EC 504 – Fall 2020 – Take Home Midterm

Due Friday , Nov 6, 2020 at 11:59. Both written and coding problems submitted in the directory `/projectnb/alg504/yourname/MT` on your SCC account by Friday Nov 6, 2020 at 11:59PM. All your work must be your own. NO collaborations.

1. (10 pts) Answer True or False to each of the questions below. Each question is worth 2 points. Answer true only if it is true as stated, with no additional assumptions. No explanation is needed, but any explanation is likely to earn partial credit, and no explanation will not earn any credit if the answer is wrong.

   (a) All heaps can be represented by full trees.

   (b) Any sorting algorithms for 32 bit positive integers that only uses the $\leq$ comparison test must have worst case performance $\Omega(N \log(N))$.

   (c) It is possible to formulate Quick Sort to be worst case $O(N \log N)$ including the cost of picking a suitable pivot.

   (d) Given an array of N integers, the best algorithm has a worst case time to build the heap $\Theta(Nlog(N))$.

   (e) The two traversal BFS (bread first search) and DFS (depth first search) for a tree can implemented by entering the nodes one at time into Stack and Queue data structure respectively.

   (f) $N^2 e^{\ln(N^{-1})} \in \Theta(N)$.

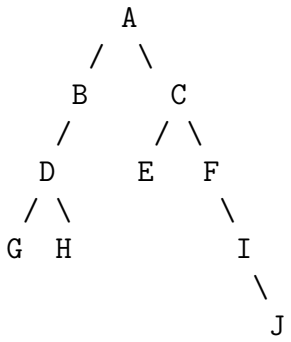
   (g) $\log(N)(1 + 1/N)^{N^2} \in \Theta(e^N log(N))$ HINT: $ln(1 + x) \simeq x$ for small x

   (h) $N^3 + 200N^2 \in \Theta(200N^3 + N^2)$

   (i) If $f(N) \in O(g(N))$ and $g(N) \in \Theta(h(N))$, then $f(N) \in O(h(N))$

   (j) If $f_i(N) \in O(N^2)$, then $\sum_{i=1}^{N} f_i(N) \in O(N^3)$

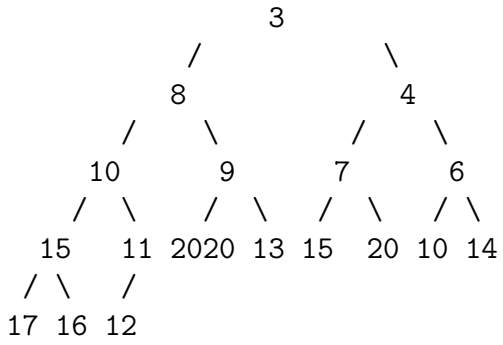
2. (10 pts) Consider the following tree:



(a) List the height and depth of each of the nodes A-J

(b) List the nodes in pre-order, in-order and post-order.

3. (15 pts) Consider the following min-heap.

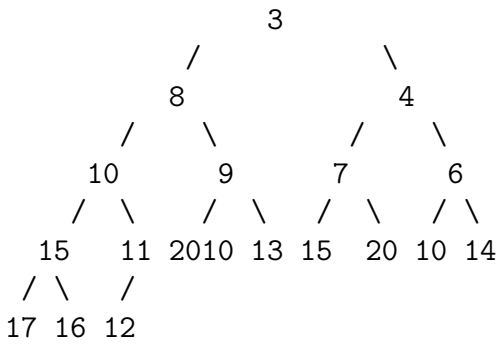


(a) Show the min-heap which results after inserting the element 2 in the heap. (Indicate the sequence of steps with arrows.) Then in this new heap show the steps required to delete element 3. Draw the final min-heap

(b) Consider min-heap as an array **int a[N+1]** with the size of heap stored in **a[0] = N**. (For example the one above has **a[0] = 18** and $a[1], a[2], \cdots, a[18]$. Describe carefully an $\Theta(N \log N)$ algorithm to sort `in place in place` the array elements $a[1], a[2], \cdots, a[N]$ in descending order. Use $a[0]$ the decreasing number remaining in the heap as you proceed with the sort.

(c) Now use $a[0]$ as a temporary to re-arrange `in place` the sort into ascending order with $O(N)$ extra swaps.

(d) Re-arrange the original min-heap (repeated below) into a max-heap by a "bottom up" $O(N)$ algorithm. Describe the steps level by level.



4. (15 pts) This problem is to construct step by step BST and AVL trees given the following list of $N = 10$ elements.

$$8 \quad 12 \quad 2 \quad 4 \quad 8 \quad 1 \quad 36 \quad 33 \quad 35 \quad 64$$

   (a) Insert them sequentially into a BST (Binary Search Tree).

   (b) 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).

   (c) Relative to the BST has AVL tree decreased the total height $T_H(N)$ and the total depth $T_D(N)$? Give the amount that these have change.

   (d) For $N = 2^{16} - 1$ what is the max and minimum values for $T_H(N) + T_D(N)$.

5. (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:	G	R	E	A	T	F	U	L
weights:	8	6	60	30	4	12	16	15

(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.)

(b) Identity the code for each letter and list the number of bits for each letter and compute the average number of bits per symbole 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.)

(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?

6. (10 pts) Given an array of n-distinct positive integers $a[n]$ and a second array of n-distinct positive integers $b[n]$ to construct the sum:

$$\sum_{i=0}^{n-1} a[i] * b[i] \tag{1}$$

   (a) Describe in a few words, an algorithm to find the maximun allowing different permutation both arrays $a[n]$ and $b[n]$. Give your best estimate of a bound on $T(n)$ the optimal method for this problem.

   (b) Now consider maximizing this sum by only permuting the the elements of $b[i]$. How does this maximun compare to the result of the first method? Describe you best algorithm for this method and estimate is scaling in n.

7. (25 pts) **Coding Exercise:** On GitHUB there is a working code for the BST problem for HW5 in directory `MT_codes` with the Height of each node labelled. The main file is called called `mainAVL.cpp`. The main file `standard BST` form [1] runs the standard BST algorithm for random list for range of sizes: n = 8, 16, 32, 64, 128, 256, 512, 1024, 2048,4096 and averages it over 10 permutations of or each list to construct average heights.

   Starting with this code in `MT_codes` the exercise is to add a new function

   `SearchTree InsertAVL( ElementType X, SearchTree T )`

   that satisfies the AVL balancing condition explained in the class slides. The relevant slides form class for the AVL insertion algorithm are copied into `MT_codes` on GitHUB. You may also look at a very nice description with animation on Wikipedia:

   https://en.wikipedia.org/wiki/AVL_tree

   Of course the AVL tree is still a BST tree but with optimal balance. With this new routine develop the main routine `mainAVL.cpp` to perform the following analysis:

```
(1) Rum the AVL version of the code  starting with a  random list for a
    range  10 sizes: n = 8, 16, 32, 64, 128, 256, 512, 1024, 2048,4096

(2) Permute each list over set of least 10 random permutation for each
    list and take average of the heights.

(3) Do the same average  total height of the standard BST forma and compare
    this average with the AVL form. Plot them as a function of n.

(4) For extra credit (10 pts)  do a standard BST deletion and
    reinseetion of the standard BST insertion (No balance!) of half the
    elements in the largest list (n = 4096) for both standard and AVL trees and
    compare the resultant average height  between the two methods.
```

[1] NOTE: By `standard BST` I mean a BST tree without AVL balancing.