Name:_____

CSE332, Spring 2012, Midterm Examination
April 27, 2012

Please do not turn the page until the bell rings.

Rules:

- The exam is closed-book, closed-note.
- **Please stop promptly at 3:20.**
- You can rip apart the pages, but please staple them back together before you leave.
- There are **100 points** total, distributed **unevenly** among **9** questions (many with multiple parts).

Advice:

- Read questions carefully. Understand a question before you start writing.
- **Write down thoughts and intermediate steps so you can get partial credit. But clearly circle your final answer.**
- The questions are not necessarily in order of difficulty. **Skip around.** Make sure you get to all the problems.
- If you have questions, ask.
- Relax. You are here to learn.

Name:_____

1. (**12** points) Give a big-O bound on the worst-case running time of each code fragment below in terms of n.

   - Assume integer arithmetic (division rounds down).
   - Each bound should be as "tight" and "simple" as possible.

   (a)
   ```
   for(i = 0; i < n; i++) {
      for(j = 0; j < n; j++) {
        for(k = 0; k < i + j - 4; k++) {
           sum++;
        }
      }
   }
   ```

   (b)
   ```
   x = n;
   while (x > 0) {
     sum++;
     x = x / 2;
   }
   ```

   (c)
   ```
   sum = 0;
   i = 0;
   j = 7n;
   while(i < j) {
     sum++; i++; j--;
   }
   ```

   (d)
   ```
   x = n;
   while (x > 0) {
     y = n;
     while (y > 0) {
       sum++;
       y = y / 2;
     }
     x = x / 2;
   }
   ```

   (e)
   ```
   tree = new AVLTree();
   for(i = 0; i < n; i++) {
     tree.insert(foo()); // foo produces some element in O(1) time
   }
   sum = 0;
   for(i = 0; i < n; i++) {
     sum += tree.findMin();
     tree.insert(foo()); // foo produces some element in O(1) time
   }
   ```

   Solution:
   (a) $O(n^3)$ (b) $O(\log n)$ (c) $O(n)$ (d) $O(\log^2 n)$ (which means $O((\log n)(\log n))$, not $O(\log \log n)$) (e) $O(n \log n)$

Name:_____

2. (**10** points) Assume there are two functions over n called $f_1(n)$ and $f_2(n)$ that have positive results for all n.
Let $g(n)$ be defined as $g(n) = \max(f_1(n), f_2(n))$.
Let $h(n)$ be defined as $h(n) = f_1(n) + f_2(n)$.

   (a) Using the **formal definition** of big-O, show that $f_1(n)$ is in $O(g(n))$.

   (b) Using the **formal definition** of big-O, show that $g(n)$ is in $O(h(n))$.

   (c) Using the **formal definition** of big-O, show that $h(n)$ is in $O(g(n))$.

   None of your answers should require more than 1–3 sentences.

   Solution:

   (a) We need to give a c and n_0 such that $f_1(n) \leq cg(n)$ for all $n \geq n_0$. Any $c \geq 1$ and $n_0 \geq 0$ suffices because $f_1(n) \leq \max(f_1(n), f_2(n))$ for any n.

   (b) We need to give a c and n_0 such that $g(n) \leq ch(n)$ for all $n \geq n_0$. Any $c \geq 1$ and $n_0 \geq 0$ suffices because $\max(f_1(n), f_2(n)) \leq f_1(n) + f_2(n)$ for any n since $f_1(n)$ and $f_2(n)$ have positive results for all n (so their sum must exceed their max).

   (c) We need to give a c and n_0 such that $h(n) \leq cg(n)$ for all $n \geq n_0$. Any $c \geq 2$ and $n_0 \geq 0$ suffices because $f_1(n) + f_2(n) \leq 2\max(f_1(n), f_2(n))$ for any n since the definition of max ensures $f_1(n) \leq \max(f_1(n), f_2(n))$ and $f_2(n) \leq \max(f_1(n), f_2(n))$.

Name:_____

3. (**9** points) Recall:

   - The structure property for a binary min-heap is that it is a *complete binary tree*.
   - A *complete binary tree* may or may not be a *perfect binary tree*.

   (a) What is the definition of a complete binary tree? In addition to an English sentence or two, draw a picture to demonstrate.

   (b) What is the definition of a perfect binary tree? In addition to an English sentence or two, draw a picture to demonstrate.

   (c) Complete this alternate definition of a complete binary tree. For each blank, put either a height (e.g., h, $h-1$, etc.) or a kind of tree (e.g., complete or perfect).

   Any binary tree of height 0 is a perfect binary tree and therefore also a complete binary tree.
   Any complete tree of height $h > 0$ is either:

   - a root whose left child is a _____ binary tree of height _____ and whose right child is a _____ binary tree of height _____,
   - or a root whose left child is a _____ binary tree of height _____ and whose right child is a _____ binary tree of height _____.

   (The two bullets above are the same, but they will not be the same after you correctly fill in the blanks.)

Solution:

(a) A complete binary tree is a binary tree where each row of the tree is completely full except the bottom row may be partially full, but must be full from left to right. [Picture omitted from sample solution.]

(b) A perfect binary tree is a binary tree where each row of the tree is completely full. (A height h tree must have $2^{h+1} - 1$ nodes.) [Picture omitted from sample solution.]

(c) Any complete tree of height $h > 0$ is either:

- a root whose left child is a complete binary tree of height $h-1$ and whose right child is a perfect binary tree of height $h-2$,
- or a root whose left child is a perfect binary tree of height $h-1$ and whose right child is a complete binary tree of height $h-1$.

Name:_____

4. (**10** points) We studied a particular way to use an array to represent a binary tree. When we represented a binary min-heap with n elements this way, we did not waste much space because an array of size $n + 1$ was enough. For a general binary tree, much more space may be wasted (unused).

   (a) Using the array representation we studied, what is the largest array that might be necessary to hold a binary tree with 5 elements?

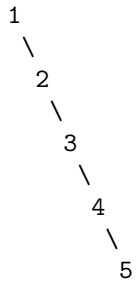
   (b) Give an example binary search tree that leads to the largest-array case for your answer to part (a). (Draw the tree, *not* its array representation.)

   (c) Using the array representation we studied, what is the largest array that might be necessary to hold a tree with n elements? Answer precisely in terms of n.

   Solution:

   (a) 32-element array (root at index 1, right child at 3, its right child at 7, its right at 15, its right child at index 31)

   (b) Any tree with 5 nodes where there is 1 leaf, all interior nodes have only a right child, and it obeys the BST order property. For example:



   (c) 2^n

   For (a) and (c), it is okay to *consistently* assume that the values start at index 0 instead of index 1, leading to answers of 31 and $2^n - 1$.

Name:_____

5. (**16** points)

   (a) There are six possible AVL trees containing elements with keys 2, 4, 6, 8, and 10. *Draw all six trees.* (Each tree will have all five keys in it.)

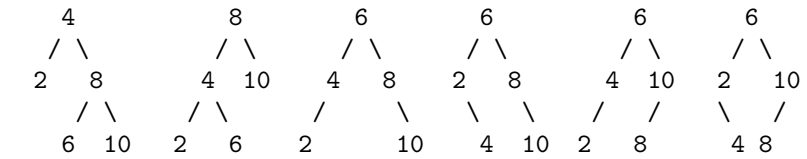
   (b) Choose *one* of your six trees from part (a) such that inserting an element with key 7 into it requires *no rotations*. Clearly indicate what tree you start with and show the result of the insertion.

   (c) Choose *one* of your six trees from part (a) such that inserting an element with key 7 into it requires *one single rotation*. Clearly indicate what tree you start with and show the result of the insertion.

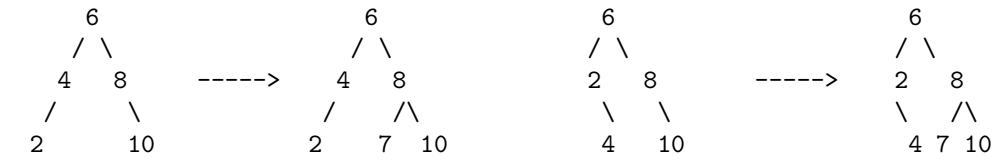
   (d) Choose *one* of your six trees from part (a) such that inserting an element with key 7 into it requires *one double rotation*. Clearly indicate what tree you start with and show the result of the insertion.

Solution:

(a)

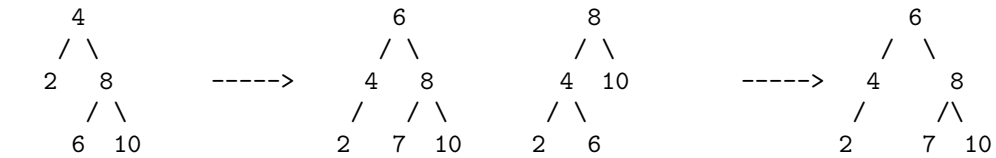


(b) Two correct solutions:



(c) Two correct solutions:

(d) Two correct solutions:



Name:_____

6. (**8** points) On a homework, we investigated a clever data structure for a double-ended priority queue, with $O(\log n)$ operations `insert`, `deleteMin`, `deleteMax` and $O(1)$ operations `findMin` and `findMax`. A colleague of yours suggests there is a simpler data structure with the same asymptotic behavior:

   - Keep a binary min-heap and perform the operations on it as usual.
   - Also keep a separate field that holds the index with the max element so that `findMax` can still be $O(1)$ and you know where the max element is to delete it. When inserting an element, if it is greater than the max element, update the separate field appropriately.

   This approach does not work. Which operation will require an $O(n)$ algorithm and why?

   Solution:
   `deleteMax` will be $O(n)$. After we delete the max element, we have to find the *new* max element at least before completing the next `deleteMax` operation. But the next max could be in any leaf of the binary min-heap and there are $O(n)$ such leaves. (We can't also store the second-max element without having the same problem with three `deleteMax` operations, and if you keep a sorted list of elements by priority, you have a much inferior priority queue implementation based on a sorted list.)

   [Note the question really should have asked which operation would require an $\Omega(n)$ or $\Theta(n)$ operation since any $O(1)$ operation is also $O(n)$.]

Name:_____

7. (**16** points)

   (a) What is the worst-case big-O running-time for the `find` operation in a B tree with n elements, at most M children at an internal node, and at most L data items at a leaf?

   (b) Suppose the `find` algorithm uses linear search instead of binary search whenever possible. Now what is the worst-case big-O running-time?

   (c) True or false and *explain your answer briefly*: If every leaf of a B tree is *not* full, then an `insert` operation definitely will *not* require a split operation.

   (d) True or false and *explain your answer briefly*: If every leaf of a B tree *is* full, then an `insert` operation definitely *will* require a split operation.

   (e) True or false and *explain your answer briefly*: If the root of a B tree is *not* full, then an `insert` operation definitely will *not* increase the height of the tree.

   (f) True or false and *explain your answer briefly*: If the root of a B tree *is* full, then an `insert` operation definitely *will* increase the height of the tree.

Solution:

(a) $O(\log_2 L + \log_2 M \log_M n)$

(b) $O(L + M \log_M n)$

(c) True: the new element will go in some leaf and there is room in it

(d) True: whatever leaf should hold the new element must be split

(e) True: even in the rare case that all nodes between the root and the leaf where the new element goes must be split, there will still be room for the two new children of the root to be pointed to from the root

(f) False: for example, there is likely room in the leaf where the new element goes, in which case no splits are needed

Name:_____

8. (**6** points) Consider this simple type for representing complex numbers:

```
class ComplexNumber {
   double realPart;
   double imaginaryPart;
   ...
}
```

If we define arithmetic operations over these numbers, we could use them instead of `double` since they are a generalization (we can think of every real number as a complex number).

If we want to put complex numbers in a hashtable, we need a hash function to convert them to integers. Give a reason why making the hash function for `c` be `(int)(c.realPart * c.imaginaryPart)` is likely to be a very poor choice.

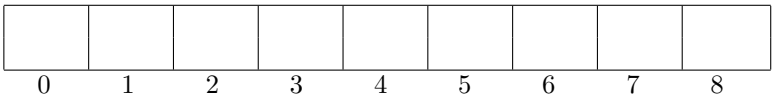
Solution:
The best answer is that it will cause a collision between *any* two numbers that have a `realPart` or an `imaginaryPart` of 0. In particular, all numbers with no imaginary part will hash to the same integer. Significant partial credit for multiplication being commutative (so $2 + 3i$ and $3 + 2i$ would collide) or for the fact that the cast to `int` rounds down, ignoring all the information "after the decimal point."

Name:_____

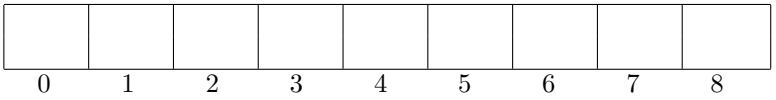
9. (**13** points)

Consider inserting data with integer keys 20, 18, 11, 47, 36, 19 in that order into a hash table of size 9 where the hashing function is `h(key) % 9`.

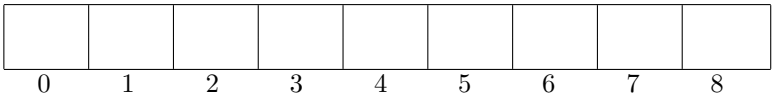
- Show a chaining hash table after doing the insertions:



- Show an open addressing with linear probing hash table after doing the insertions.



- Show an open addressing with quadratic probing hash table after doing the insertions.



Solution:

(a) At index 0: a list holding 36 and 18
At index 1: a list holding 19
At index 2: a list holding 47, 11, and 20

(b) `18 36 20 11 47 19 X  X X`

(c) `18 36 20 11 X  19 47 X X`