CS3000: Algorithms & Data — Summer ’23 — Laney Strange

Heapsort

The Heapsort algorithm has a similar idea to Selection Sort – given an array, repeatedly find the smallest element and put it where it belongs. It tries to beat Selection Sort’s $\Theta(n^2)$ run-time by, rather than trying a different algorithmic technique, changes up the data structure.

The input to this procedure is an array, A, which has already been arranged as a Min-Heap. In addition to its usual attribute *A.length*, we assign an attribute *A.heapsize* to tell us where the heap ends. A Min-Heap maintains two properties: (1) it is a complete binary tree (the levels get filled in left-to-right), and (2) every node is smaller than all nodes below it. We assume the array comes in as a Min-Heap, just like we assumed we were given a Binary Search Tree when we developed our Binary Search algorithm back in week one. Can we convert a plain unsorted array to a Min-Heap? Absolutely! It’s jut not part of this particular algorithm.

Heapsort on a Min-Heap orders the values from the array in reverse-sorted order. It runs in-place, by assuming some of A is the sorted result, and some is the Min-Heap we haven’t gotten to yet. It relies on a procedure HEAPIFY, that takes in the array A and the position of the current root i. Heapify’s job is to assume whatever’s in the root might not be right, but the whole rest of the heap is. It ”bubbles down” the root node to its correct position within the heap.

```
HEAPSORT($A$)
1  $A.heapsize = A.length$
2  for $i = A.length$ downto 2
3      swap $A[1], A[i]$
4      $A.heapsize = A.heapsize - 1$
5      HEAPIFY($A, 1$)
```

We typeset the Heapsort procedure above with the following LaTeX:

```
\begin{codebox}
\Procname{$\proc{Heapsort}(A)$}
\li $\id{A.heapsize} = \id{A.length}$
\li \For $i \gets \id{A.length} \Downto 1$
\Do
\li swap $A[1], A[i]$
\li $\id{A.heapsize} = \id{A.heapsize} - 1$
\li $\proc{Heapify}(A, 1)$
\End
\end{codebox}
```