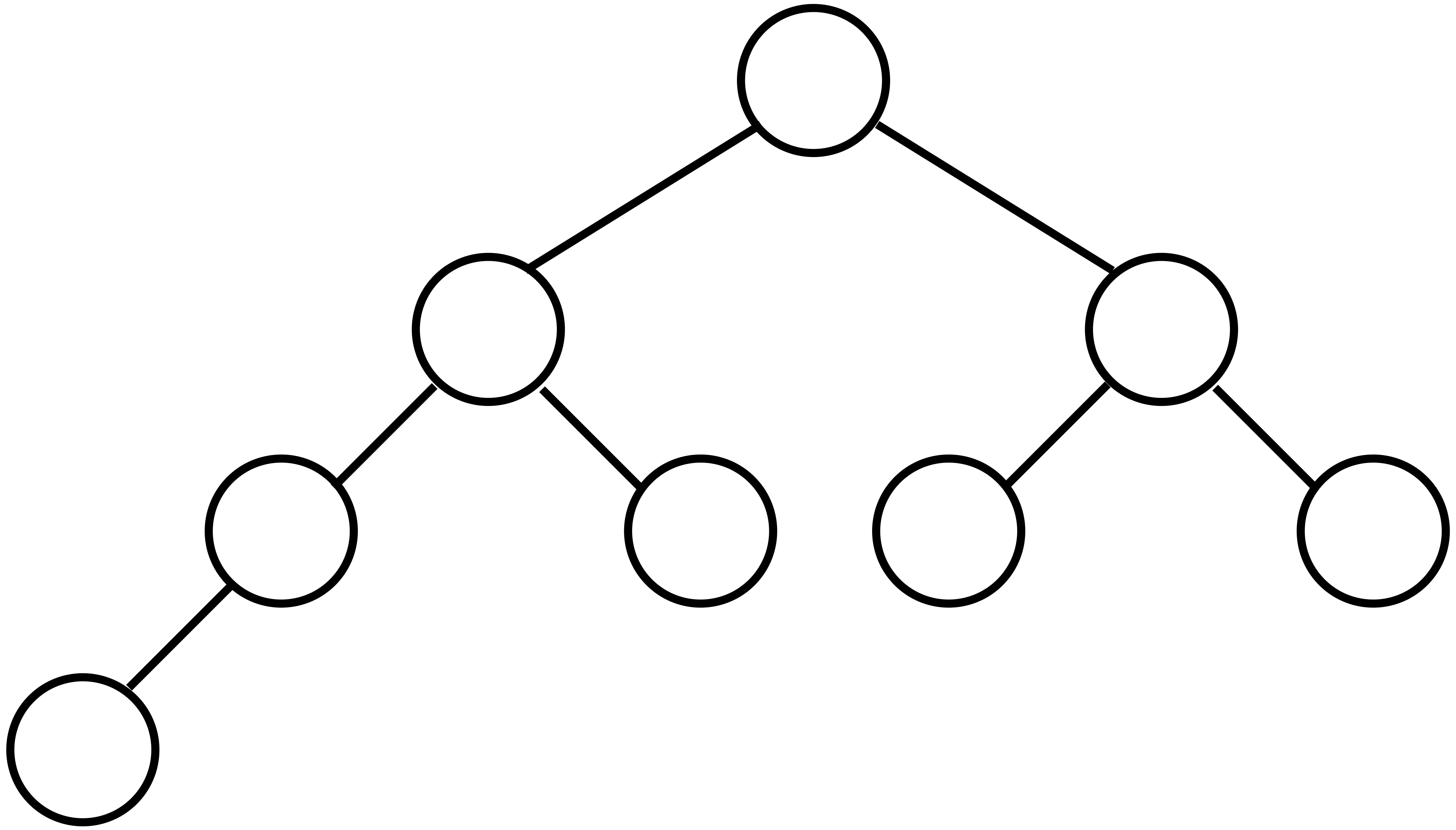
Insert (Push)

Let's insert a new key into our min heap.



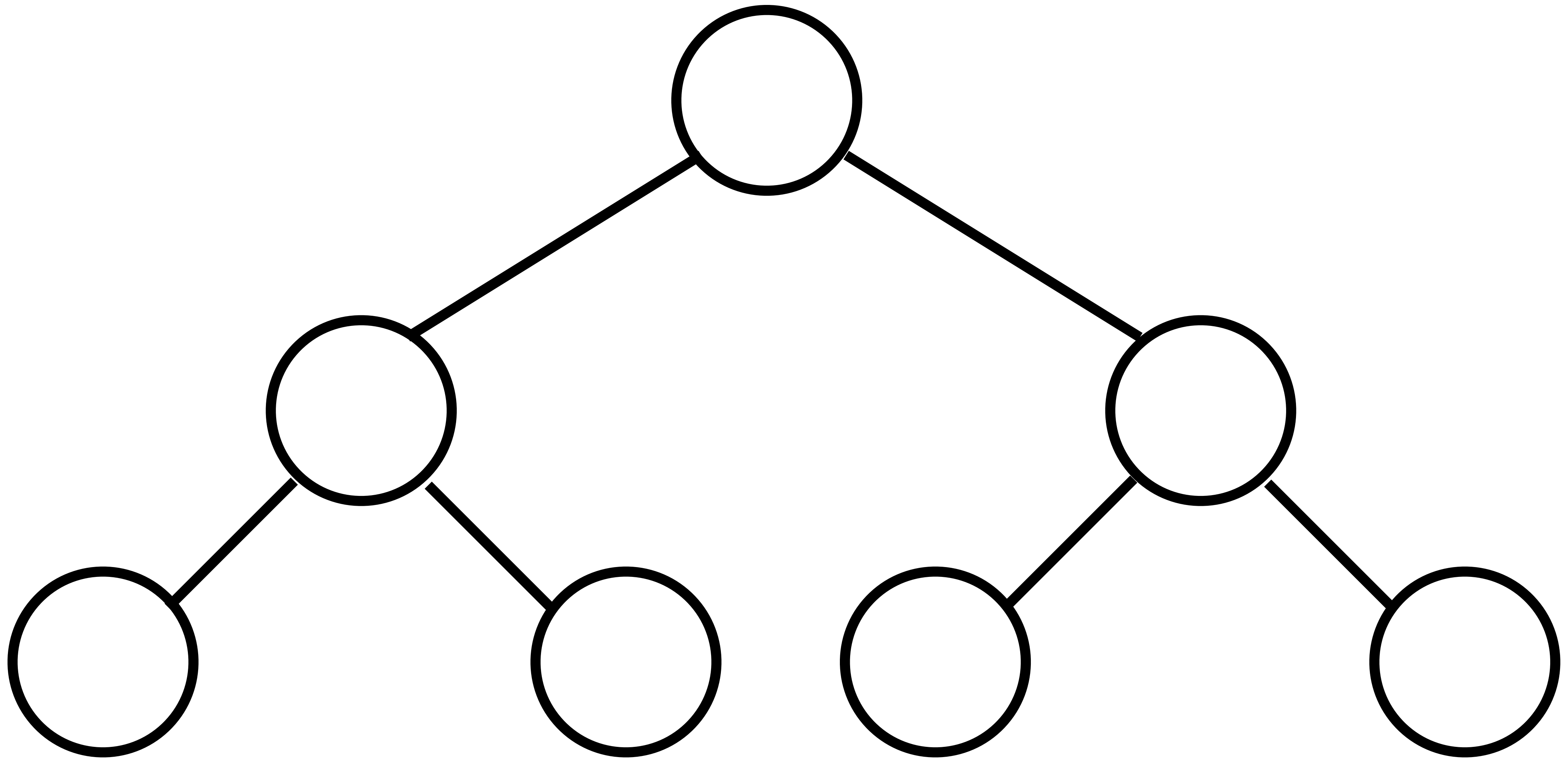
Remove the minimum (Pop)

Now let's see how to remove the minimum element in a min heap.



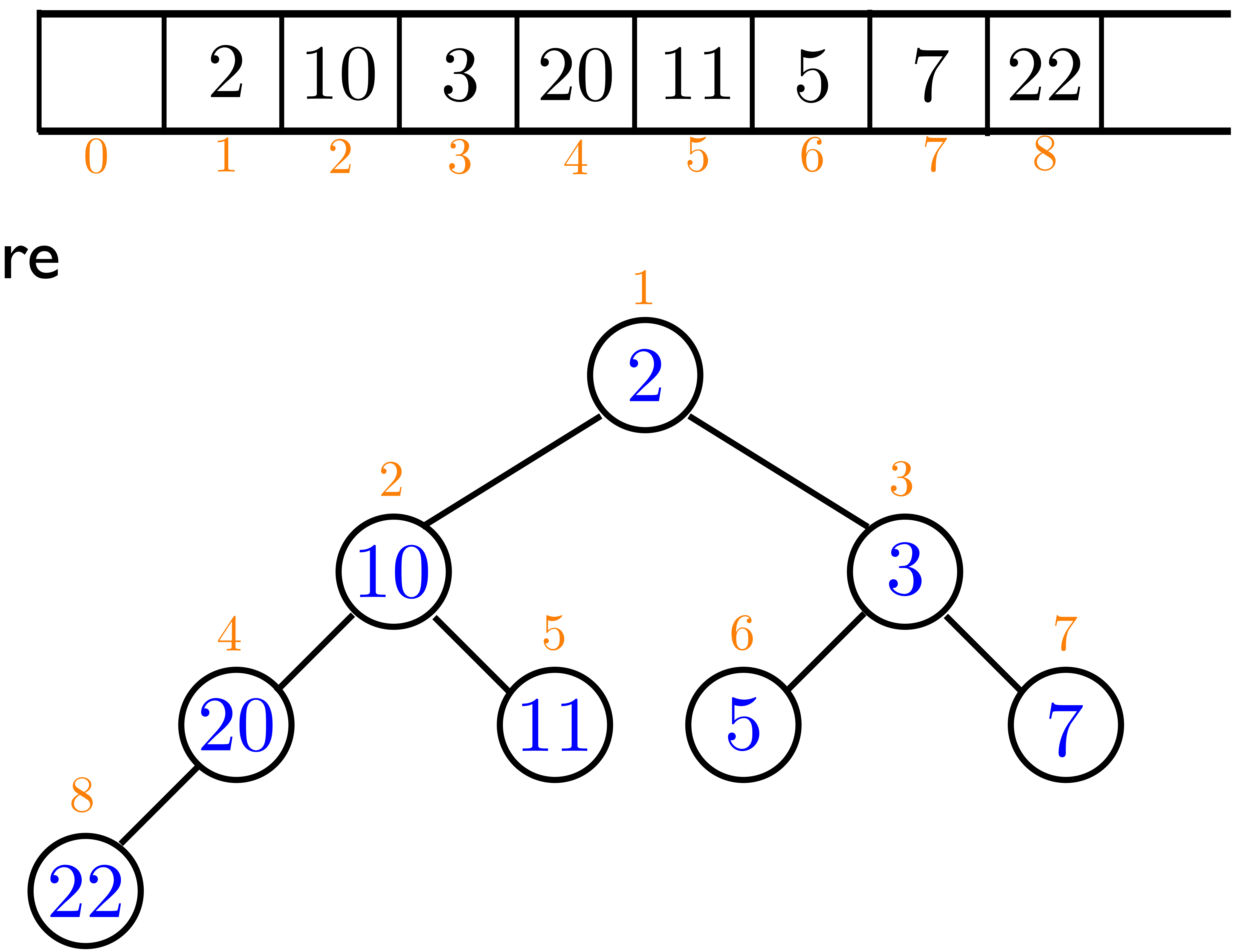
Remove the minimum

Now let's see how to remove the minimum element in a min heap.



A nice feature of heaps is that they are relatively simple to implement.

We can represent the heap by a vector.



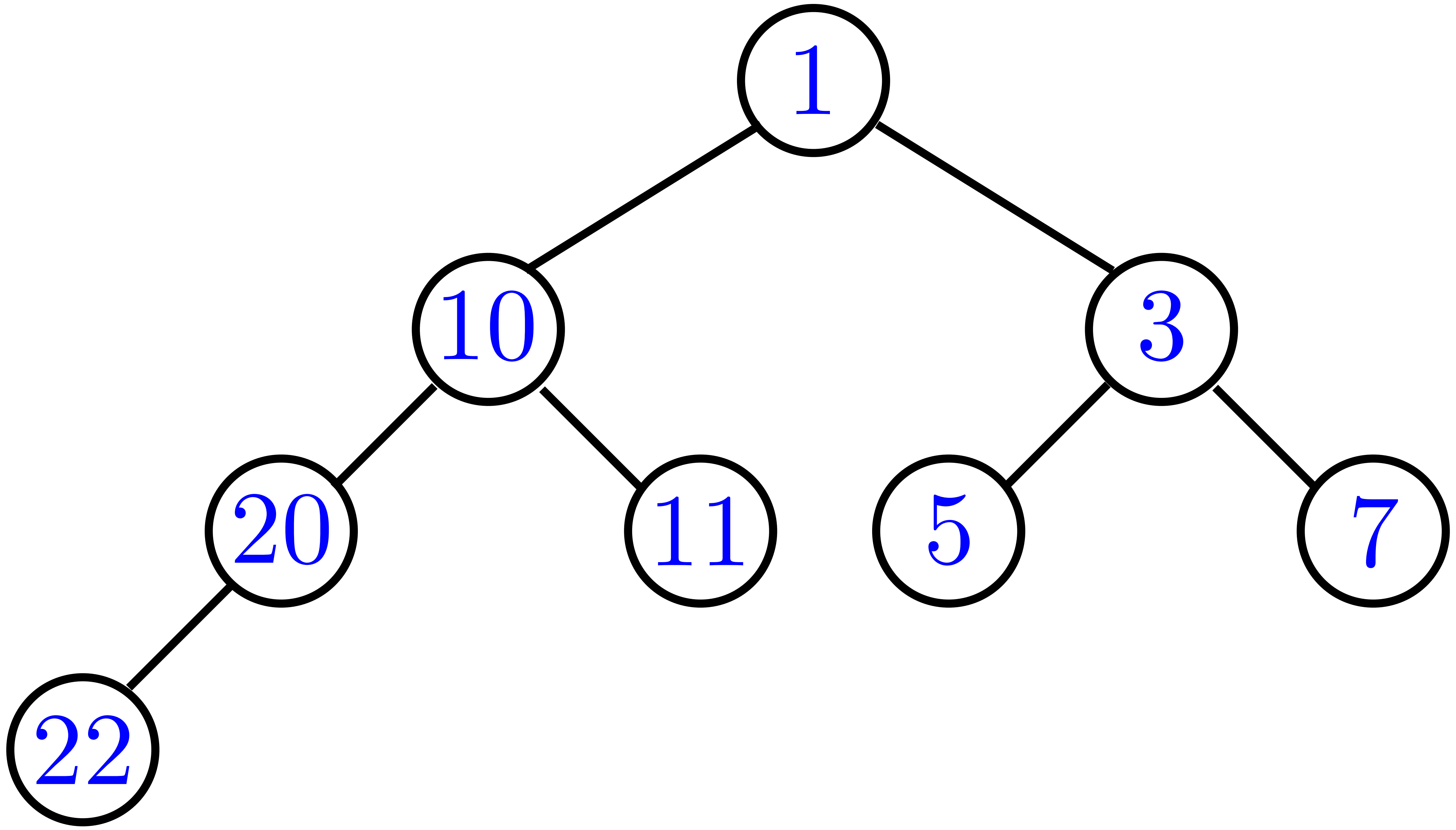
Complexity

What is the complexity of our 3 operations on a min heap?

peek:

insert:

remove_min:



Heapsort

Let's see how we can use a heap to sort a vector of n elements.

	3	7	6	5	3	5

Heapsort

Let's see how we can use a heap to sort a vector of n elements.

	3	7	6	5	3	5

To sort the vector from smallest to largest, it is best to use a max heap.

Heapsort

Let's see how we can use a heap to sort a vector of n elements.

	3	7	6	5	3	5

To sort the vector from smallest to largest, it is best to use a max heap.

In a max heap the key at a node is not smaller than the keys of its children.

Heapsort

Let's see how we can use a heap to sort a vector of n elements.

	3	7	6	5	3	5

To sort the vector from smallest to largest, it is best to use a max heap.

In a max heap the key at a node is not smaller than the keys of its children.

In a max heap, the root holds the largest element.

Heapsort

Heapsort consists of two phases. In the first phase we create a max heap with the elements of the vector.

	3	6	7	5	3	5

We grow a heap by inserting each element of the vector into it.

Heapsort

Heapsort consists of two phases. In the first phase we create a max heap with the elements of the vector.

	3	6	7	5	3	5

We grow a heap by inserting each element of the vector into it.

Heapsort

Heapsort consists of two phases. In the first phase we create a max heap with the elements of the vector.

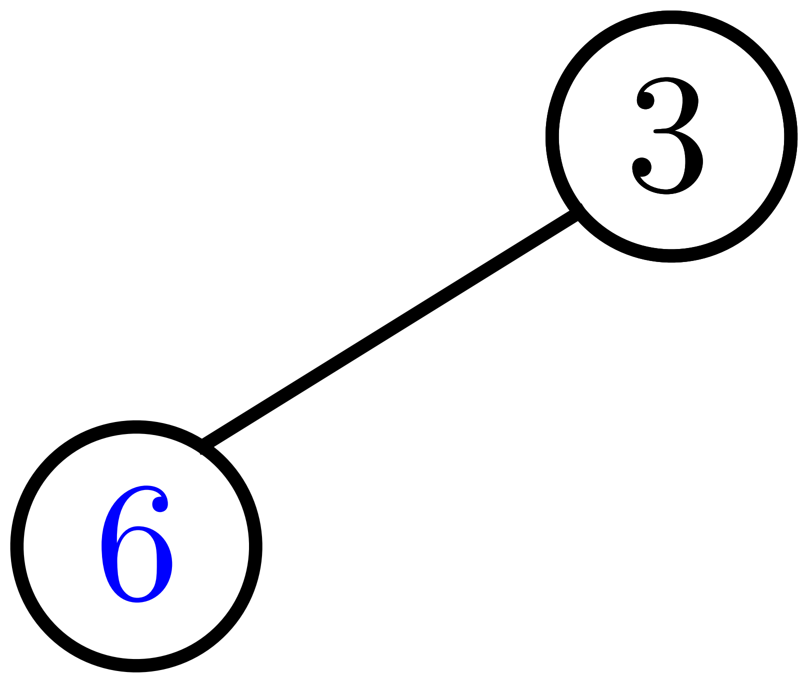
We grow a heap by inserting each element of the vector into it.

Not much to do with the first element.

Heapsort

In the first phase we create a max heap with the elements of the vector.

	3	6	7	5	3	5

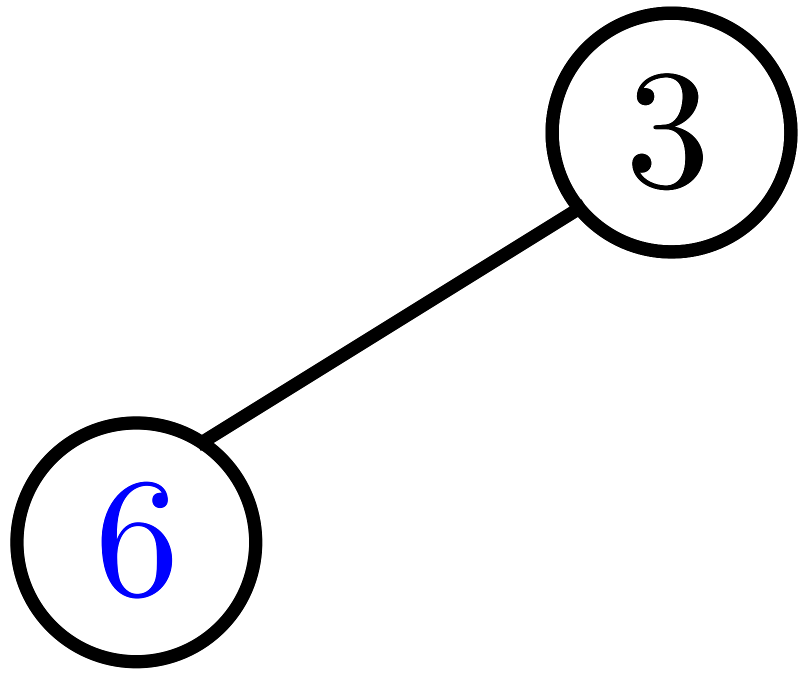


Next we insert 6 .

Heapsort

In the first phase we create a max heap with the elements of the vector.

	3	6	7	5	3	5



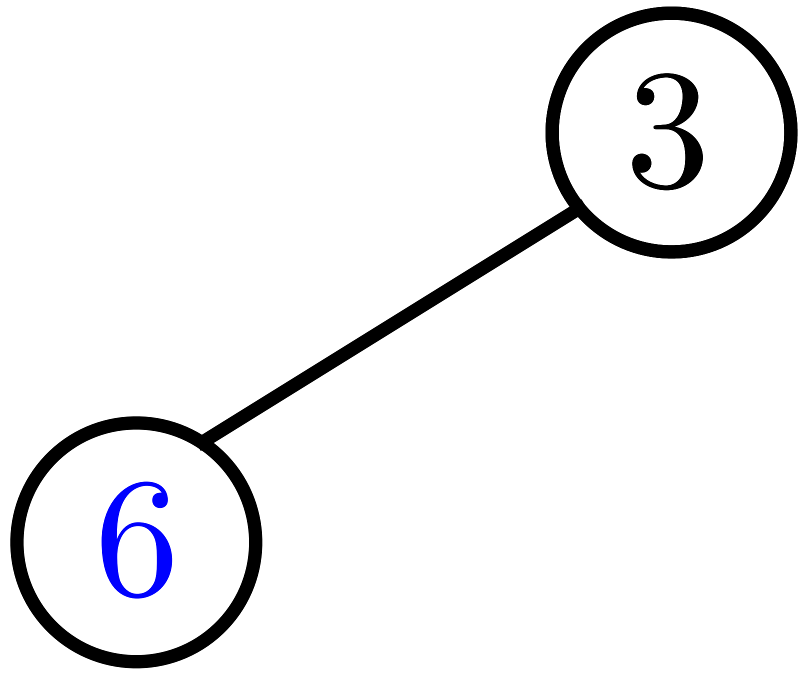
Next we insert 6.

Now we "swim" with 6.

Heapsort

In the first phase we create a max heap with the elements of the vector.

	3	6	7	5	3	5



Next we insert 6.

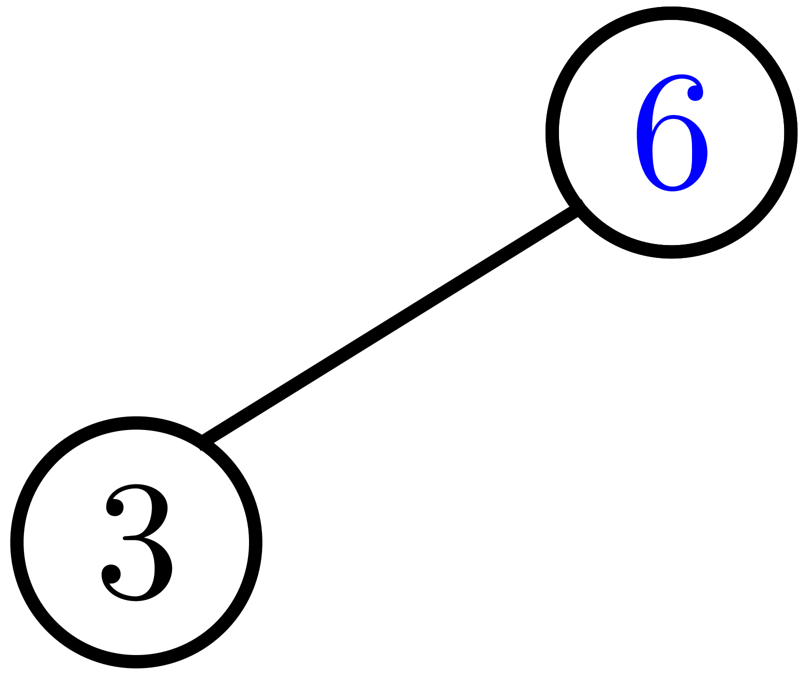
Now we "swim" with 6.

Is $3 < 6$? Yes, so the max heap property is violated. We swap them.

Heapsort

In the first phase we create a max heap with the elements of the vector.

	6	3	7	5	3	5

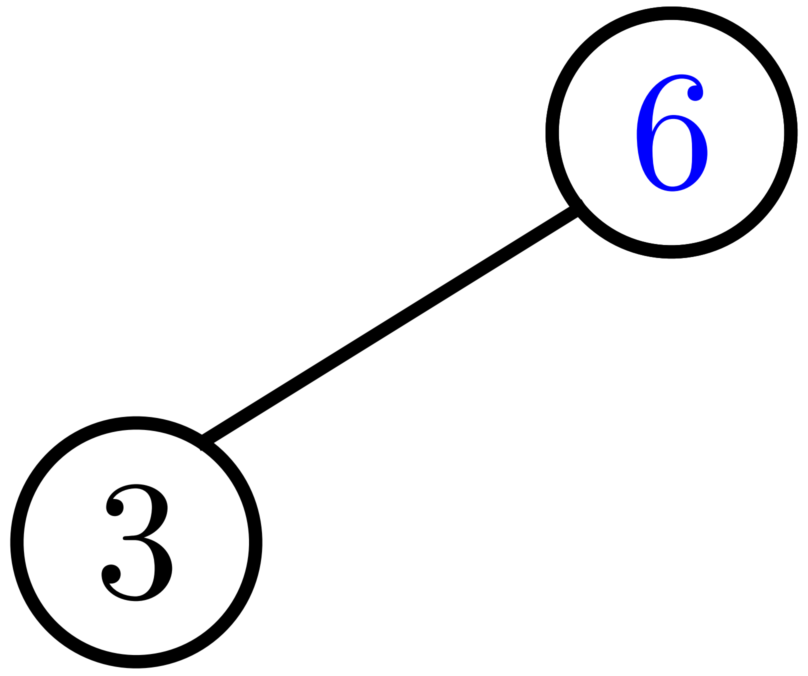


Now we have a max heap with the first two elements of the vector.

Heapsort

In the first phase we create a max heap with the elements of the vector.

	6	3	7	5	3	5



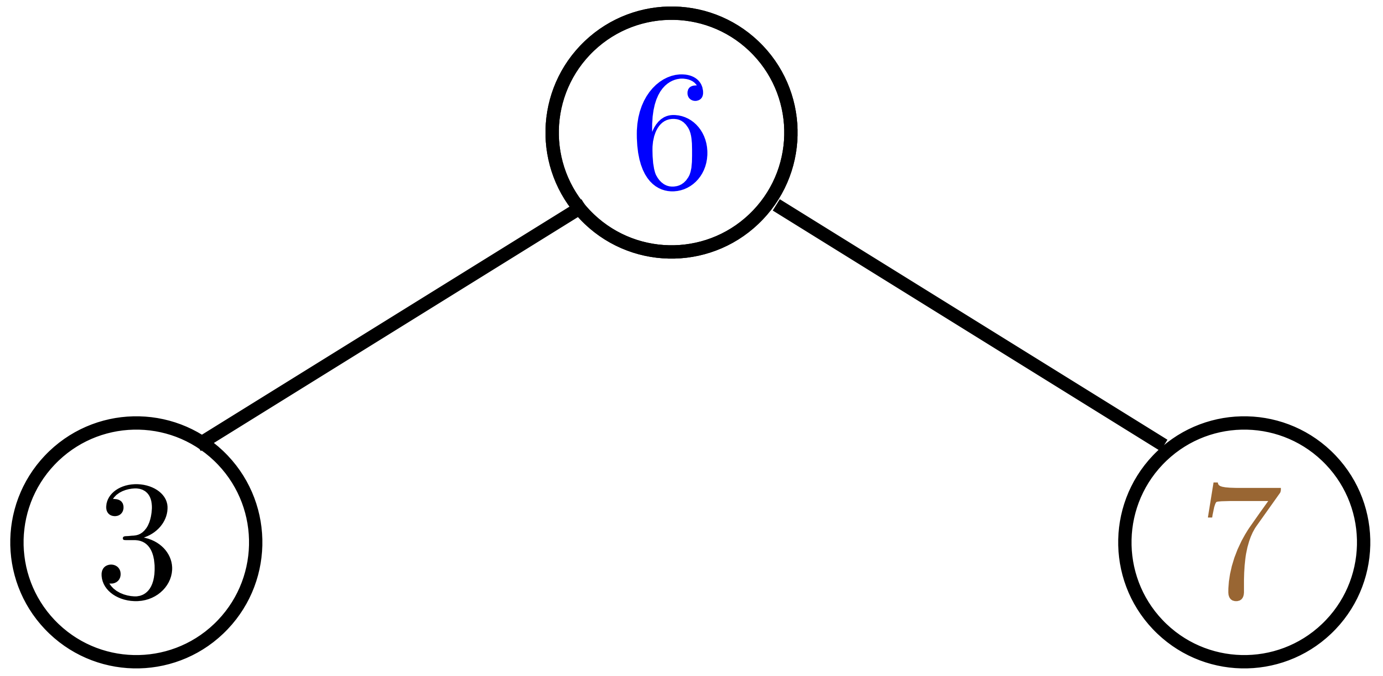
Next we insert 7 into the heap.

Heapsort

In the first phase we create a max heap with the elements of the vector.

	6	3	7	5	3	5

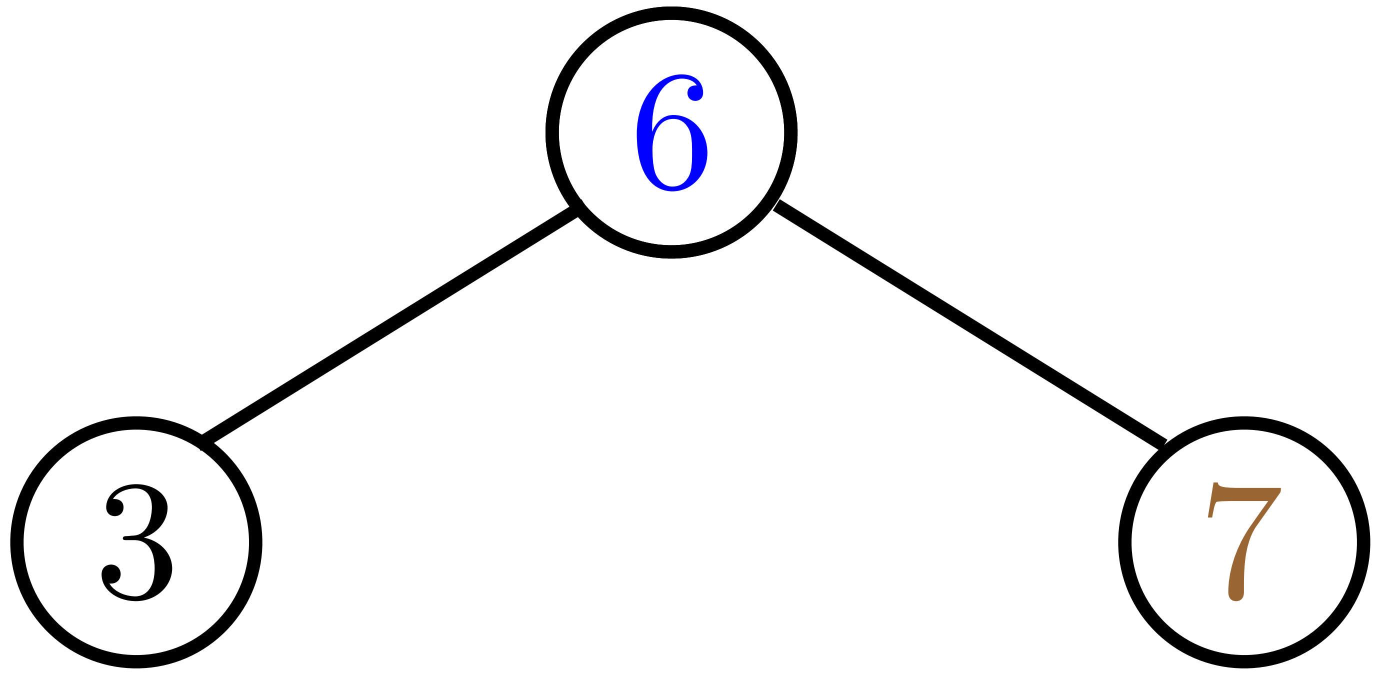
Next we insert 7 into the heap.



Heapsort

In the first phase we create a max heap with the elements of the vector.

	6	3	7	5	3	5



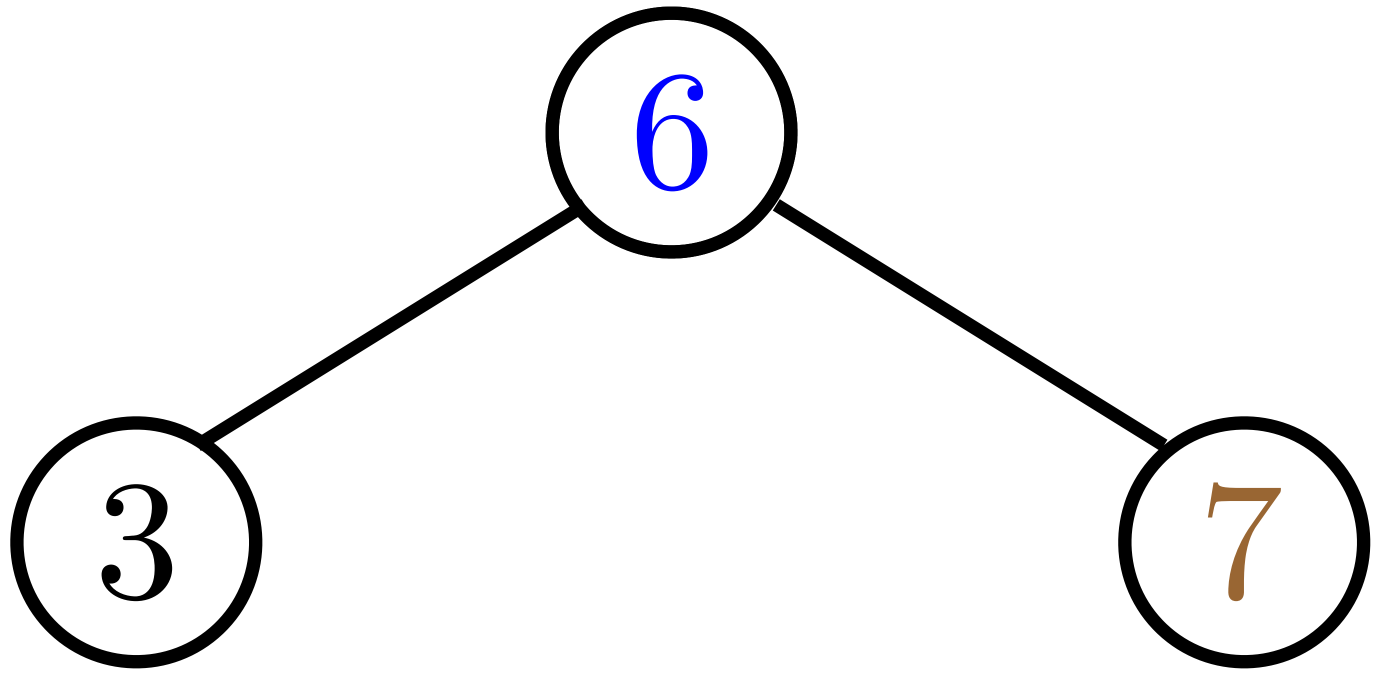
Next we insert 7 into the heap.

Now "swim" with 7.

Heapsort

In the first phase we create a max heap with the elements of the vector.

	6	3	7	5	3	5



Next we insert 7 into the heap.

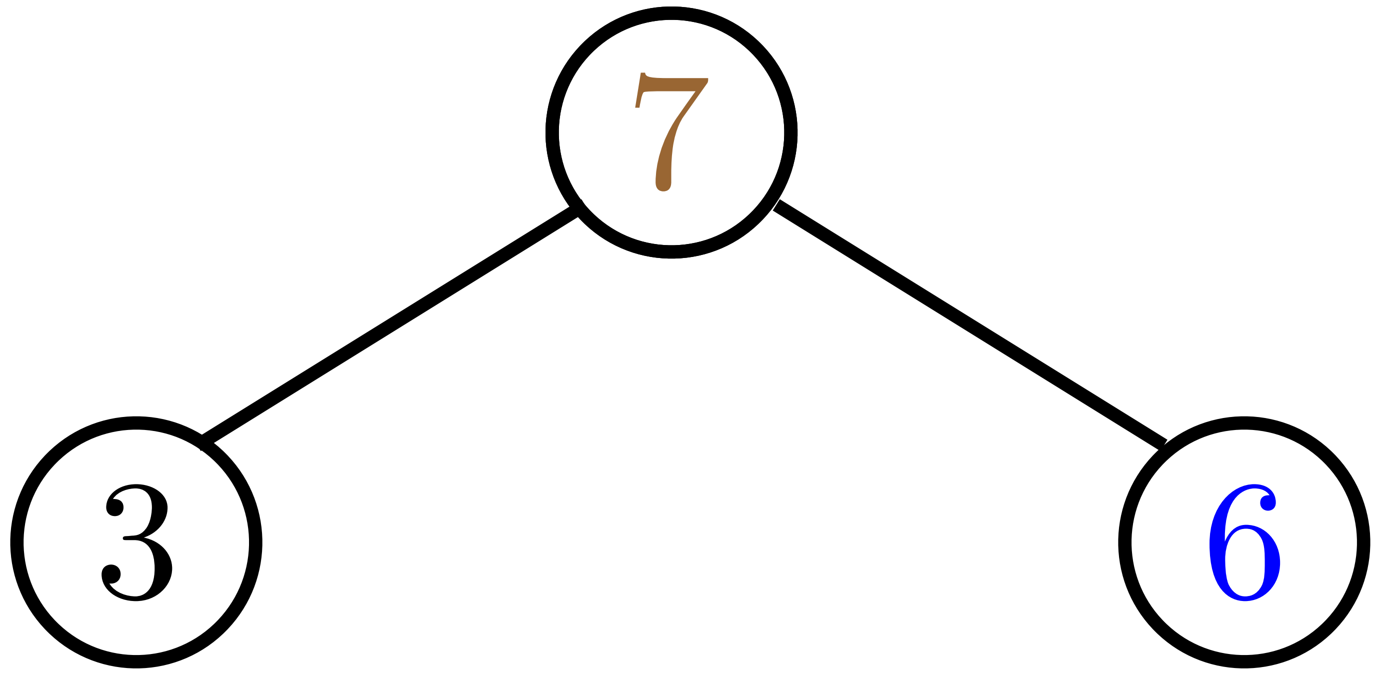
Now "swim" with 7.

Is $6 < 7$? Yes, so the max heap property is violated. We swap them.

Heapsort

In the first phase we create a max heap with the elements of the vector.

	7	3	6	5	3	5

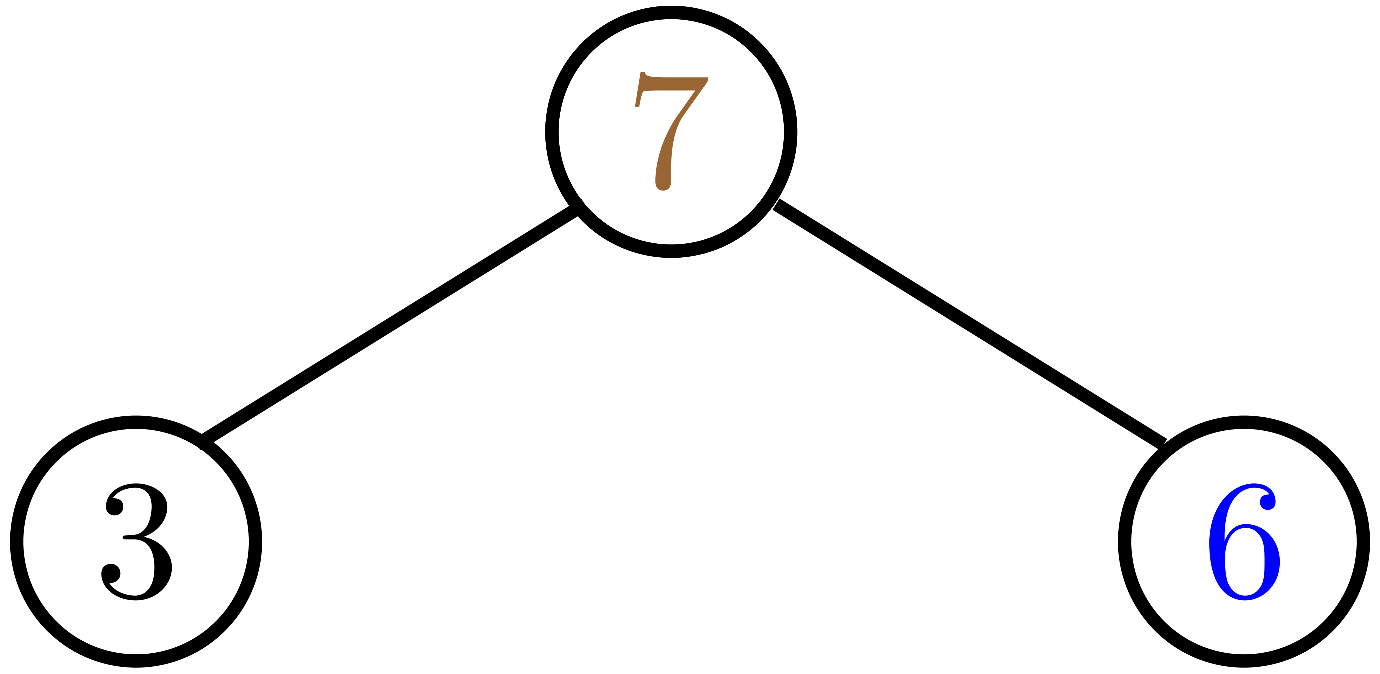


Now we have a max heap with the first three elements of the vector.

Heapsort

In the first phase we create a max heap with the elements of the vector.

	7	3	6	5	3	5



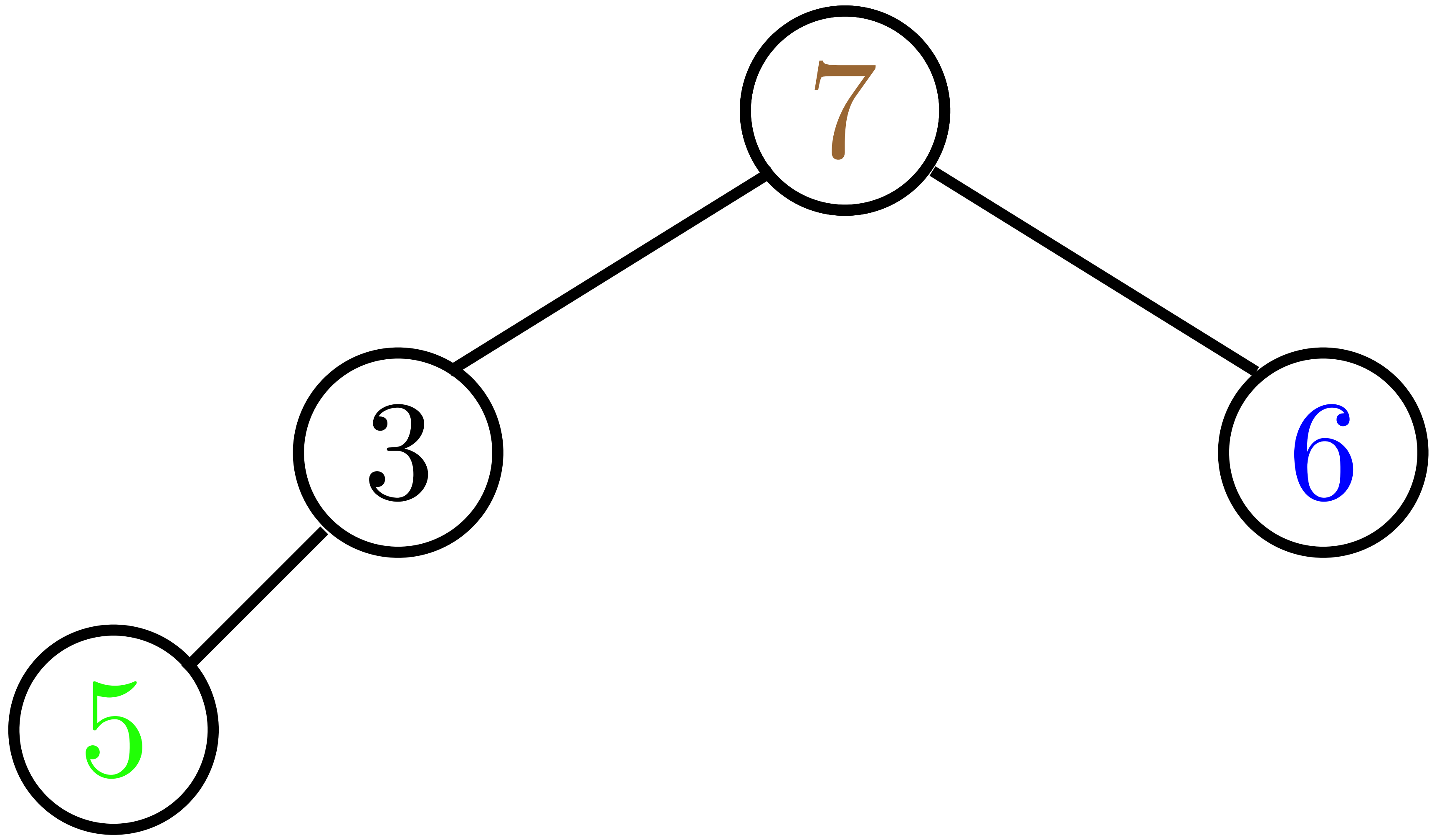
Next we insert 5.

Heapsort

In the first phase we create a max heap with the elements of the vector.

	7	3	6	5	3	5

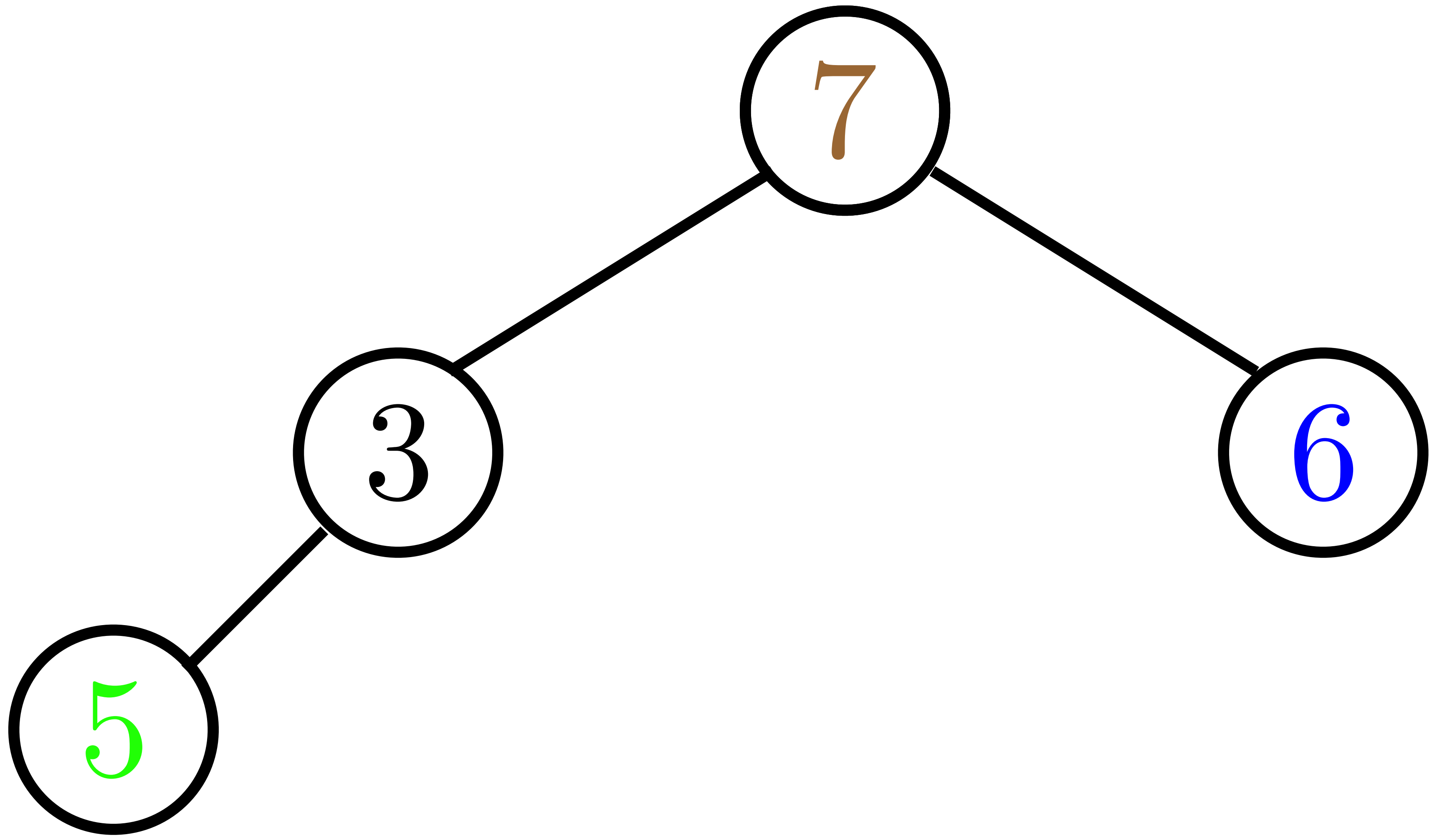
Next we insert 5.



Heapsort

In the first phase we create a max heap with the elements of the vector.

	7	3	6	5	3	5



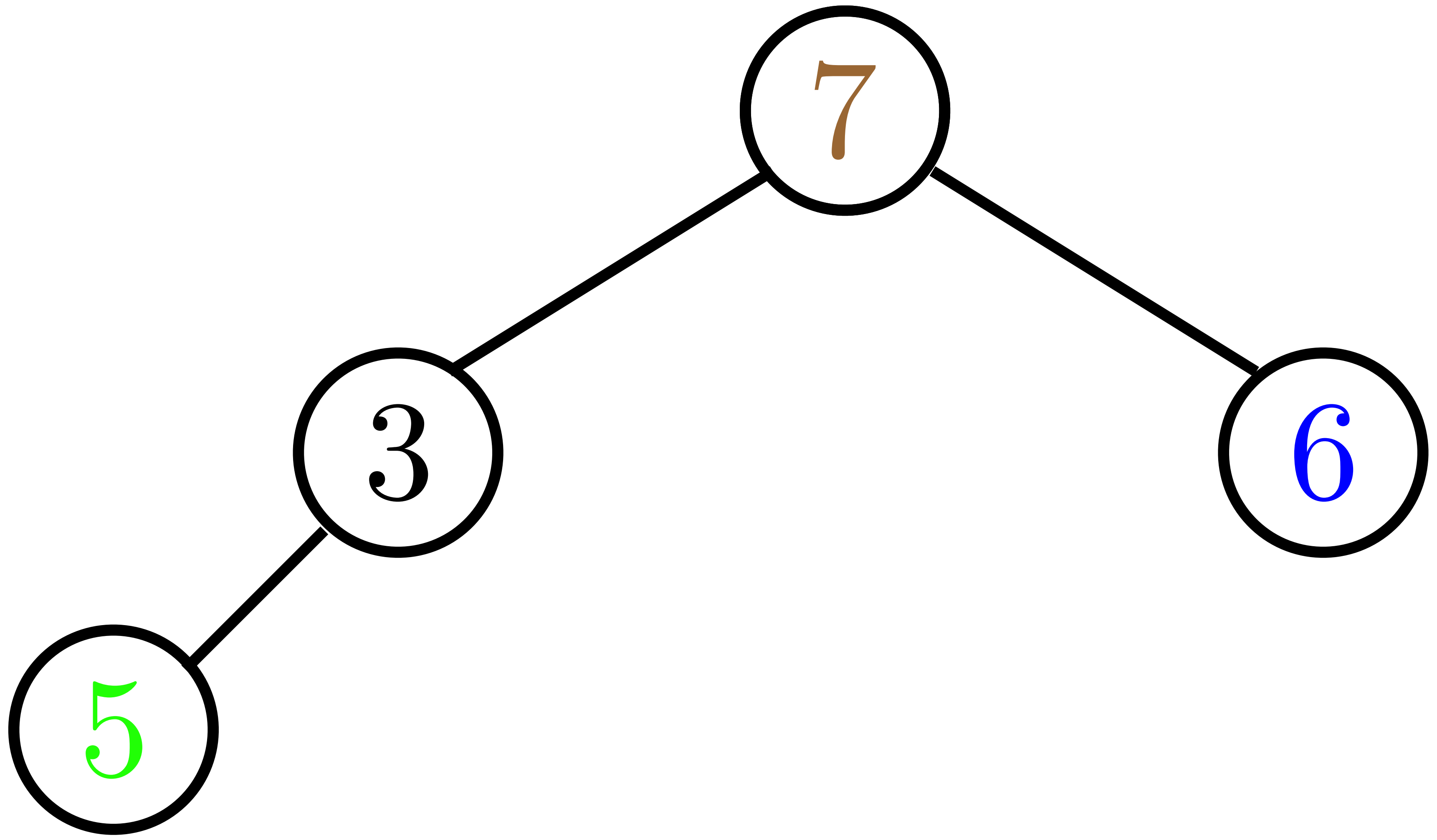
Next we insert 5.

Swim with 5 .

Heapsort

In the first phase we create a max heap with the elements of the vector.

	7	3	6	5	3	5



Next we insert 5.

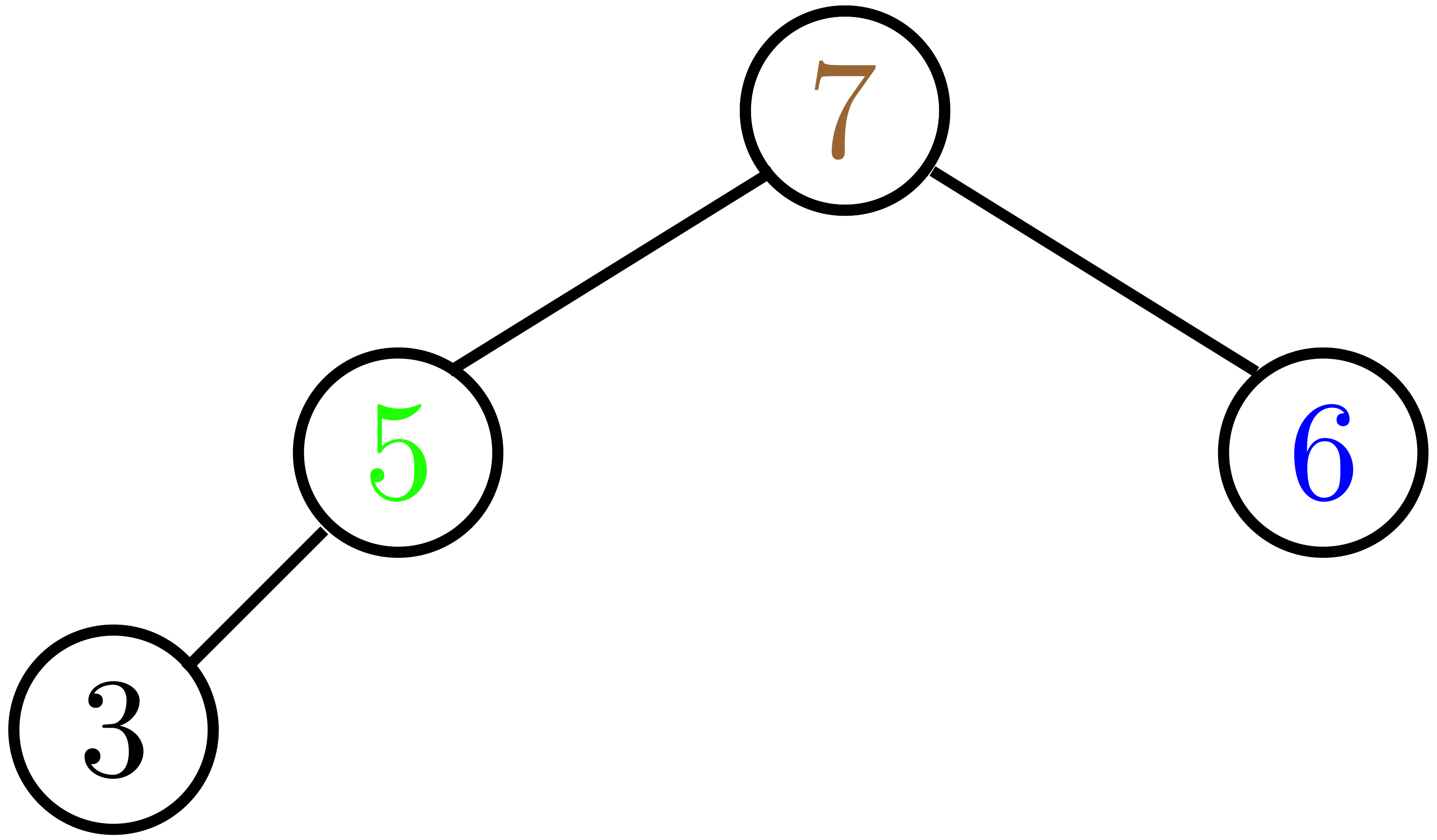
Swim with 5 .

Is $3 < 5$? Yes, so the max heap property is violated. We swap them.

Heapsort

In the first phase we create a max heap with the elements of the vector.

	7	5	6	3	3	5



Swim with 5 .

Is $7 < 5$?

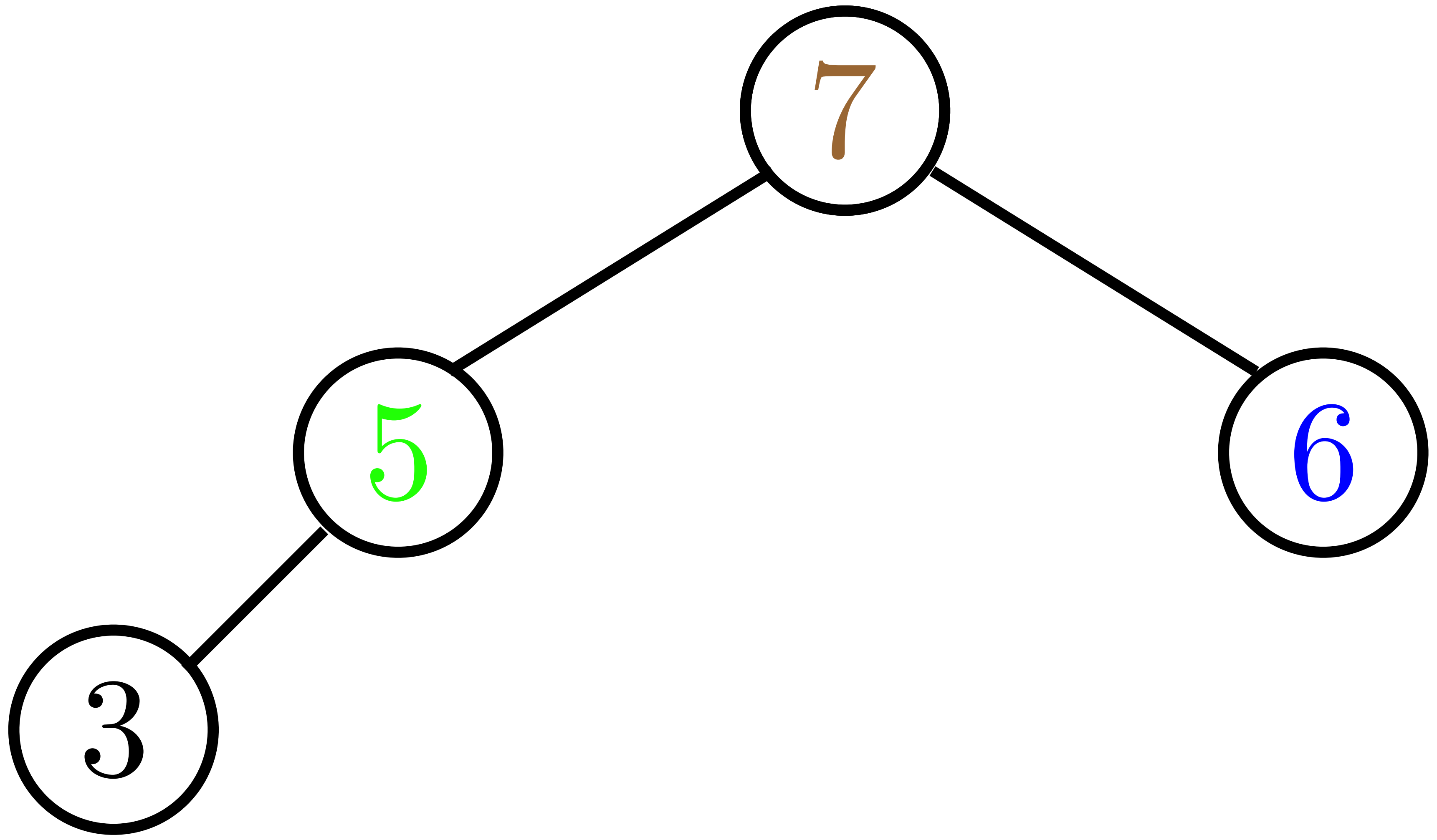
No, we have a max heap on the first four elements of the vector.

Heapsort

In the first phase we create a max heap with the elements of the vector.

	7	5	6	3	3	5

Next we insert 3 .

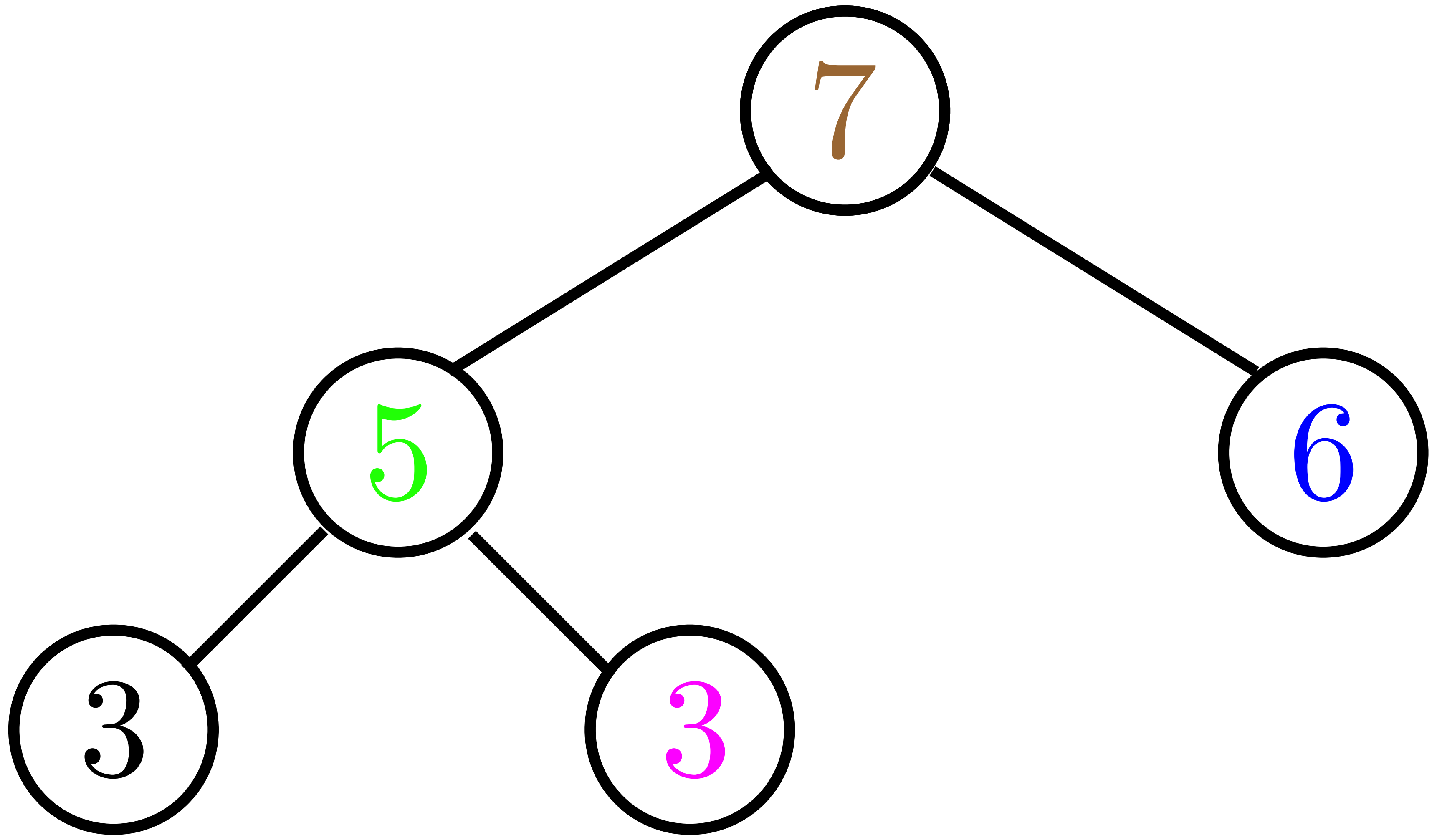


Heapsort

In the first phase we create a max heap with the elements of the vector.

	7	5	6	3	3	5

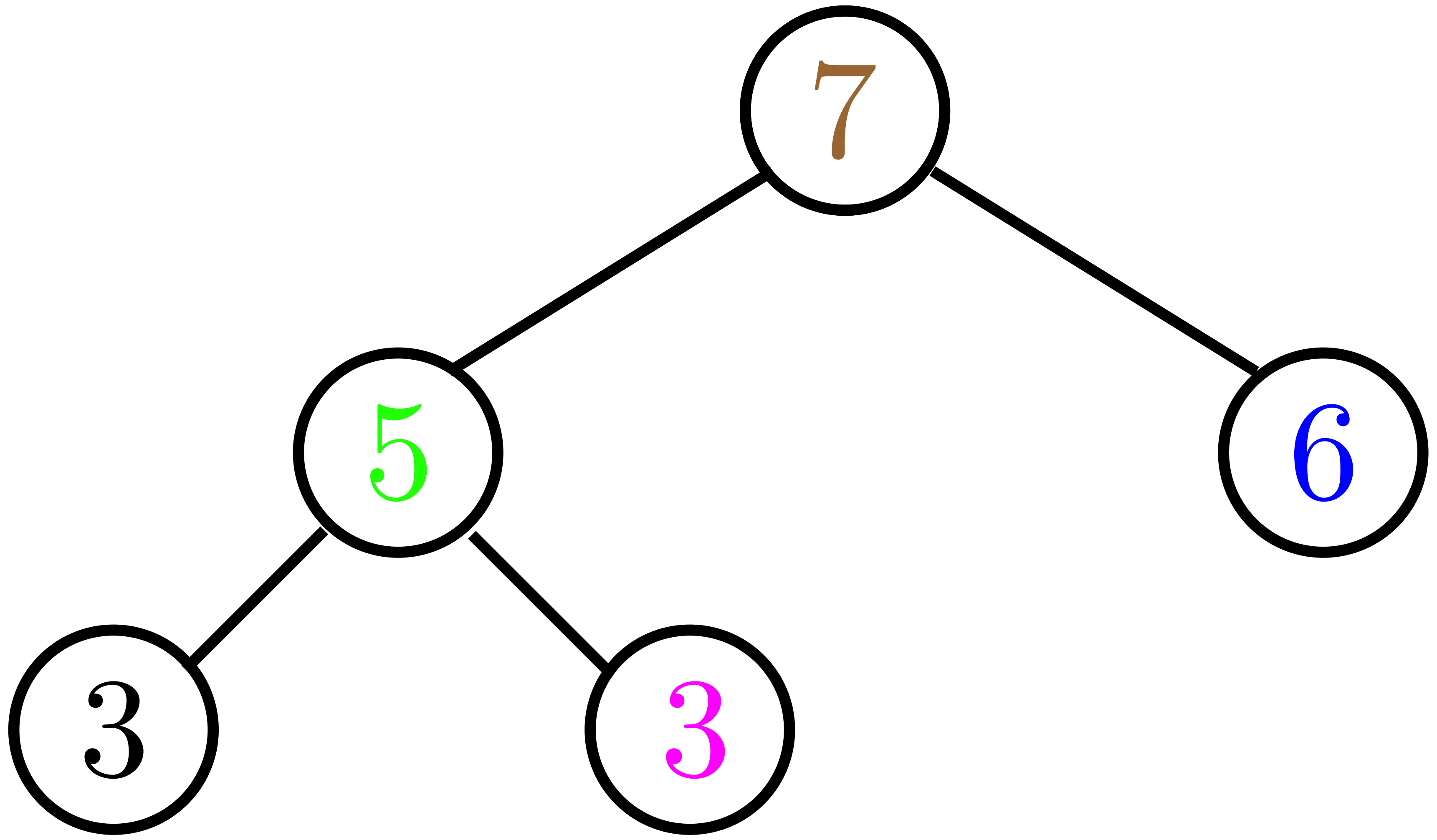
Next we insert 3 .



Heapsort

In the first phase we create a max heap with the elements of the vector.

	7	5	6	3	3	5



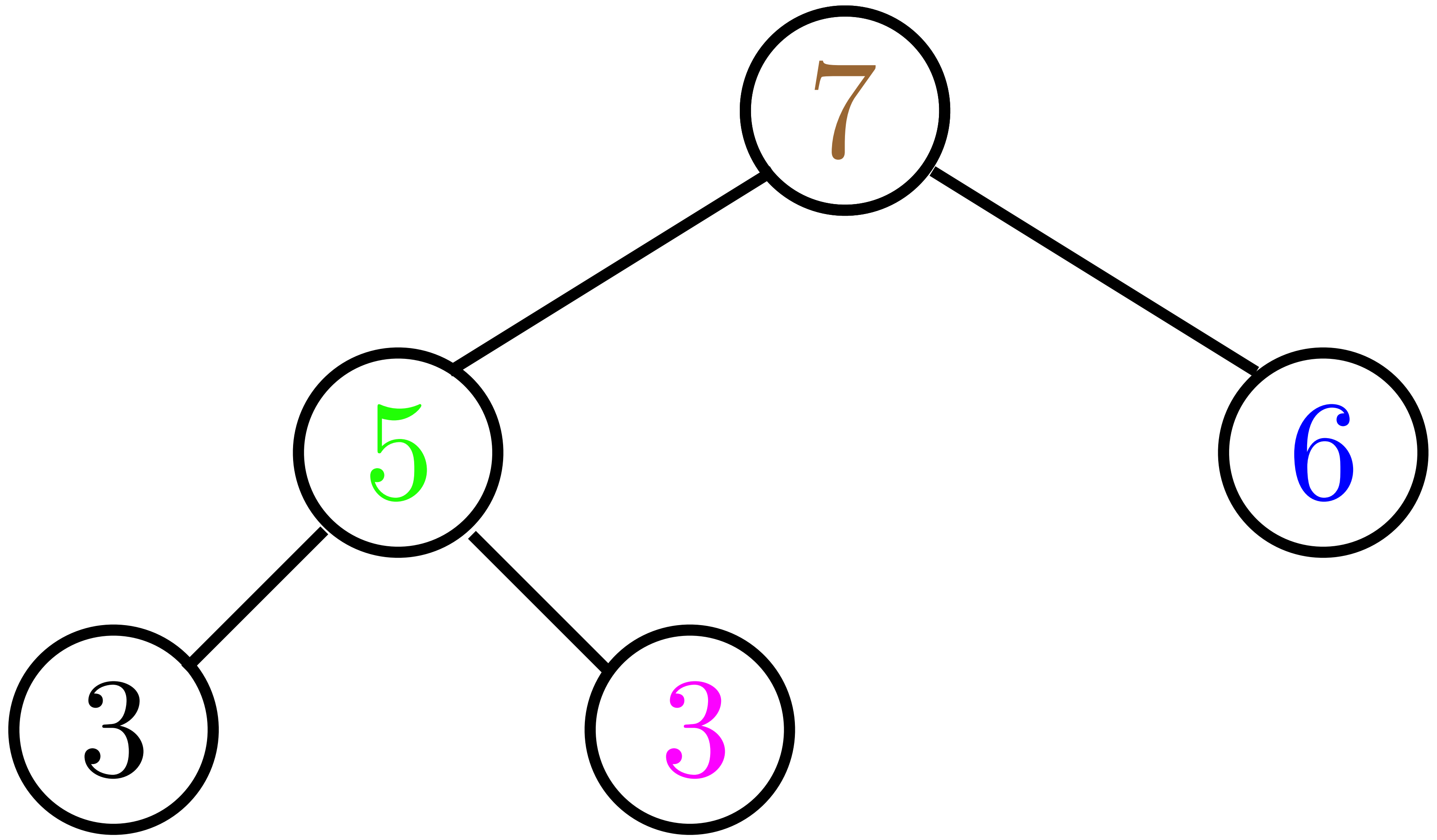
Next we insert 3 .

Now swim with 3 .

Heapsort

In the first phase we create a max heap with the elements of the vector.

	7	5	6	3	3	5



Next we insert 3 .

Now swim with 3 .

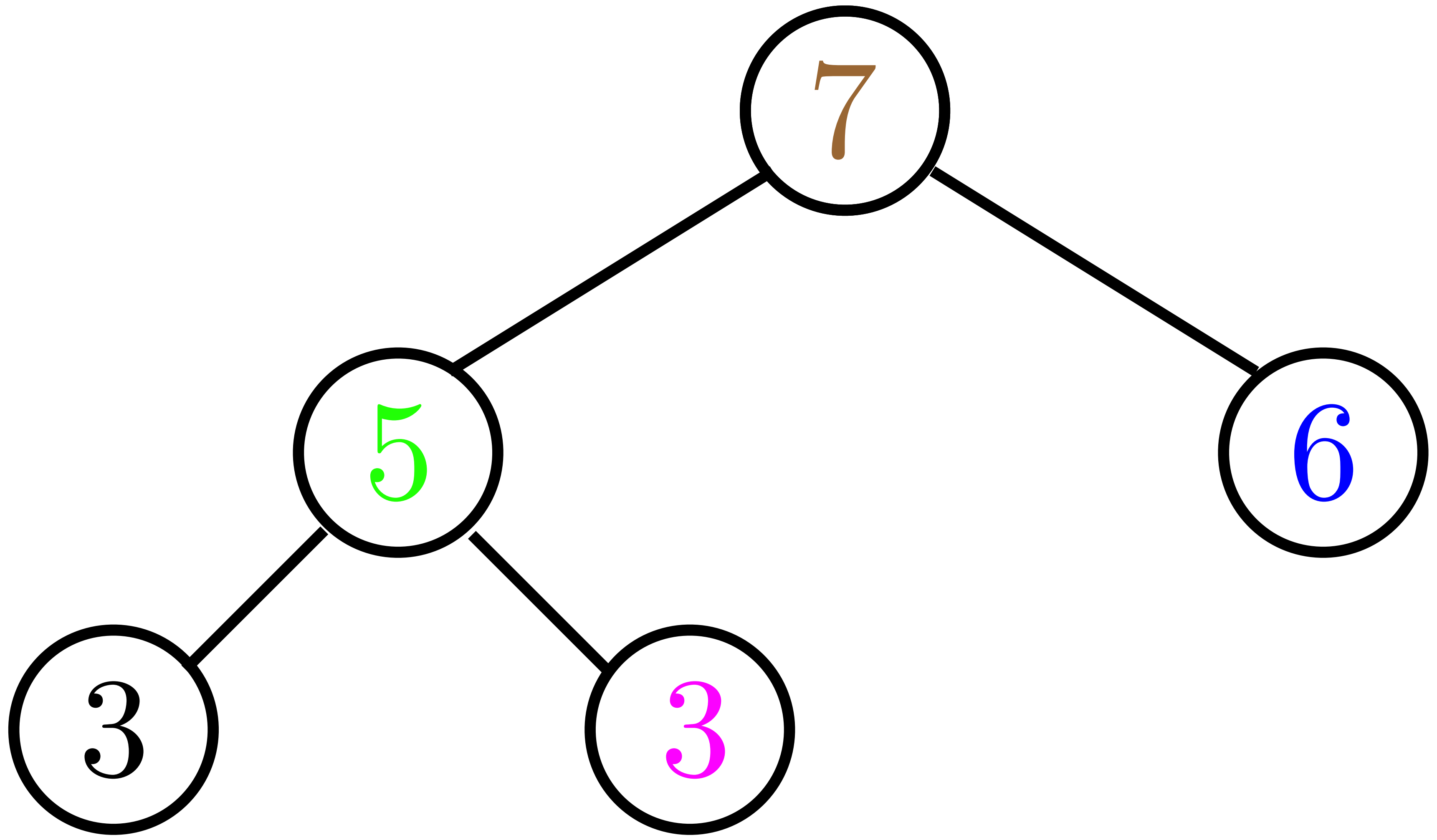
Is $5 < 3$? No, so we have a max heap on the first 5 elements.

Heapsort

In the first phase we create a max heap with the elements of the vector.

	7	5	6	3	3	5

Finally, we insert 5 .

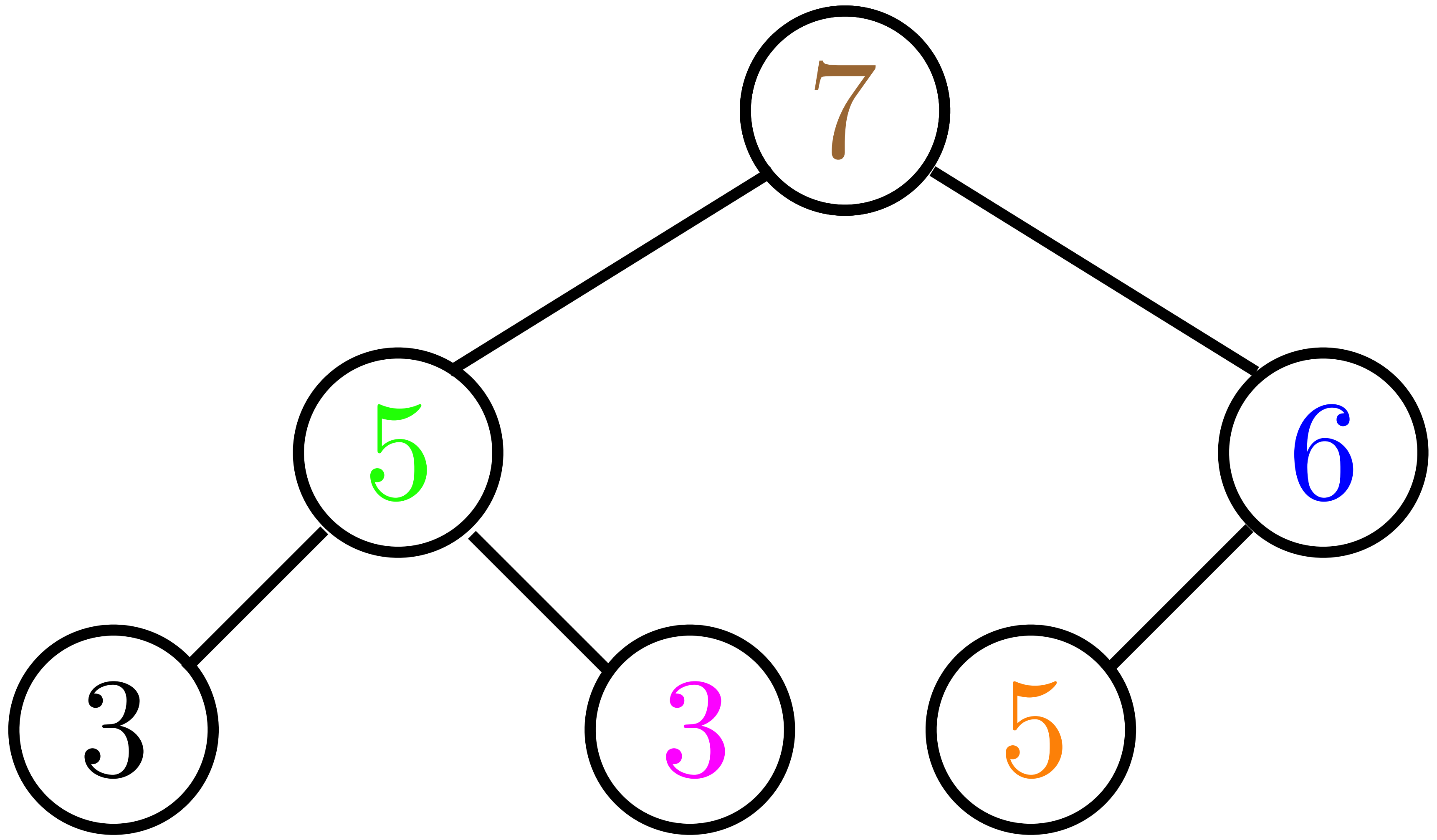


Heapsort

In the first phase we create a max heap with the elements of the vector.

	7	5	6	3	3	5

Finally, we insert 5 .



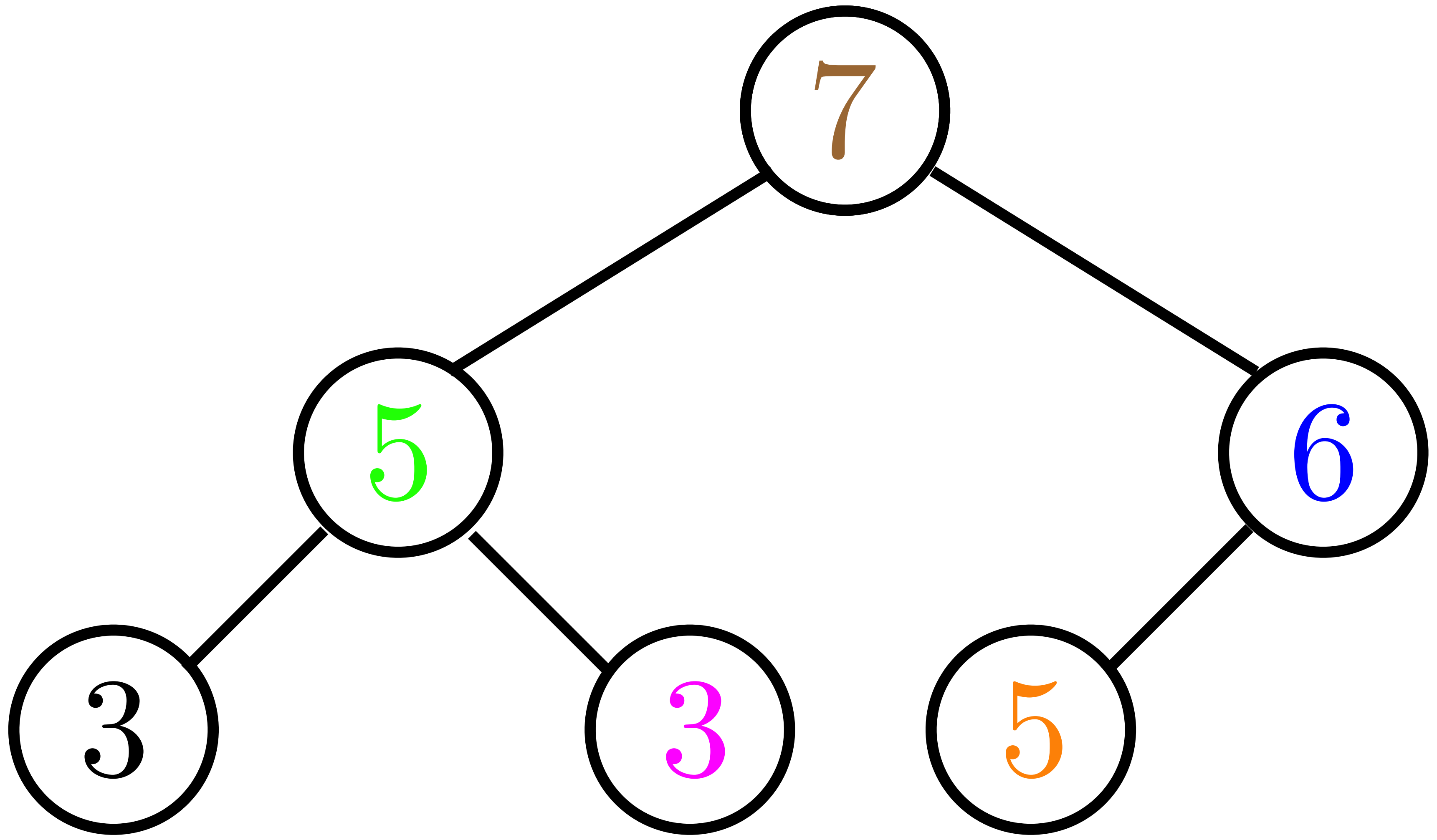
Heapsort

In the first phase we create a max heap with the elements of the vector.

	7	5	6	3	3	5

Finally, we insert 5 .

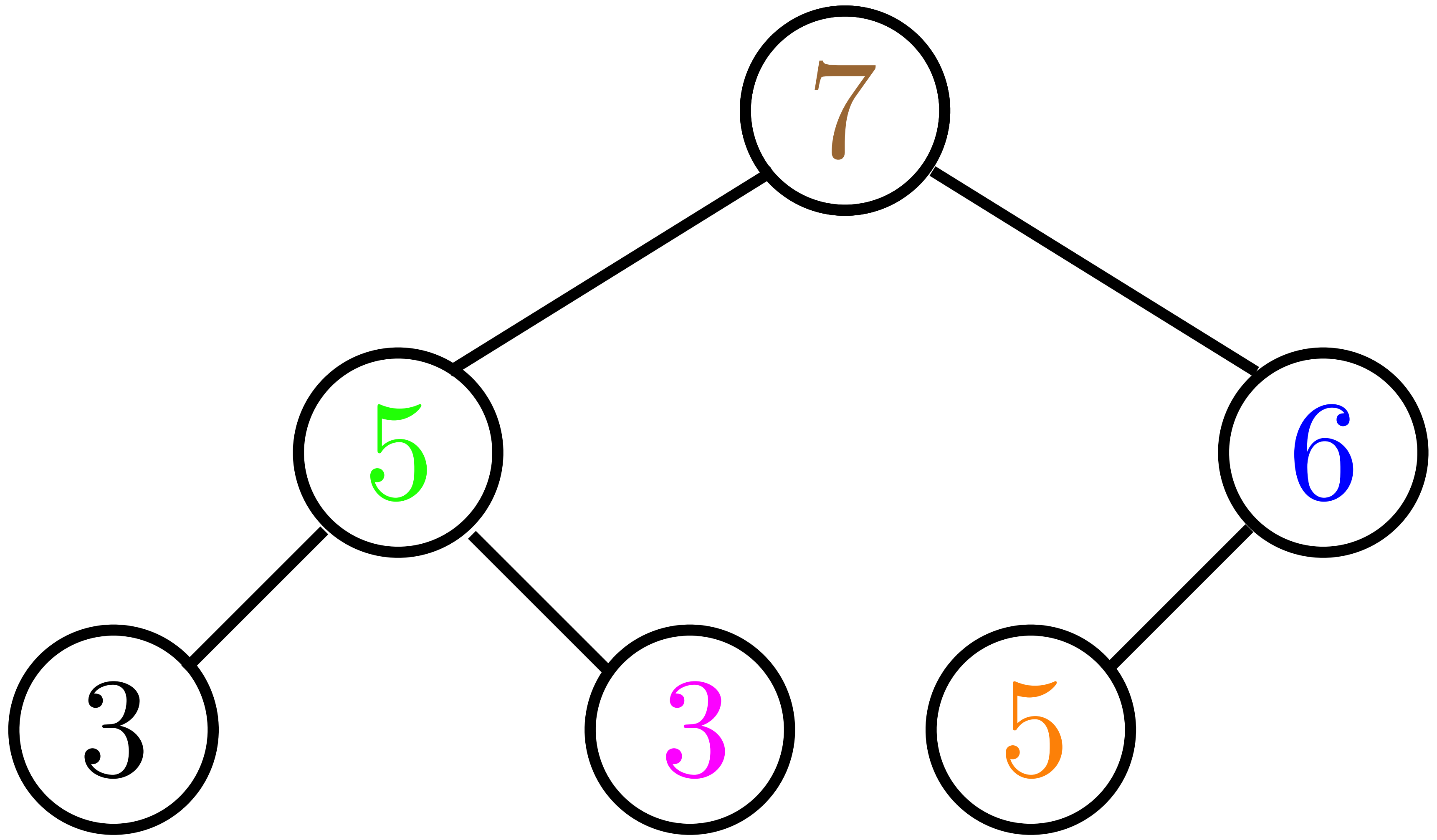
Now swim with 5.



Heapsort

In the first phase we create a max heap with the elements of the vector.

	7	5	6	3	3	5



Finally, we insert 5 .

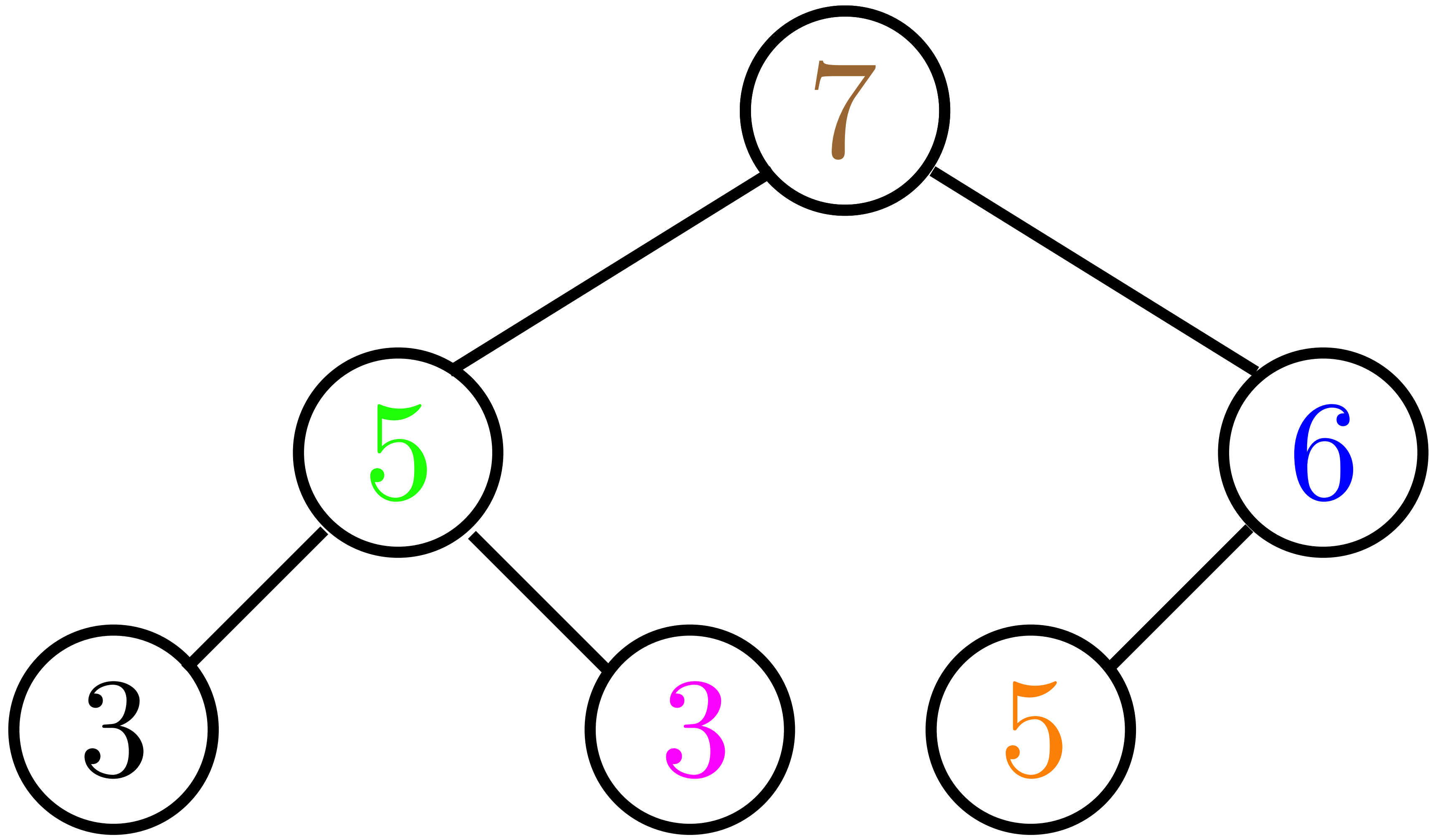
Now swim with 5.

Is $6 < 5$? No, so now we have a max heap with our initial vector.

Heapsort

We have completed the first phase.

	7	5	6	3	3	5



We now pop the elements one by one.

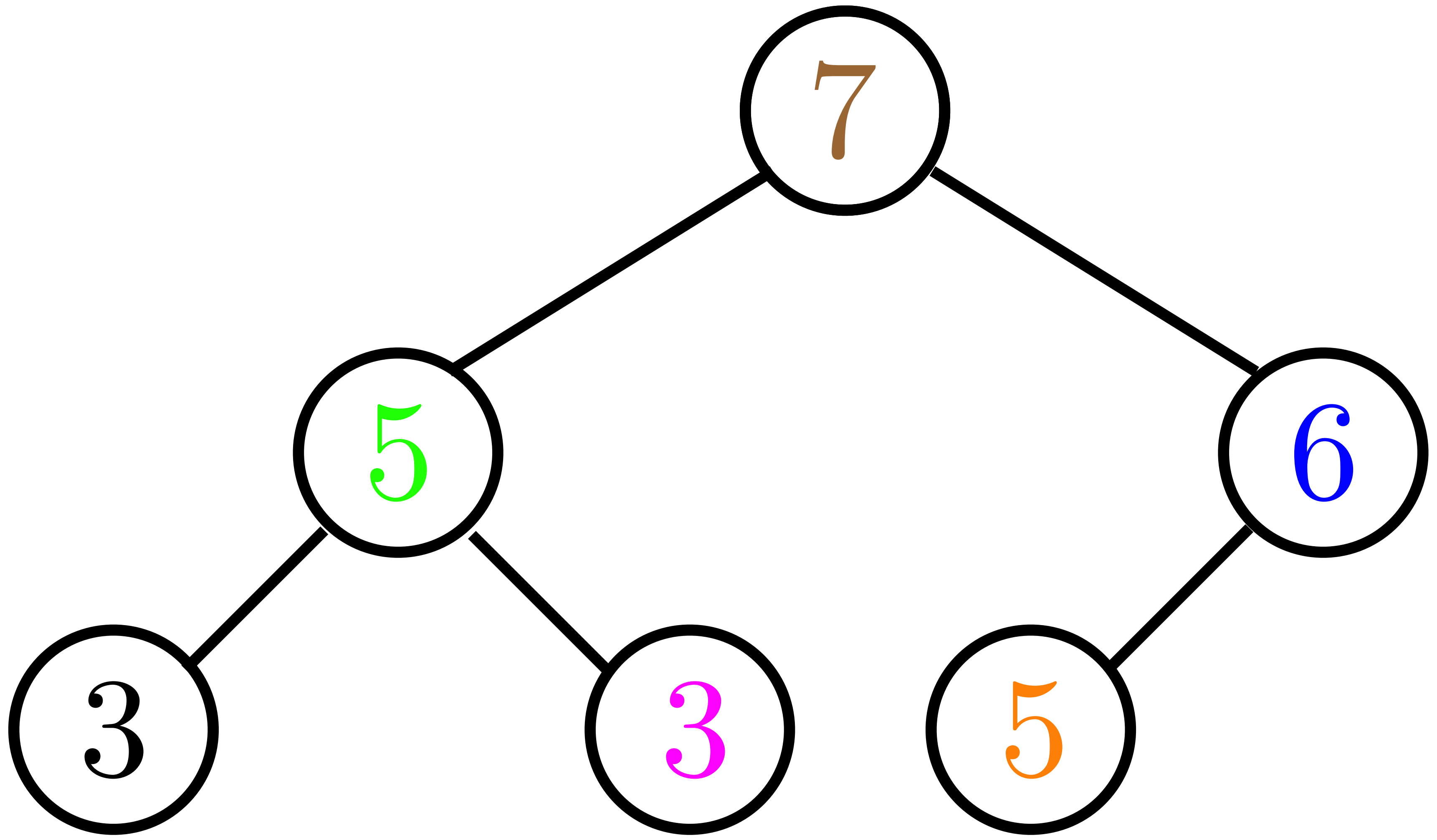
After popping, we store the elements at the back of the vector.

Heapsort

We have completed the first phase.

	7	5	6	3	3	5

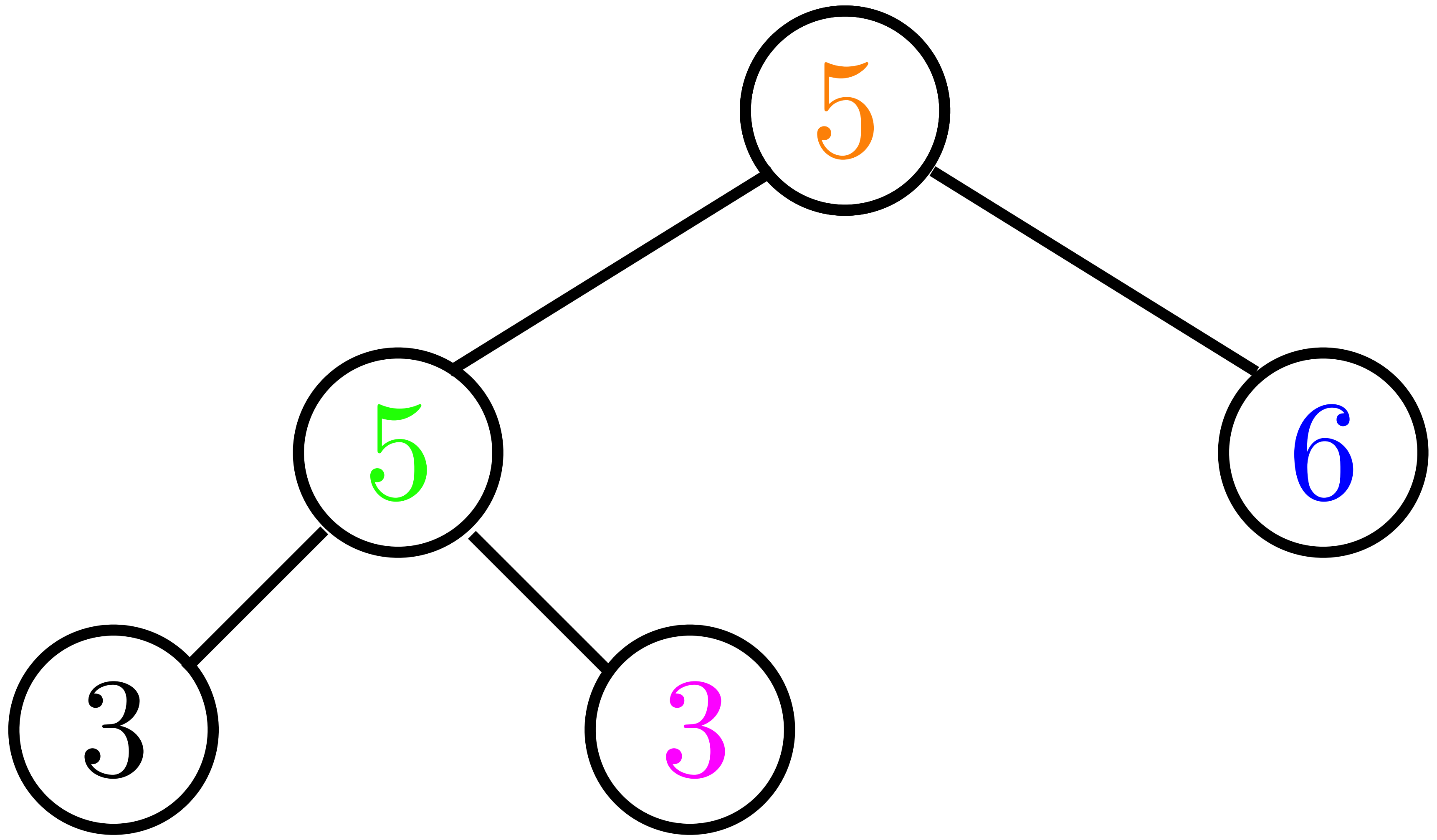
Pop 7. We replace 7 with 5.



Heapsort

We have completed the first phase.

	5	5	6	3	3	7



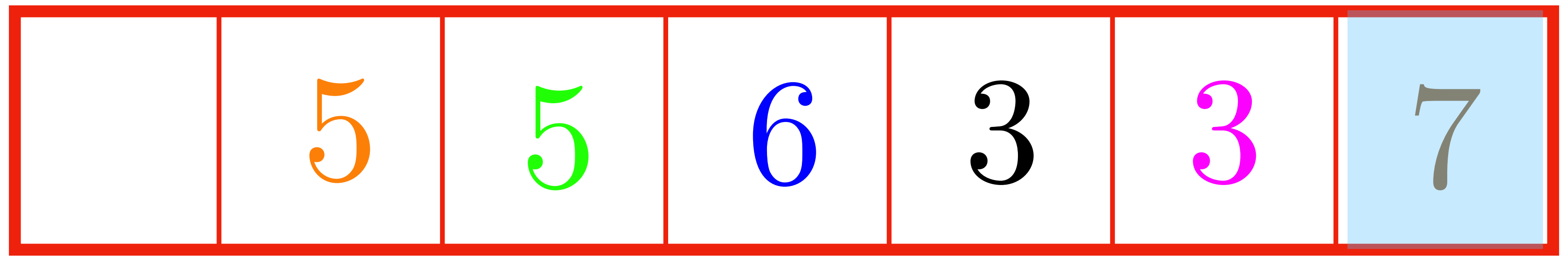
Pop 7. We replace 7 with 5.

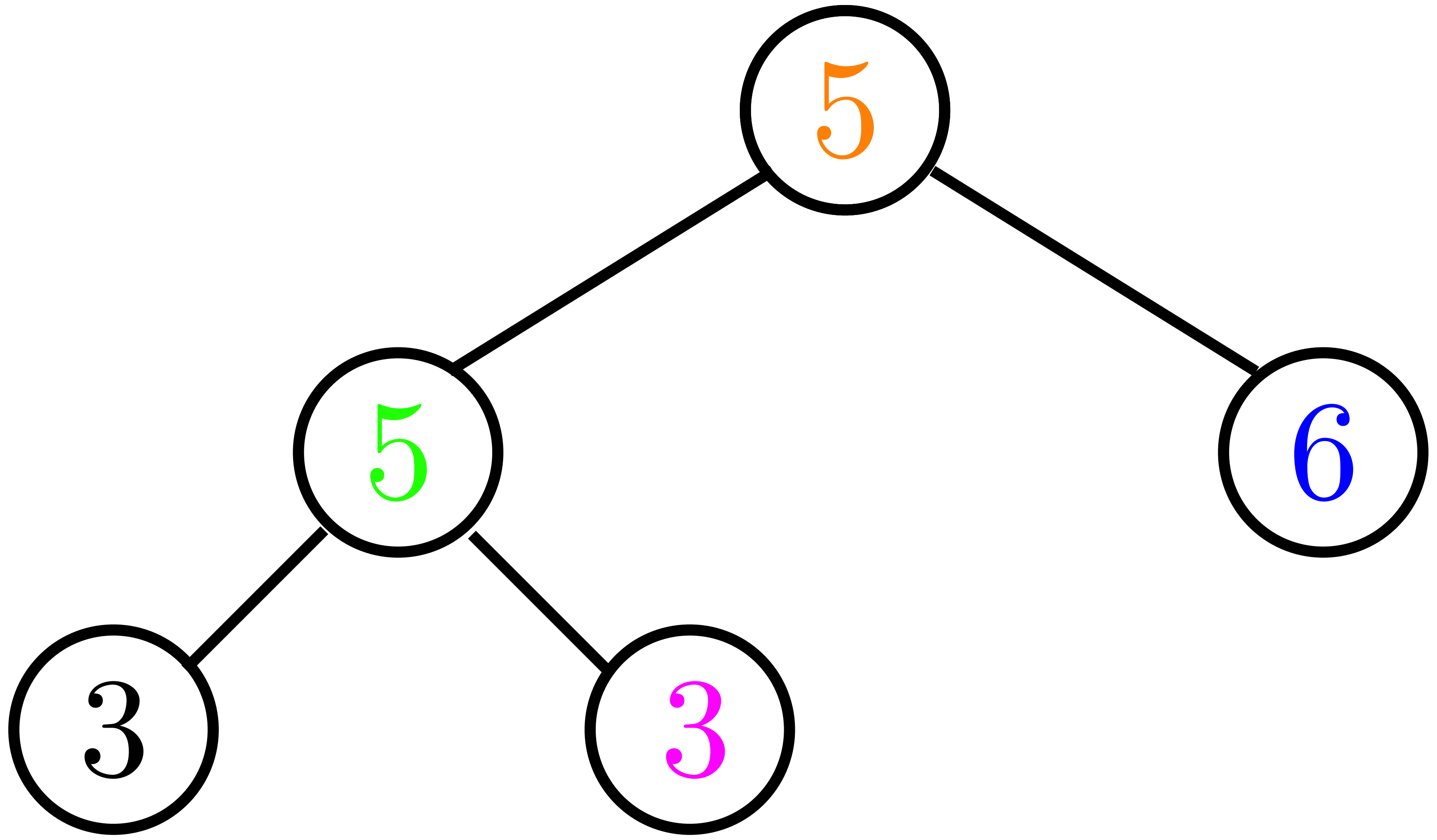
We store 7 at the old position of in the vector 5.

We no longer think of 7 as being in the heap. It will not move again.

Heapsort

We have completed the first phase.





Pop 7. We replace 7 with 5.

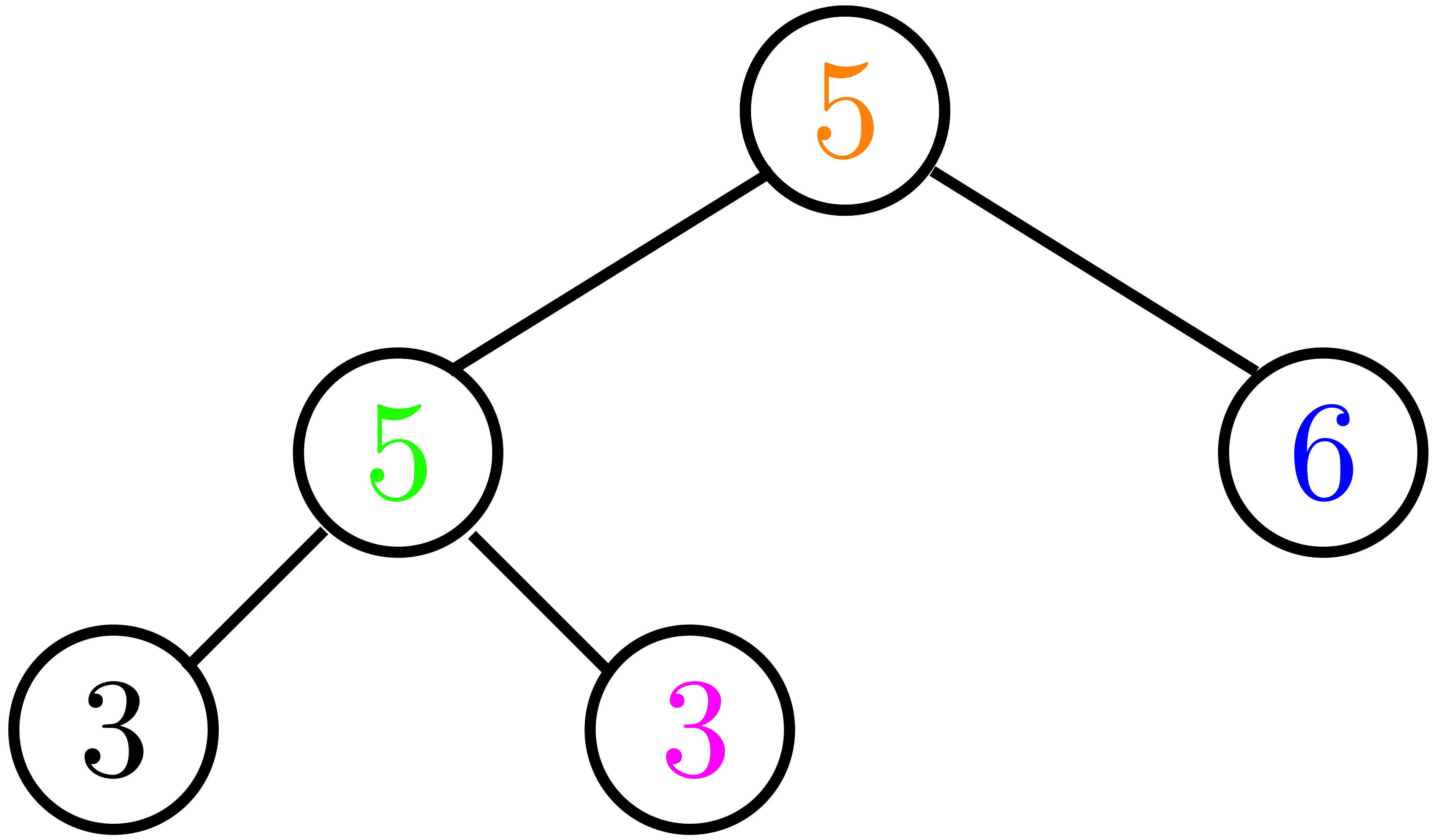
We store 7 at the old position of in the vector 5.

We no longer think of 7 as being in the heap. It will not move again.

Heapsort

We have completed the first phase.

	5	5	6	3	3	7



Now "sink" with 5 .

Is $5 < 6$? Yes, so we swap them.

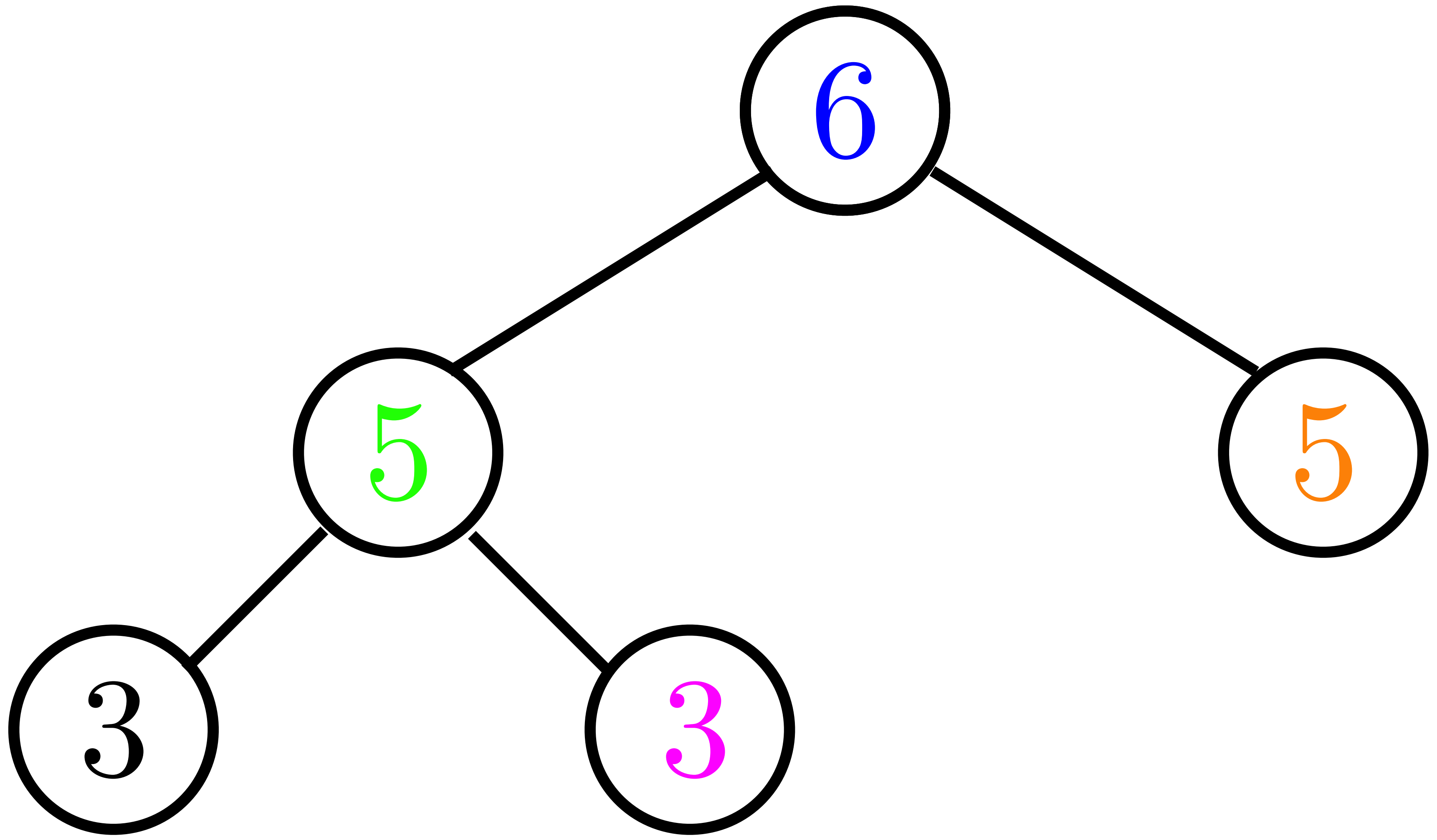
Heapsort

We have completed the first phase.

	6	5	5	3	3	7

Now "sink" with 5 .

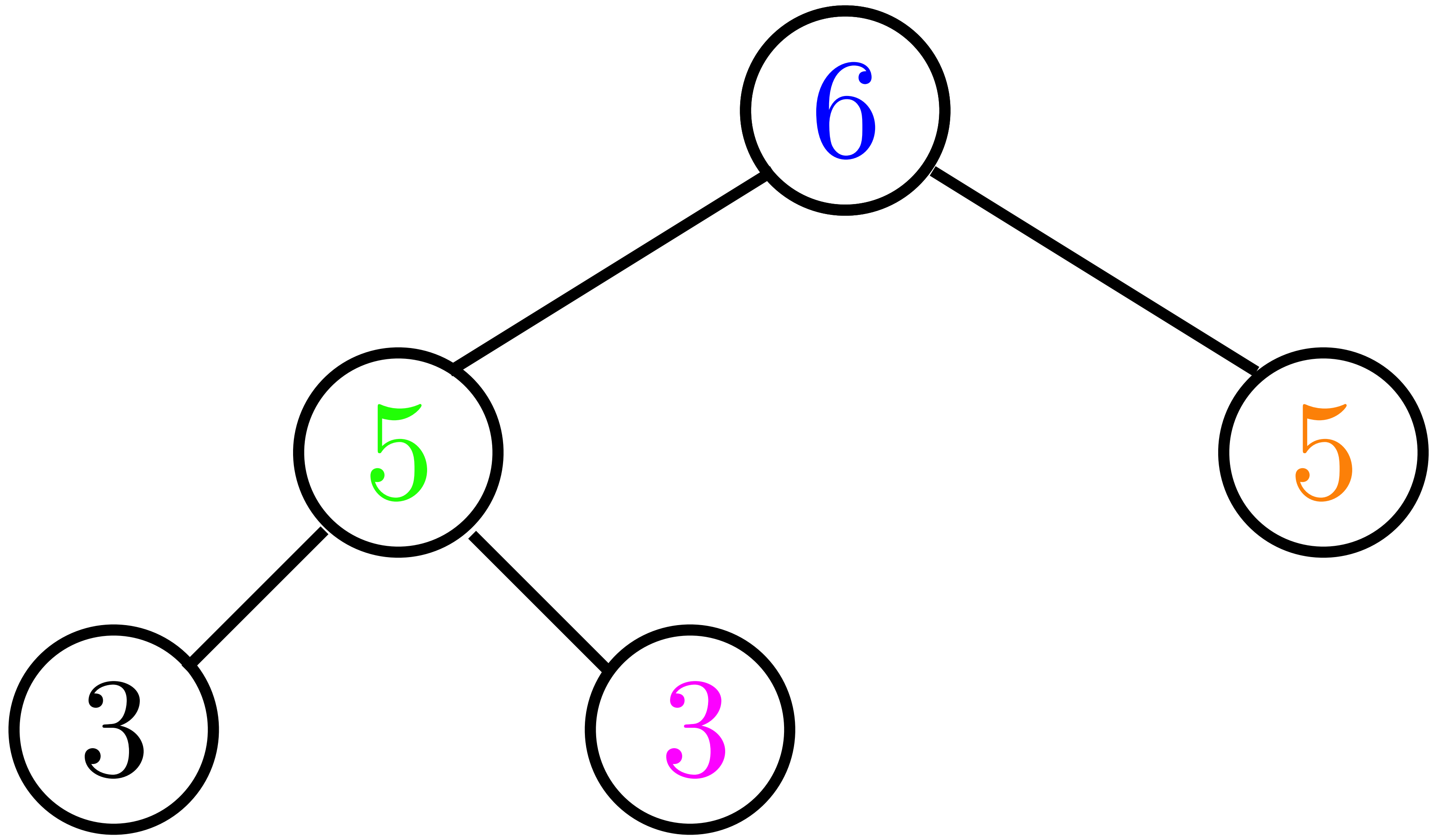
We have restored the max heap property.



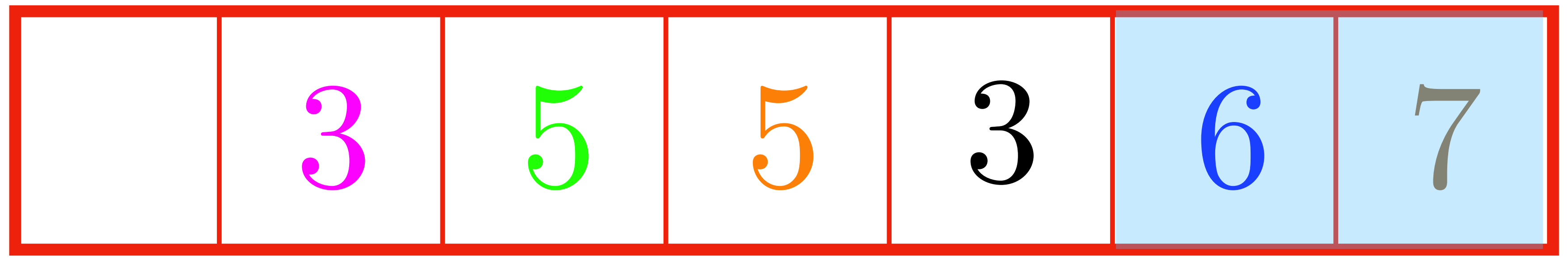
Heapsort

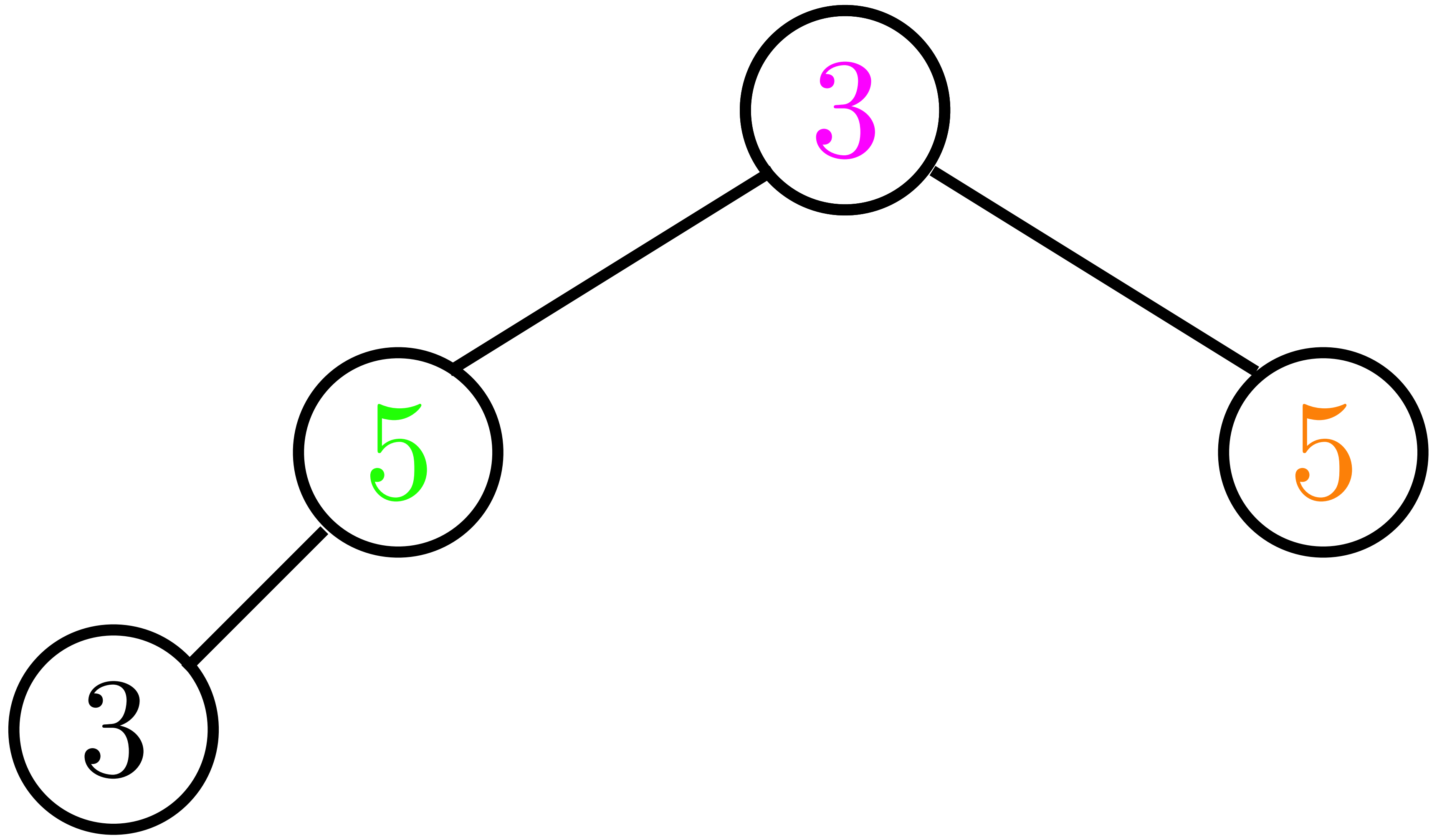
	6	5	5	3	3	7

Now we pop again. We swap 6 and 3 in the vector.



Heapsort



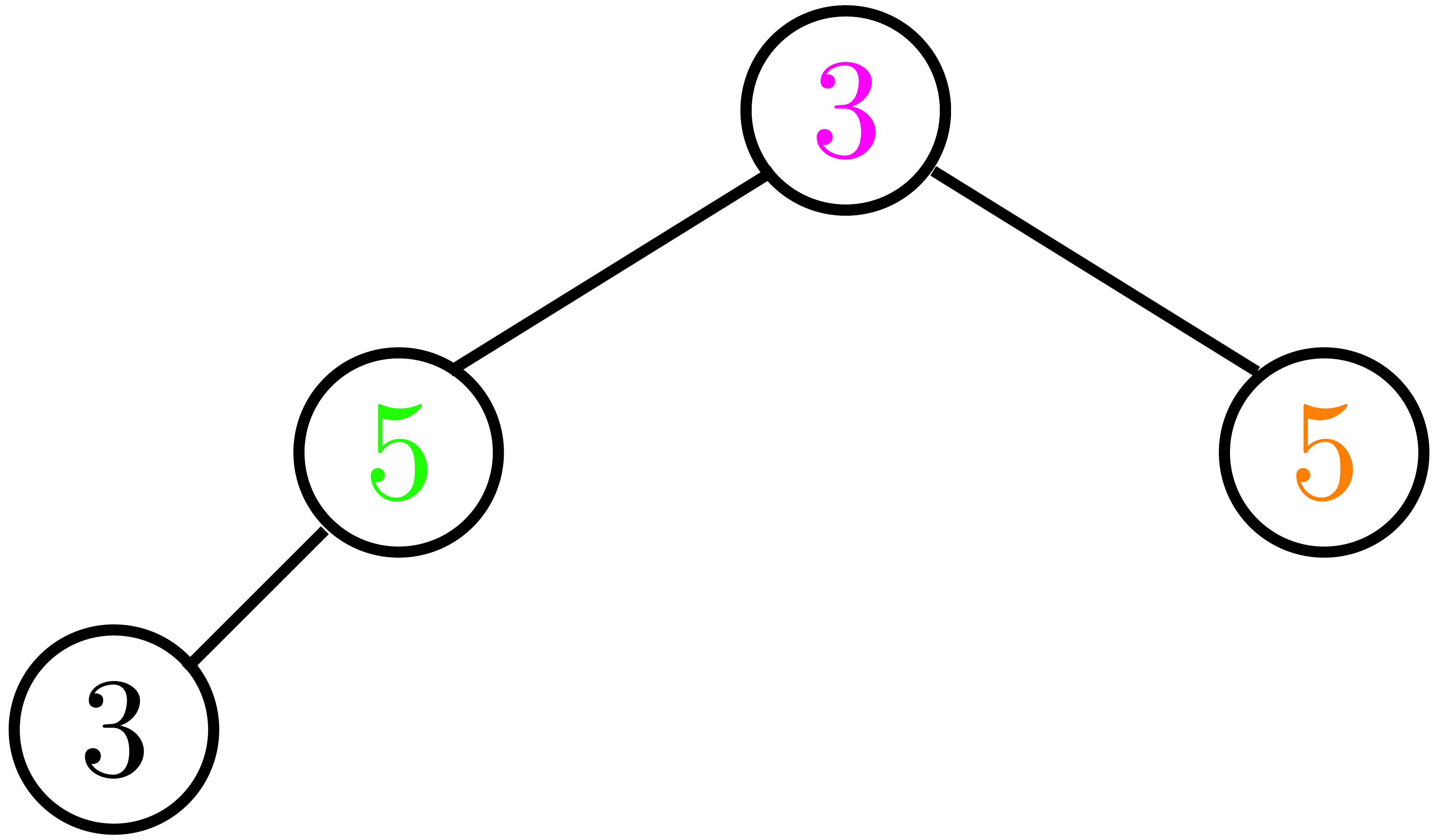


Now we pop again. We swap 6 and 3 in the vector.

We no longer consider 6 as being in the heap. It will not move again.

Heapsort

	3	5	5	3	6	7

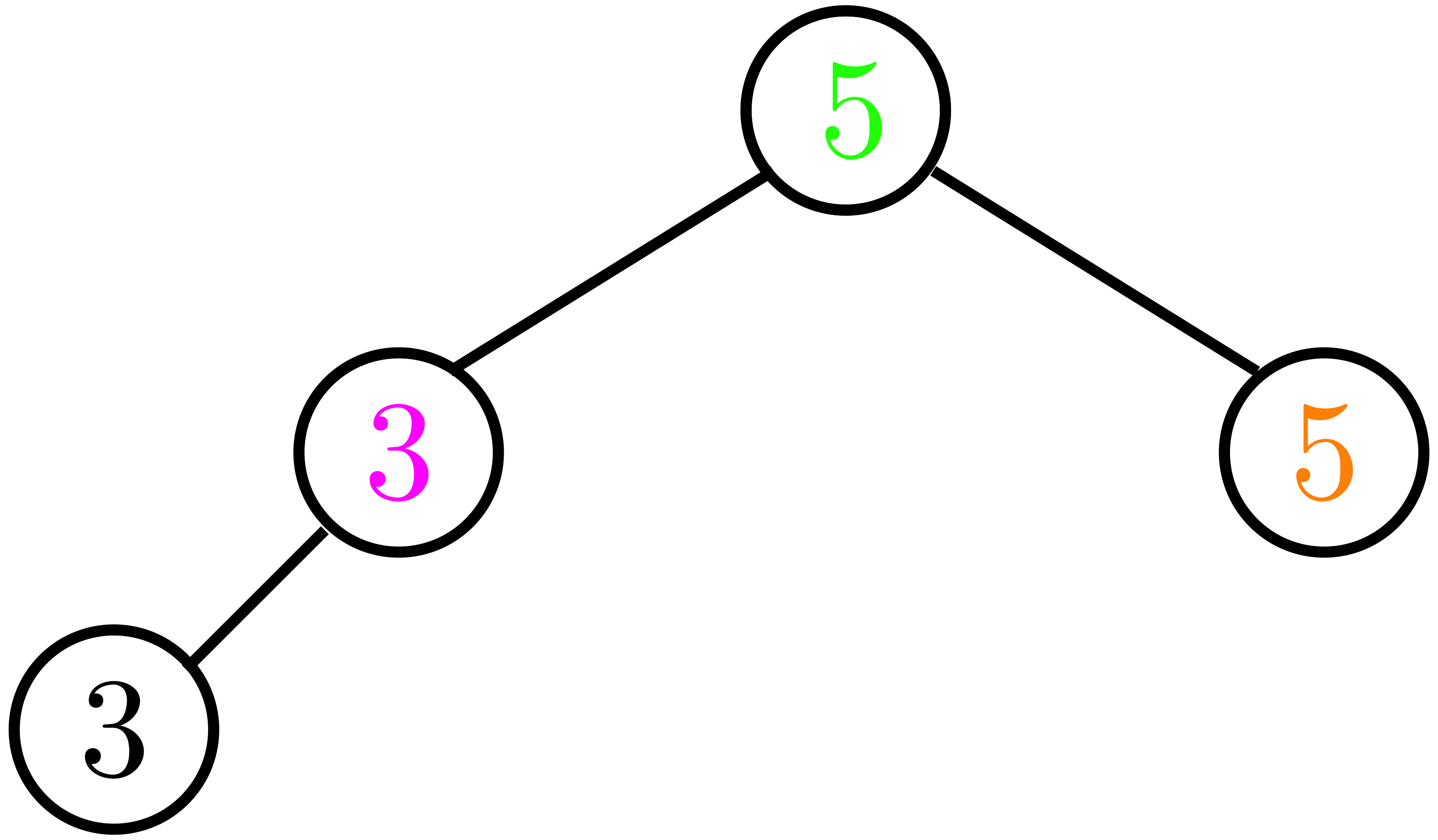


Now we sink with 3 .

Is $3 < 5$? Yes, so we swap them.

Heapsort

	5	3	5	3	6	7

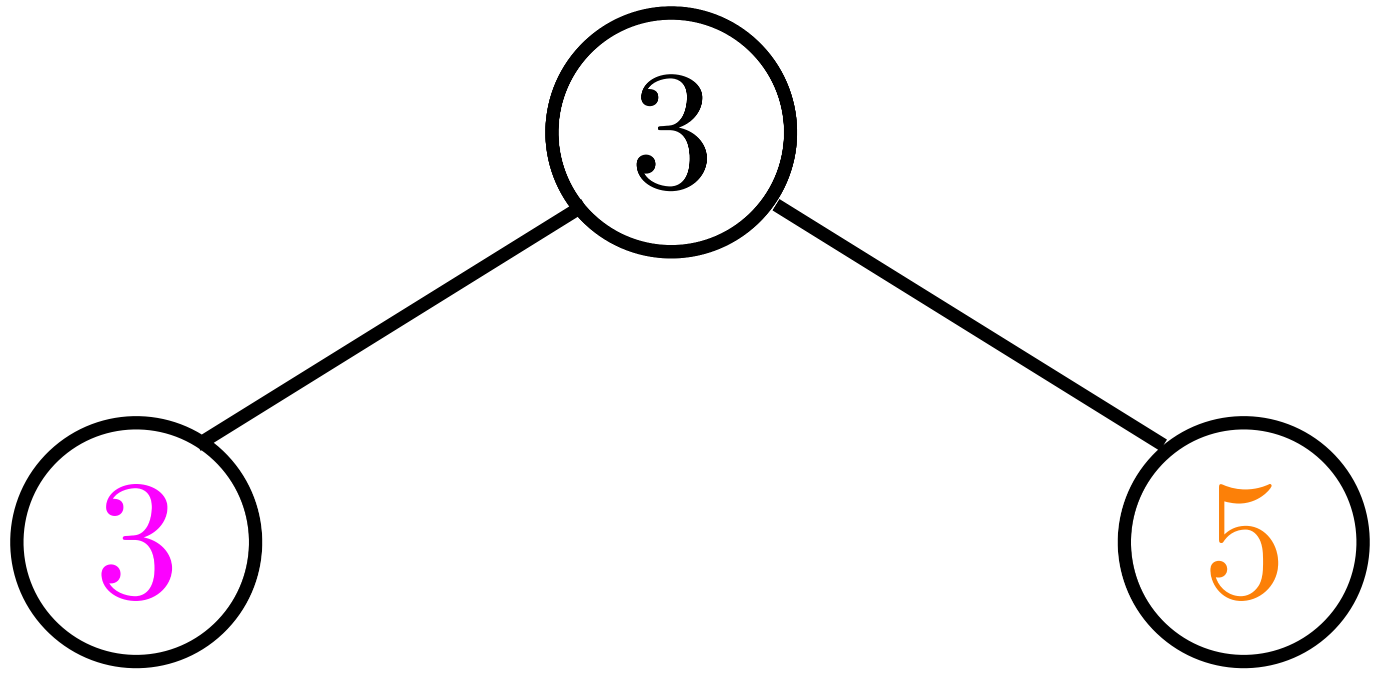


We have restored the heap property.

Next we pop 5 .

Heapsort

	3	3	5	5	6	7

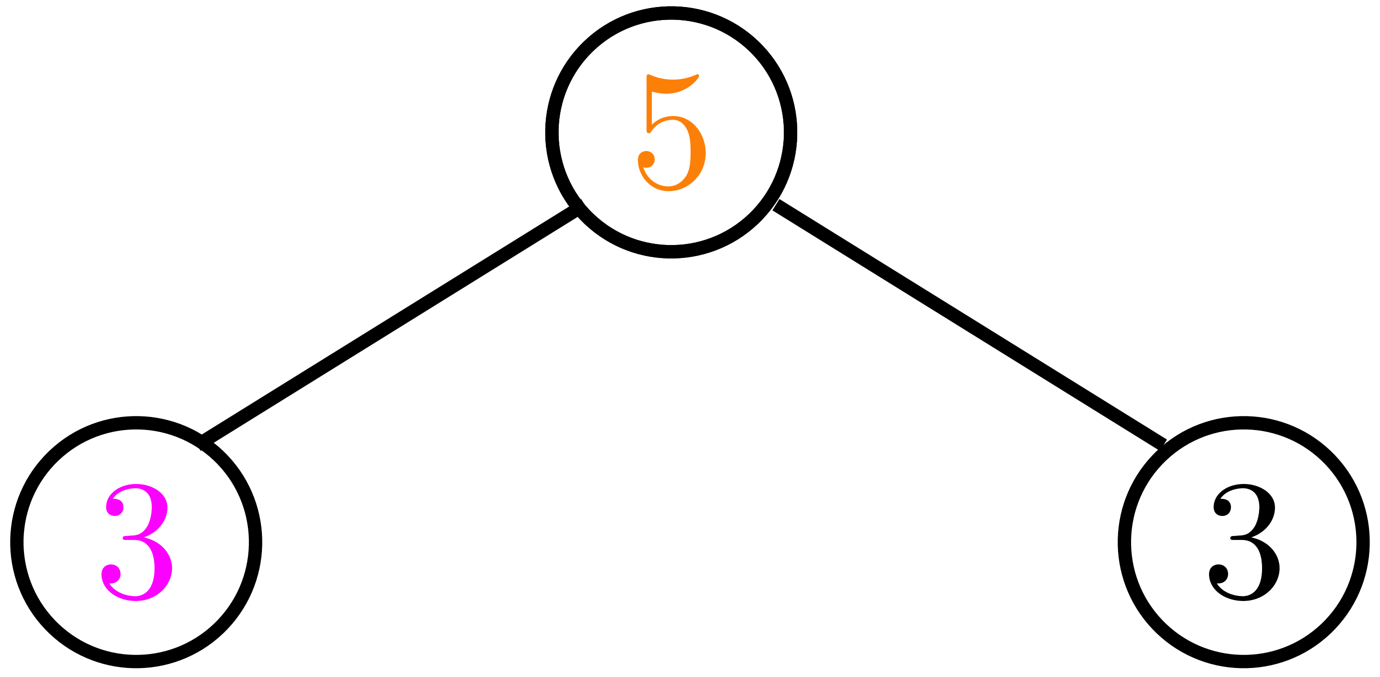


And sink with 3 .

We swap 3 and 5.

Heapsort

	5	3	3	5	6	7

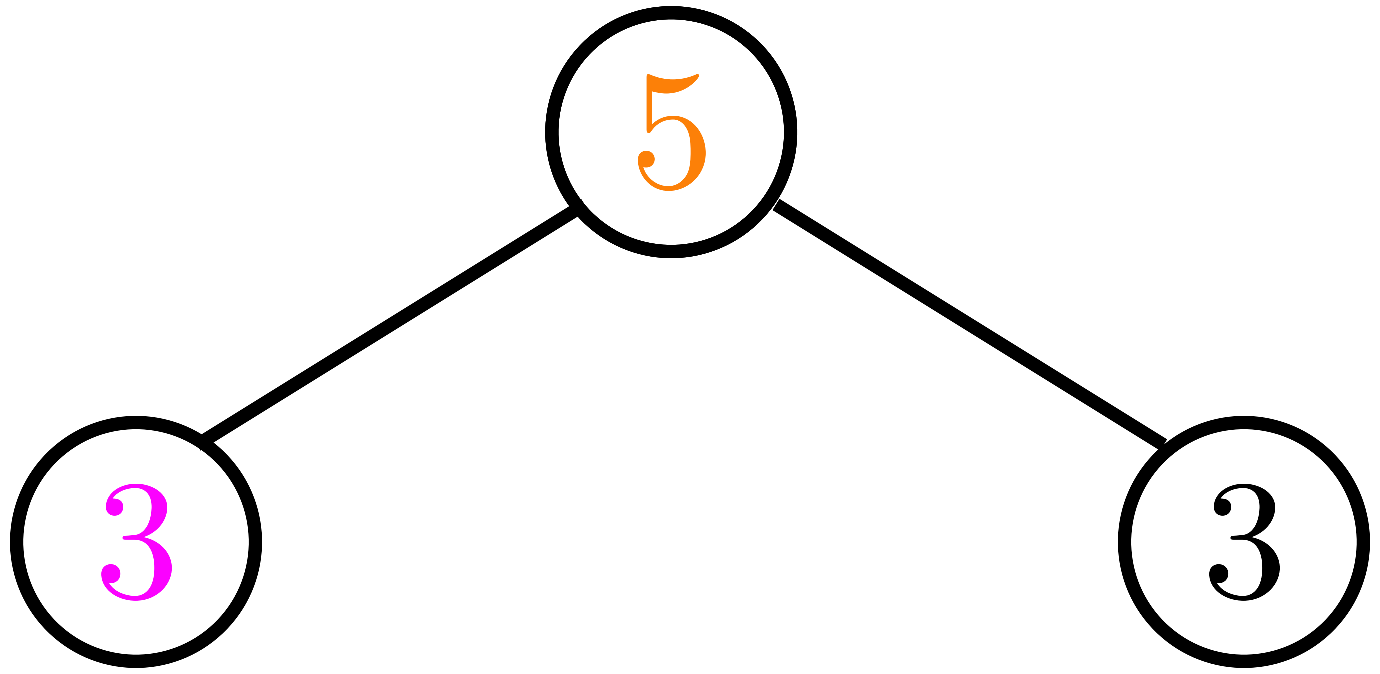


And sink with 3 .

We swap 3 and 5.

Heapsort

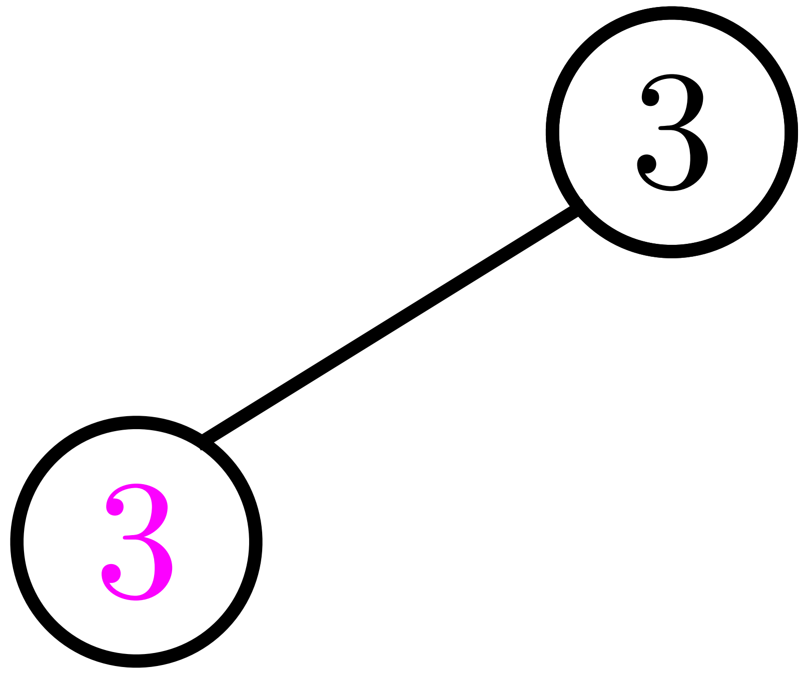
	5	3	3	5	6	7



Pop 5.

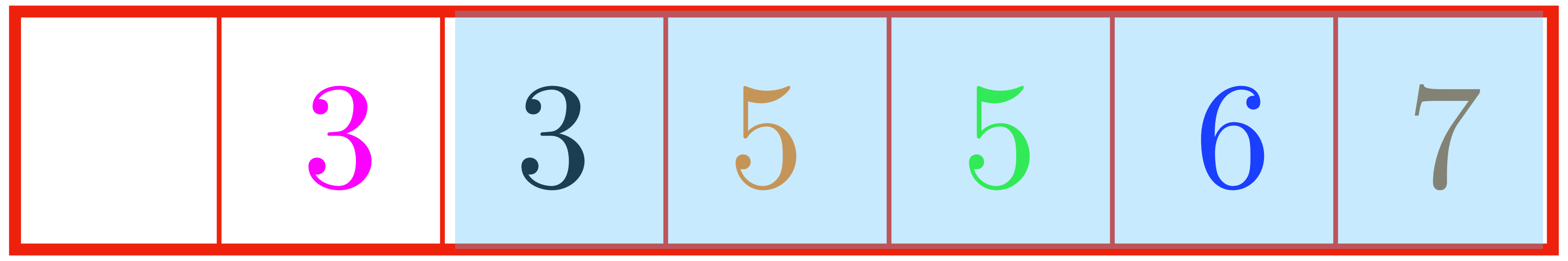
Heapsort

	3	3	5	5	6	7



Pop 3.

Heapsort



Pop 3.

Heapsort

	3	3	5	5	6	7

Now our vector is in sorted order.

What is the time complexity of heap sort?

Heapsort

input vector		3	7	6	5	3	5
sorted vector		3	3	5	5	6	7

Heapsort is not a stable sorting algorithm. The order of keys with the same value is not necessarily the same in the output as in the input.

Benefit of Stable Sort

McBeal, Ally
Smith, Jack
McBeal, Diana
Smith, Bob

sort by first name

McBeal, Ally
Smith, Bob
McBeal, Diana
Smith, Jack

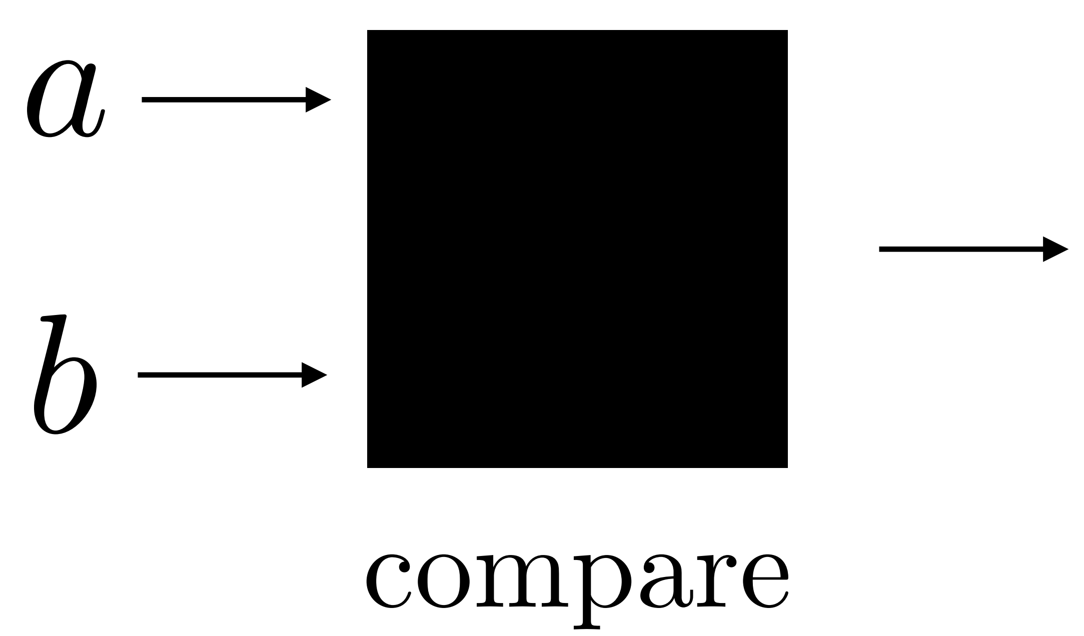
stable sort by last name

McBeal, Ally
McBeal, Diana
Smith, Bob
Smith, Jack

now fully sorted

Comparison-based sort

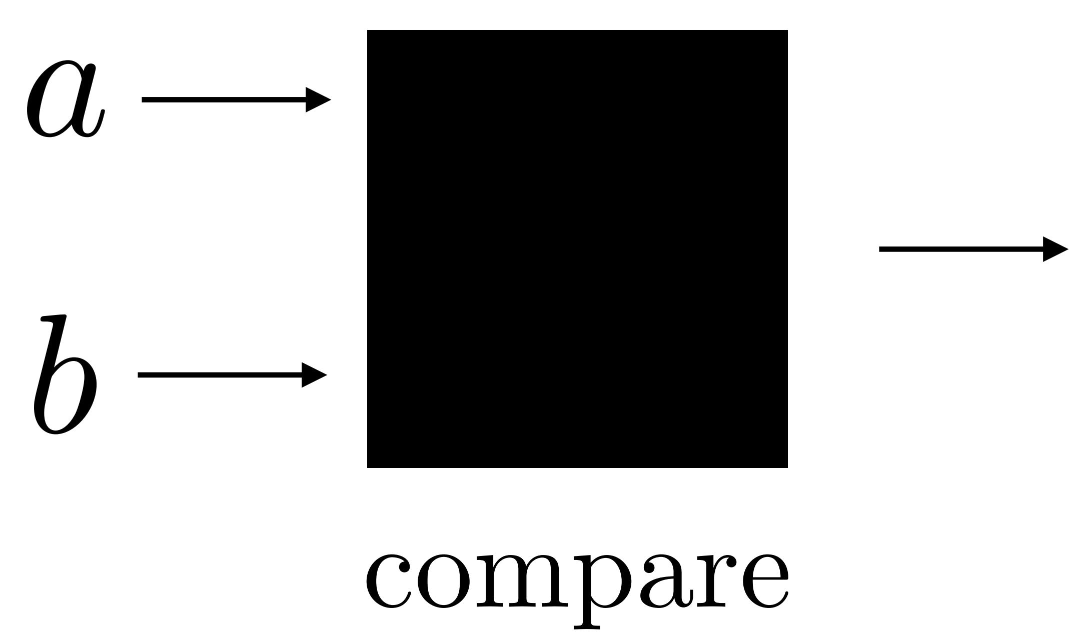
A comparison-based sorting algorithm only interacts with the data via a "compare" function.



$$\begin{cases} 1 & \text{if } a < b \\ 0 & \text{otherwise} \end{cases}$$

Comparison-based sort

A comparison-based sorting algorithm only interacts with the data via a "compare" function.

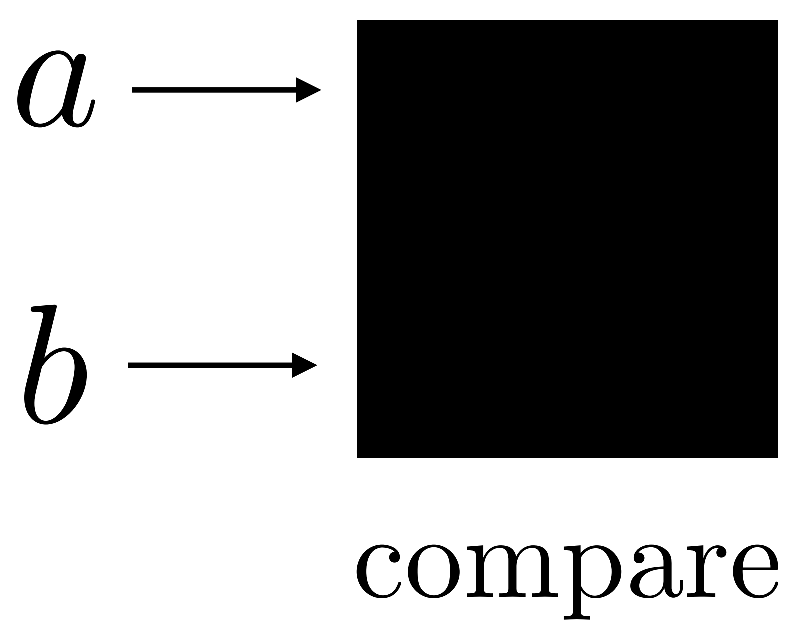


$$\begin{cases} 1 & \text{if } a < b \\ 0 & \text{otherwise} \end{cases}$$

Is heapsort a comparison based sorting algorithm?

Comparison-based sort

A comparison-based sorting algorithm only interacts with the data via a "compare" function.



$$\longrightarrow \begin{cases} 1 & \text{if } a < b \\ 0 & \text{otherwise} \end{cases}$$

Is heapsort a comparison based sorting algorithm?

What is an example of a sorting algorithm we have seen that is not comparison based?

Comparison-based sort

Any comparison based sorting algorithm must make at least a constant times $n \log n$ comparisons in the worst case.

Heapsort is optimal with respect to number of comparisons.

We cannot expect to have worst case $O(1)$ insert and pop operations on a heap.

In-Place

Heapsort is also an in-place sorting algorithm.

In-Place

Heapsort is also an in-place sorting algorithm.

We just needed a constant number of helper variables, in addition to the original input array.

Intro Sort

Implementation of the standard library sorting algorithm `std::sort` typically uses an algorithm called introspective sort.

It starts out doing quicksort, but if this takes too long it switches to heapsort.

This allows it to have $O(n \log n)$ worst-case running time (which is required by the standard since C++11).