Name: Sample Solution

UWNetID: _____

CSE 332 Autumn 2019: Midterm Exam

(closed book, closed notes, no calculators)

Instructions: Read the directions for each question carefully before answering. We will give partial credit based on the work you **write down**, so show your work! Use only the data structures and algorithms we have discussed in class so far.

Note: For questions where you are drawing pictures, please circle your final answer.

Good Luck!

Total: 100 points. Time: 60 minutes.

Question	Max Points	Score
1	22	Big-O
2	16	Code Analysis
3	9	Theta, etc
4	8	Find Recurrence
5	9	Solve Recurrence
6	10	BTrees
7	8	BTrees ++
8	8	Heaps
9	10	AVL
Total	100	

1. (22 pts) Big-Oh

(2 pts each) For each of the operations/functions given below, indicate the tightest bound possible (in other words, giving $O(2^N)$ as the answer to every question is not likely to result in many points). Unless otherwise specified, all logs are base 2. **Your answer should be as "tight" and "simple" as possible.** For questions that ask about running time of operations, assume that the most efficient implementation is used. For array-based structures, assume that the underlying array is large enough.

You do not need to explain your answer.

a) Pop in a ***stack*** containing N elements implemented as an array (worst case) — $O(1)$

b) $f(N) = N \log (\log (N + N)) + N (\log^2 N)$ — $O(N \log^2 N)$

c) $T(N) = 2\ T(N/2) + N$ — $O(N \log N)$

d) *Finding the largest odd value in an* ***AVL tree*** *containing N integers. (worst case)* — $O(N)$

e) $f(N) = 50\ N + (\log N)^2$ — $O(N)$

f) $T(N) = 2\ T(N - 1) + 6$ — $O(2^N)$

g) $f(N) = \log N + N^{1/2}$ — $O(N^{\frac{1}{2}})$

h) $f(N) = \log (N^2) + \log^2 N$ — $O(\log^2 N)$

i) Deleting the smallest value from a ***binary search tree*** containing N elements. *(worst case)* — $O(N)$

j) $T(N) = T(N - 2) + 4\ N$ — $O(N^2)$

k) *Inserting the integers 1, 2, 3, 4, … N (in that order) into a* ***binary min heap***. — $O(N)$ (No percolations are required)

2. **(16 pts) Big-Oh and Run Time Analysis:** Describe the worst case running time of the following pseudocode functions in Big-Oh notation in terms of the variable `n`. Your answer should be as "tight" and "simple" as possible. ***Showing your work is not required.***

I.
```
void pumpkin(int n) {
  if (n < 10000) {
    for (int i = 0; i < n * n; i += n) {
      print("I love cse 332! ");
    }
  }
  print(n);
}
```

Runtime: $O(1)$

II.
```
void boo(int n) {
  for (int i = 0; i < n; i++) {
    if (n % 4 == 0) {
      for (int j = 0; j < n; j++) {
        for (int k = 0; k < n; k++) {
          print("Happy Halloween!");
        }
      }
    }
  }
}
```

$O(n^3)$

III.
```
void spooky(int n) {
  for (int i = 0; i < n; i += 2) {
    if (i % 2 == 1) {
      for (int j = 0; j < n * n * n * n; j++) {
        print("wow this is awesome");
      }
    } else {
      for (int j = 0; j < n; j++) {
        print("who knew big O was so fun");
      }
    }
  }
}
```

$O(n^2)$

IV.
```
int candy(int n, int apple) {
  if (n < 1000) {
      for (int i = 0; i < n * n; i++) {
          apple++;
      }
      return apple;
  } else if (n < 2000) {
      return candy(n / 3, apple);
  } else {
    apple = apple + candy(n / 2, apple);
  }
  return apple + candy(n / 2, apple);
}
```

$O(n)$

3. (9 pts) Big-O, Big Ω, Big Θ

(3 pts each) For parts (a) – (c) circle **ALL** of the items (if any) that are TRUE. You do not need to show any work or give an explanation.

a) $N \log N + N^{(3/2)}$ is:

$O(N^{(3/2)})$ $\Theta(N \log N)$ $\Omega(N^2)$ $\Theta(N^{(3/2)})$

b) $N \log (N^2)$ is:

$\Theta(N \log^2 N)$ $\Omega(N \log N)$ $O(N \log N)$ $\Theta(N^2)$

c) $N^3 + 4^N \log N$ is:

$O(N^6)$ $\Omega(N^4)$ $\Theta(4^N)$ $\Theta(N^N)$

4. (8 pts) Recurrences

Give a base case and a recurrence for the runtime of the following function. Use variables appropriately for constants (e.g. c_1, c_2, etc.) in your recurrence (you do not need to attempt to count the exact number of operations). **YOU DO NOT NEED TO SOLVE** this recurrence.

```
int dontTrieAtHome(int n) {
   if (n <= 3) {
      return 25;
   }
   if (dontTrieAtHome(n - 3) <= 5) {
      for (int i = 0; i < log(n); i++) {
         print("we're what you'd call professionals.");
      }
   }
   for (int i = 0; i < n; i++) {
      print("busted!");
   }
   return 2 * dontTrieAtHome(n / 2) + 5 * n * n;
}
```

T(n) = c_0 For n =< 3

T(n) = $T(\frac{n}{2}) + T(n-3) + c_1 \cdot n + c_2$ For n > 3

Yipee!!!! YOU DO NOT NEED TO SOLVE *this* recurrence…

5. (9 pts) Solving Recurrences

Suppose that the running time of an algorithm satisfies the recurrence relationship:

$T(1) = 7.$

and

$T(N) = 2 * T(N/2) + 6$ for integers $N > 1$

Find the closed form for T(N). **You may assume that N is a power of 2.** Your answer should *not* be in Big-Oh notation – show the relevant ***exact*** constants and bases of logarithms in your answer (e.g. do NOT use "c_1, c_2" in your answer). You should not have any summation symbols in your answer and you should combine like-terms. The list of summations on the last page of the exam may be useful. ***You must show your work to receive any credit***.

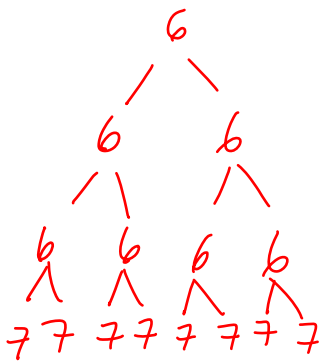
e.g. $N=8$

$$T(N) = 7 \cdot N + 6 \cdot \sum_{i=0}^{(\log_2 N)-1} 2^i$$

$$= 7 \cdot N + 6 \cdot (2^{\log_2 N} - 1)$$

$$= 7N + 6(N-1)$$

$$= 7N + 6N - 6 = \boxed{13N - 6}$$



$$T(N) = 2 \cdot T\left(\frac{N}{2}\right) + 6$$

$$= 2\left[2 \cdot T\left(\frac{N}{4}\right) + 6\right] + 6$$

$$= 4 \cdot T\left(\frac{N}{4}\right) + 2 \cdot 6 + 6$$

$$= 4 \cdot \left[2 \cdot T\left(\frac{N}{8}\right) + 6\right] + 3 \cdot 6$$

$$= 8 \cdot T\left(\frac{N}{8}\right) + 4 \cdot 6 + 3 \cdot 6$$

$$= 8 \cdot \left[2 \cdot T\left(\frac{N}{16}\right) + 6\right] + 7 \cdot 6$$

$$= 16 \cdot T\left(\frac{N}{16}\right) + 8 \cdot 6 + 7 \cdot 6$$

$$= 16 \cdot T\left(\frac{N}{16}\right) + 15 \cdot 6$$

$$= K \cdot T\left(\frac{N}{K}\right) + (K-1) \cdot 6$$

We want $\frac{N}{K} = 1$, $K = N$

$$= N \cdot T(1) + (N-1) \cdot 6$$

$$= N \cdot 7 + N \cdot 6 - 6 = \boxed{13N - 6}$$

Min # of leaves possible = $\frac{200}{10}$ = 20 (all full)

Max # of leaves possible = $\frac{200}{5}$ = 40 (all half full)

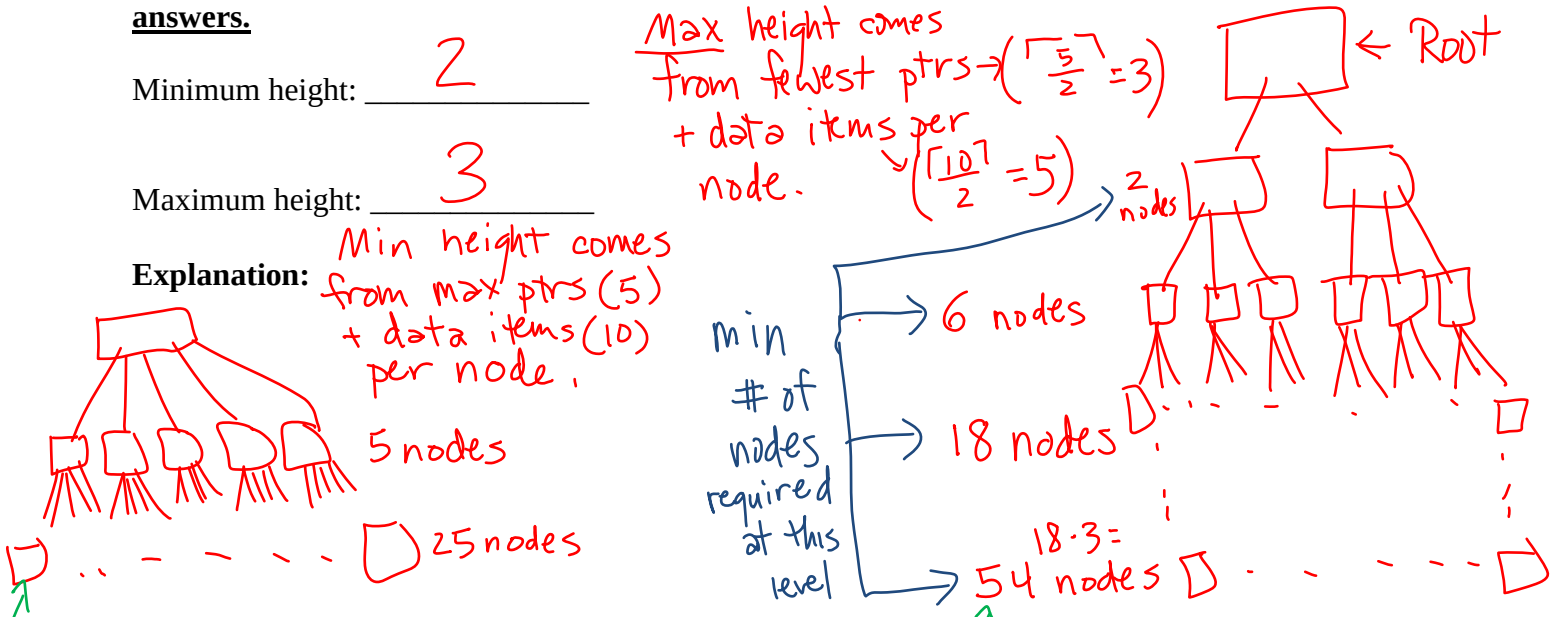
6. (10 pts total) B-Trees:

a) (6 pts) Given a B-tree (as defined in lecture and in Weiss) with M = 5 and L = 10, if you have inserted 200 items into the tree, **what is minimum and maximum height of the tree**? Give an exact number, not a formula for your answer. **Explain briefly how you got your answers.**

Minimum height: 2

Maximum height: 3

Max height comes from fewest ptrs → $\lceil \frac{5}{2} \rceil = 3$ + data items per node. $\lceil \frac{10}{2} \rceil = 5$



Explanation:

Min height comes from max ptrs (5) + data items (10) per node.

We could fit all of our data items at this level if these were leaf nodes and we cannot make the tree any shorter, so min height = 2.

We cannot make this many leaf nodes, so height = 3 is max height possible.

b) (4 pts) Given the following parameters for a B-tree with M = 11 and L = 5:

Key Size = 10 bytes

Disk Block Size (also known as Page size) = 128 bytes

Data Size = 20 bytes per record (***includes*** the key)

Assuming that M and L were chosen appropriately, **what is the likely size of a pointer** in the environment where this implementation will be deployed? Give a numeric answer in bytes (not a formula) and **a short justification *based on an equation(s)*** using the parameter values above.

$(M \cdot p) + (M-1) \cdot k \leq 128$

$11 \cdot p + (11-1) \cdot 10 \leq 128$

$11 \cdot p + 100 \leq 128$

$11 \cdot p \leq 28$

$p \leq \frac{28}{11}$

$p = \lfloor \frac{28}{11} \rfloor = 2$

Likely size of a pointer in bytes:
2

7. (8 pts) B-tree Modifications

You are given a B-tree (as defined in lecture and in Weiss) with appropriately picked values for M and L. Your friend suggests the following modifications to improve the performance of the find operations.

a. Replace each current leaf node with an AVL tree.
b. Replace each current leaf node with a binary min heap.

For each idea, explain why you think this **would or would NOT** lead to an improvement in the performance of the **find** operation on the overall B-tree. Assume that no further modifications to the tree are made (no further insertions or deletions). In your answer, you may assume you have access to (but may not modify) the underlying structure of the AVL trees and binary min heaps.

a. Replace each current leaf node with an AVL tree.

Would (Would NOT) improve the performance of find operation on overall tree (circle one)

Find in leaf node of B-tree is currently worst case $O(\log_2 L)$ - binary search in a sorted array, and entire leaf is on one disk block. Find in AVL tree would also be worst case $O(\log_2 L)$, however all the nodes of the AVL tree would not be guaranteed to be on the same disk block due to the pointers in each node. Thus we would expect performance of find to be much worse if using an AVL tree as leaf node.

b. Replace each current leaf node with a binary min heap.

Would (Would NOT) improve the performance of find operation on overall tree (circle one)

Find in leaf node of B-tree is currently worst case $O(\log_2 L)$ - binary search in a sorted array, and entire leaf is on one disk block. "Find" is not a standard operation on a binary min heap. If you had access to the underlying array you could implement the find operation with a linear scan of the heap array in time $O(L)$. While the heap would likely fit on a disk block and a linear scan would have good spatial locality, a binary search with worst case runtime of $O(\log_2 L)$ would perform better.

8. (8 pts) Heaps

Given a 3-heap (Min heap) of height 3, backed by an array (**Assume indexing starts at index 1**):

a) (3 pts)

What is the index of the first child of node i? 3*i - 1

What is the index of the second child of node i? 3*i

What is the index of the third child of node i? 3*i + 1

b) (5 pts) Finish the implementation of an insert in the 3-heap (Min heap). (**Assume indexing starts at index 1**)

Let **arr** be the backing array and let **size** be the size of the 3-heap.

```
void insert(int val) {
      if (size == arr.length - 1) {
            resize();
      }
      size++;
      int i = percolate( size, val);
      arr[i] = val;
}

int percolate(int hole, int val) {
      while ( (hole >1) && (val< arr[(hole+1)/3]) ) {

            arr[hole] = arr[ (hole+1)/3 ];

            hole = (hole+1)/3;
      }
      return hole;
}
```

9. (10 pts) AVL Trees: Suppose you want to keep track of the parent node of each **`AVLNode`** in a field called **`parent`**, such that this is the format of an **`AVLNode`**:

```
class AVLNode {
      AVLNode parent;
      String value;
      int key;
      int height;
      AVLNode[] children;
}
```

Note that the parent of the root node will be null.

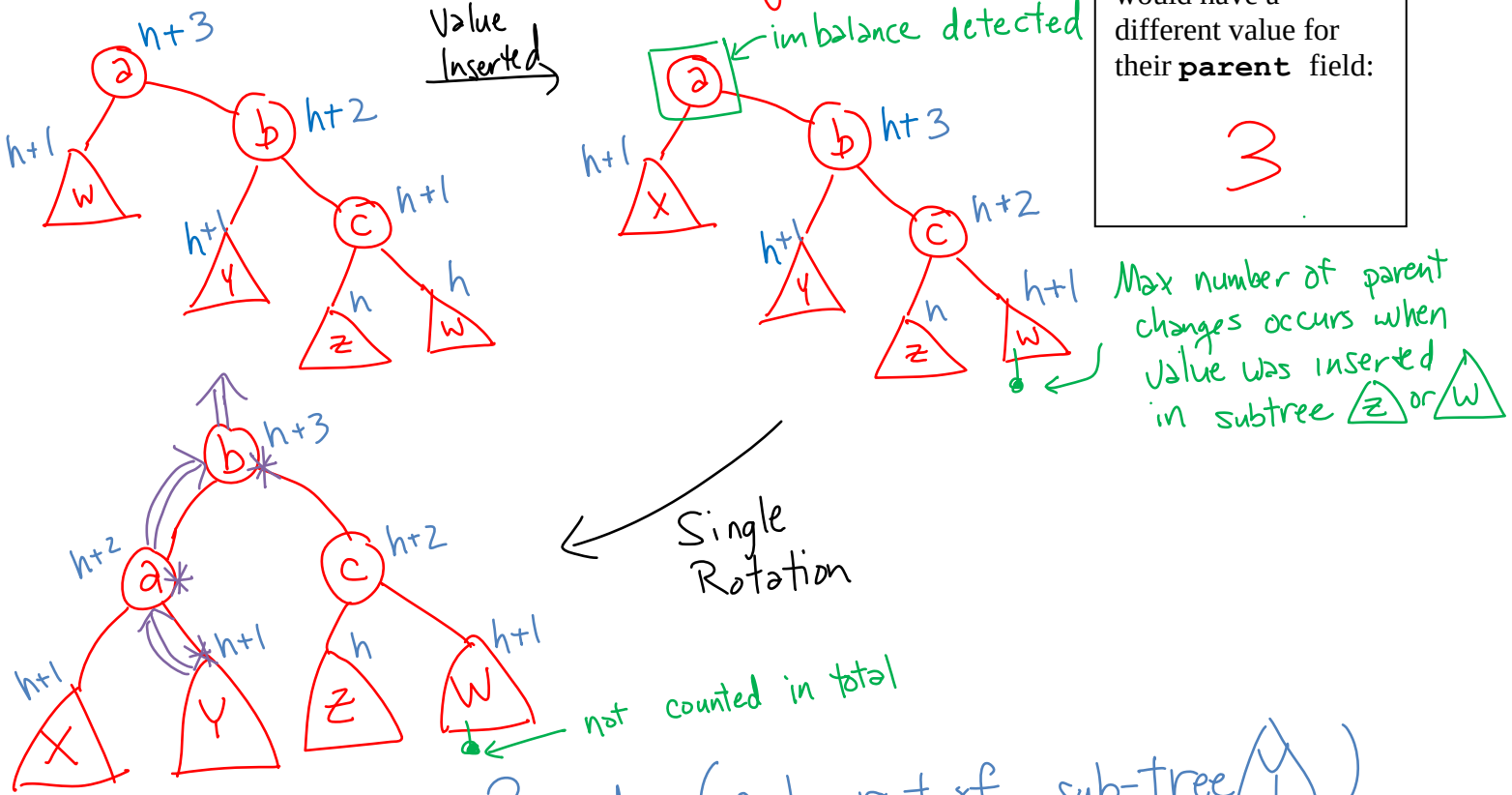
<u>You are originally given an AVL tree of height 10.</u> You insert a (key, value) pair into the tree creating a new **`AVLNode`**.

a) Assuming this insertion causes a **SINGLE ROTATION** to occur, what is the maximum *total* number of **`AVLNode`**s that would have a different value for their **`parent`** field than before the insertion? Do NOT count the new node you just inserted in this total. **For any credit, briefly explain your answer (a labelled diagram is fine).**

Maximum *total* number of **`AVLNode`**s that would have a different value for their **`parent`** field:

3

Using the example of a "case 4" single rotation, "case 1" would be mirror image.

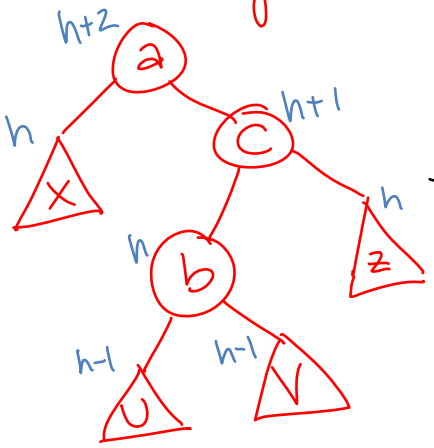


3 nodes (a, b, root of sub-tree Y) with new value for parent field. No other parts of the tree need to be changed since height has been restored to what it was before the insertion.

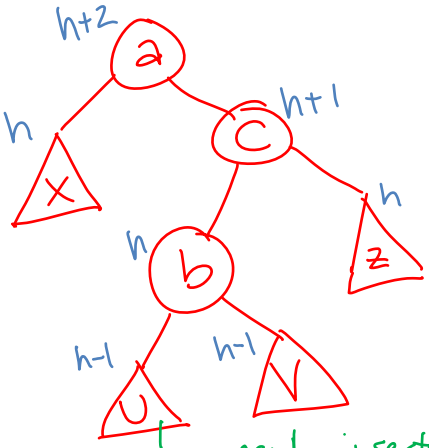
9. AVL Trees (continued)

b) Assuming this insertion causes a **DOUBLE ROTATION** to occur, what is the maximum *total* number of **`AVLNode`**s that would have different value for their **`parent`** field than before the insertion? Do NOT count the new node you just inserted in this total. **For any credit, briefly explain your answer (a labelled diagram is fine).**

Using a "case 3", "case 2" is mirror image.

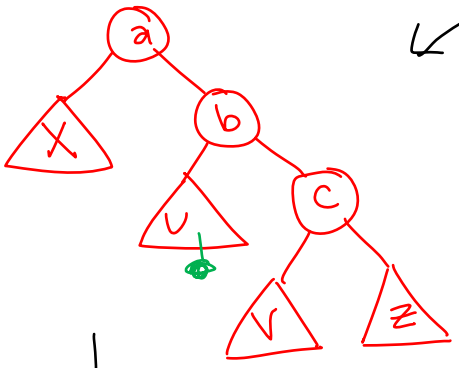


value inserted

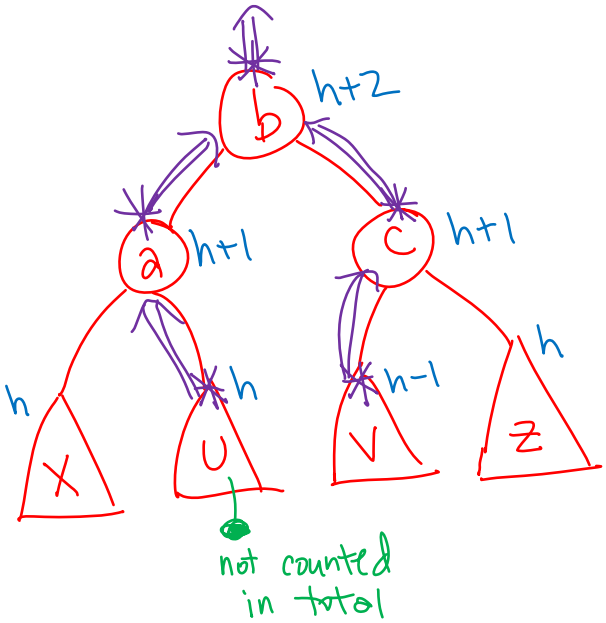


newly inserted value was somewhere in this sub-tree

step 1



Step 2



not counted in total

Maximum *total* number of **`AVLNode`**s that would have a different value for their **`parent`** field:

5

5 nodes (a, b, c, roots of sub-trees U and V) with new value for parent field. No other parts of the tree need to be changed since height has been restored to what it was before the insertion.