Lab 4 | Heap Sort | Sept. 20 2019

Resources for visualizing Heaps and Heap Sort:

<http://btv.melezinek.cz/binary-heap.html>

<https://www.cs.usfca.edu/~galles/visualization/HeapSort.html>

1. Important Notes:
   1. Range for build max array is **reversed** length(array)//2 to **0**
   2. Range for Heap sort is **reversed** length(array)-1 to **1**
   3. Make HeapSize a variable you pass to Max-Heapify
2. Get your Build Max Heap / Max-Heapify function to work first.

Example: [6,4, 8, 10, 2] => [10, 6, 8, 4, 2]

* 1. Left & Right are indexes:
     1. Left always equals 2\*i + 1
     2. Right always equals 2\*i +2
  2. HeapSize represents the cutoff index for the unsorted part of array
     1. From 0 to Heapsize, array is unsorted (should be made into max heap)
     2. From Heapsize+1 to end, array is sorted
     3. When Heapsize = length, none of array is sorted yet (first max heap)

Example of array mid sort:

[10, 6, 8, 4, 2, 12, 14, 20]

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 6 | 8 | 4 | 2 | 12 | 14 | 20 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Sorted section of array starts on 5, so:

Heapsize = 4.

1. Order of Heap Sort:
   1. Make entire array max heap
      1. Moves largest element to index 0
      2. Don’t forget loop is reversed! Starts at largest *parent* index, goes to 0.
      3. Length//2 means it is only running max-heap on parents (for efficiency)
   2. For all elements in the array (another reversed loop: starting from largest index and ending at 1, **not zero):**
      1. Swap index 0 for last element that is *unsorted* (this is index i)
         1. That number is now sorted
      2. Decrease HeapSize by one
         1. Part of array that needs to be Heaped is one less
         2. Need to move the next largest index to 0 (make another heap!)
      3. Create New Heap with unsorted section
         1. Max Heap starting at root
2. In English:

**HeapSort:**

First, build a max heap (gets index 0 as biggest element)

#Then, to actually sort the list:

For all elements (starting with largest index | excluding index 0):

Exchange index 0 [biggest element] to the end [sorted part of array]

Decrease HeapSize (i.e. decrease part of array to be heaped)

Re-Heap it w/ MaxHeapify(to move next biggest element to index 0)

**BuildMaxHeap**(array):

#You’re max heaping the whole array, so:

Heapsize = Length of array - 1

For all parents in the array (starting w/ largest index |

i.e. elements w/ index <= len(array)//2):

Max\_Heapify(array, i, Heapsize)

**Max-Heapify**(array, parent\_index, sorted\_cutoff\_index):

Left = 2\*parent\_index +1

Right = 2\*parent\_index +2

If Left Child index is less or equal to sorted\_cutoff\_index AND

If the Left child value is greater than its parent:

Left is the largest

Else:

Parent is the largest value

If Right Child index is less or equal to sorted\_cutoff\_index AND

If Right Child value is greater than current Largest:

Right is largest value

If Parent isn’t the Largest Value:

Switch the parent for the largest value

Max\_Heapify new Parent (make sure it is not larger than its parent)