Solutions -- Topic 9, Class 02/17: Heaps

Copyright (c) 2016 Dan Suthers. All rights reserved. These solution notes may only be used by students in ICS 311 Spring 2016 at the University of Hawaii.

1. (2 points) Heap-Delete(A, i)

Procedure **Heap-Delete(*A*, *i*)** deletes the node at index *i* in heap *A* (represented as an array). **Give an implementation of Heap-Delete that runs in O(lg *n*) time** for a heap of size *n* = A.heapSize. You may use instance variable A.heapSize and any of the other procedures already defined in the text. You do not need to include error checking. You do not need to return anything. *Hint: What other heap procedure is similar?*

2 points. Take off 0.5 for minor code error, and give 1.0 if there was some right idea but a logic error.

```
Heap-Delete(A,i)
A[i] = A[A.heapSize]
A.heapSize = A.heapSize - 1
Max-Heapify(A, i)
```

2. (3 points) Heapsort on Sorted Data

Previously we noted that Insertion-Sort runs faster on already sorted data, but Merge-Sort does not. What about Heapsort? How is it affected by sorted data? **Give your reasoning: justification is more important than getting the answer right**. <u>Refer to line numbers in code</u> (appended) when discussing your analyses. *Hints: Consider Build-Max-Heap and the for loop separately. You may want to work examples, but don't get bogged down in details: return to asymptotic reasoning as soon as you see what is going on.*

For each: Give 0.5 points for saying O(n lg n) and 1.0 points for argument (reduce this if the argument is lacking).

(a) What is the **asymptotic running time of Heapsort** using a Max-Heap on an array *A* of *n* elements that is <u>already sorted in ***increasing*** order</u>? Justify your claim.

Solution: O(n lg n). Build-Max-Heap is O(n) in general, and indeed it has to do some work to convert the sorted list into a max heap (which generally has the larger items earlier in the array). But the loop that follows has O(n) passes, and each pass has a call to Max-Heapify at cost O(lg n), so the loop dominates with O(n lg n). (If you work out detailed examples, you will see that once Build-Max-Heap is done, the array looks similar to a reverse-sorted array, since larger elements must be higher up in the tree = more towards the left.)

(b) What is the **asymptotic running time of Heapsort** using a Max-Heap on an array *A* of *n* elements that is <u>already sorted in ***decreasing*** order</u>? Justify your claim.

Solution: Also O(n lg n). Build-Max-Heap has to do less work in this case, as a reverse sorted list is already a max heap, so it does not have to actually swap anything (each call to Max-Heapify is constant time for the two assignments and conditional test). However, this is a constant reduction: Build-Max-Heap is still O(n) on reverse sorted data as it still runs the loop through half the items. Furthermore, the time savings does not matter: the loop that follows is O(n lg n), as it includes an O(lg n) Max-Heapify on each of the n items processed. (Another way to look at it: the sorting does not help because Max-Heapify is called after putting one of the leaves in the root position; the leaf keys are small; and it has to propagate down the tree.)

Extra Credit

3. Ternary Heaps (2 points)

You have just studied binary max-heaps. Suppose we want to represent ternary max-heaps (each node has three children with smaller keys) in an array, using 1-based indexing (the root is at array index 1). Given the index i of a heap element, how do you compute the index of:

0.5 points each. There may be algebraic variations of these!

(a) The left child?

3i - 1

(b) The middle child?

3i

(c) The right child?

3i+1

(d) The parent of *any* node?

floor((i+1)/3)

Note: Homework will generalize to d-ary case.