Partners: Robert Johnson, Chris Johnson

\*Indexing starts at 1 for this problem so for an array A = [1,2,3,4], A[1] = 1, A[4] = 4, and so on\*

\* n // 2 stands for floor(n/2) so if n is 5, n // 2 = 2\*

**MaxHeapify(A, n, i)**

**Note: This works as long as the sub-tree’s are max heaps, to make sure that’s always true make sure that you start one level above the leaf nodes and work up**

**Input:**array A of n distinct integers and i the root node to heapify

**Post Condition:** the node i and its children (if it has any) will satisfy max heap conditions

root = i

left = i \* 2

right = i \* 2 + 1

//implementation so you can ignore values during sort

if left <= n and A[left] > A[root]

  root = left

if right <= n and A[right] > A[root]

  root = right

if root != i

  swap(A[i], A[root])

  maxHeapify(A, n, root)

**HeapSort(A)**

**Input:**Array A of n distinct integers

**Post Condition:** Array A will be sorted from lowest to highest

n = len(A)

i = n // 2

while i > 0

  maxHeapify(A, n, i)

  i = i - 1

while n > 1

  swap(A[1], A[n])

  maxHeapify(A, n-1, 1)

  n = n – 1

2. Building the heap will at most take n / 2 \* log(n) operations.

Sorting the heap will at most take n log n operations.

Since they are executed sequentially the run time is O(n log n).