Name:________________________________ UW NetID:__________

CSE332 Summer 2012 Midterm Exam, July 18, 2012

Please do not turn the page until the bell rings.

Rules:

- The exam is closed-book and limited-note. You are permitted a single, handwritten 3x5 index card of notes. You must turn in this card with your exam.
- Calculators are also permitted but not necessary.
- Please stop promptly at 12:20.
- You can rip apart the pages, but please staple them back together before you leave.
- Blank paper for extra room are available upon request.
- This exam contains 10 questions (many with multiple parts). There are **110 points** total, but the exam is worth **100 points**, meaning that you may earn some extra points.

Advice:

- The questions are not necessarily in order of difficulty. Read through the entire exam first and then **skip around** as you see fit. Make sure you get to all the problems.
- Read questions carefully. Understand a question before you start writing.
- Write down thoughts and intermediate steps to earn partial credit. **Circle your final answer**.
- If you have questions, ask.
- Relax. You are here to learn.

EXAM SCORE	/ 100

This page is for instructor use only

	TOTAL SCORE	/ 100
1.	Algorithmic Analysis	/ 14
2.	Recurrence Relations	/ 8
3.	Move-to-Front Structure	/ 10
4.	Min-Heaps	/ 10
5.	AVL Trees	/ 14
6.	AVL Confirm	/ 10
7.	Splay Trees	/ 6
8.	Splay Tree Range Restrict	/ 10
9.	B Trees	/ 10
10.	Hash Tables	/ 18

1. (14 pts) Algorithmic Analysis

The following questions all refer to big-O, big-Θ, and big-Ω notation and algorithmic analysis. For parts (a) and (b), you will need to provide [dis]proof of the bounds. For parts (c)-(l), you only need to write down the asked-for bound in the underlined area. Each bound should be as tight and simple as possible

(a) Prove (give a c and n_0) or disprove that $5n^2 - 20n + 3$ is $O(n^3)$

(b) Prove (give a c and n_0) or disprove that 2^{3n} is $O(2^n)$

(c) What is the tightest bound you can give for $f(n) = 2n^3 - 3n\log n$? (c) ____________

(d) What is the tightest bound that you can give for the summation $\sum_{i=0}^{n} i^k$? (d) ____________

(e) What is the worst-case performance for a building a binary min-heap of n items **without** using a call to BuildHeap? (e) ____________

(f) What is the big-O bound for performing find in a perfect BST? (f) ____________

(g) What is the tightest bound you can give for deletion in an unsorted array (ignoring the cost of finding the element to delete)? (g) ____________

(h) What is the average time for insertion in a B tree with $M = 32$ and $L = 8$? (h) ____________

(i) What is the worst-case time for insertion into a 5-heap? (i) ____________

(j) What is the worst-case time for a **single** call to enqueue in an array-based queue? (j) ____________

(k) What is the **amortized** time for enqueue in an array-based queue? (k) ____________

(l) What is the run-time for the following code? (l) ____________

```
for(i=1; i<n; i++) {
   x = n;
   while (x > 0) {
      sum++;
      x = x / 2;
   }
}
```

2. (8 pts) Recurrence Relations

For this problem you will be working with the following recurrence relation:

$$T(n) = \begin{cases} 4 & n = 1 \\ T(n) = 4 + 4T(\lfloor \frac{n}{2} \rfloor) & n > 1 \end{cases}$$

(a) Show the values for $T(n)$ for all integers in the range $1 \leq n \leq 8$

(b) Provide the closed form for $T(n)$. Show your work; do not just show the final equation. You may assume n is sufficiently large such that the floor function does not lead to rounding issues.

3. (10 pts) Move-to-Front Structure

The splay tree's idea of moving recently touched elements to positions that are more easily fetched can be applied to other data structure. In particular, consider a linear data structure (i.e., not a tree) called *MoveToFront*. MoveToFront provides the following functionality:

- *insert(x)*: The item *x* is inserted at the front of MoveToFront.
- *find(x)*: Returns *true* or *false* depending on if *x* is currently in MoveToFront. After a call to find, *x* is move to the front so that it will be accessed first on the next find.
- *remove(x)*: Removes element *x* from the structure. Other than this removal, the ordering of the other elements remains unchanged.

For this question, you need to do the following:

(a) Provide a description of how you would implement MoveToFront. Provide pseudocode for both *insert* and *find*. You do NOT need to implement *remove*. Your pseudocode should be clear and unambiguous but does not need to be exact Java code.

(b) Provide a brief justification for your design decisions.

(c) Describe the best AND worst case performance for both insert AND find.

You may make any assumptions you feel are necessary as long as you state them in your writeup.

4. (10 pts) Min-Heaps

For this question, you will be working with binary min-heaps.

(a) What are the two properties required by a heap?

(b) Draw the heap produced by *BuildHeap()* given the following sequence of keys:

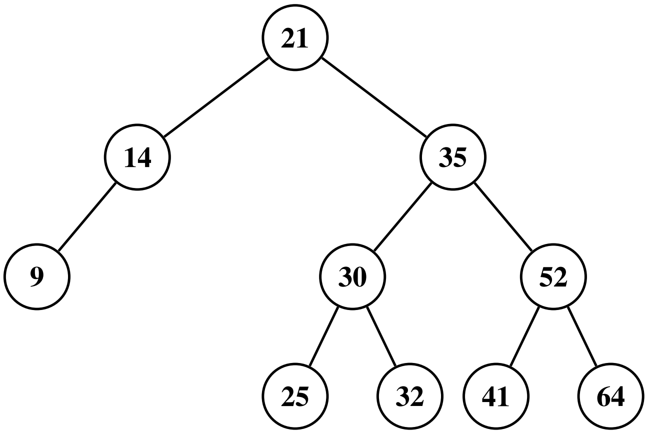
15, 29, 3, 8, 11, 35, 25, 57, 7, 1, 44, 16

(c) Using your answer from (b), draw the heap after a call to *deleteMin()*

(d) Using your answer from (c), draw the heap after a call to *insert(6)*

(e) Using your answer from (d), show the array representation for the heap.

5. (14 pts) AVL Trees



(a) What is the balance for the following nodes in the AVL tree: **14**, **21**, **35**, and **41**?

(b) Show the AVL tree after inserting **3** into the provided tree.

(c) Show the AVL tree after inserting **36** into your answer from (b).

(d) Show the AVL tree after inserting **70** into your answer from (c).

(e) Show the AVL tree after deleting **52** from your answer from (d). Your deletion should use the immediate predecessor for replacement.

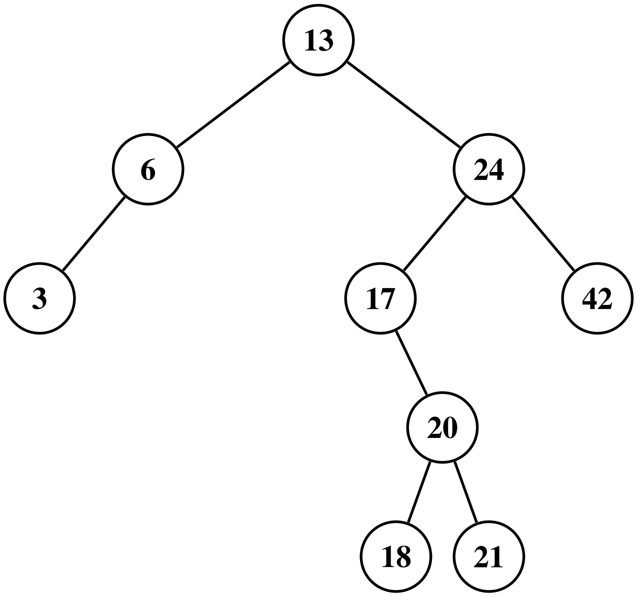
(f) Show the AVL tree after deleting **36** from your answer from (e). Your deletion should use the immediate predecessor for replacement.

6. (10 pts) AVL Confirmation

Give pseudocode for a O(n) algorithm that veries that an AVL tree is correctly maintained. Assume every node has fields key, data, height, left, and right and that keys can be compared with $<$, $==$, and $>$. The algorithm should verify all of the following:

- The tree is a binary search tree
- The height information of every node is correct
- Every node is balanced

7. (6 pts) Splay Trees



(a) Show the splay tree after a call to **find(20)**.

(b) Using your answer from (a), show the splay tree after a call to insert **insert(8)**. You should use the find/split approach for inserting.

(c) Using your answer (c), show the splay tree after a call to **delete(17)**. Your should use the find/join approach for deletion and use the max from the left subtree for replacement.

8. (10 pts) Splay Tree Range Restrict

A common enhancement to the Dictionary ADT is **RangeRestrict(x,y)**. This method constrains the dictionary to only contain keys between x and y inclusive. In other words, it removes all nodes whose keys are $< x$ or $> y$.

For this question, you are to provide pseudocode for how you would implement **RangeRestrict(x,y)** in a splay tree. You may assume that the splay tree class is a subclass of a general binary search tree class. The BST class has implementations of *insert*, *find*, *remove*, *findMax*, *findMin*, *findPredecessorOf*, and *findSuccessorOf*. The Splay class implements its own versions of *insert*, *find*, and *remove*. Thus, you may use any of the standard splay tree operations or their BST equivalents (i.e., be sure to state if you are using BSTfind or SplayFind). Your solution should be efficient and correct.

It is up to you to decide how to handle situations where $y \geq x$.

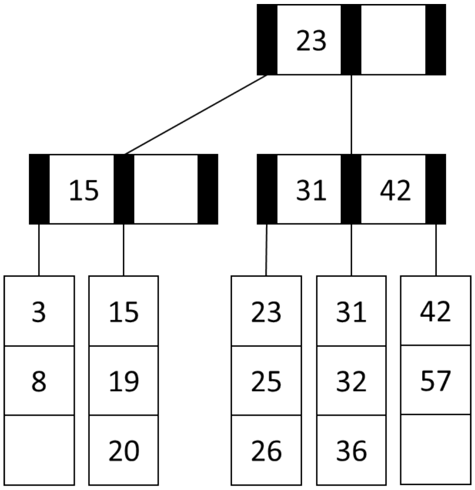
If you make any further assumptions, be sure to state them clearly.

9. (10 pts) B Trees

(a) Using the B tree ($M = 3$ and $L = 3$) to the right, show the resulting tree after inserting **43**.

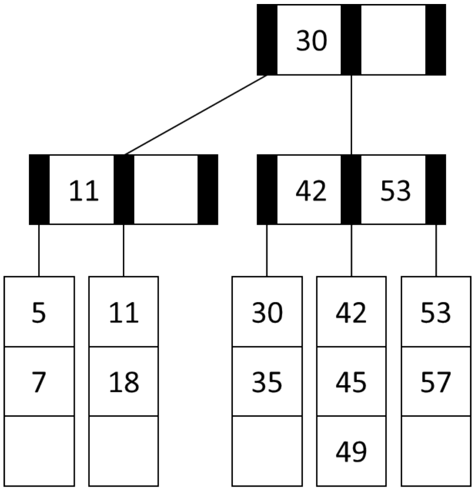
(b) Using your answer from (a), show the resulting tree after inserting **17**.

(c) Using your answer from (b), show the resulting tree after inserting **39**.



(d) Using the B tree ($M = 3$ and $L = 3$) to the right, show the resulting tree after deleting **57**.

(e) Using your answer from (d), show the resulting tree after deleting **5**.



10. (18 pts) Hash Tables

For each of the following versions of hash tables, insert the following elements in this order:

34, 16, 45, 53, 6, 29, 37, 78, and 1

For each table, $TableSize = 11$, and you should use the primary hash function $h(k) = k\%11$. If an item cannot be inserted into the table, please indicate this and continue inserting the remaining values.

For double hashing, the secondary hash is $g(k) = 1 + (k/4)\%T$, where T is the table size.

For separate chaining, insert at the front of the list.

Linear Probing

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Quadratic Probing

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Double Hashing

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Separate Chaining

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

What is the load factor of the hash table using *linear probing*?

What is the load factor of the hash table using *separate chaining*?