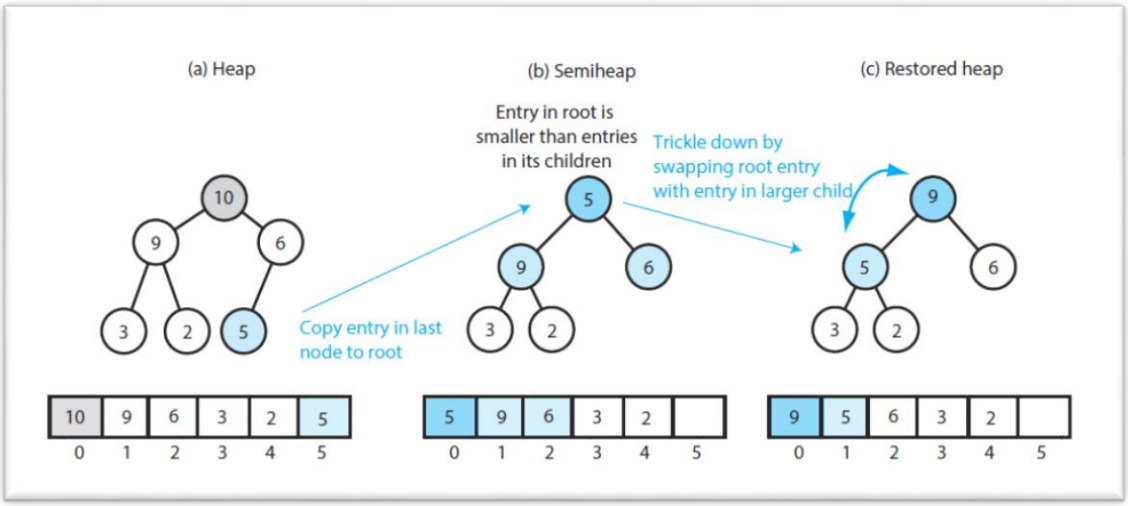
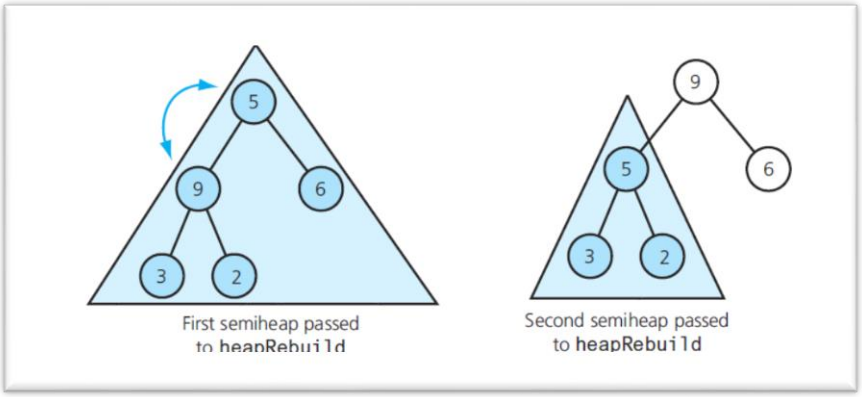
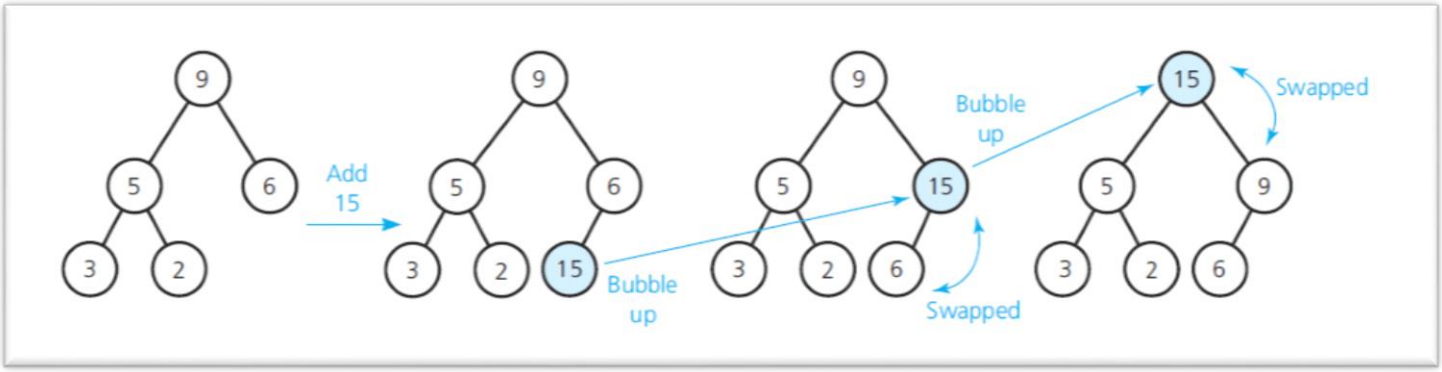





```
// Converts a semiheap rooted at index nodeIndex into a heap.
heapRebuild(nodeIndex: integer, items: ArrayType, itemCount: integer): void
{
   // Recursively trickle the item at index nodeIndex down to its proper position by
   // swapping it with its larger child, if the child is larger than the item.
   // If the item is at a leaf, nothing needs to be done.
   if (the root is not a leaf)
   {
      // The root must have a left child; find larger child
      leftChildIndex = 2 * rootIndex + 1
      rightChildIndex = leftChildIndex + 1
      largerChildIndex = rightChildIndex // Assume right child exists and is the larger
      // Check whether right child exists; if so, is left child larger?
      // If no right child, left one is larger
      if ((largerChildIndex >= itemCount) ||
                              (items[leftChildIndex] > items[rightChildIndex]))
         largerChildIndex = leftChildIndex; // Assumption was wrong
     if (items[nodeIndex] < items[largerChildIndex])
     {
        Swap items[nodeIndex] and items[largerChildIndex]
        // Transform the semiheap rooted at largerChildIndex into a heap
        heapRebuild(largerChildIndex, items, itemCount)
     }
   }
   // Else root is a leaf, so you are done
}
```




```
add(newData: itemType): boolean
{
   // Place newData at the bottom of the tree
   items[itemCount] = newData
   // Make new item bubble up to the appropriate spot in the tree
   newDataIndex = itemCount
   inPlace = false
   while ((newDataIndex >= 0) and !inPlace)
   {
      parentIndex = (newDataIndex - 1) / 2
      if (items[newDataIndex] <= items[parentIndex])
         inPlace = true
      else
      {
         Swap items[newDataIndex] and items[parentIndex]
         newDataIndex = parentIndex
      }
   }
   itemCount++
   return inPlace
}
```

```
================================
// chapter 17  Heaps  source code
================================
//  Created by Frank M. Carrano and Timothy M. Henry.
//  Copyright (c) 2017 Pearson Education, Hoboken, New Jersey.

template<class ItemType>
ArrayMaxHeap<ItemType>::
ArrayMaxHeap(const ItemType someArray[], const int arraySize):
itemCount(arraySize), maxItems(2 * arraySize)
{
  // Allocate the array
  items = std::make_unique<ItemType[]>(maxItems);

  // Copy given values into the array
  for (int i = 0; i < itemCount; i++)
    items[i] = someArray[i];

  // Reorganize the array into a heap
  heapCreate();
} // end constructor

template<class ItemType>
void ArrayMaxHeap<ItemType>::heapCreate()
{
  for (int index = itemCount / 2 - 1; index >= 0; index--)
  {
    heapRebuild(index);
  }  // end for
}  // end heapCreate


-----=====
//  Created by Frank M. Carrano and Timothy M. Henry.
//  Copyright (c) 2017 Pearson Education, Hoboken, New Jersey.

template<class ItemType>
int ArrayMaxHeap<ItemType>::getLeftChildIndex(const int
nodeIndex) const
{
  return (2 * nodeIndex) + 1;
}  // end getLeftChildIndex

-----=====
//  Created by Frank M. Carrano and Timothy M. Henry.
//  Copyright (c) 2017 Pearson Education, Hoboken, New Jersey.

template<class ItemType>
ItemType ArrayMaxHeap<ItemType>::peekTop() const
throw(PrecondViolatedExcept)
{
  if (isEmpty())
    throw PrecondViolatedExcept("Attempted peek into an empty
                                                    heap.");
  return items[0];
}  // end peekTop


============-
// Listing 17-3.
//  Created by Frank M. Carrano and Timothy M. Henry.
//  Copyright (c) 2017 Pearson Education, Hoboken, New Jersey.

/** ADT priority queue: Heap-based implementation.
 @file HeapPriorityQueue.h */

#ifndef HEAP_PRIORITY_QUEUE_
#define HEAP_PRIORITY_QUEUE_

#include "ArrayMaxHeap.h"
#include "PriorityQueueInterface.h"

template<class ItemType>
class Heap_PriorityQueue : public
PriorityQueueInterface<ItemType>,
                  private ArrayMaxHeap<ItemType>
{
public:
  Heap_PriorityQueue();
  bool isEmpty() const;
  bool enqueue(const ItemType& newEntry);
  bool dequeue();

  /** @pre  The priority queue is not empty. */
  ItemType peekFront() const throw(PrecondViolatedExcept);
}; // end Heap_PriorityQueue

#include "Heap_PriorityQueue.cpp"
#endif



============--17.1==
// Listing 17-1.
//  Created by Frank M. Carrano and Timothy M. Henry.
//  Copyright (c) 2017 Pearson Education, Hoboken, New Jersey.

/** Interface for the ADT heap.
 @file HeapInterface.h */

#ifndef HEAP_INTERFACE_
#define HEAP_INTERFACE_

template<class ItemType>
class HeapInterface
{
public:
  /** Sees whether this heap is empty.
   @return  True if the heap is empty, or false if not. */
  virtual bool isEmpty() const = 0;
  /** Gets the number of nodes in this heap.
   @return  The number of nodes in the heap. */
  virtual int getNumberOfNodes() const = 0;
  /** Gets the height of this heap.
   @return  The height of the heap. */
  virtual int getHeight() const = 0;
  /** Gets the data that is in the root (top) of this heap.
      For a maxheap, the data is the largest value in the heap;
      for a minheap, the data is the smallest value in the heap.
   @pre  The heap is not empty.
   @post  The rootâ€™s data has been returned, and the heap is unchanged.
   @return  The data in the root of the heap. */
  virtual ItemType peekTop() const = 0;
  /** Adds a new data item to this heap.
   @param newData  The data for the new node.
   @post  The heap has a new node that contains newData.
   @return  True if the addition is successful, or false if not. */
  virtual bool add(const ItemType& newData) = 0;
  /** Removes the data that is in the root (top) of this heap.
   @return  True if the removal is successful, or false if not. */
  virtual bool remove() = 0;
  /** Removes all data from this heap. */
  virtual void clear() = 0;
  /** Destroys this heap and frees its assigned memory. */
  virtual ~HeapInterface() { }
}; // end HeapInterface
#endif
```

```
//=================
// Listing 17-2.
//  Created by Frank M. Carrano and Timothy M. Henry.
//  Copyright (c) 2017 Pearson Education, Hoboken, New Jersey.

/** Array-based implementation of the ADT heap.
 @file ArrayMaxHeap.h */

#ifndef ARRAY_MAX_HEAP_
#define ARRAY_MAX_HEAP_
#include <memory>
#include "HeapInterface.h"
#include "PrecondViolatedExcept.h"

template<class ItemType>
class ArrayMaxHeap : public HeapInterface<ItemType>
{
private:
   static const int ROOT_INDEX = 0;       // Helps with readability
   static const int DEFAULT_CAPACITY = 21; // Small capacity
                                           to test for a full heap
   std::unique_ptr<ItemType[]> items;     // Array of heap items
   int itemCount;                         // Current count of heap items
   int maxItems;                          // Maximum capacity of the heap

   // ------------------------------------------------------------
   // Most of the private utility methods use an array index as a
   // parameter and in calculations. This should be safe, even though
   // the array is an implementation detail, since the methods are
   //  private.
   // ------------------------------------------------------------
  // Returns the array index of the left child (if it exists).
   int getLeftChildIndex(const int nodeIndex) const;

   // Returns the array index of the right child (if it exists).
   int getRightChildIndex(int nodeIndex) const;

   // Returns the array index of the parent node.
   int getParentIndex(int nodeIndex) const;

   // Tests whether this node is a leaf.
   bool isLeaf(int nodeIndex) const;

   // Converts a semiheap to a heap.
   void heapRebuild(int nodeIndex);

   // Creates a heap from an unordered array.
   void heapCreate();

public:
   ArrayMaxHeap();
   ArrayMaxHeap(const ItemType someArray[], const int arraySize);
   virtual ~ArrayMaxHeap();

   // HeapInterface Public Methods:
   bool isEmpty() const;
   int getNumberOfNodes() const;
   int getHeight() const;

   ItemType peekTop() const throw(PrecondViolatedExcept);
   bool add(const ItemType& newData);
   bool remove();
   void clear();
}; // end ArrayMaxHeap

#include "ArrayMaxHeap.cpp"
#endif
```

```
// ==========
// Listing 17-4.
//  Created by Frank M. Carrano and Timothy M. Henry.
//  Copyright (c) 2017 Pearson Education, Hoboken, New Jersey.

/** Heap-based implementation of the ADT priority queue.
 @file HeapPriorityQueue.cpp */

#include "HeapPriorityQueue.h"

template<class ItemType>
HeapPriorityQueue<ItemType>::HeapPriorityQueue()
{
  ArrayMaxHeap<ItemType>();
}  // end constructor

template<class ItemType>
bool HeapPriorityQueue<ItemType>::isEmpty() const
{
  return ArrayMaxHeap<ItemType>::isEmpty();
}  // end isEmpty

template<class ItemType>
bool HeapPriorityQueue<ItemType>::enqueue(const ItemType&
newEntry)
{
  return ArrayMaxHeap<ItemType>::add(newEntry);
}  // end enqueue

template<class ItemType>
bool HeapPriorityQueue<ItemType>::dequeue()
{
  return ArrayMaxHeap<ItemType>::remove();
}  // end dequeue

template<class ItemType>
ItemType HeapPriorityQueue<ItemType>::peekFront() const
throw(PrecondViolatedExcept)
{
  try
  {
    return ArrayMaxHeap<ItemType>::peekTop();
  }
  catch (PrecondViolatedExcept e)
  {
    throw PrecondViolatedExcept("Attempted peek into an empty
priority queue.");
  }  // end try/catch
}  // end peekFront
```