# **Heap Sort**

* Let’s say we have just built a minheap and wanted to get the elements in the heap in sorted order. This algorithm is called **heap sort**.
* As its name implies, the heap sort algorithm uses a **heap** to sort an **unsorted array** of items.
* How do we implement heap sort?
* We can easily sort a heap by **repeatedly deleting the minimum value**.
* Suppose that we have

1. A min-heap

and

1. An empty array whose size is the number of items in the heap.

* If the min-heap has N items, then we can build N calls to peek() and poll() to get and remove the largest item in the heap.
* We place each next-smallest item from the priority queue and place them sequentially into increasing positions in a new array. The result is an array sorted into ascending order.
* While this approach would work, it uses more memory and time than is necessary.
* We can actually accomplish this in-place.

**In Place**

* Let’s start over with an array of unsorted values.
* Our first step would be to transform the array into a heap using heapify(collection).
* Heap sort partitions an array into two regions

A picture containing table

Description automatically generated

public void inPlaceHeapSort(collection) {

E[] heap = buildHeap(collection)  
for (e in heap)

output[n – i - 1] = removeMin(heap)

}

* This yields an algorithm that is O(n log n) in the worst case (unlike quicksort) and that doesn’t require extra memory (unlike mergesort).
* Complication: final array is reversed!
  + Run reverse afterwards (O(n))
  + Use a max heap
  + Reverse compare function to emulate max heap
* Worst case runtime? Best case runtime? Average runtime?
  + O(nlogn)

A picture containing graphical user interface

Description automatically generated