Solution 1

```
  }

  public class Heap {
    public List<int> heap = new List<int>();
    public Func<int, int, bool> comparisonFunc;
    public int length;

    public Heap(Func<int, int, bool> func, List<int> array) {
      this.heap = buildHeap(array);
      this.comparisonFunc = func;
      this.length = heap.Count;
    }

    public int peek() {
      return heap[0];
    }

    public int remove() {
      this.swap(0, heap.Count - 1);
      int valueToRemove = heap[heap.Count - 1];
      this.heap.RemoveAt(heap.Count - 1);
      this.length -= 1;
      this.siftDown(0, heap.Count - 1, heap);
      return valueToRemove;
    }

    public void Insert(int value) {
      this.heap.Add(value);
      this.length += 1;
      this.siftUp(heap.Count - 1, heap);
    }

    public List<int> buildHeap(List<int> array) {
      int firstParentIdx = (array.Count - 2) / 2;
      for (int currentIdx = firstParentIdx; currentIdx >= 0; currentIdx--) {
        this.siftDown(currentIdx, array.Count - 1, array);
      }
      return array;
    }

    public void siftDown(int currentIdx, int endIdx, List<int> heap) {
      int childOneIdx = currentIdx * 2 + 1;
      while (childOneIdx <= endIdx) {
        int childTwoIdx = currentIdx * 2 + 2 <=
          endIdx ? currentIdx * 2 + 2 : -1;
        int idxToSwap;
        if (childTwoIdx != -1) {
          if (comparisonFunc(heap[childTwoIdx], heap[childOneIdx])) {
            idxToSwap = childTwoIdx;
          } else {
            idxToSwap = childOneIdx;
          }
        } else {
          idxToSwap = childOneIdx;
        }
        if (comparisonFunc(heap[idxToSwap], heap[currentIdx])) {
          swap(currentIdx, idxToSwap);
          currentIdx = idxToSwap;
          childOneIdx = currentIdx * 2 + 1;
        } else {
          return;
        }
      }
    }

    public void siftUp(int currentIdx, List<int> heap) {
      int parentIdx = (currentIdx - 1) / 2;
      while (currentIdx > 0) {
        if (comparisonFunc(heap[currentIdx], heap[parentIdx])) {
          swap(currentIdx, parentIdx);
          currentIdx = parentIdx;
          parentIdx = (currentIdx - 1) / 2;
        } else {
          return;
        }
      }
    }

    public void swap(int i, int j) {
      int temp = this.heap[j];
      this.heap[j] =  this.heap[i];
      this.heap[i] =  temp;
    }
  }
}

```