Binary Tree Application Heap Sorting

Revised based on textbook author's notes.

The Heapsort

- The simplicity and efficiency of the heap structure can be applied to the sorting problem.
  - Build a heap from a sequence of unsorted keys.
  - Extract the keys from the heap to create a sorted sequence.
- Very efficient: $O(n \log n)$

Heapsort Implementation

- A simple implementation is provided below.

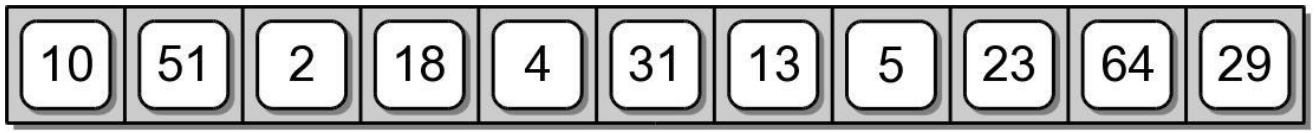
```
def simple_heap_sort( the_seq ):
  # Create an array-based max-heap.
  n = len(the_seq)
  heap = MaxHeap( n )

  # Build a max-heap from the list of values.
  for item in the_seq :
    heap.add( item )

  # Extract each value from the heap and store
  # them back into the list.
  for i in range( n-1, -1, -1 ) : # small to large
  # for I in range( n ) :  # large to small
    theSeq[i] = heap.extract()
```

In-Place Heapsort

- The previous version required additional storage for the heap.
- The entire process can be done in-place within the original sequence.
  - Suppose we are given the following array

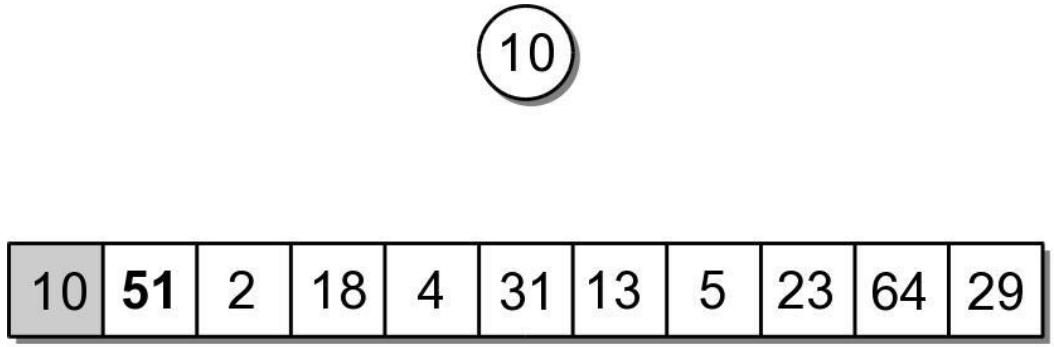


In-Place Heapsort

- The first step is to construct a max-heap.
  - The heap nodes occupy the array from front to back.
  - We can keep the heap elements in the front and those yet to be added at the back.
  - Keep track of where the heap ends.

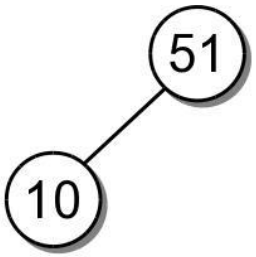
In-Place Heapsort

- The first value in the array represents a max-heap of one element.



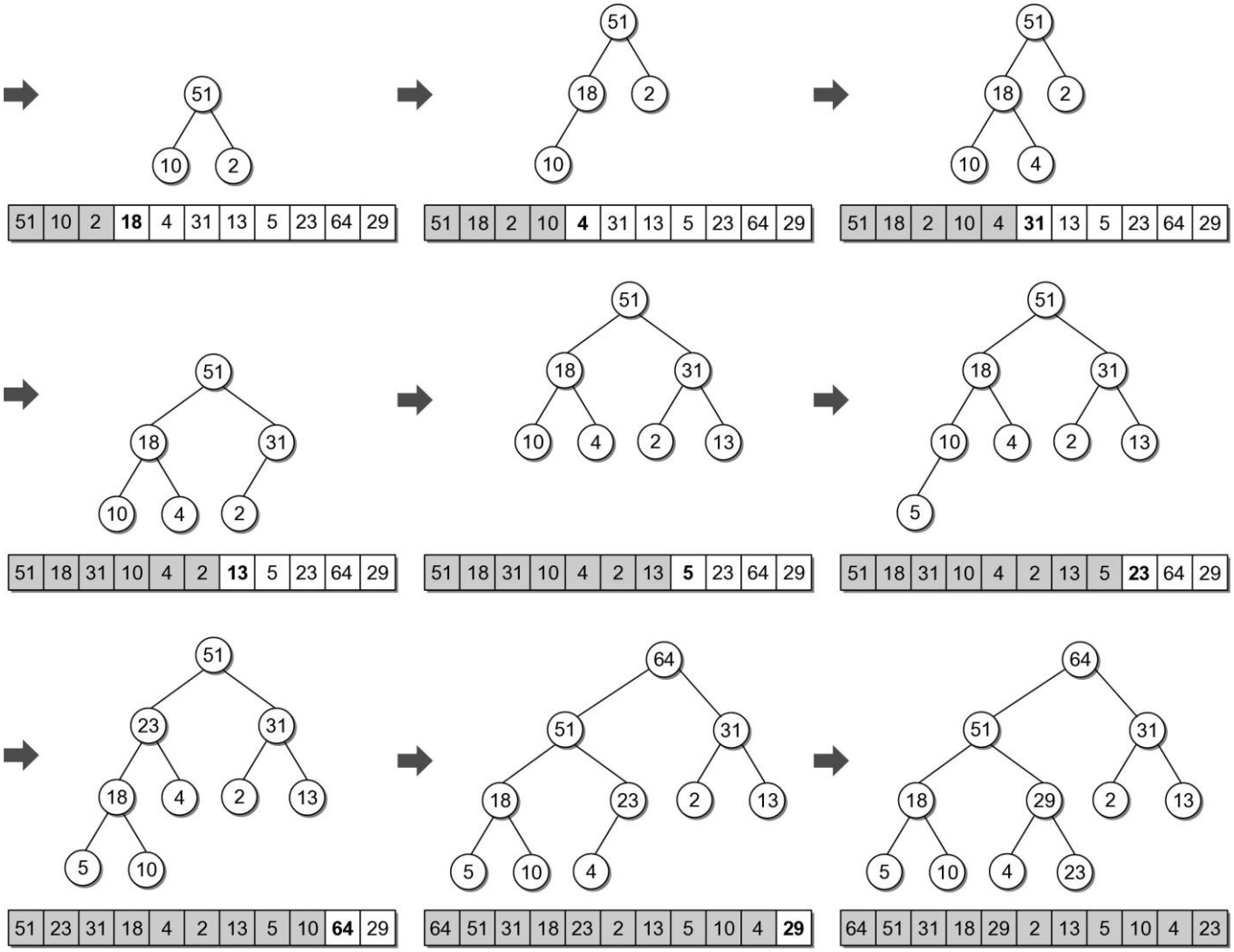
In-Place Heapsort

- The next value to be added, is the next in the array.
  - The value is copied to the root node (position 0).
  - Then sifted down.



51	10	**2**	18	4	31	13	5	23	64	29

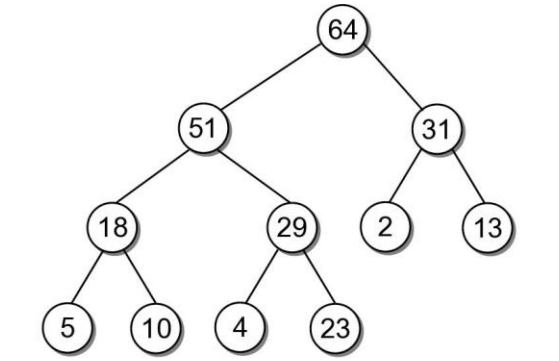
In-Place Heapsort



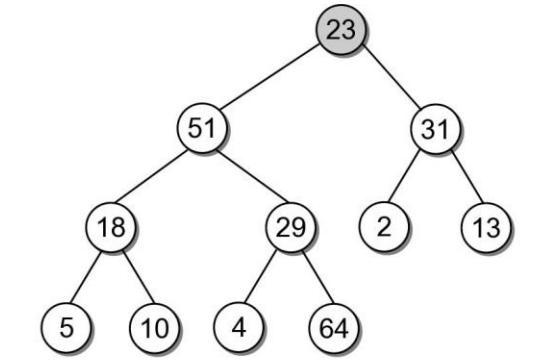
In-Place Heapsort

- The next step is to extract the values from the heap and build the sorted sequence.
  - When the root is extracted, the last child node is copied to the root.
  - The last child node is stored in the last element of the heap.
  - We can simply swap the two values.

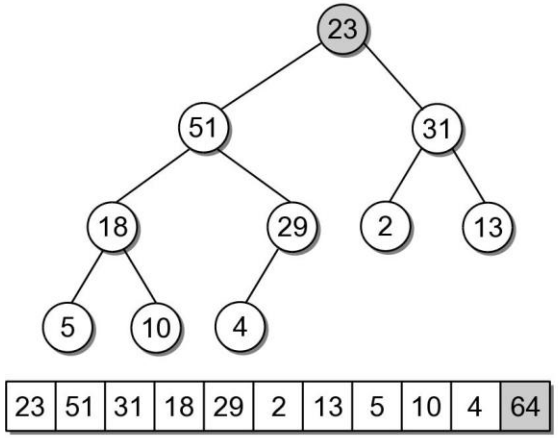
In-Place Heapsort



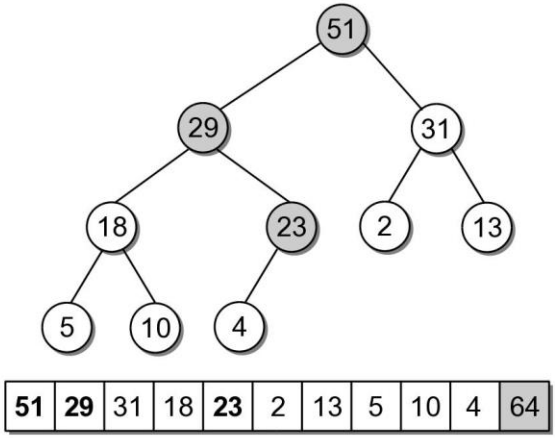
(a) the original max-heap.



(b) swap the first and last items in the heap.

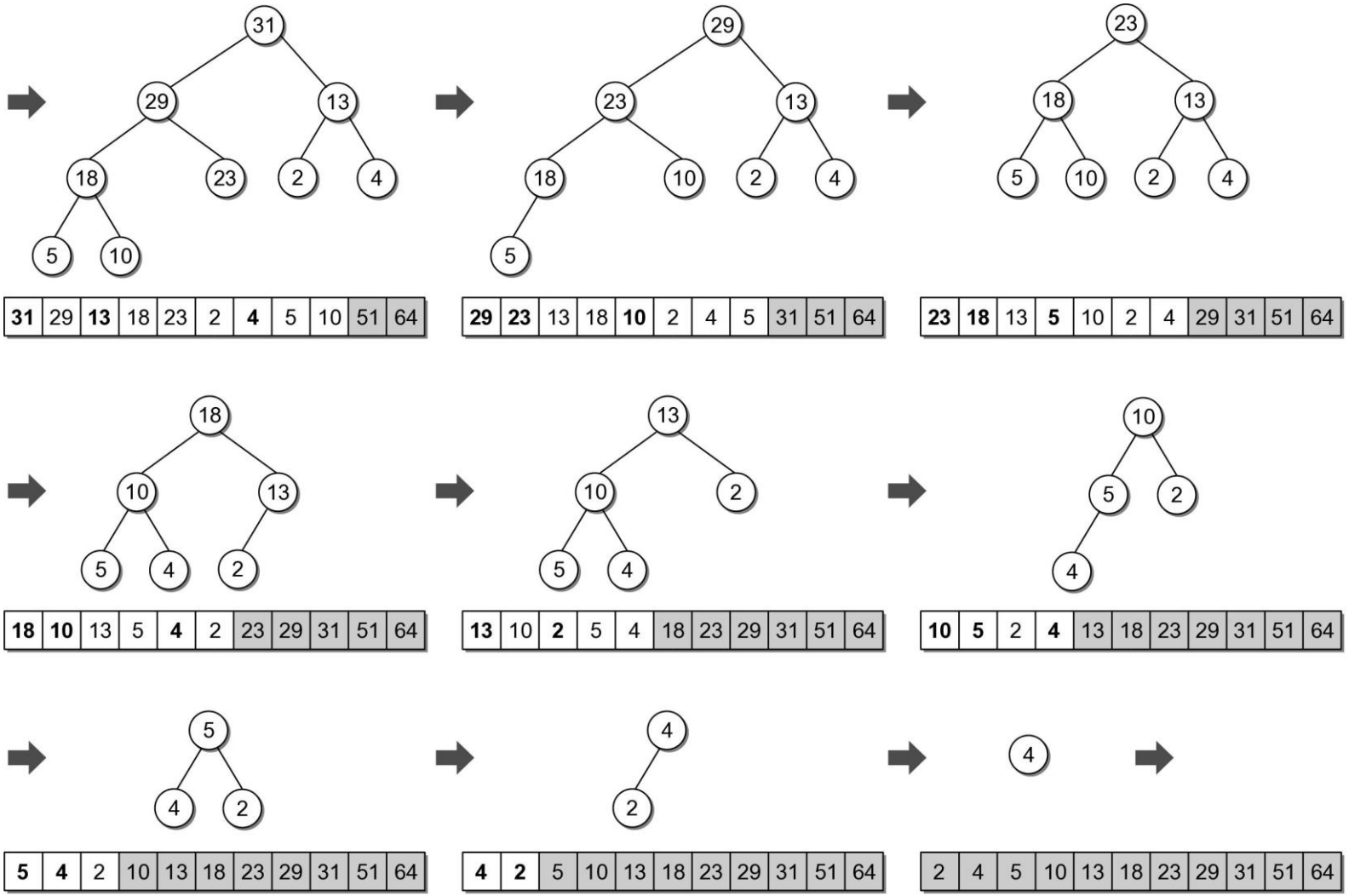


(c) remove the last item from the heap.



(d) sift the root value down the tree.

In-Place Heapsort



In-Place Heapsort Implementation

`sift_up()` and `sift_down()` are helper functions based on those used in the heap class.

```
def heapsort( the_seq ):
  n = len(the_seq)
   # Build a max-heap within the same array.
  for i in range( n ) :
    siftUp( the_seq, i )

   # Extract each value and rebuild the heap.
  for j in range( n-1, 0, -1) :
    tmp = the_seq[j]
    the_seq[j] = the_seq[0]
    the_seq[0] = tmp
    siftDown( the_seq, j-1, 0 )
```

Try heapsort.py