```
Test 1  - heapInsert
Print heap array: 70  37  52  -55  0  44

Test 2: removeMax
max = 70  Print heap array: 52  37  44  -55  0
max = 52  Print heap array: 44  37  0  -55
max = 44  Print heap array: 37  -55  0
max = 37  Print heap array: 0  -55
max = 0  Print heap array: -55
max = -55  Print heap array: Heap array is empty

Test 3:  max() and size
max = 60   heapsize = 3
Print heap array: 60  5  55

Test 4 :  modifyHeap
Print heap array: 30  20  8  15  10  -10
Print heap array: 30  15  8  5  10  -10
Print heap array: 32  15  30  5  10  8

Test 5: HeapException: Error: Heap is full, cannot insert 3.
Print heap array: 6  5  4  2

Test 6:  HeapException: Heap is empty, cannot remove max.

Test 7:
Print heap array: -4  -40
-4   -40     Heap size = 0
HeapException: Heap is empty, cannot retrieve max.

Test 8:  HeapException: Node index 3 currently does not exist in this heap, thus
cannot be modified.

Testing Complete
```

```java
/* NAME: CHRIS TUFENKJIAN
 * DUE DATE: 09/30/2014
 * COMP282 - PROJECT 2
 */

public class MaxIntHeap
{
    int [] heap;
    private int heap_size;

    public MaxIntHeap()
    {
        heap = new int [20];
    }

    public MaxIntHeap(int m)
    {
        if (m <= 0)
        {
            heap = new int [20];
        }
        else
        {
            heap = new int [m];
        }
    }

    public boolean isEmpty()
    {
        if (heap_size == 0)
            return true;
        else
            return false;
    }

    public int size()
    {
        return heap_size;
    }

    public void heapInsert(int v) throws HeapException
    {
        if( heap_size == heap.length )
        {
            throw new HeapException("Error: Heap is full, cannot insert " + v + ".");
        }
        else
        {
            heap[heap_size] = v;
            trickleUp(heap_size);
        }
        heap_size++;
    }

    private void swap_parent (int i)
    {
//      swap algorithm ex: x = 20, y = 5
//      x = x + y;         x = 20 + 5 = 25
```

```
59 //      y = x - y;          y = 25 - 5 = 20
60 //      x = x - y;          x = 25 - 20 = 5
61         heap[i]              = heap[i] + heap[get_Parent(i)];
62         heap[get_Parent(i)] = heap[i] - heap[get_Parent(i)];
63         heap[i]              = heap[i] - heap[get_Parent(i)];
64     }
65
66     private void swap (int i, int j)
67     {
68         heap[i] = heap[i] + heap[j];
69         heap[j] = heap[i] - heap[j];
70         heap[i] = heap[i] - heap[j];
71     }
72
73     public int removeMax() throws HeapException
74     {
75         if( size() == 0 )
76         {
77             throw new HeapException("Heap is empty, cannot remove max.");
78         }
79         else
80         {
81             int removedValue = heap[0];
82
83             heap[0] = heap[heap_size-1];
84             heap_size--;
85             trickleDown(0);
86
87             return removedValue;
88         }
89     }
90
91     public int max() throws HeapException
92     {
93         if (heap_size == 0)
94         {
95             throw new HeapException("Heap is empty, cannot retrieve max.");
96         }
97         else
98         return heap[0];
99     }
100
101    public void modifyHeap (int k, int newvalue) throws HeapException
102    {
103        if (k <0 || k >= heap.length )
104        {
105            throw new HeapException("Index value " + k + " is beyond "
106                    + "heap limit, thus cannot exist nor be modified.");
107        }
108        else if (k <0 || k >= (size() ){
109
110            throw new HeapException("Node index " + k + " currently does "
111                    + "not exist in this heap, thus cannot be modified.");
112        }
113        else
114        {
115            heap[k] = newvalue;
116            trickleUp(k);
```

```
117             trickleDown(k);
118         }
119     }
120 
121     private void trickleUp(int k)
122     {
123         if (k == 0){}
124         else if( heap[k] > heap[get_Parent(k)] )
125         {
126             swap_parent(k);
127             k = get_Parent(k);
128             trickleUp(k);
129         }
130     }
131 
132     private void trickleDown(int k)
133     {
134         if ( get_rightChild(k) < heap_size && get_leftChild(k) < heap_size
135                 && heap[get_rightChild(k)] > heap[k]
136                         &&  heap[get_rightChild(k)] > heap[get_leftChild(k)])
137         {
138                 swap( k, get_rightChild(k) );
139                 trickleDown(get_rightChild(k));
140         }
141         if (get_leftChild(k) < heap_size
142                 && heap[get_leftChild(k)] > heap[k])
143         {
144                 swap( k , get_leftChild(k));
145                 trickleDown(get_leftChild(k));
146         }
147     }
148 
149     private int get_leftChild(int parent_Node)
150     {
151         return 2*(parent_Node) + 1;
152     }
153     private int get_rightChild(int parent_Node)
154     {
155         return 2*(parent_Node) + 2;
156     }
157     private int get_Parent(int i)
158     {
159         return (i-1)/2;
160     }
161 
162     public int[] getHeapArray()
163     {
164         int [] truncated_heap = new int[size()];
165         for (int i = 0; i < size(); i++)
166             truncated_heap[i] = heap[i];
167         return truncated_heap;
168     }
169 }
170 
```

HeapException.java

```java
@SuppressWarnings("serial")
public class HeapException extends Exception
{
    public HeapException (String message)
    {
        super(message);
    }
}

```

```java
public class HeapTest
{
    public static void main (String [] args) throws HeapException
    {
        System.out.println("Check constructors: ");

        System.out.print("\tCreate with parameter -6, size is: ");
        MaxIntHeap heap_neg6 = new MaxIntHeap (-6);
        System.out.println(heap_neg6.heap.length);

        System.out.print("\tCreate with parameter 11, size is: ");
        MaxIntHeap heap_11 = new MaxIntHeap (11);
        System.out.println(heap_11.heap.length);

        System.out.print("\tCreate with parameter 0, size is: ");
        MaxIntHeap heap_0 = new MaxIntHeap (0);
        System.out.println(heap_0.heap.length);

        System.out.print("\tCreate with no parameter, size is: ");
        MaxIntHeap heap_default = new MaxIntHeap();
        System.out.println(heap_default.heap.length);

        System.out.println("- - - - - - - - - - - - - - - - - - - - - - - - - ");
        System.out.println("----------///// ALL FOLLOWING TESTS HEREON ARE "
                + "PERFORMED ON HEAP SIZE 11 ////----------");
        System.out.println("Heap contains: " + heap_11.size() + " elements.");
        System.out.println("Test heap empty exceptions: .max() and .removeMax()" );
        try
        {
            System.out.println("Printing maximum value: " + heap_11.max());
        }
        catch(HeapException e)
        {
            System.out.println(e);
        }
        try
        {
            heap_11.removeMax();
        }
        catch(HeapException e)
        {
            System.out.println(e);
        }
        System.out.println("- - - - - - - - - - - - - - - - - - - - - - - - - ");
        System.out.println("Insert values: 2500, 500, 0, 0, 15000 and 200.");
        heap_11.heapInsert(2500);
        heap_11.heapInsert(500);
        heap_11.heapInsert(0);
        heap_11.heapInsert(0);
        System.out.println("Insert values  15000 and 200 using try/catch."
                + "(array not full, nothing is caught.)");
        try
        {
            heap_11.heapInsert(15000);
        }
        catch(HeapException e)
        {
            System.out.println(e);
```

```
59          }
60          try
61          {
62              heap_11.heapInsert(200);
63          }
64          catch(HeapException e)
65          {
66              System.out.println(e);
67          }
68          System.out.println("Printing maximum value: " + heap_11.max());
69          System.out.println("- - - - - - - - - - - - - - - - - - - - - - - - - - - - - ");
70          System.out.print("Printing Truncated COPY of heap BEFORE modifying:");
71          for (int i = 0; i < heap_11.getHeapArray().length ; i++ )
72          {
73              System.out.print( "\n\tNode " + i + ": "+  heap_11.getHeapArray()[i] );
74          }
75          System.out.println("\nHeap contains: " + heap_11.size() + " elements.");
76          System.out.println("- - - - - - - - - - - - - - - - - - - - - - - - - - - - - ");
77          System.out.println("Modify node 1, with value 99 using "
78                  + "try/catch (try/catch should not throw exception).");
79          try
80          {
81              heap_11.modifyHeap(1,99);
82          }
83          catch(HeapException e)
84          {
85              System.out.println(e);
86          }
87          System.out.println("Modify node -14, with value 888 using try/catch.");
88          try
89          {
90              heap_11.modifyHeap(-14,888);
91          }
92          catch(HeapException e)
93          {
94              System.out.println(e);
95          }
96          System.out.println("Modify node 12, with value 55.");
97          try
98          {
99              heap_11.modifyHeap(12,55);
100         }
101         catch(HeapException e)
102         {
103             System.out.println(e);
104         }
105         System.out.println("Modify node 7, with value 60.");
106         try
107         {
108             heap_11.modifyHeap(7,60);
109         }
110         catch(HeapException e)
111         {
112             System.out.println(e);
113         }
114         System.out.println("- - - - - - - - - - - - - - - - - - - - - - - - - - - - - ");
115         System.out.print("Printing Truncated heap AFTER MODIFYING IT:");
116         for (int i = 0; i < heap_11.getHeapArray().length ; i++ )
```

```
117         {
118             System.out.print( "\n\tNode " + i + ": "+  heap_11.getHeapArray()[i] );
119         }
120         System.out.println("\nHeap contains: " + heap_11.size() + " elements.");
121         System.out.println("- - - - - - - - - - - - - - - - - - - - - - - - - - - - - ");
122         System.out.println("Insert values: 1200, 4500, 800, 90000, 90 and 30.");
123         heap_11.heapInsert(1200);
124         heap_11.heapInsert(4500);
125         heap_11.heapInsert(800);
126         heap_11.heapInsert(90000);
127         heap_11.heapInsert(90);
128         System.out.println("Insert value 30(array is full, error should display).");
129         try
130         {
131             heap_11.heapInsert(30);
132         }
133         catch(HeapException e)
134         {
135             System.out.println(e);
136         }
137         System.out.println("- - - - - - - - - - - - - - - - - - - - - - - - - - - - - ");
138         System.out.print("Printing Truncated heap:");
139         for (int i = 0; i < heap_11.getHeapArray().length ; i++ )
140         {
141             System.out.print( "\n\tNode " + i + ": "+  heap_11.getHeapArray()[i] );
142         }
143         System.out.println("\nHeap contains: " + heap_11.size() + " elements.");
144         System.out.println("- - - - - - - - - - - - - - - - - - - - - - - - - - - - - ");
145         System.out.println("Printing (not removing) new max value: " +
    heap_11.max());
146         System.out.println("Using for loop, removing all elements of heap and
    print.");
147         System.out.println("Heap contains: " + heap_11.size() + " elements.");
148 
149         for (int i = 0; i < heap_11.heap.length; i++)
150         {
151             System.out.println( "Removing Max: " + heap_11.removeMax() );
152         }
153         System.out.println("Heap contains: " + heap_11.size() + " elements.");
154     }
155 }
```

```
Check constructors:
      Create with parameter -6, size is: 20
      Create with parameter 11, size is: 11
      Create with parameter 0, size is: 20
      Create with no parameter, size is: 20
- - - - - - - - - - - - - - - - - - - - - - - - - - - -
----------///// ALL FOLLOWING TESTS HEREON ARE PERFORMED ON HEAP SIZE 11
////----------
Heap contains: 0 elements.
Test heap empty exceptions: .max() and .removeMax()
HeapException: Heap is empty, cannot retrieve max.
HeapException: Heap is empty, cannot remove max.
- - - - - - - - - - - - - - - - - - - - - - - - - - - -
Insert values: 2500, 500, 0, 0, 15000 and 200.
Insert values  15000 and 200 using try/catch.(array not full, nothing is
caught.)
Printing maximum value: 15000
- - - - - - - - - - - - - - - - - - - - - - - - - - - -
Printing Truncated COPY of heap BEFORE modifying:
      Node 0: 15000
      Node 1: 2500
      Node 2: 200
      Node 3: 0
      Node 4: 500
      Node 5: 0
Heap contains: 6 elements.
- - - - - - - - - - - - - - - - - - - - - - - - - - - -
Modify node 1, with value 99 using try/catch (try/catch should not throw
exception).
Modify node -14, with value 888 using try/catch.
HeapException: Index value -14 is beyond heap limit, thus cannot exist nor be
modified.
Modify node 12, with value 55.
HeapException: Index value 12 is beyond heap limit, thus cannot exist nor be
modified.
Modify node 7, with value 60.
HeapException: Node index 7 currently does not exist in this heap, thus
cannot be modified.
- - - - - - - - - - - - - - - - - - - - - - - - - - - -
Printing Truncated heap AFTER MODIFYING IT:
      Node 0: 15000
      Node 1: 500
      Node 2: 200
      Node 3: 0
      Node 4: 99
      Node 5: 0
Heap contains: 6 elements.
- - - - - - - - - - - - - - - - - - - - - - - - - - - -
Insert values: 1200, 4500, 800, 90000, 90 and 30.
Insert value 30(array is full, error should display).
HeapException: Error: Heap is full, cannot insert 30.
- - - - - - - - - - - - - - - - - - - - - - - - - - - -
Printing Truncated heap:
      Node 0: 90000
      Node 1: 15000
      Node 2: 1200
      Node 3: 800
```

```
      Node 4: 4500
      Node 5: 0
      Node 6: 200
      Node 7: 0
      Node 8: 500
      Node 9: 99
      Node 10: 90
Heap contains: 11 elements.
- - - - - - - - - - - - - - - - - - - - - - -
Printing (not removing) new max value: 90000
Using for loop, removing all elements of heap and print.
Heap contains: 11 elements.
Removing Max: 90000
Removing Max: 15000
Removing Max: 4500
Removing Max: 1200
Removing Max: 800
Removing Max: 500
Removing Max: 200
Removing Max: 99
Removing Max: 90
Removing Max: 0
Removing Max: 0
Heap contains: 0 elements.
```