Name: Sample Solution

UWNetID: ______________________

CSE 332 Winter 2018: Midterm Exam
VERSION A

(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: 1 hr, 30 minutes.

Question	Max Points	Score
1	20	
2	16	
3	9	
4	6	
5	8	
6	6	
7	10	
8	6	
9	7	
10	12	
Total	100	

1. (20 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) $T(N) = T(N - 3) + 27$ $O(N)$

b) *Finding (but not removing) the smallest item in a **binary min heap** containing N elements (worst case).* $O(1)$

c) increaseKey*(k, amount) on a **binary min heap** containing N elements. Assume you have a reference to the key k that should be updated. (worst case)* $O(\log N)$

d) $f(N) = N \log_2 (4^N) + (\log N)^2$ $O(N^2)$

e) *Enqueue in a **FIFOqueue** containing N elements implemented using linked list nodes (worst case)* $O(1)$

f) *Creating a **binary min heap** from the values in an **AVL tree** containing N elements. (worst case).* $O(N)$

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

*h) Inserting a value into an **AVL tree** containing N elements, where the value being inserted is larger than the current largest value in the tree. (worst case)* $O(\log N)$

i) $T(N) = T(N/2) + 1/2$ $O(\log N)$

j) *Printing the values in a **binary search tree** containing N elements in decreasing order (worst case)* $O(N)$

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.***

Runtime:

```
 I.  int popcorn(int n) {
       if (n == 0) {
         return 1;
       }
       if (n % 3 == 0) {
         return 2 * n + popcorn(n / 3) + popcorn(n / 3);
       }
       if (n % 2 == 0) {
         return 3 * n + popcorn(n / 2) + popcorn(n / 2);
       }
       return popcorn(-n / 2);
     }
```

$O(N)$

```
 II. MinFourHeap heapy(int n) {
       h = new MinFourHeap();
       for (int i = 0; i < n * n * n; i++) {
         h.insert(i);
       }
       return h;
     }
```

$0, 1, 2, 3, \ldots n^3$

Does not require any percolation up.

$O(N^3)$

```
III. int sunny(int n) {
       if (n < 100) {
         return 7;
       } else {
         return sunny(n - 1) * sunny(n - 1);
       }
     }
```

$O(2^N)$

```
 IV. int woof(int n) {
       int husky = 0;
       if (n / 3 < 15) {
         for (int i = 0; i < n * n * n; i++) {
           for (int j = 0; j < i; j++) {
             husky += j;
           }
         }
       } else {
         for (int i = 0; i < n * n; i++) {
           husky += i;
         }
       }
       return husky;
     }
```

$O(N^2)$

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) $7 \log^2 N + N \log N$ is:

[✓ $\Omega(N \log N)$] $O(\log^2 N)$ [✓ $\Omega(N)$] $O(N)$

b) $2^{3* N}$ is:

$O(N^6)$ [✓ $\Omega(2^N)$] $\Theta(2^N)$ $\Omega(N^N)$

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

$\Omega(N)$ [✓ $O(\log N)$] $\Omega(\log N)$ $\Theta(\log^2 N)$

4. (6 pts) D-Heaps

a) Give a tight big-O running time of the worst case of a single deletemin operation in terms of N and D. KEEP all D and N terms in your answer, do not simplify your answer by removing D terms.

$$O(\underset{\text{\# comparisons per level}}{\underset{\uparrow}{D}} \cdot \overbrace{\log_D N}^{\text{\# of levels}})$$

b) Increasing D will decrease the height of our heap. One idea is to set D to be equal to N. In a couple of sentences explain why this is or is not a good idea.

Circle one: This IS a good idea [✓ This IS NOT a good idea]

Explanation:

Now on deletemin we must compare to N-1 children in order to find the smallest child, bringing the cost of a deletemin to $O(\underset{\text{\# comparisons}}{\underset{\nearrow}{N}} \cdot \underset{\text{height}}{1})$, which is much worse.

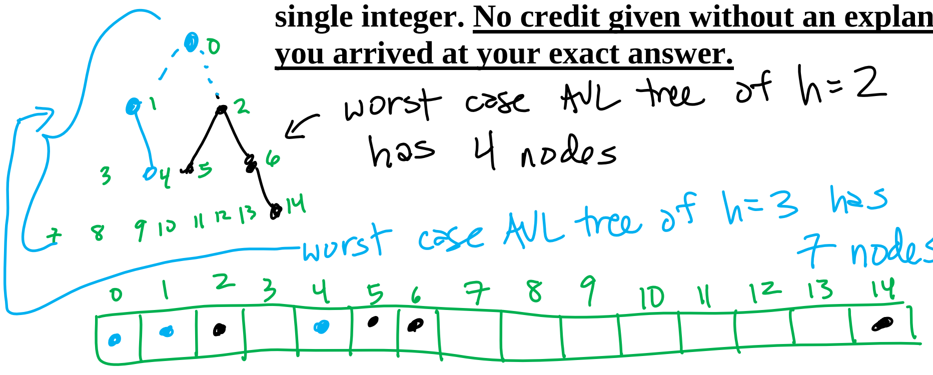
Note: Even though inserts might be faster due to shorter tree, if we assume every item that is inserted will also be deleted with deletemin then this still worse overall.

5. (8 pts) AVL Trees

We used an array to store a heap by reading the values off a row at a time and storing them in an array. What is the maximum number of locations in the array we would **waste** (would be left empty) if we decided to store an AVL tree of height 5 using the same approach? Assume that the root of the AVL tree is stored in index 0, its left child is stored in index 1 and its right child is stored in index 2, etc. You can make the array be exactly the size you want (no extra space at the end), but you must use the same storage approach we used with heaps. Show your work. **Give your answer as a single integer.** **<u>No credit given without an explanation that makes it clear how you arrived at your exact answer.</u>**

Maximum Locations wasted:

43



worst case AVL tree of h=2 has 4 nodes

worst case AVL tree of h=3 has 7 nodes

e.g. 8 locations wasted

$S(h) = 1 + S(h-1) + S(h-2)$

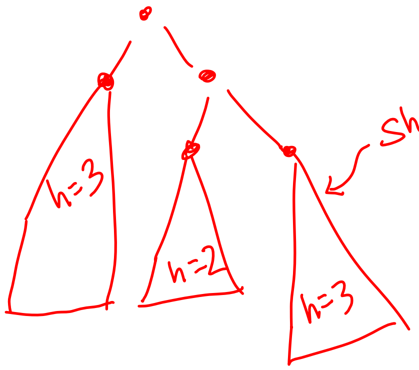
$S(4) = 1 + 7 + 4 = 12$

$S(5) = 1 + 12 + 7 = 20$ ← # nodes in W.C. AVL tree h=5

Perfect Binary Tree of h=5 has

$2^{5+1} - 1$ nodes $= 2^6 - 1 = 64 - 1 = 63$

Thus an array with 63 locations would be required to hold the worst case AVL tree if its shape was:



shaped like the tree shown above

Therefore $63 - 20 = 43$ locations in the array would be empty/wasted

6. (6 pts) Recurrences

Give a base case and a recurrence for the runtime of the following function. Use variables appropriately for constants (e.g. c1, c2, etc.) in your recurrence (you do not need to attempt to count the exact number of operations). **<u>YOU DO NOT NEED TO SOLVE</u>** this recurrence.

```
 int mystery(int n) {
     int x = 0;
     if (n <= 1) {
        for (int i = 7 * n; i > 0; i -= n) {
           x += 8;
        }
        return x;
     } else {
        x = 5 * mystery(n - 1);
        for (int i = 0; i < n * n; i += n) {
           x += 7;
        }
        return x + mystery(n - 1) * mystery(n - 3);
     }
}
```

- Even if $n \geq 1$, this loop only executes 7 times
- If $n \leq 0$ it does not execute at all

T(n) = c_1 For n <= 1

T(n) = $c_2 + c_3 \cdot n + 2 \cdot T(n-1) + T(n-3)$ For n > 1

Yipee!!!! YOU DO NOT NEED TO SOLVE ***<u>this</u>*** recurrence…

7. (10 pts) Solving Recurrences

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

$$T(1) = 5.$$

and

$$T(N) = T(N - 1) + 2N \qquad \text{for integers } N > 1$$

Find the closed form for T(N). Your answer should *not* be in Big-Oh notation – show the relevant ***exact*** constants in your answer (e.g. do NOT use "c1, c2" in your answer). You should not have any summation symbols in your answer. The list of summations on the last page of the exam may be useful. ***You must show your work to receive any credit***.

$$T(N) = T(N-1) + 2N$$

$$= T(N-2) + 2(N-1) + 2N$$

$$= T(N-3) + 2(N-2) + 2(N-1) + 2N$$

$$= T(N-4) + 2(N-3) + 2(N-2) + 2(N-1) + 2N$$

$$= T(N-4) + 2\sum_{i=0}^{3}(N-i)$$

$$= T(N-K) + 2\sum_{i=0}^{K-1}(N-i)$$

$$= T(N-K) + 2\left[\sum_{i=0}^{K-1} N - \sum_{i=0}^{K-1} i\right]$$

$N-K=1$

when $K=N-1$

$$= T(N-K) + 2 \cdot K \cdot N - 2\left(\frac{(K-1)K}{2}\right)$$

$$= T(N-K) + 2 \cdot K \cdot N - K^2 + K$$

$$= T(1) + 2 \cdot (N-1) \cdot N - (N-1)^2 + (N-1)$$

$$= 5 + 2(N^2 - N) - (N^2 - 2N + 1) + N - 1$$

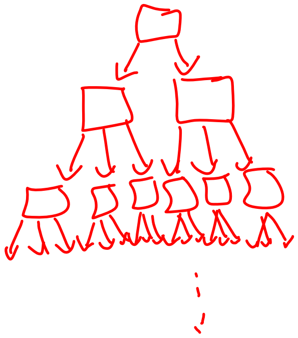
$$= 5 + 2N^2 - 2N - N^2 + 2N - 1 + N - 1$$

$$= N^2 + N + 3$$

8. (6 pts) B-Trees

Given M=5 and L=15, what is the minimum and maximum number of **pointers** in a B-Tree (as defined in lecture and in Weiss) of height 5? By pointers, we mean pointers that point to an interior or leaf node. Do not count the pointer that points to the root node. **Give a single number or a single number with an exponent for your answers, not a formula. For any credit, explain briefly how you got your answers.**

Min # pointers →



Minimum number of pointers: 242

Maximum number of pointers: 3905

Explanation:

$$\text{Min \# pointers} = \underset{\uparrow \atop \text{root} \to \text{1st level}}{2} + \underbrace{2\cdot 3}_{\text{1st level} \downarrow \text{2nd level}} + \underbrace{2\cdot 3^2}_{\text{2nd level} \downarrow \text{3rd level}} + 2\cdot 3^3 + 2\cdot 3^4$$

$$= 2\cdot\left(\sum_{i=0}^{4} 3^i\right) = 2\cdot\left(\frac{3^5-1}{3-1}\right) = 2\cdot\left(\frac{243-1}{2}\right) = 242$$

$$\text{Max \# pointers} = 5 + 5^2 + 5^3 + 5^4 + 5^5$$

$$= \left(\sum_{i=0}^{5} 5^i\right) - 1$$

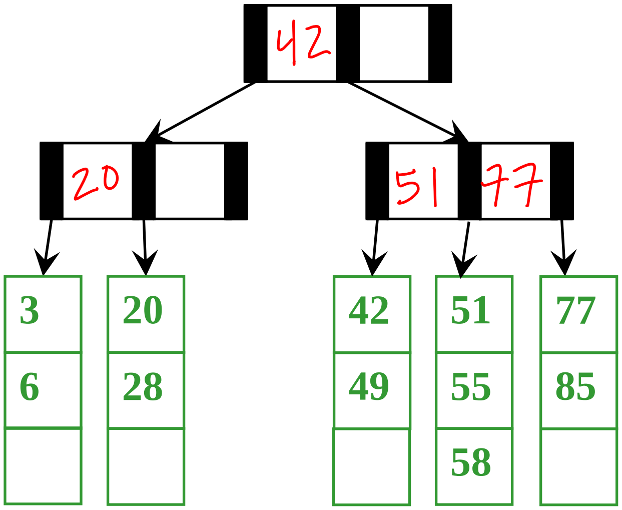
$$= \left(\frac{5^6-1}{5-1}\right) - 1 = \left(\frac{15625-1}{4}\right) - 1 = \left(\frac{15624}{4}\right) - 1 = 3906 - 1$$

$$= 3905$$

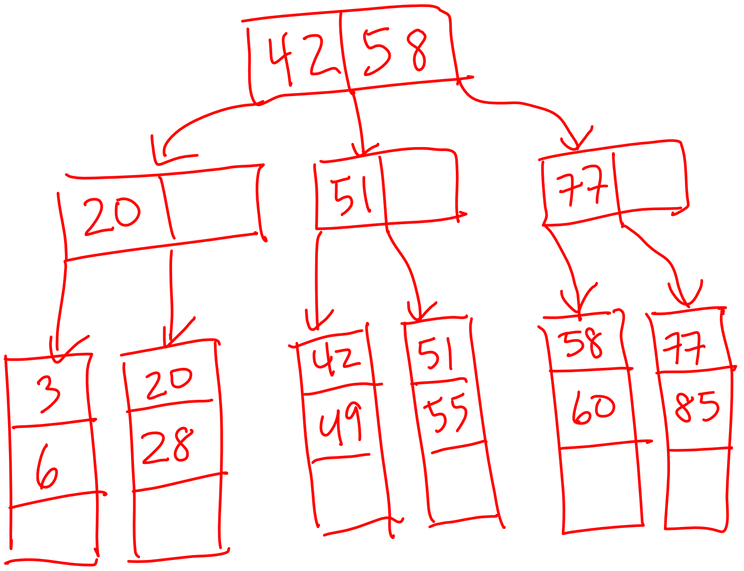
9. (7 pts) B-trees

a) (1 pt) In the B-Tree shown below, **write in the values for the interior nodes**.

ORIGINAL:



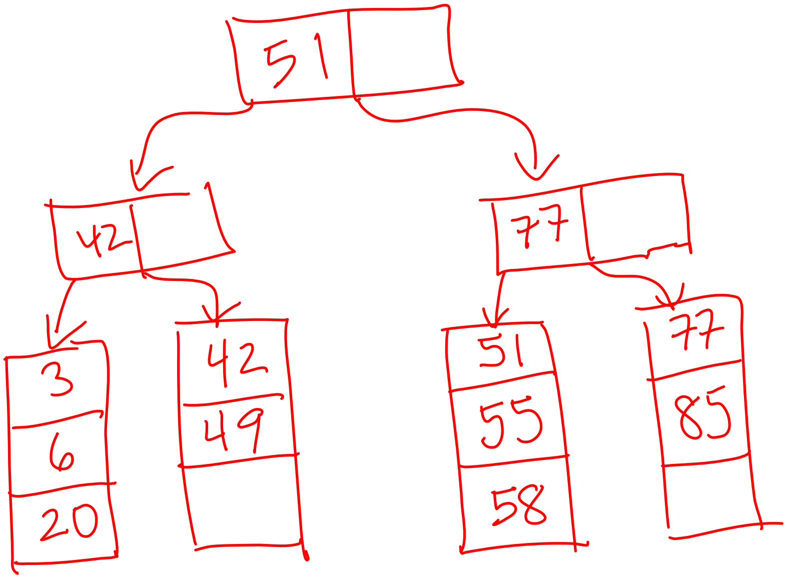
After inserting 60:



b) (3 pts) Starting with the **ORIGINAL** B-tree shown above, in the box above, draw the tree resulting after inserting the value 60 (*including values for interior nodes*). Use the method for insertion described in lecture and in the book.

c) (3 pts) Starting with the **ORIGINAL** B-tree shown above, below, draw the tree resulting after deleting the value 28 (*including values for interior nodes*). Use the method for deletion described in lecture and in the book.

After deleting 28:



10. (12 pts) Data Structure Analysis

The Allen School has developed a new TA Management System that assigns TAs to courses based upon how many times the TA has previously TAed for that course. Due to the new AI Virtual TA-bots that will be now be used for courses, each course is allocated only one actual human TA. Furthermore, each TA can be assigned to only one course. The school wants to prioritize the largest courses, thus a TA will be assigned first to the CSE course that has the largest enrollment for that quarter, then one will be assigned to the CSE course with the second largest enrollment, etc..

The proposed data structure, called a Course-heap, is a binary max heap containing a total of C CSE courses, prioritized by enrollment. Each course in the Course-heap contains a reference to a binary max heap of TAs who could potentially TA for that course, called a TA-heap. The priority of each TA is how many times the TA has TAed that particular course. Ties are broken alphabetically by userID, and even TAs who have never TAed a course before are included in that course's TA-heap. Once a TA has been assigned as the one human TA for a particular course, they should be removed from all other TA-heaps (since they can only serve as the TA for one course). Please use these variables in your answers:

C = total number of CSE courses in the system
M = maximum number of TAs in any TA-heap associated with a CSE course

Since you have no idea how C relates to M, you will need to keep all factors of C and M in your answers (e.g. you should NOT assume that C > M, or that either one is a constant).

Assume that all data structures have sufficient capacity for new inserts and that you may have access to the underlying data structure if needed.

10. (Continued)

a) (6 pts) The process of assigning a single TAs to a course is as follows:

i. First, the highest priority course (the one with the largest enrollment) is deleted from the Course-heap.

ii. Next, the TA with the most experience TAing that course is assigned to that course and is deleted from the TA-heap associated with that course.

iii. Finally, this new-ly assigned TA must be removed from the TA-heaps associated with all the other courses.

What would be the worst case big-O run-time of this operation (assigning a single TA to a single course) on this data structure? Give a simplified answer in terms of C and M. **Explain your answer briefly for any credit.**

Simplified big-O run-time:

$O(C \cdot M)$

i) deletemax on Course-heap
$O(\log C)$

ii) deletemax on TA-heap for that course
$O(\log M)$

iii) traverse the Course-heap array $(O(C))$, and for each TA-heap, traverse the TA-heap array $(O(M))$, call remove $(O(\log M))$ once, when the assigned TA has been found.

Thus:

$O(\log C + \log M + C \cdot (M + \log M))$

$\longrightarrow O(C \cdot M)$

10. **(Continued)**

b) (6 pts) Adding a course to the Course-heap can be done using the **addCourse** function. **addCourse** takes three parameters: a course name, course enrollment, and a dictionary of TAs who could potentially TA for that course implemented as an AVL tree. The key to the dictionary is the TA's userID and the value is the number of times the TA has TAed for the given course. **addCourse** does this:

i. Creates a TA-heap from the given dictionary
ii. Adds the course into the Course-heap.

Describe briefly the most efficient way to implement addCourse and what the worst case big-O run-time would be. Give a simplified answer in terms of C and M. If you use any algorithms we talked about in class, you may just refer to that algorithm without describing how to implement it.

Simplified big-O run-time:

$O(M + \log C)$

i) - Traverse the given AVL tree $(O(M))$, place the contents of the AVL tree into an array with "# of times TAed for this course" as the primary value.

- Convert the array into a TA-heap by calling buildheap $(O(M))$ to create a max heap.

ii) Insert this new course into the Course-heap $(O(\log C))$ using the course enrollment as the priority and the TA-heap as created in step i)

Thus: $O(M + M + \log C) \longrightarrow O(M + \log C)$