CS 260
Heap Sort

Heap Sort

Heapsort is the final sort covered in this module. It is also not a stable sort since items are moved around during the heapification stage. Heapsort requires no additional memory, being within the original array. The sort itself is not recursive, but it does use recursion in the bubble-up and trickle-down methods.

It is very predictable since it performs similarly in best, worst, and average cases. Its performance is also order of N log N.

Unlike mergesort and quicksort, heapsort does not recursively split the array, instead it turns the array into a heap and then removes each value from the heap in turn and places it at the end of the array. Note that this algorithm uses the bubbleUp and trickleDown algorithms defined earlier as part of implementing a heap.

To heapify an array, it is possible to either start at the leaves and trickleDown repeatedly to form a heap or start at the root and bubbleUp on each node in return. For efficiency, it is better to skip the leaf nodes and start the trickleDown at the first node with children (length/2).

Example

Here the starting row is the array at the beginning. The heap row shows it after it has been heapified. The final row shows it after the largest value was placed in the final spot.

5	3	9	7	2	8	4	6	1	Starting
9	7	8	6	2	5	4	3	1	Heap
1	2	3	4	5	6	7	8	9	Done

Algorithm

Here are the steps necessary to sort an array using heapsort.

- Heapify the array
    1. For each node starting at length/2 (nodes with children)
    2. Use trickledown to order that node
    3. Repeat with each node at a smaller index until done
- Sort the array
    1. Swap the root with the last element in the array
    2. Decrement the size of the heap
    3. Use trickledown to place the new root properly
    4. Continue until all values are sorted