EC 504 – Fall 2019 – Homework 2

Due Tuesday, Oct 8, 2019 in the beginning of class. Coding problems submitted in the directory `/projectnb/alg504/yourname/HW2` on your SCC account by Tuesday, Oct. 8, 11:59PM.

Reading Assignment: CLRS Chapters 6, 7, 9, 10, 12 and Appendix B .5

1. (20 pts) Determine whether the following statements are true or false, and explain briefly why.

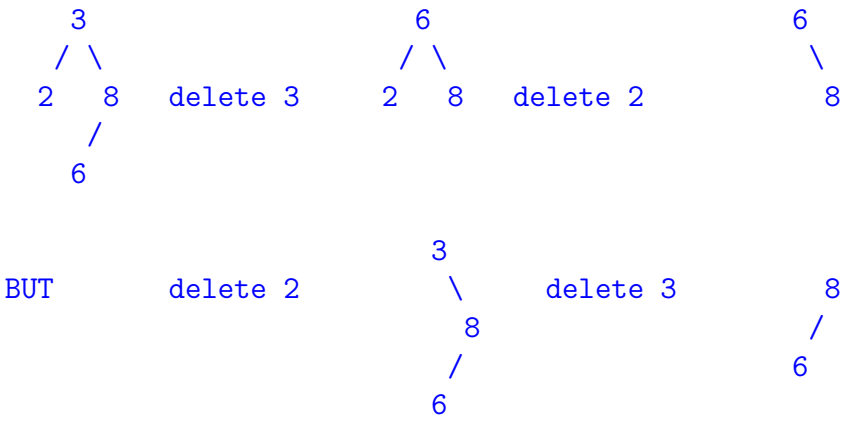
   (a) If doubling the size ($N \to 2N$) causes the execute time $T(N)$ of an algorithm to increase by a factor of 4, then $T(N) \in O(4N)$.

   Solution: False: If $T(N) \sim \text{const}N$ then when $N \to 2N$ then $TN) \to T(2N) \sim$)const$2N$ – a fatctor of 2. Need to squared it $T(N) \in O(N^2($ does it. This is grows larger so in general $T(N) \in O(4N) = O(N)$ would be false

   (b) The height of a binary tree is the maximum number of edges from the root to any leaf path. The maximum number of nodes in a binary tree of height h is $2^{h+1} - 1$.
   Solution: True. See that it works for a complete binary tree at all levels. One level is h = 0, so 1 node. 2 levels is h = 1, so it has 3 nodes. Continue by induction.

   (c) In a binary search tree with no repeated keys, deleting the node with key x, followed by deleting the node with key y, will result in the same search tree as deleting the node with key y, then deleting the node with key x.

   Solution: Flase: Deletes do not commute. When you delete with one child you replace it directly but when you delete with two children you replace it by the minimun in the right subtree. Here is very simple coutere exaample. One is enough.



   (d) Inserting numbers 1, . . . , n into a binary min-heap in that order will take $O(n)$ time.
   Solution: True: This the result is already min-heap order just when you load the array.

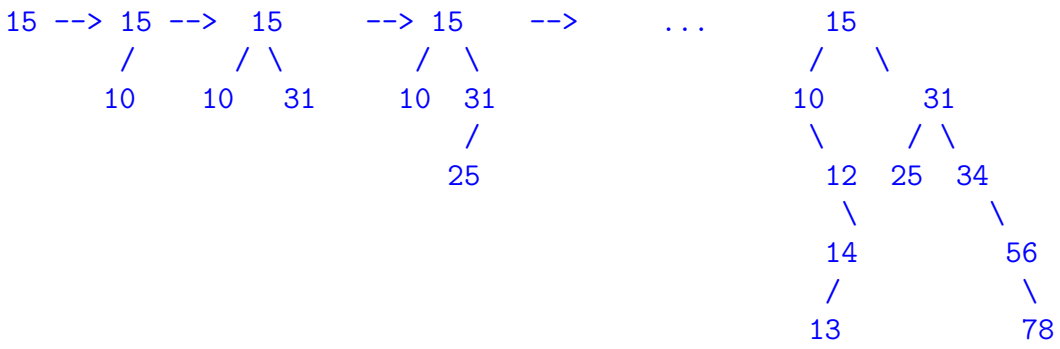
(e) The second smallest element in a binary min-heap with all elements with distinct values will always be a child of the root.

Solution: True: Suppose it is were false. Then its parent would be larger that violats the min-heap order'

2. (20 pts) This exercise is to learn binary search tree operations

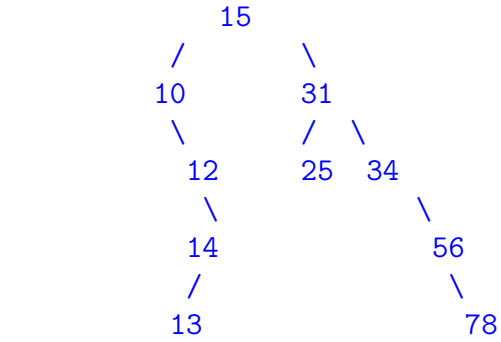
(a) Draw the sequence of binary search trees which results from inserting the following values in left-to-right order, assuming no balancing. 15, 10, 31, 25, 34, 56, 78, 12, 14, 13

Solution: Trial rule. Hard to type. So I don't do every step!



(b) Starting from the tree at the end of the previous part, draw the sequence that results from deleting the following nodes in left-to-right order: 15, 31, 12, 14.

Solution:



(c) After deleting them Draw the sequence of reinserting left-to-right in reverse order: 14, 12, 31, 15. in order into the tree and comment on the result?

Solution:

Comment: Not returned to the same because deletes and inserts are not inverse operations.

3. (20 pts) rat by reading CRLS Chapter 6 and do the written

   (a) Exercises: exercises 6.1-3, 6.1-4, 6.1-6, 6.2-1, 6.3-1 and 6.3-3 **Solution:**

   6.1 - 3:

   If there largest element in the subtee were somewhere other than the root, it has a parent that is in the subtree. So, it is larger than it's parent, so, the heap property is violated at the parent of the maximum element in the subtree

   6.1 - 4:

   The smallest element must be a a leaf node. Suppose that node x contains the smallest element and x is not a leaf. Let y denote a child node of x. By the max-heap property, the value of x is greater than or equal to the value of y. Since the elements of the heap are distinct, the inequality is strict. This contradicts the assumption that x contains the smallest element in the heap.

   6.1 - 6:

   No, the array is not a max-heap. 7 is contained in position 9 of the array, so its parent must be in position 4, which contains 6. This violates the max-heap property.

   6.2 - 1:

   27 17 3 16 13 10 1 5 7 12 4 8 9 0

   27 17 10 16 13 3 1 5 7 12 4 8 9 0

   27 17 10 16 13 9 1 5 7 12 4 8 3 0

   6.3 - 1:

   5 3 17 10 84 19 6 22 9

   5 3 17 22 84 19 6 10 9

   5 3 19 22 84 17 6 10 9

   5 84 19 22 3 17 6 10 9

   84 5 19 22 3 17 6 10 9

   84 22 19 5 3 17 6 10 9

   84 22 19 10 3 17 6 5 9

   6.3 - 3:

   All the nodes of height h partition the set of leaves into sets of size between $2^{h-1}+1$and 2h, where all but one is size 2h. Just by putting all the children of each in their own part of the partition. Recall from 6.1-2 that the heap has height $[log(n)]$, so, by looking at the one element of this height (the root), we get that there are at most $2^{[log(n)]}$ leaves. Since each of the vertices of height h partitions this into parts of size at least 2^{h-1} +

1, and all but one corresponds to a part of size 2h , we can let k denote the quantity we wish to bound, so,

$(k-1)*2^h + k*(2^{h-1}+1) <= 2^{[log(n)]} <= n/2$

so

$k <= \frac{n+2^h}{k*(2^{h+1}+2^h+1} <= \frac{n}{2^{h+1}} <= [\frac{n}{2^{h+1}}]$

This Chapter 6 give a background to the fist coding exercise to use an array for a Max Heap. Also there is of course a nice Wikipedia article to look at https://en.wikipedia.org/wiki/Heapsort.

Coding Exercises – More details in class

```
There is template program on the GitHub but you all the basic
framework can be adapted from the coding exercise in HW1.  One goal of
these coding exercises it to have  a set of C tools and Gnuplot tools
to reuse.
```

4. (20pts) Implement a Max Heap for n elements as and array `int HeapArray[n+1];` of $n+1$ setting elements by placing the integers setting `HeapArray[0] = n` and copying the elements putting the elements in sequence into `HeapArray[i],  for i = 1,2,..., n` (May choose to have an longer array with extra space and a save a value `heapSize` to tell how many are in the heap. This is a useful index in any case!

```
(1) Insert random sequence to Heap array
(2) Bottom up Heapify for Max Heap
(3) Delete any key and restore Max Heap
(4) Insert new key and restore Max Heap
(4) Sort in place and print out array
```

Implement your algorithm as a C/C++ function. Put final code with makefile `makeHeap` in your top level CCS account directory HW2.)

`/projectnb/alg504/username/HW2`

Extra credit to be pass in with the written exercises.

```
(1) Plot timing for range of sise n = 8, 16, 32,....2^20
(2)  Histogram the performance over random permutation of UnsortedList100.txt
(3) Combine these two part to define the average for
n = 8, 16, 32,....2^20 and fit T(n) = a + b n Log[n] + c n*n
```

Implement your algoristhm as a C/C++ function. Put final code with makefile `makeHeap` on your top level CCS account in directory HW2 (e.g. folder!)

Solution: See Solution code on GitHub

5. (20 pts) Suppose you are given two sorted arrays A, B of integer values, in increasing order, sizes n and m respectively, and $m + n$ is an odd value (so we can define the median value uniquely). Develop an algorithm for finding the median of the combined sorted arrays, and analyze the complexity of your algorithm. It is straight forward to develop an algorithm that is $O(min(n, m))$. Explain in words. You should attempt to construct a code that is worst case complexity is $O(min(n, m))$ to get good performance. (The written remarks should be pass in with the written exercises.)

```
(1) Submit Timings for the input files provided.
```

Implement your algorithm as a C/C++ function. Put final code with makefile `makeAB` in your top level CCS account directory HW2 (e.g. folder!)

`/projectnb/alg504/username/HW2`

Solution: See Solution code on GitHub