Priority queues

Johan Montelius

KTH

HT23

a queue

a queue

First in first out.

a queue

First in first out.

- enqueue(): Add item to the queue.

a queue

First in first out.

- enqueue(): Add item to the queue.
- dequeue() : Remove item that has been the longest in the queue.
- empty() : Is the queue empty?

a priority queue

a priority queue

Items ordered by priority.

a priority queue

Items ordered by priority.

- enqueue(): Add item with a given priority to the queue.

a priority queue

Items ordered by priority.

- enqueue(): Add item with a given priority to the queue.
- dequeue() : Remove item with the highest priority.
- empty() : Is the queue empty?

a priority queue

Items ordered by priority.

- enqueue(): Add item with a given priority to the queue.
- dequeue() : Remove item with the highest priority.
- empty() : Is the queue empty?

Let's say that low numbers have high priority.

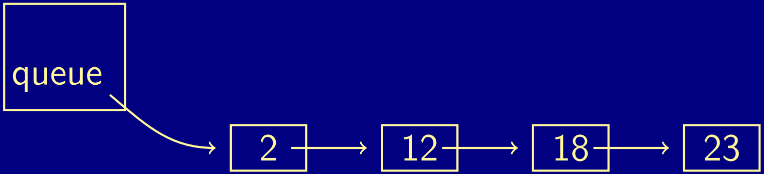
a linked list implementation

a linked list implementation

Let's keep the list sorted.

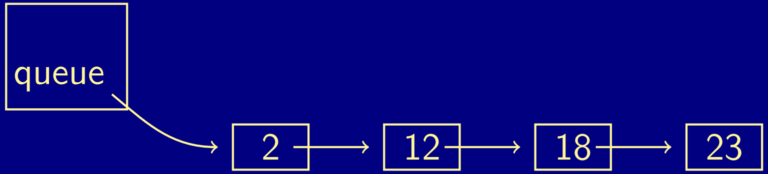
a linked list implementation

Let's keep the list sorted.



a linked list implementation

Let's keep the list sorted.



How do we implement add and remove?

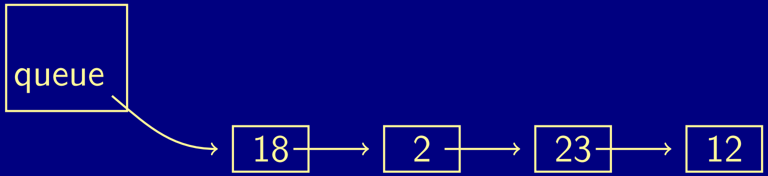
a linked list implementation

a linked list implementation

Let's not bother keeping the list sorted.

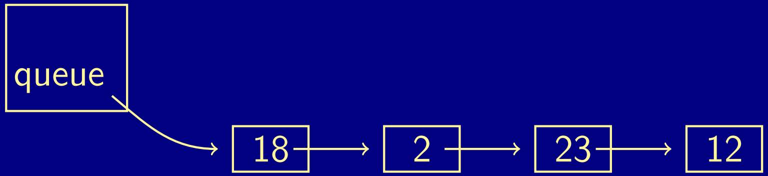
a linked list implementation

Let's not bother keeping the list sorted.



a linked list implementation

Let's not bother keeping the list sorted.

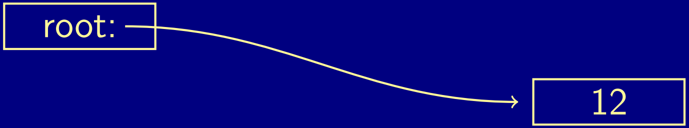


How do we implement add and remove?

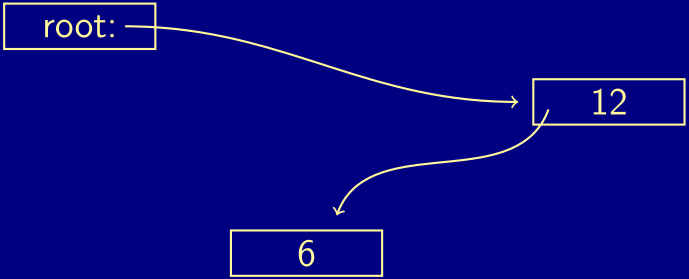
why no a tree

root:

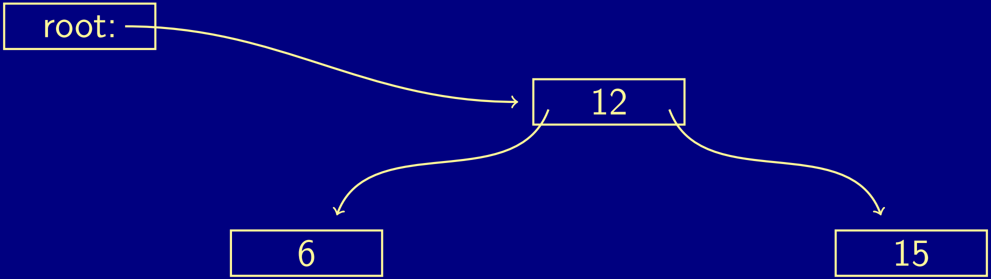
why no a tree



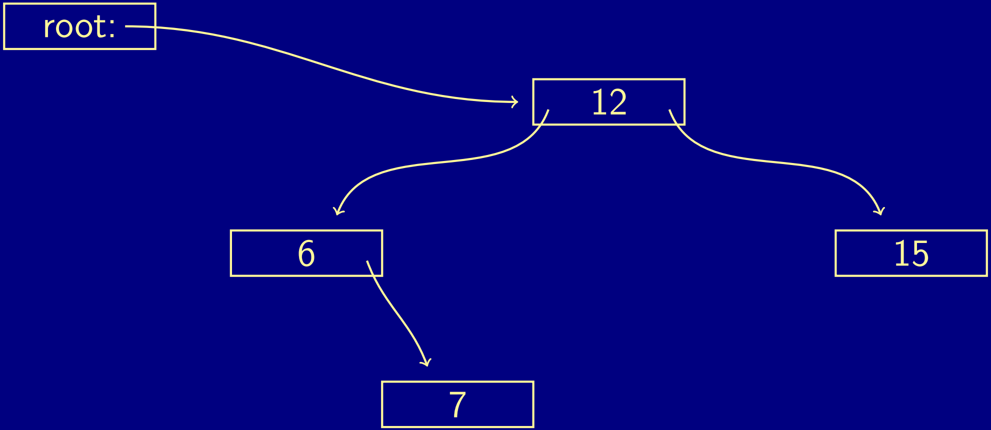
why no a tree



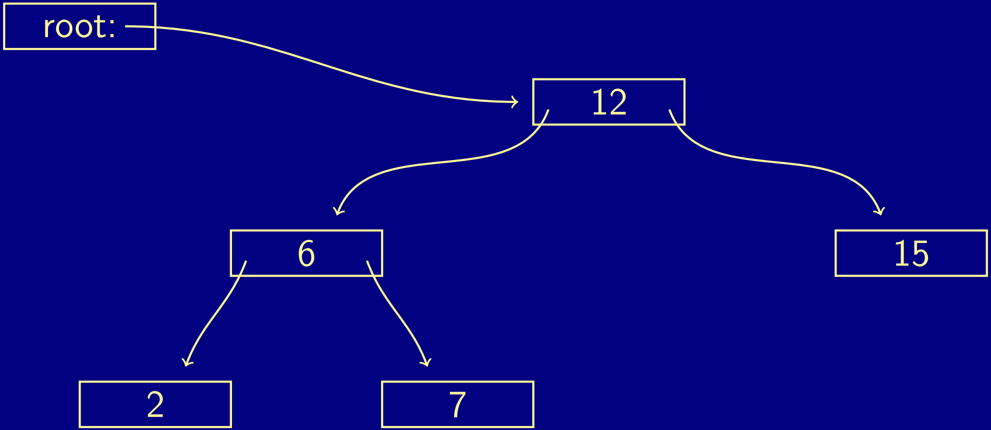
why no a tree



why no a tree



why no a tree



a sorted tree

a sorted tree

- A sorted tree gives us $O(lg(n))$ add opperation and $O(lg(n))$ remove operation.

a sorted tree

- A sorted tree gives us $O(lg(n))$ add opperation and $O(lg(n))$ remove operation.
- Excellent for searching but ....

a sorted tree

- A sorted tree gives us $O(lg(n))$ add opperation and $O(lg(n))$ remove operation.
- Excellent for searching but ....
- we know which element to remove next.

a sorted tree

- A sorted tree gives us $O(lg(n))$ add opperation and $O(lg(n))$ remove operation.
- Excellent for searching but ....
- we know which element to remove next.
- Arrange the tree such that the higest priority is always the root node.

the heap

the heap

A heap :

- The element with highest priority is in the root.

A heap :

- The element with highest priority is in the root.
- The left branch is a heap, and so is the right branch.

the heap

A heap :

- The element with highest priority is in the root.
- The left branch is a heap, and so is the right branch.
- There is no relationship between the left and right branch.

the heap

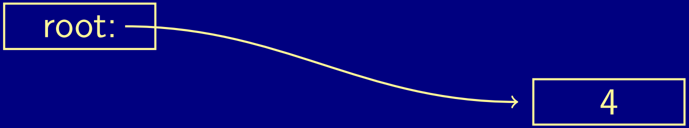
A heap :

- The element with highest priority is in the root.
- The left branch is a heap, and so is the right branch.
- There is no relationship between the left and right branch.
- We need add() and remove() operations that maintain this order.

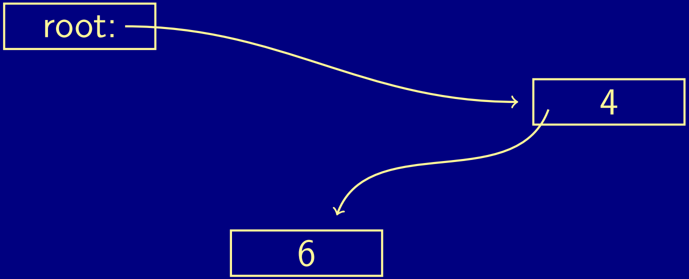
the heap

root:

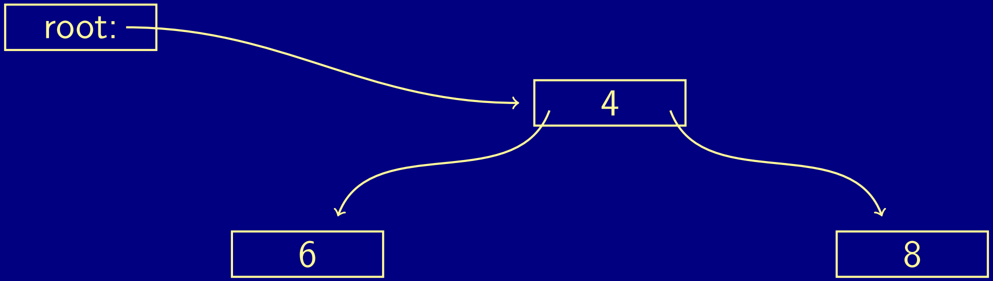
the heap



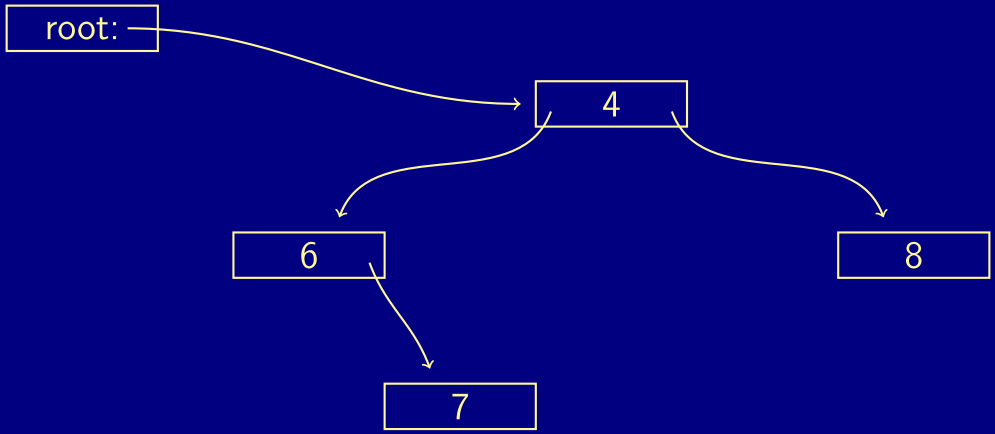
the heap



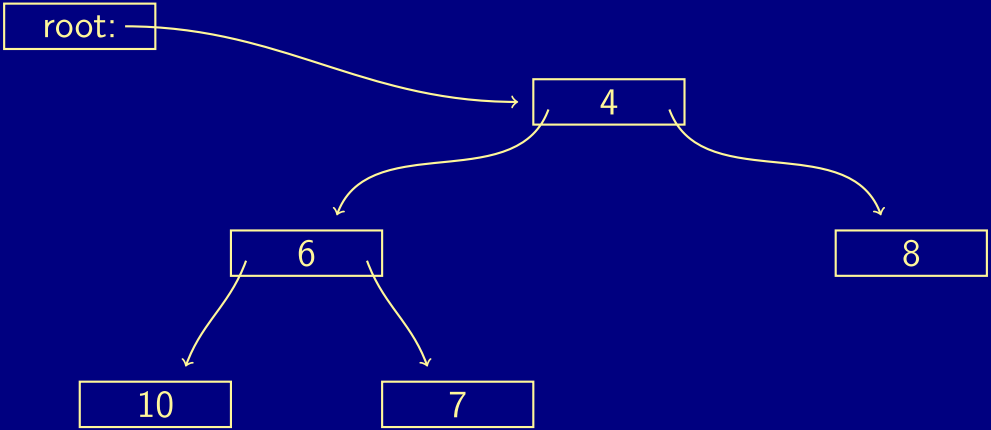
the heap



the heap



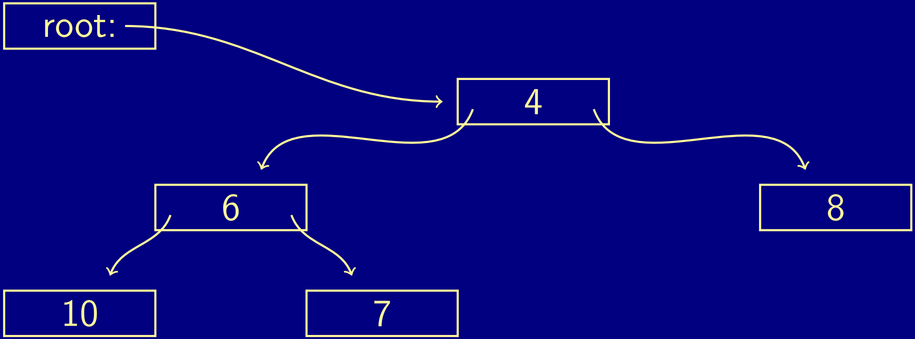
the heap



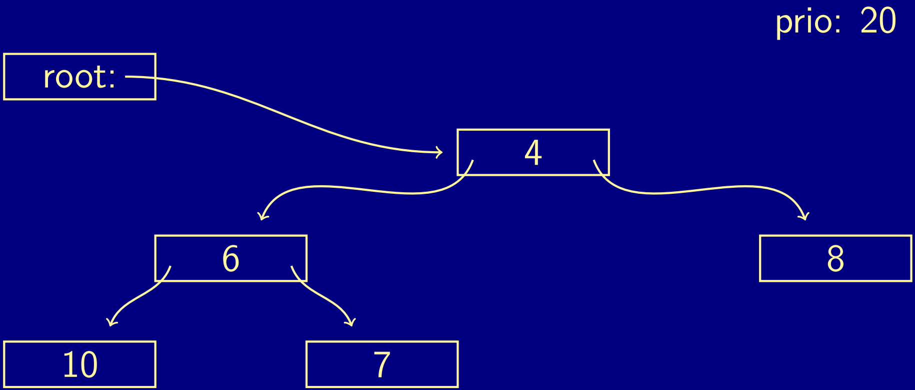
add an element to a heap

root:

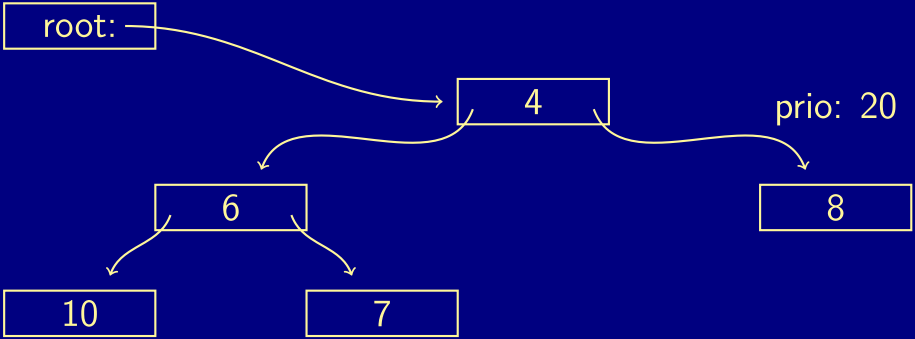
add an element to a heap



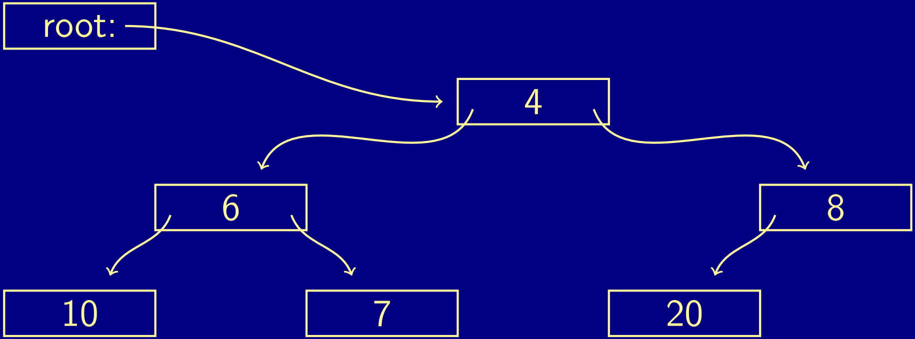
add an element to a heap



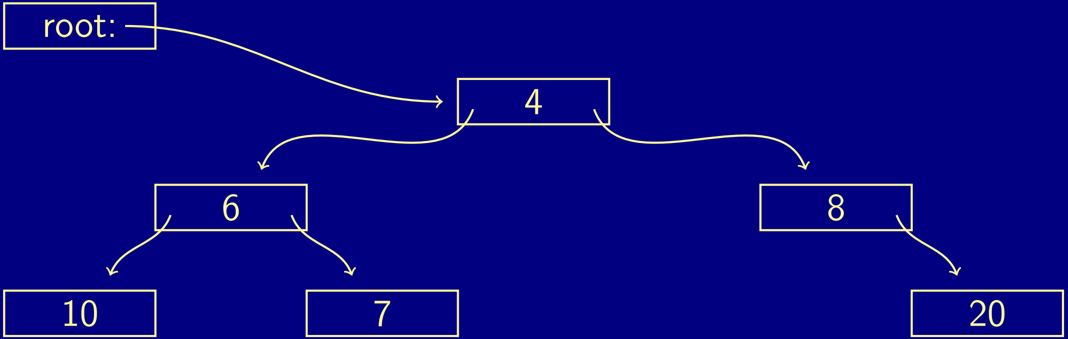
add an element to a heap



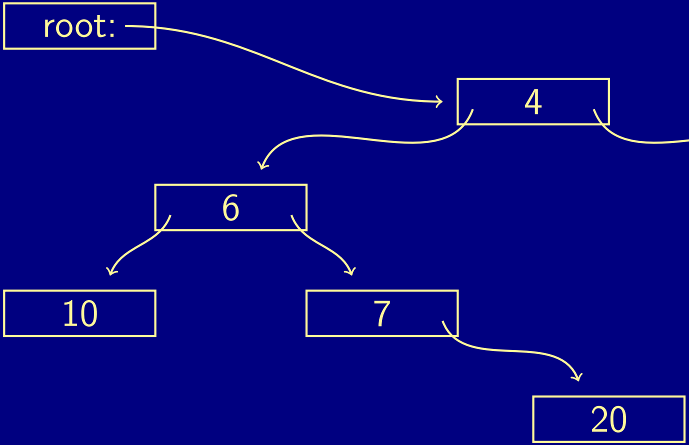
add an element to a heap



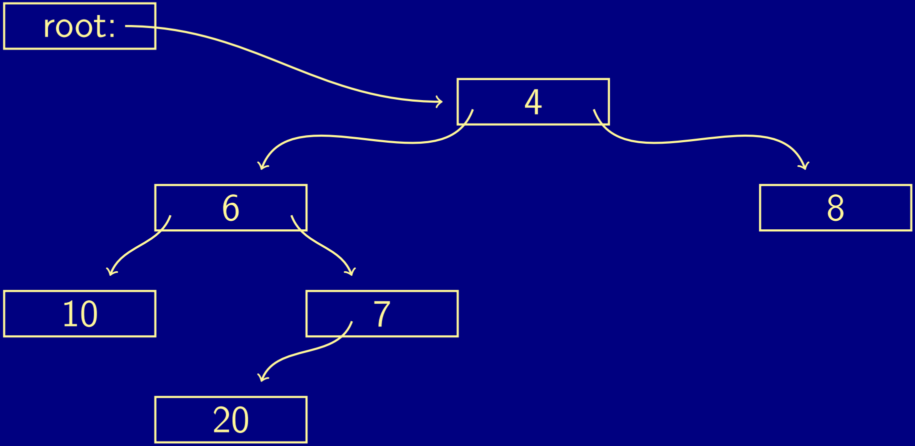
add an element to a heap



add an element to a heap



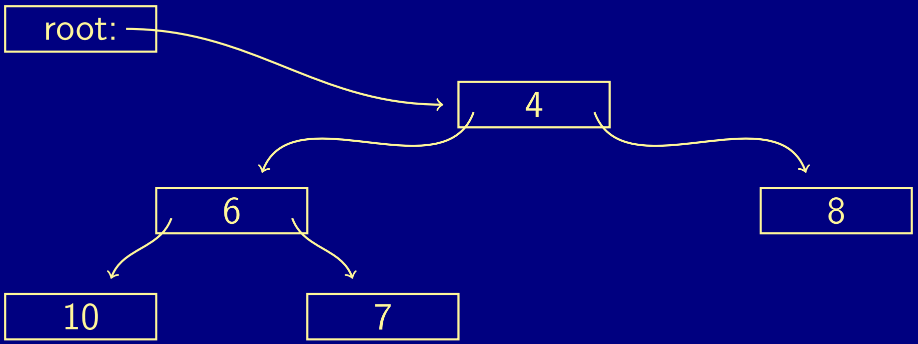
add an element to a heap



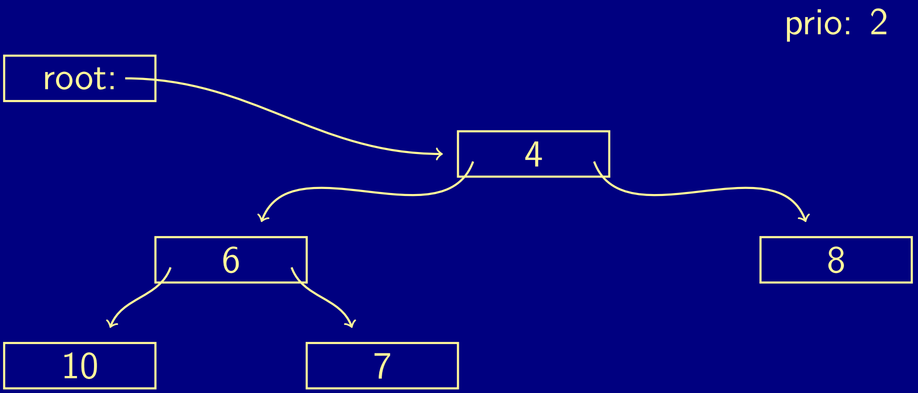
add an element to a heap

root:

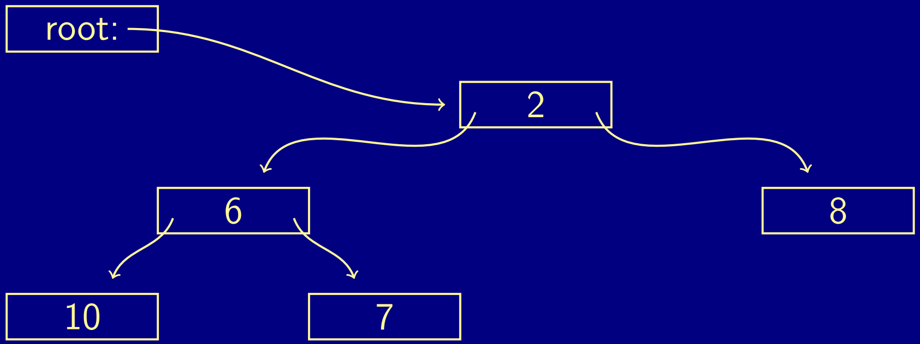
add an element to a heap



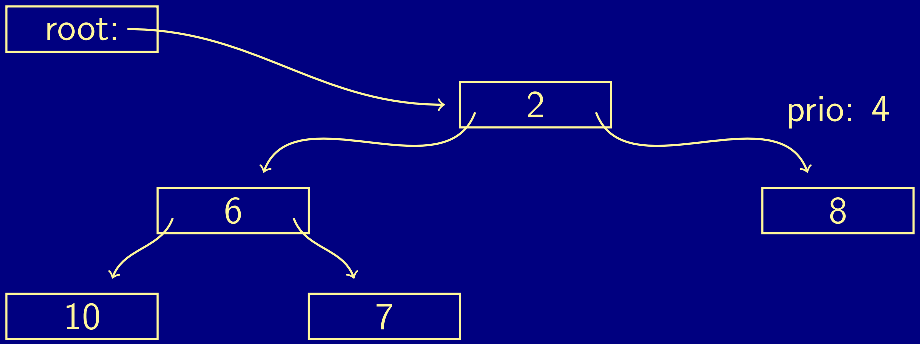
add an element to a heap



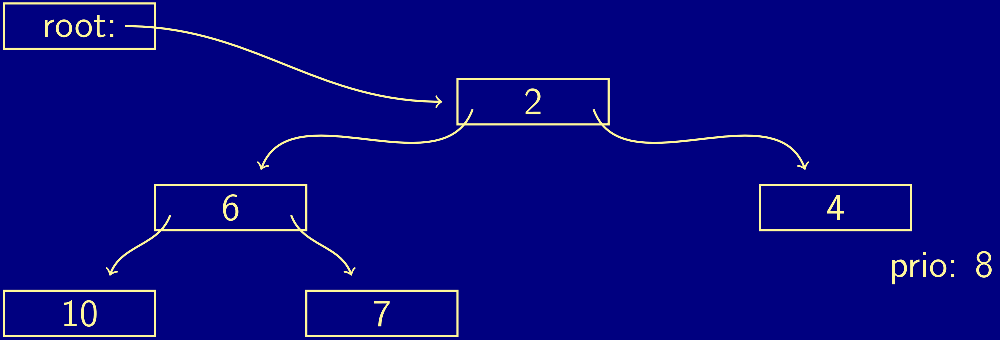
add an element to a heap



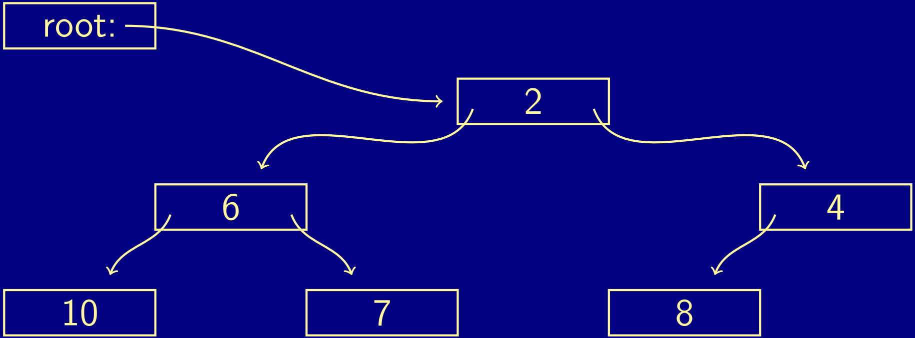
add an element to a heap



add an element to a heap



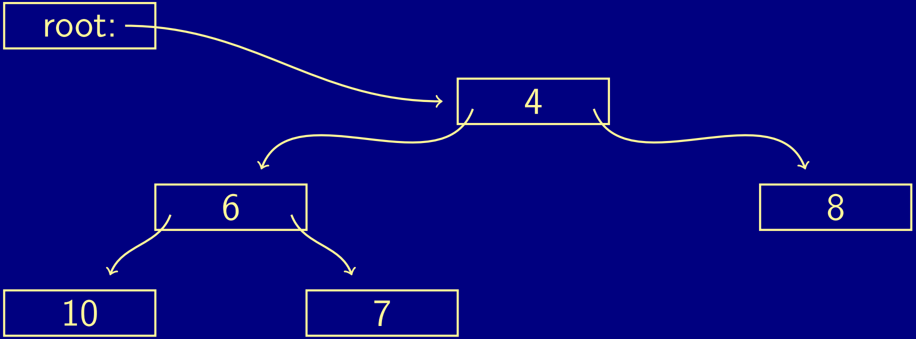
add an element to a heap



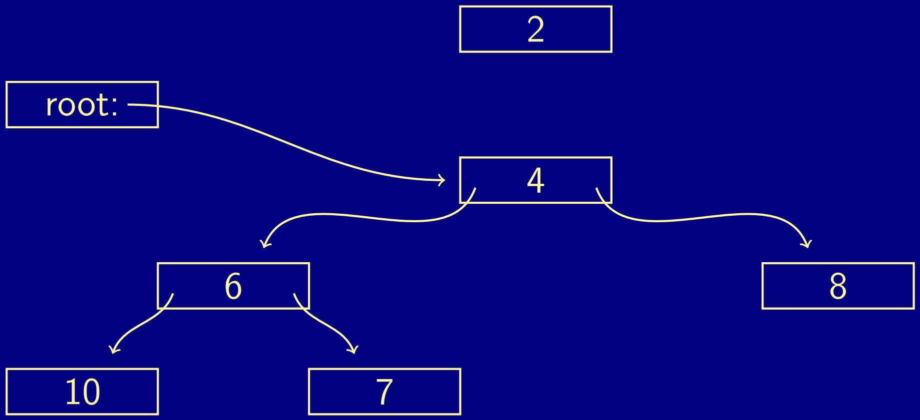
an alternative

root:

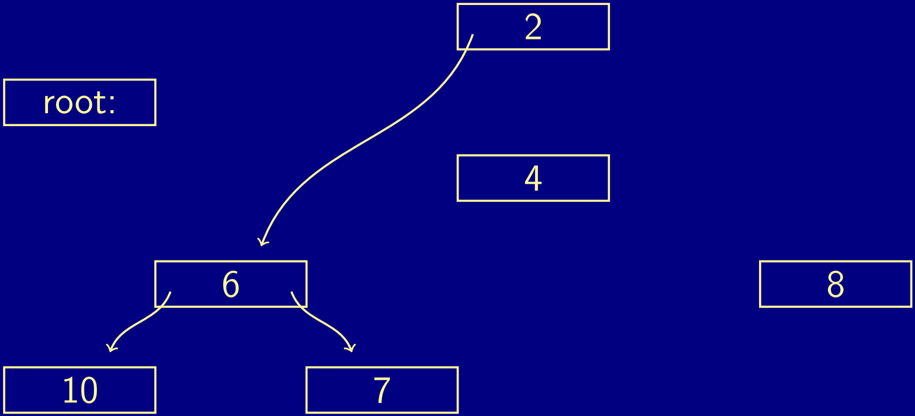
an alternative



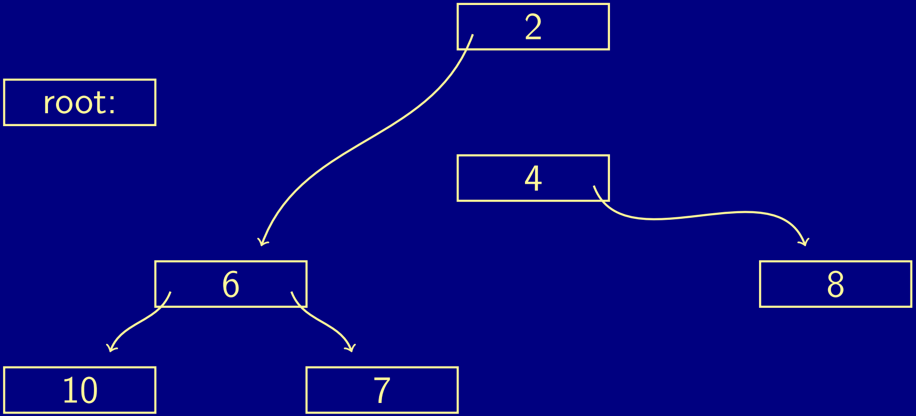
an alternative



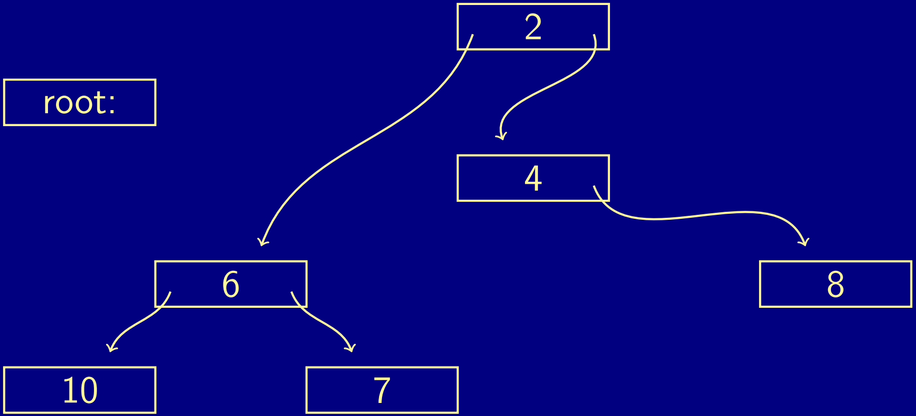
an alternative



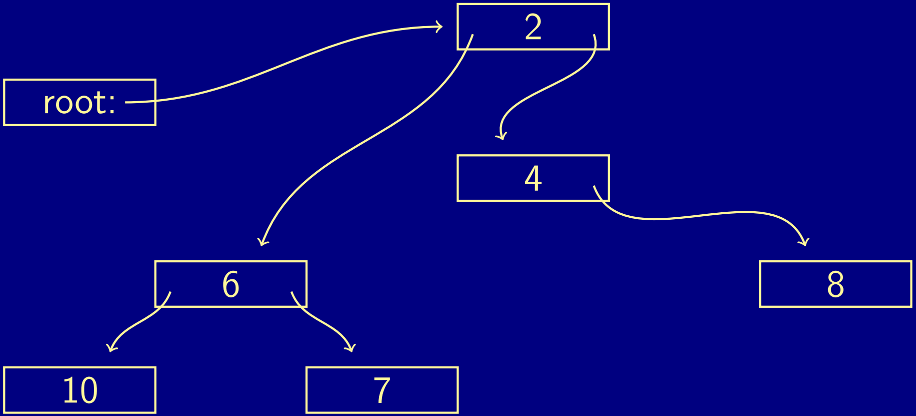
an alternative



an alternative



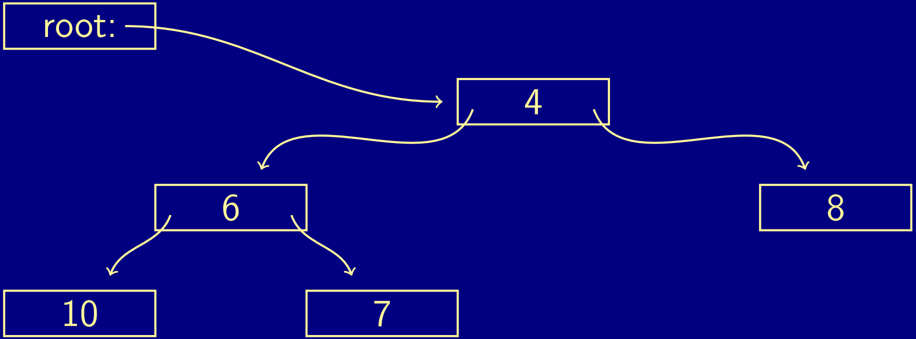
an alternative



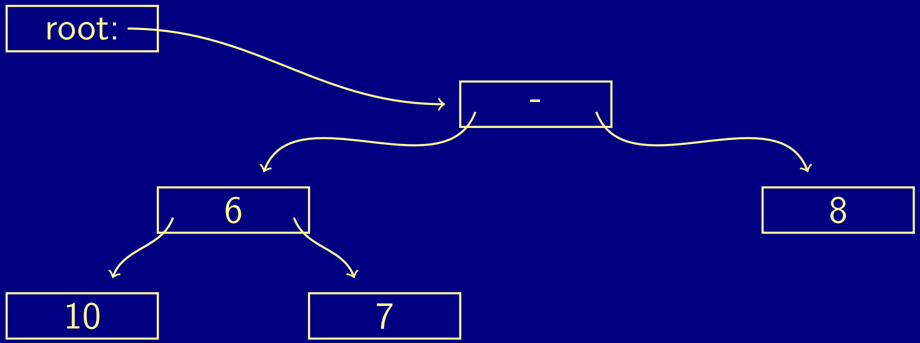
remove the next item

root:

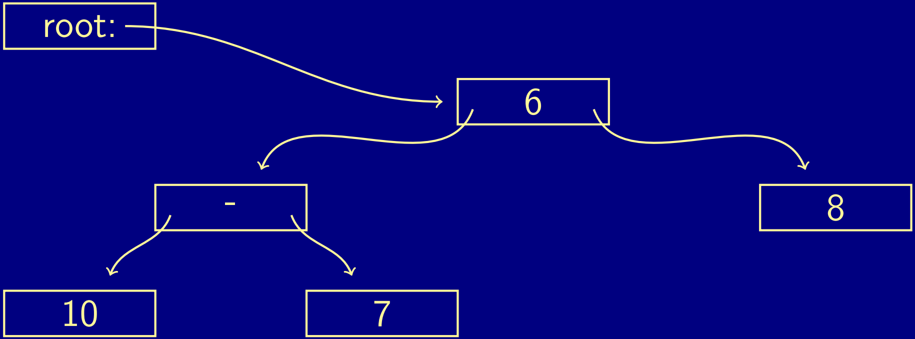
remove the next item



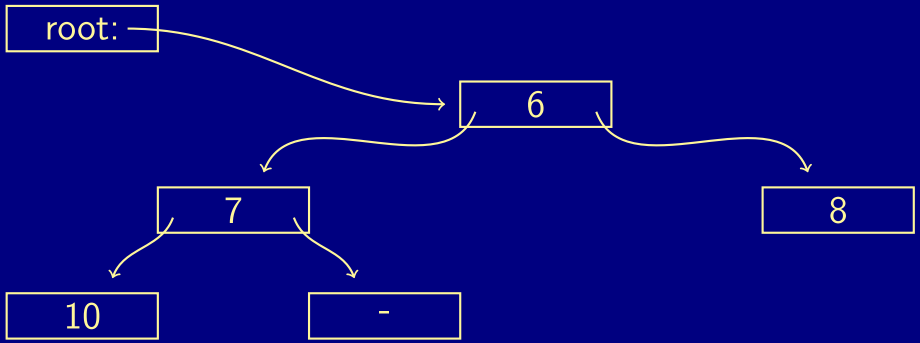
remove the next item



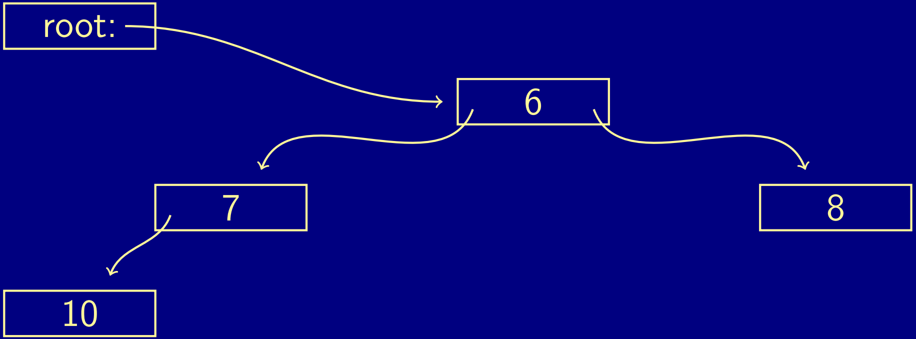
remove the next item



remove the next item



remove the next item



a push operation

a push operation

A frequent operation is to remove and then immediately add the same item with a lower priority.

a push operation

A frequent operation is to remove and then immediately add the same item with a lower priority.

- look at the item with highest priority

a push operation

A frequent operation is to remove and then immediately add the same item with a lower priority.

- look at the item with highest priority
- change the priority and update the tree.

not ideal

not ideal

The solution works but:

- tree will become unbalanced,

not ideal

The solution works but:

- tree will become unbalanced,
- needs to keep track of size

not ideal

The solution works but:

- tree will become unbalanced,
- needs to keep track of size
- add to smallest branch

not ideal

The solution works but:

- tree will become unbalanced,
- needs to keep track of size
- add to smallest branch
- adjust the branches when removing items

not ideal

The solution works but:

- tree will become unbalanced,
- needs to keep track of size
- add to smallest branch
- adjust the branches when removing items

it's a fun exercise

a linked implementation

```
class Heap<T> {

  Node root;

  private class Node {

    T item;
    int prio;
    int size;
    Node Left, right;

    :
    }
  :
}
```

enqueu an item given priority

```
public void enqueue(T itm; int pr) {
  if (root == null)
    root = new Node(itm, pr);
  else
    root.enqueu();
}
```

enqueu an item given priority

```
private void enqueue(T itm, int p) {
    size = size+1;
    if (p < prio) {
      // swap item and priority
    }
    if( right != null )
        if( left != null )
            // add to smallest branch
        else
            left = ...
    else
      right = ...
}
```

dequeu the next item

```
public T dequeue() {
  if (root == null)
     return null;
  else {
    T itm = root.item;
    root = root.remove();
    return itm;
  }
}
```

remove the node and promote the node with higest priority

```
public Node remove() {
  if ( right == null && left == null )
     return null;
  if ( right == null )
     return left;
  if ( left == null )
     return right;
  // the tricky part
       :
       :
  return this;
}
```

it works .... but

it works .... but

can we do better?

a complete tree

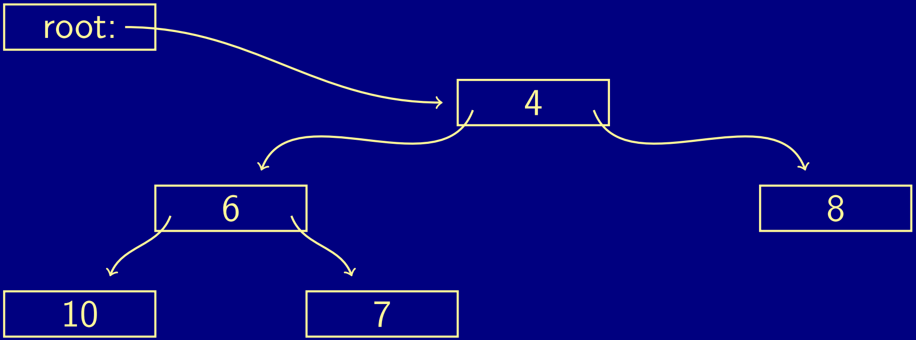
a complete tree

- The tree is *complete* i.e. all levels are filled apart from the last level that is filled from the left.

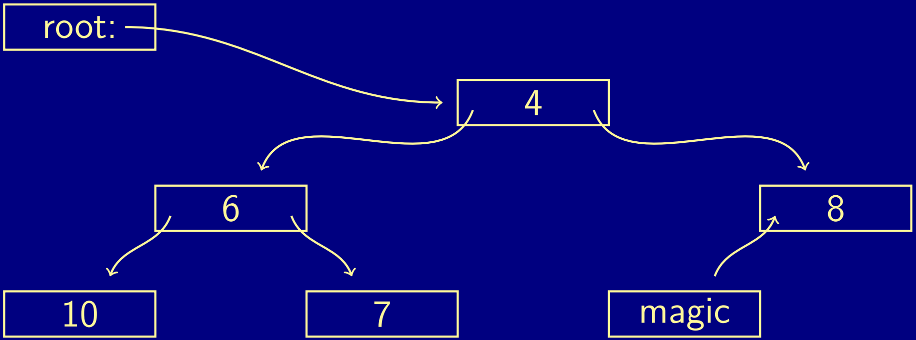
a complete tree

- The tree is *complete* i.e. all levels are filled apart from the last level that is filled from the left.
- The tree is still complete after an add or remove operation.

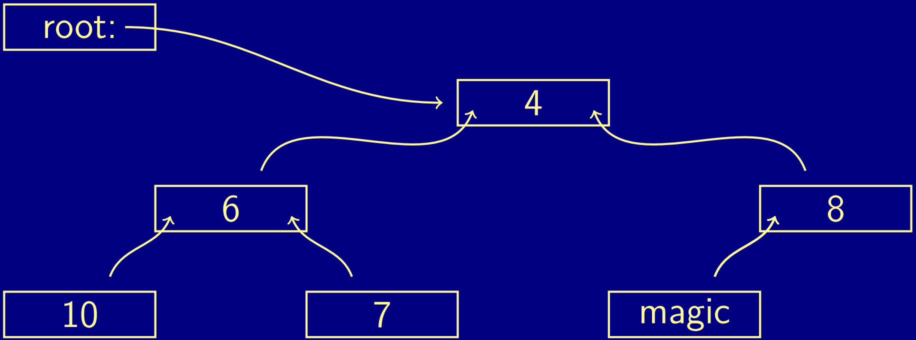
the complete tree - add



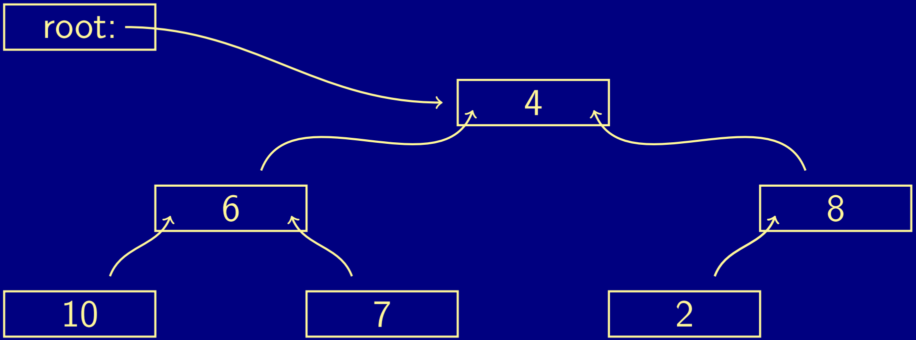
the complete tree - add



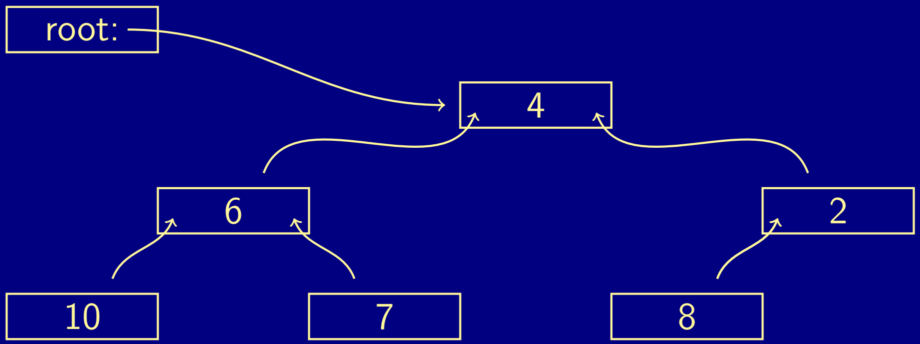
the complete tree - add



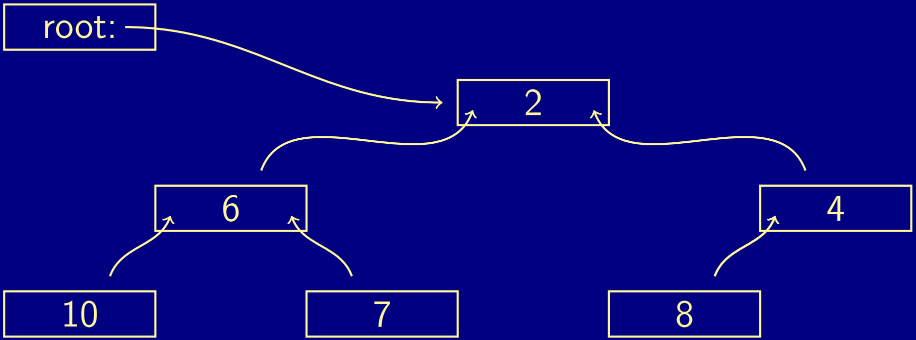
the complete tree - add



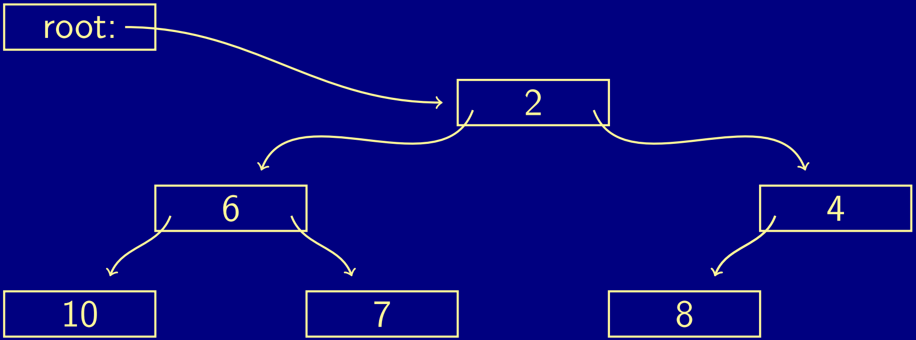
the complete tree - add



the complete tree - add



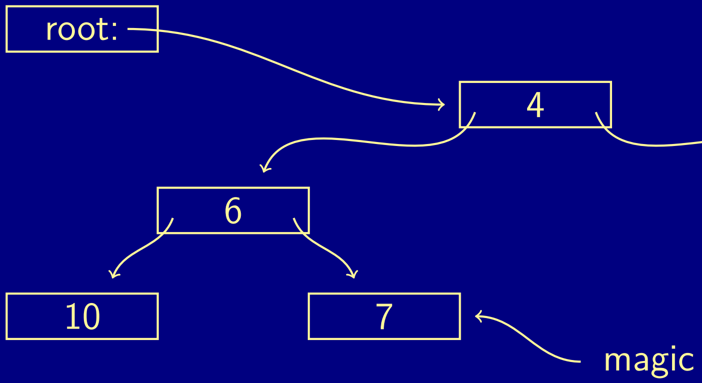
the complete tree - add



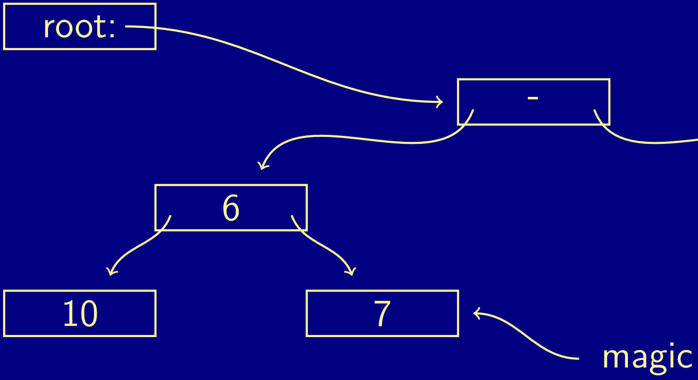
the complete tree - remove

root:

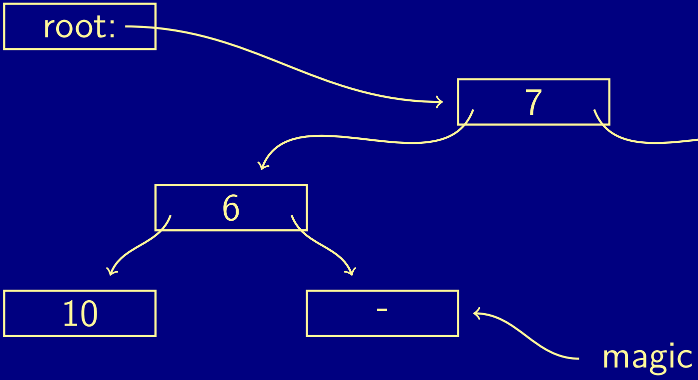
the complete tree - remove



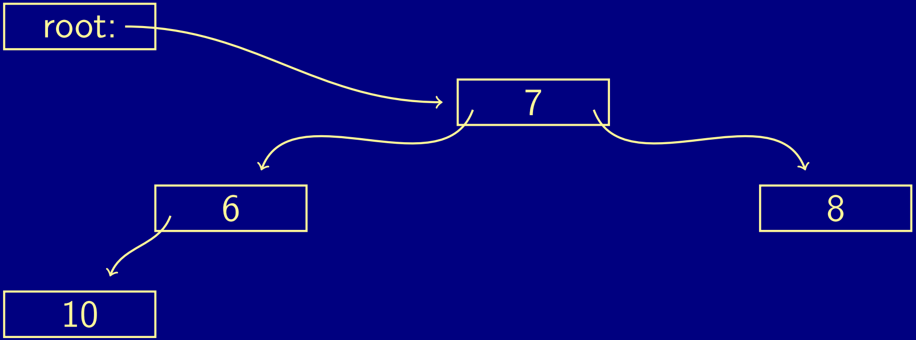
the complete tree - remove



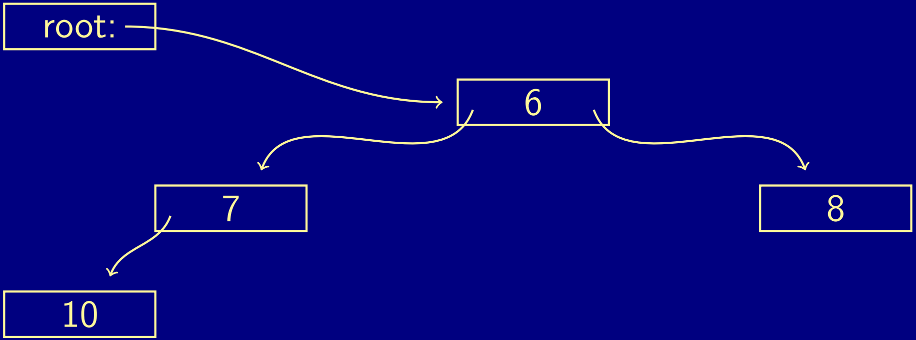
the complete tree - remove



the complete tree - remove



the complete tree - remove

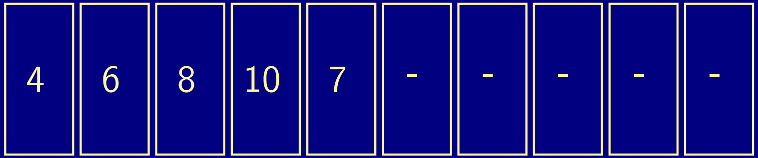


this is hard

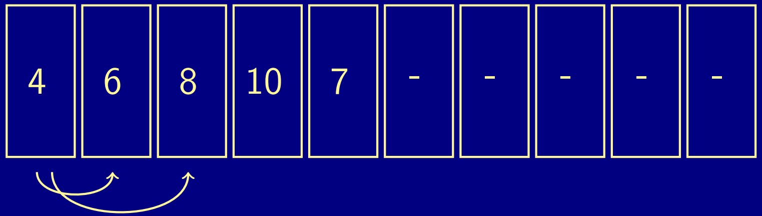
this is hard

There will be a lot of bookkeeping to make this work.

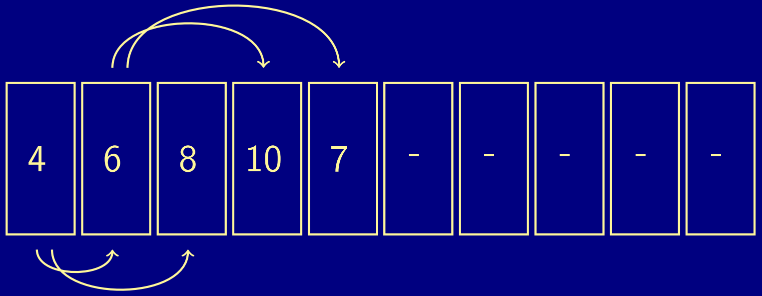
an array implementation



an array implementation



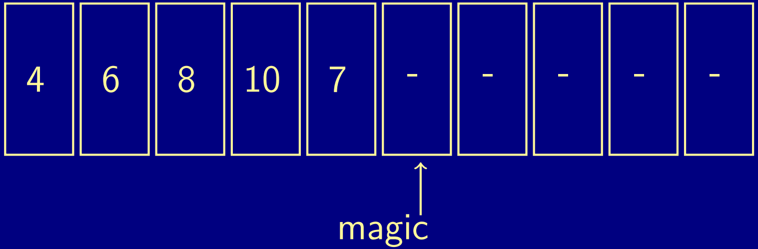
an array implementation



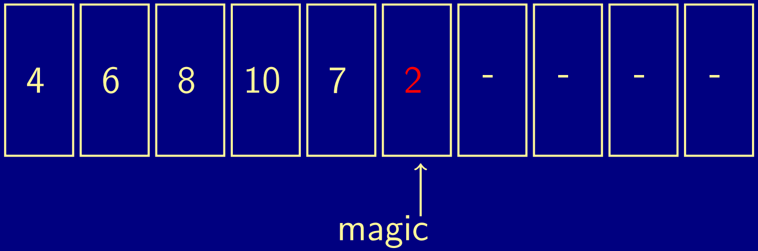
add operation

4	6	8	10	7	-	-	-	-	-

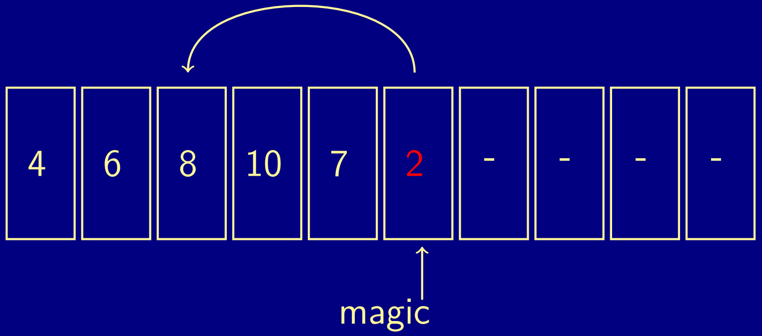
add operation



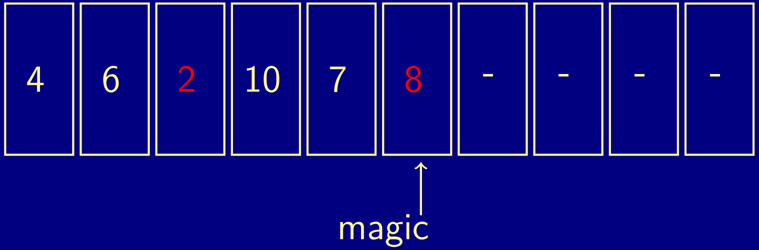
add operation



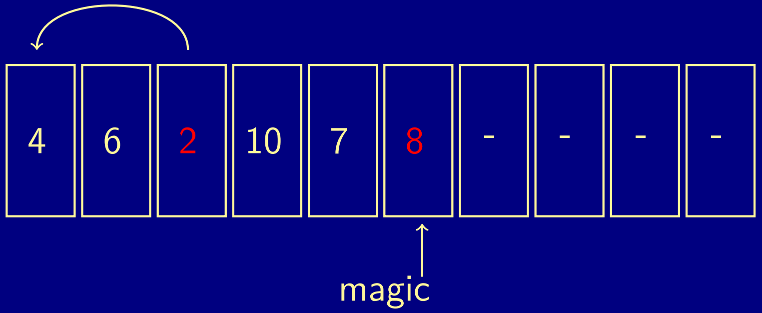
add operation



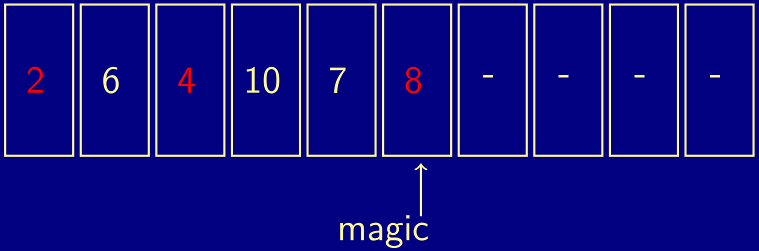
add operation



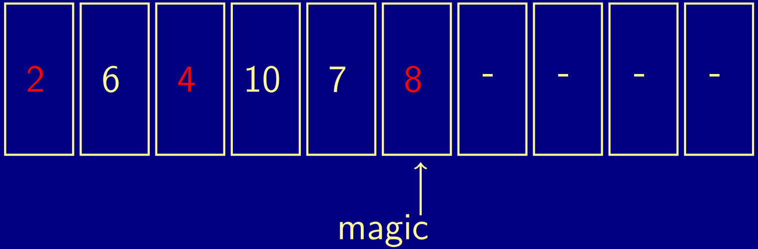
add operation



add operation



add operation



The new item bubbles upwards.

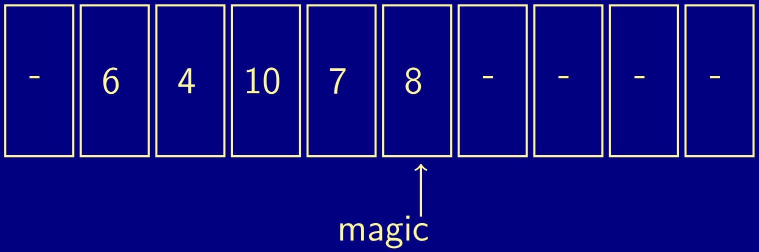
remove operation

2	6	4	10	7	8	-	-	-	-

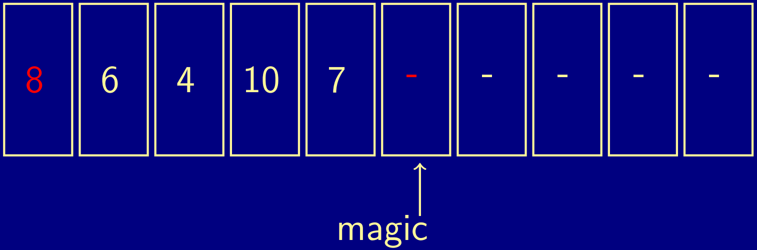
remove operation

-	6	4	10	7	8	-	-	-	-

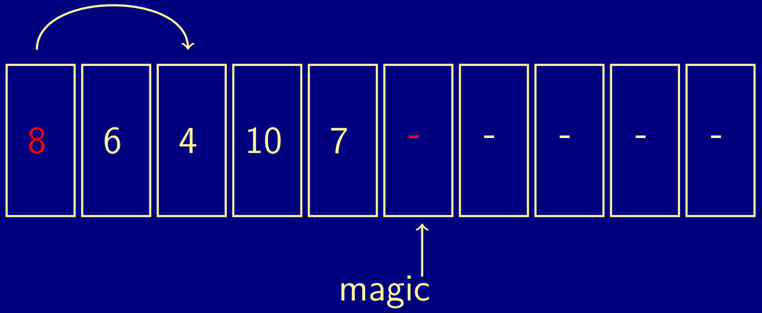
remove operation



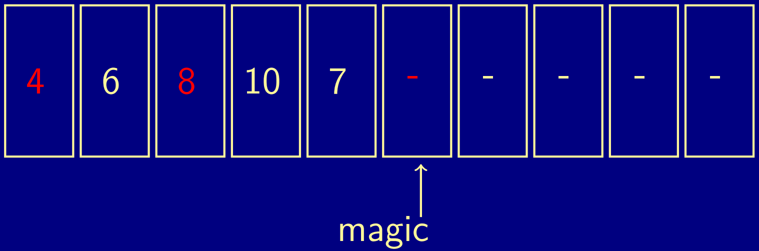
remove operation



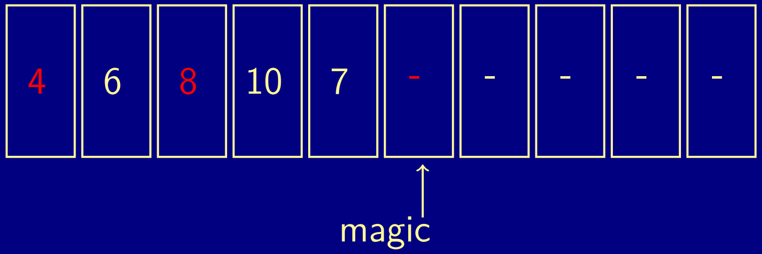
remove operation



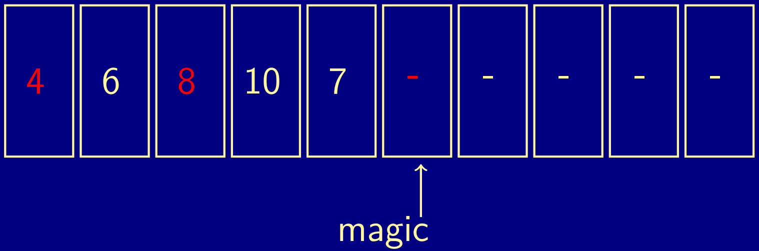
remove operation



remove operation



remove operation



The promoted item sinks.