Heap Algorithms

We assume A[] is an array that is indexed from 1 to n. (We ignore index 0).

Percolate Down (Sink): If an item has at least one child that is smaller, swap that item with the larger of its two children. Continue at the updated child. Stop when there is no swap.

```
sink(A[], int k)
  while (2*k <= n)    // n is size of heap
    j = 2*k
    if (j < n and A[j] < A[j+1]))
      j++
    if A[k] >= A[j]
      break
    swap A[k] and A[j]
  k = j
```

Percolate Up (Swim): If an item is larger than its parent, swap. Continue at parent. Stop when there is no swap.

```
swim(A[], int k)
  while (k > 1 and A[k/2] < A[k])
    swap A[k/2] and A[k]
    k = k/2
```

Heapsort

- Start with array A[] of unsorted elements in positions 1 through n (ignore position 0).
- Create a heap in place with those elements:
  - Interpret A[1..n] as a heap structure with many violations of the heap property.
  - Repeatedly call sink() on each element, starting from position n/2 and progressing backwards to position 1.
  - This creates a heap.
- Repeatedly swap A[1] (max element in heap) with A[n] last element in heap. This moves the largest element in the heap to its correct spot at the end of the array.
- Call sink() to repair the heap from the root node.

```
heapsort(A[])
  n = size of array A              # this first step is sometimes
  for (int k = n/2; k >= 1; k--)   # known as the “heapify”
    sink(A, k)                     # algorithm

  while (n > 1)
    swap A[1] and A[n]
    n--  // decrease size of heap
    sink(A, 1)   // Sink takes into account new heap size here.
```