# EC 504 – Fall 2023 – Homework 3

**Due Friday Oct 13, 2023 on your SCC account by 11:59PM to Gradescope.**

Reading Assignment on GitHub – In CRLS add to past assignments Sorting_Data_Structure (Chapters 6, 7,8,9,10) , new material on Trees.pdf (12, 13) and TreeMath_B.pdf (Appendix B .5) and Huffman Coding 16.3

NOTE: MidTerm Exam in class on October 26, 2023 – only written questions on algorithm scaling, sorting and trees.

1. (10 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 $T(N) \in O(4N)$.

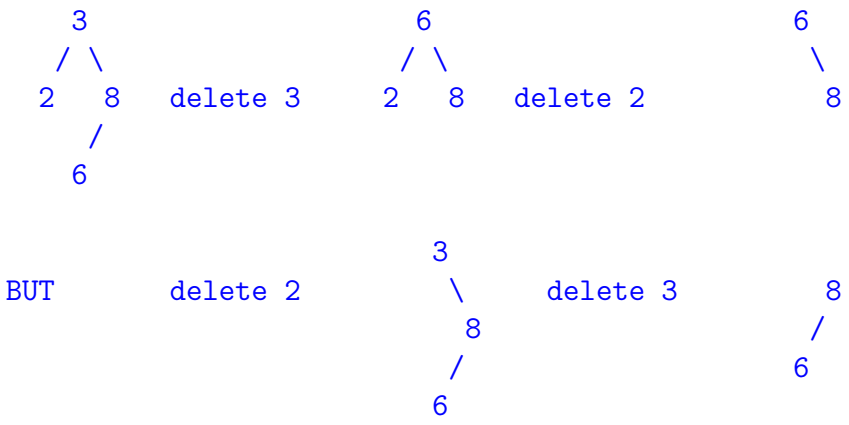
   **Solution:** False: If $T(N) \sim \text{const}N$ then when $N \to 2N$ then $TN) \to T(2N) \sim$ )const$2N$ – a factor of 2. Need to squared it $T(N) \in O(N^2($ does it. This is grows larger so in general $T(N) \in O(4N) = O(N)$ would be false

   (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:** False: 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 minimum in the right subtree. Here is very simple counter example. 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.

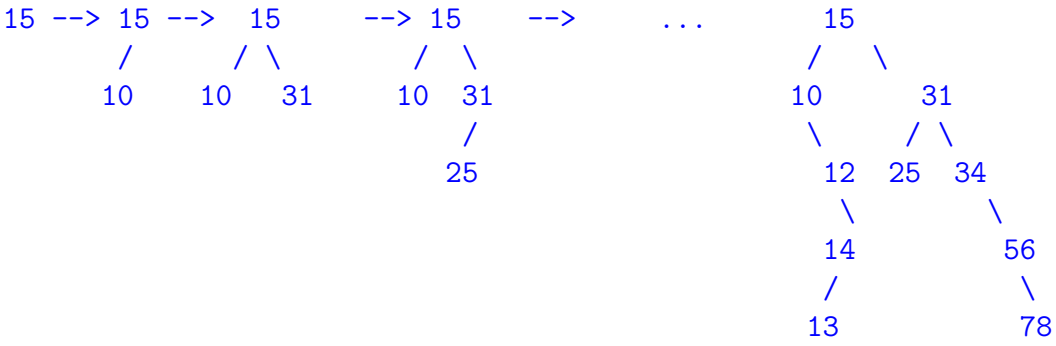
(e) The second smallest element in a binary min-heap with all elements with distinct values will always be a child of the root.

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

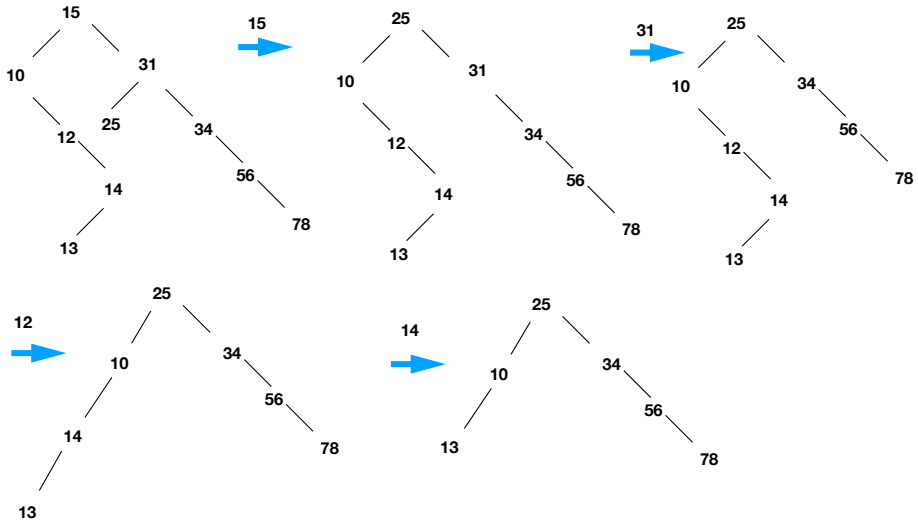
2. (15 pts) This exercise is to learn binary search tree operations

(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!



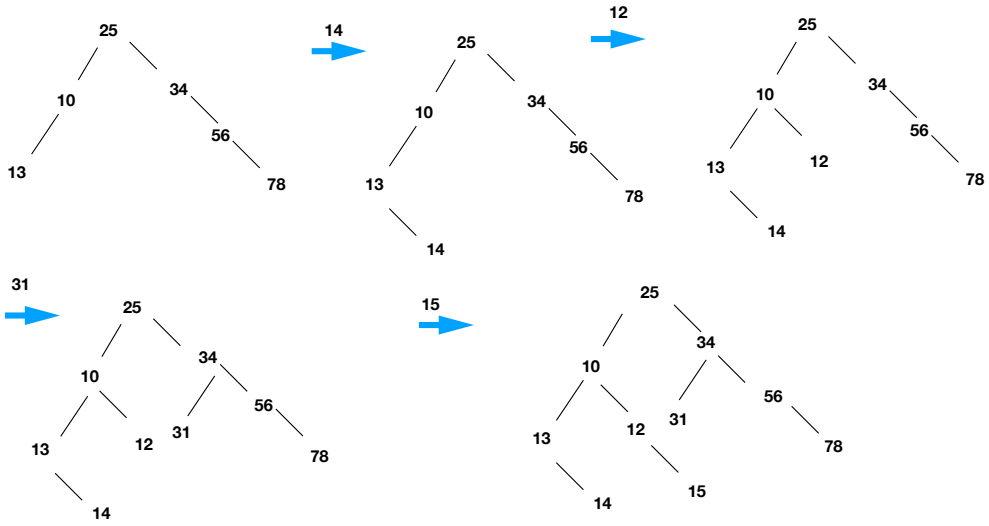

(b) Starting from the tree at the end of the previous part, draw the sequence that results from deleting the following nodes in left-to-right order: 15, 31, 12, 14.




**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. (15 pts) Reading CRLS Chapter 6 and do the written

   (a) Exercises: 6.1-3, 6.1-4, 6.1-6, and 6.3-3

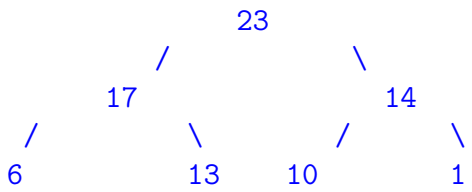
   (Note chapter 6 gives background to the coding exercise to use an array for a Max Heap.
   Also there is of course a nice Wikipedia article to look at https://en.wikipedia.org/wiki/Heapsort.)

   **Solution:**

   Exercise 6.1-3: In a max head the path from the root as you descend to any leaf is a sequence a (sorted) values deceasing. is sorted as you descend. So by the recursive definition of a binary tree each node is itself a max heap of that subtree.

   Exercise 6.1-4: The max heap (with no equal values) has strictly order sequence to smaller values as descend to the leaves (i.e. nodes with no children). Therefore the smallest element must be on leaf at the bottom.

   Exercise 6.1-6: Try it:



Exercise 6.3-3: Consider with full heap (perfect tree) has rows with $2^d$ at depth d. With the last row at depth d = D there are $n = 1 + 2 + 4 + ... + 2^D = 2^{D+1} - 1$ nodes. So now going back up the number of at height h are

$$2^{D-h} = 2^{D+1}/2^{h+1} = n/2^{h+1} + 1//2^{h+1}$$

So in this case we have ceiling of $n/2^{h+1}$. Now is you deplete the lowest row you reduce the height of some elements so this is a maximum.

(b) Exercises: 12.3-2, 12.3-3, 12.3-4, B.5-4

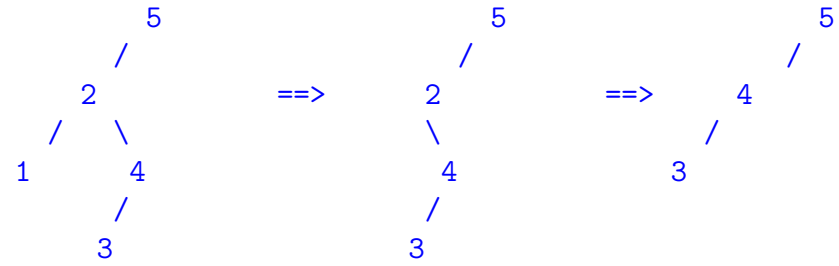
(Note that the degree of in an undirected graph (or tree) is the number links incident on the node. A leaf is a node with degree 1.)

**Solution:**

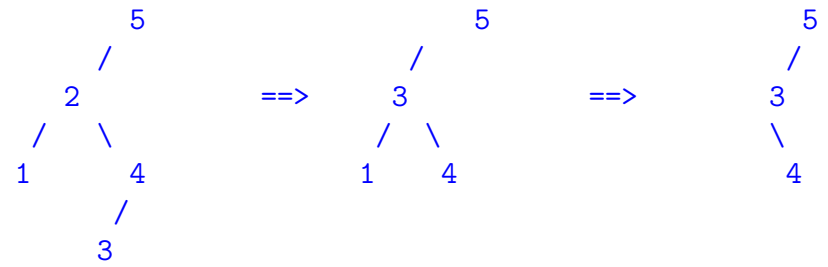
Exercises 12.3-2: When you insert an element it follows a path by comparing nodes as you descend util it is inserted. If there are n comparison it has gone to depth $n$. The search follows the same path but needs to compare the final value so this is $n + 1$.

Exercises 12.3-3: The ironware walk is walk around the outside of the tree. It has to visit every link twice. To the complexity is the number of links. Since every node has at most 3 links shared. There are at most $O(N)$ links. The minimum number of links is N so this this worst case is also $O(N)$ (Obvious you can't do better than visiting every node!) If you also assume the building the tree as part of the algorithm then the bests is $O(NlogN)$ to insert into complete tree and $O(N^2)$ totally in one line.

Exercises 12.3-4: No it is not commutative. Here is an example:; build in 5 2 1 4 3 order. First 1 then 2 (left only for 1 and left only for 2 )



consider. First 2 then 1 ( Right Left for 2 and left only for 1




Exercises B.5-4: Consider the minimum height H(N) for a tree with N nodes. A binary tree with N nodes is a recursive structure with a left sub-tree, $T_L$ and a right sub-tree $T_R$. Let $H(N)$ be the minimum height of a tree with N nodes. Base case $N = 1$ we check that $H(1) = \lfloor lg(1) \rfloor = 0$. We kwon the height of $T$ is the 1 plus the max of the height of $T_L$ and $T_R$ we have

$$H(N) = 1 + Max[H(k)] \text{for} \quad k = 0, ..., N/2 - 1$$

but now we know $H(k) = \quad \lfloor lg(k) \rfloor$ so

$$H(N) = 1 + \quad \lfloor lg(N/2) \rfloor = 1 - 1 + \quad \lfloor lg(N) \rfloor$$

4. (15 pts) Determine whether the following statements are true or false, and explain briefly why.

   (a) A heapsort algorithm for a given list first forms a max-heap with the elements in that list, then extracts the elements of the heap one by one from the top. This algorithm will sort a list of n elements in time of O(log(n)).

   **Solution:** False. Forming the heap is $\Theta(n)$ and extracting each element is $O(\log(n)))$. Altogether that gives you a runtime of $O(n \log(n)))$ Also the fastest sorting algorithm is $O(n \log(n)))$ so this should've been obvious.

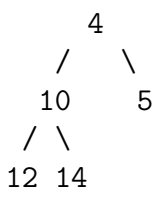
   (b) A connected undirected acyclic graph with N nodes and N-1 edges is a tree.

   **Solution:** True. Start with N=1 a single node and zero links. Each time you add a link it has one more node if it does not make a cycle.

   (c) A binary heap of n elements is a full binary tree for all possible values of n.

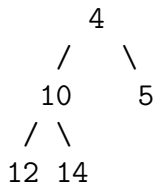
   **Solution:** False. It is a complete binary tree.

   (d) The following tree is a valid min-heap.



**Solution:** True. It is a complete binary tree satisfying the min heap property.

(e) The following tree can appear in a binomial min-heap. .




**Solution:** False. It is not a binomial tree, as the number of elements is not a power of 2.

(f) Given a array of positive integers $a[i]$, maximizing $\sum i * a[i]$ over permutations of the array requires sorting $a[i]$ in assigning order.

**Solution:** True. This is the same argument used for the Huffman coding proof. One way is to use induction. Consider n = 2. Have two possibilities. If $a[1] < a[2]$ but

$$(2-1)(a[2]-a[1]) > 0 \implies 1a[1] + 2a[2] > 1a[2] + 2a[1]$$

Now assume for n-1 and add another element. Put the max into the last position $a[n]$.

$$Max[\sum_{i=1}^{n} i * a[i]] = Max[\sum_{i=1}^{n-1} i * a[i]] + na[n]$$

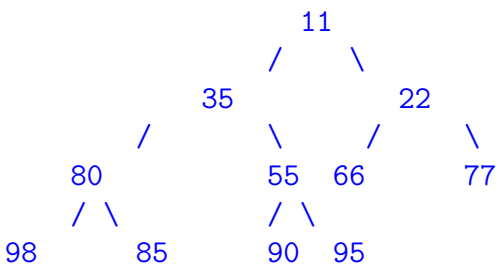
The n-element list must sorted by assumption and $na[n]$ is as large as it can be. q.e.d.

5. (15 pts) Given the following list of elements,

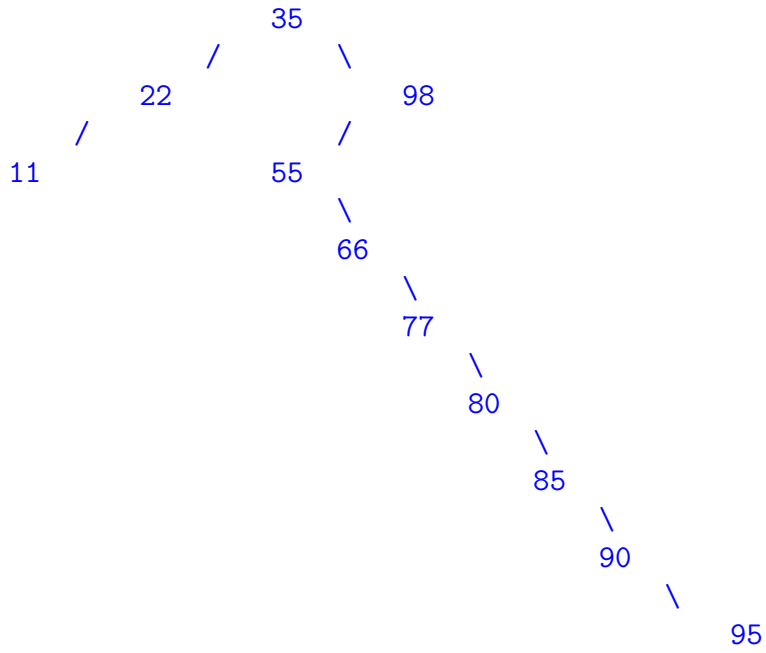
$$35, 22, 11, 98, 55, 66, 77, 80, 85, 90, 95$$

(a) Draw the sequence of binary min-heaps which results from inserting the following values in the order in which they appear into an empty heap.

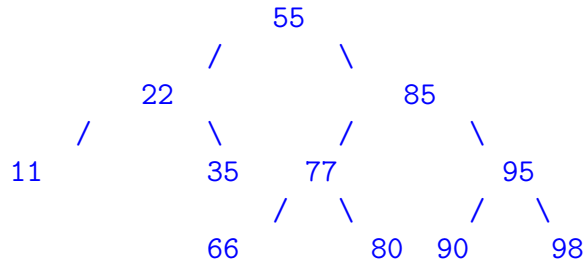
**Solution:** Final Solution Shown:




(b) Compare this answer with inserting them into the BST tree. **Solution:** Final Solution Shown:



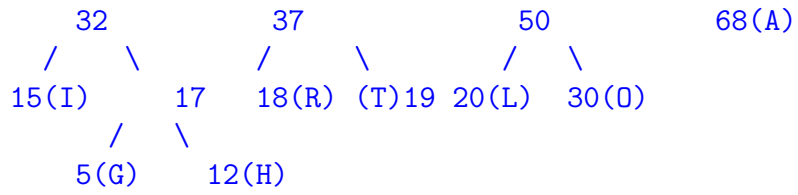

(c) Compare this answer with inserting them into the AVL tree. **Solution:** Final Solution Shown:




6. (15 pts) You are interested in compression. Given a file with characters, you want to find the binary code which satisfies the prefix property (no conflicts) and which minimizes the number of bits required. As an example, consider an alphabet with 8 symbols, with relative weights (frequency) of appearance in an average text file give below:

| alphabet: | $A$ | $L$ | $G$ | $O$ | $R$ | $I$ | $T$ | $H$ |
|---|---|---|---|---|---|---|---|---|
| weights: | 68 | 20 | 5 | 30 | 18 | 15 | 19 | 12 |

(a) Determine the Huffman code by constructing a tree with **minimum external path length**: $\sum_{i=1}^{8} w_i d_i$. (Arrange tree with smaller weights to the left.) **Solution:** Solution Shown:




(b) Identity the code for each letter and list the number of bits for each letter and compute the average number of bits per symbols in this code. Is it less than 3? (You can leave the answer as a fraction since getting the decimal value is difficult without a calculator.) **Solution:** Solution Shown:

| LETTER | WEIGHT | CODE | BITS |
|---|---|---|---|
| A | 68 | 11 | 2 |
| O | 30 | 101 | 3 |
| L | 20 | 100 | 3 |
| T | 19 | 011 | 3 |
| R | 18 | 010 | 3 |
| I | 15 | 000 | 3 |
| H | 12 | 0011 | 4 |
| G | 5 | 0010 | 4 |

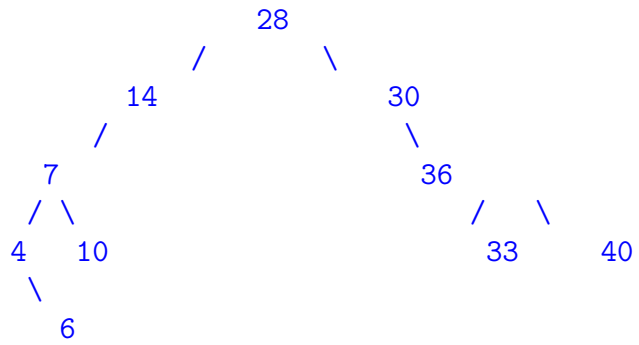
AVG BITS PER CHAR < 3

(c) Give an example of weights for these 8 symbols that would saturate 3 bits per letter. What would the Huffman tree look like? Is a Huffman code tree always a full tree? **Solution:** Solution Shown: If all letters had the same weight it would saturate the 3 bits because all 8 letters would need 3 bits to represent it, any example is a complete full binary tree.

7. (15 pts) Given the following list of $N = 10$ elements.

$$28 \quad 14 \quad 7 \quad 4 \quad 6 \quad 30 \quad 36 \quad 33 \quad 10 \quad 40$$

(a) Insert them sequentially into a BST (Binary Search Tree). Compute the total height $T_H(N)$ and the total depth $T_D(N)$, where H is the height of the root. (Note the height of a node is the longest path length to a leaf whereas its depth is its unique distance from the root.) **Solution:** Final Solution Shown:



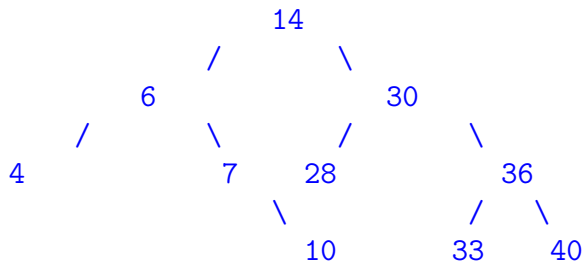
(b) Find $H$, $T_H(N)$, $T_D(N)$ and check the bounds for any N that

$$\log_2(N)N \leq T_H(N) + T_D(N) \leq (N-1)N$$

**Solution:**

```
H=4
To(N) = 22
Th(N) = 13
To(N) + Th(N) > HN
35 < 40
```

(c) Insert them sequentially into an empty AVL tree, restoring the AVL property after each insertion. Show the AVL tree which results after each insertion and name the type of rotation (RR or LL zig-zig or versus RL or LR zig-zag). **Solution:** Final Solution Shown:




(d) Has the final AVL tree decreased the total height $T_H(N)$ and the total depth $T_D(N)$? What are the new values? What is the new value of $T_H(N) + T_D(N)$? **Solution:** Yes

```
Th(N) = 9
To(N) = 19
Both values decreased with the AVL tree, new value of To(N) + Th(N) = 28
```