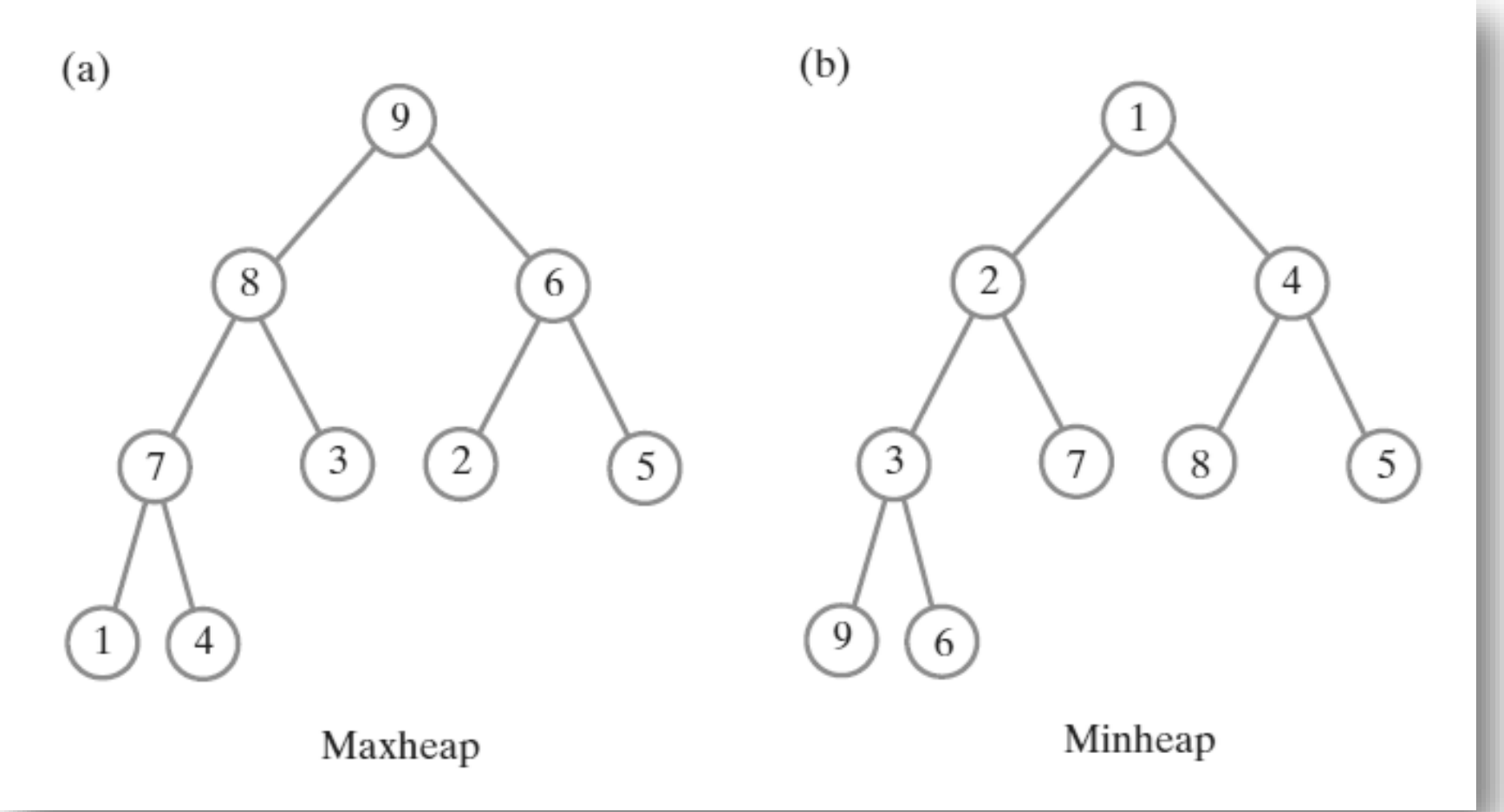
A Heap Implementation

Heaps

- Definition: A **heap** is a *complete binary tree* whose nodes contain Comparable objects and are organized as follows.
  - Each node contains an object that is no smaller (or no larger) than the objects in its descendants.

- In a **maxheap**, the object in a node is greater than or equal to its descendant objects.
- In a **minheap**, the relation is less than or equal to.



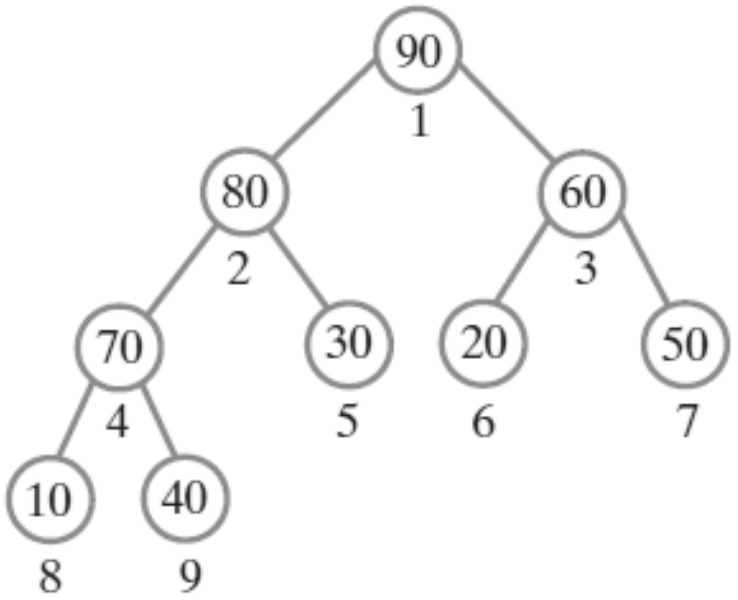
- The root of a **maxheap** contains the largest object in the heap. Notice that the subtrees of any node in a maxheap are also maxheaps.
- For simplicity, we use integers instead of objects in our illustrations.

```java
public interface MaxHeapInterface<T extends Comparable<? super T>>
{
   /** Adds a new entry to this heap.
       @param newEntry  An object to be added. */
   public void add(T newEntry);

   /** Removes and returns the largest item in this heap.
       @return  Either the largest object in the heap or,
                if the heap is empty before the operation, null. */
   public T removeMax();

   /** Retrieves the largest item in this heap.
       @return  Either the largest object in the heap or,
                if the heap is empty, null. */
   public T getMax();

   /** Detects whether this heap is empty.
       @return  True if the heap is empty, or false otherwise. */
   public boolean isEmpty();

   /** Gets the size of this heap.
       @return  The number of entries currently in the heap. */
   public int getSize();

   /** Removes all entries from this heap. */
   public void clear();
} // end MaxHeapInterface
```

Using an array to represent a Heap

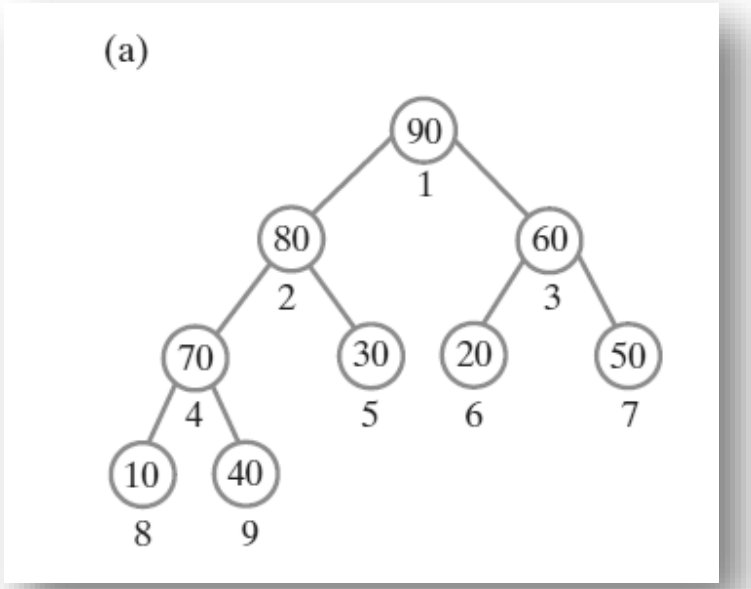
- Use an array to represent a complete binary tree
- Number nodes in the order in which a level-order traversal would visit them, beginning with 1
- Can locate either the children or the parent of any node
  - Perform a simple computation on the node's number

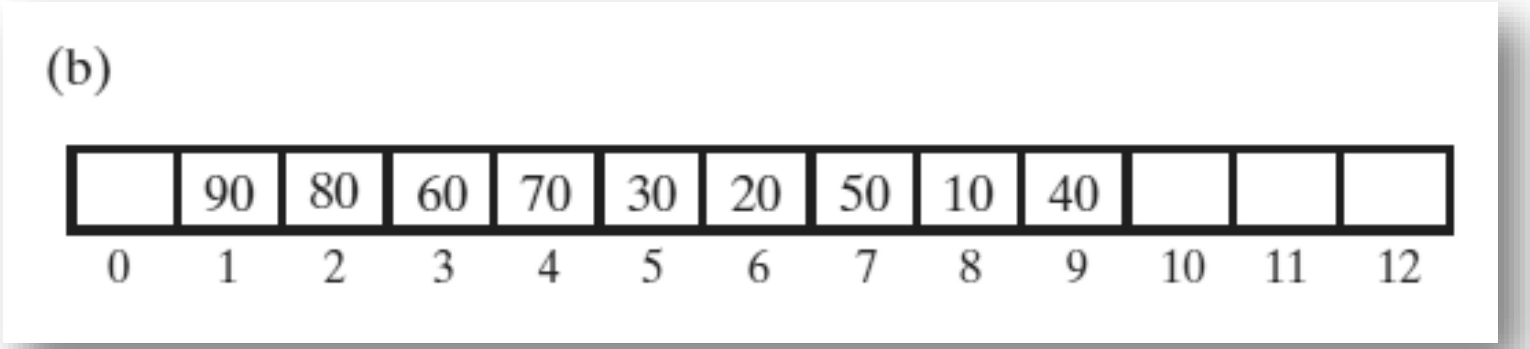
(a)



(b)

	90	80	60	70	30	20	50	10	40			
0	1	2	3	4	5	6	7	8	9	10	11	12

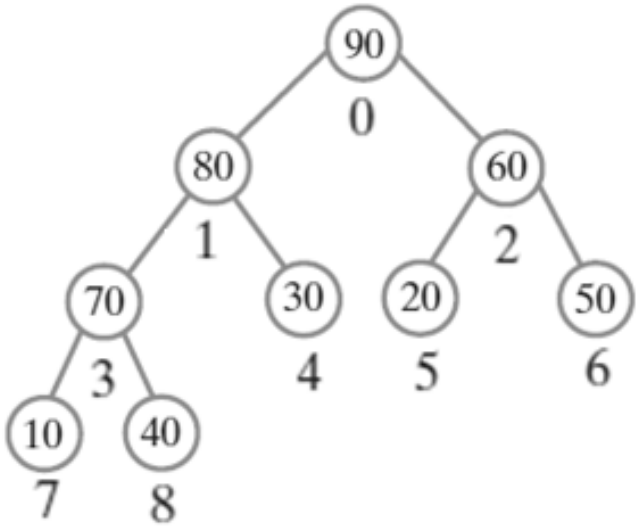




- Since the tree is complete, we can locate either the children or the parent of any node by performing a simple computation on the node's number. This number is the same as the node's corresponding array index.
- Thus, the children of the node $\boldsymbol{i}$ – if they exist – are stored in the array at indices $\boldsymbol{2 \times i}$ and $\boldsymbol{2 \times i + 1}$.
- The parent of this node is at array index $\boldsymbol{i/2}$, unless, of course, the node is the root. In that case, $\boldsymbol{i/2}$ is $\boldsymbol{0}$, since the root is at index $\boldsymbol{1}$.

Question

- If an array contains the entries of a heap in level order beginning at index 0, what array entries represent a node's parent, left child, and right child?



90	80	60	70	30	20	50	10	40			
0	1	2	3	4	5	6	7	8	9	10	11

```java
import java.util.Arrays;
public class MaxHeap<T extends Comparable<? super T>>
            implements MaxHeapInterface<T>
{
    private T[] heap;      // Array of heap entries
    private int lastIndex; // Index of last entry and number of entries
    private boolean initialized = false;
    private static final int DEFAULT_CAPACITY = 25;
    private static final int MAX_CAPACITY = 10000;

    public MaxHeap()
    {
        this(DEFAULT_CAPACITY); // Call next constructor
    } // end default constructor

    public MaxHeap(int initialCapacity)
    {
        checkCapacity(initialCapacity);

        // The cast is safe because the new array contains null entries
        @SuppressWarnings("unchecked")
        T[] tempHeap = (T[])new Comparable[initialCapacity + 1];
        heap = tempHeap;
        lastIndex = 0;
        initialized = true;
    } // end constructor
```

```java
public interface
 MaxHeapInterface<T extends Comparable<? super T>>
{
 public void add(T newEntry);
 public T removeMax();
 public T getMax();
 public boolean isEmpty();
 public int getSize();
 public void clear();
} // end MaxHeapInterface
```

```java
private void ensureCapacity()
{
 if(lastIndex >= heap.length)
 {
  int newCapacity = 2*(heap.length - 1);
  checkCapacity(newCapacity);
  heap = Arrays.copyOf(heap, newCapacity);
 } // end if
} // end ensureCapacity
```

```java
private void checkInitialization()
{
 if(!initialized)
  throw new SecurityException("MaxHep object is not initialized properly.");
} // end checkInitialization
```

```java
private void checkCapacity(int capacity)
{
 if(capacity < DEFAULT_CAPACITY)
  capacity = DEFAULT_CAPACITY;
 else
  throw new IllegalStateException("Attempt to create a heap "+
        "whose capacity is larger than "+MAX_CAPACITY);
} // end checkCapacity
```

```java
public T getMax()
{
    checkInitialization();
    T root = null;

    if (!isEmpty())
        root = heap[1];

    return root;
} // end getMax
```

```java
public boolean isEmpty()
{
    return lastIndex < 1;
} // end isEmpty
```

```java
public int getSize()
{
    return lastIndex;
} // end getSize
```

```java
public void clear()
{
    checkInitialization();
    while (lastIndex > -1)
    {
        heap[lastIndex] = null;
        lastIndex--;
    } // end while

    lastIndex = 0;
} // end clear
```

References

[1] F. M. Carrano and T. M. Henry, “Data Structures and Abstractions with Java”, 4th edition. 2015.