READ THIS NOW!

- Print your name in the space provided below.
- Unless a question involves determining whether given C++ code is syntactically correct, assume that it is. Unless a question specifically deals with compiler #include directives, you should assume the necessary header files have been included.
- There are 8 short-answer questions, priced as marked. The maximum score is 100.
- When you have finished, sign the pledge at the bottom of this page and turn in the test and your fact sheet.
- Aside from the allowed one-page fact sheet, this is a closed-book, closed-notes examination.
- No laptops, calculators, cell phones or other electronic devices may be used during this examination.
- You may not discuss this examination with any student who has not taken it.
- Failure to adhere to any of these restrictions is an Honor Code violation.

Name (Last, First) _____

printed

Pledge: On my honor, I have neither given nor received unauthorized aid on this examination.

_____

signed

1. It is a fact that for all $n \geq 1,\ 2^n \leq 2n!$.

   a) [10 points] Prove the fact stated above.

proof: If n = 1, then 2^n = 2 and 2n! = 2x1! = 2, so the result holds.

Suppose that for some integer k >= 1, 2^k < 2k!. Then:

$$
\begin{aligned}
2^{k+1} &= 2 \cdot 2^k && \text{factoring} \\
&\leq 2 \cdot 2k! && \text{inductive hypothesis} \\
&\leq (k+1)2 \cdot k! && \text{since k >= 1, k+1 >= 2} \\
&= 2 \cdot k!(k+1) && \text{commutativity of multiplication} \\
&= 2(k+1)! && \text{definition of factorial}
\end{aligned}
$$

Hence, for all n >= 1, we have that 2^n <= 2n!.

   b) [10 points] Assuming the fact stated above is correct, what does <u>that fact</u> imply regarding any big-O, big-Ω and/or big-Θ relationships between the functions 2^n and $n!$?

Two things are implied:

- **2^n is O(2n!)**
- **2n! is Ω(2^n)**

There is no implication of a Θ-relationship between the two functions. In fact, they don't have a Θ-relationship.

2. Consider the following algorithm for computing the sum of two matrices:

```
for (i = 0; i < N; i++) {                  // 1 before, 2 each pass, 1 on exit
   for (j = 0; j < N; j++) {               // 1 before, 2 each pass, 1 on exit
      sum[i][j] = a[i][j] + b[i][j];       // 8 operations
   }
}
```

a) [10 points] Using the rules given in class for counting operations, find a simplified function T(N) that counts the number of operations the algorithm would perform.

$$T(N) = 1 + \sum_{i=0}^{N-1}\left(2 + 1 + \sum_{j=0}^{N-1}(2+8) + 1\right) + 1$$

$$= 2 + \sum_{i=1}^{N}\left(4 + \sum_{j=1}^{N} 10\right)$$

$$= 2 + \sum_{i=1}^{N}(4 + 10N)$$

$$= 2 + 4N + 10N^2$$

b) [5 points] What big-Θ complexity class does your answer to the previous part belong to? (No proof is necessary.)

Θ(N^2)

3. [10 points] Using any applicable theorem or definition covered in class, prove or disprove the following statement:

$$f(n) = \frac{n^3}{\log n} \text{ is } O\left(n^3\right) \text{ but not } \Theta\left(n^3\right)$$

First of all, observe that:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\frac{n^3}{\log n}}{n^3} = \lim_{n\to\infty} \frac{1}{\log n} = 0$$

By the Corollary to Theorem 8, this implies that f is O(g) but f is not Θ(g).

4. [15 points] Recall that the access time for a hard drive is the sum of three separate measures of performance. Suppose that an existing hard drive design was modified by making the following two changes:

- the rotational speed is increased by 30%
- the number of tracks on each surface is increased by 20%, with a corresponding increase in the radius of the platters

No other characteristics of the original drive design are changed, aside from the obvious increase in the storage capacity. Describe what effect these changes, if any, will have on each of the three measures, and why.

Measure	Effect
Seek time	**increased, because seek time is largely determined by the average number of tracks that must be crossed and that has been increased by 20%**
Latency	**decreased, because average latency is half of the time for the platter to revolve, and that has been decreased by 30%**
Transfer time	**decreased, for the same reason as thelatency**

5. [10 points] Suppose that a web-hosted database application stores its data in a single file, which is just a large collection of GIS records (as used in the 2606 projects so far). The records in the file are sorted into ascending order by their feature identifiers (FIDs). Lots of clients use this application, looking up geographic features that they find interesting for a variety of typical reasons.

   Discuss whether, and why, the application is likely to exhibit spatial locality in its accesses of the data file.

   It is not likely that the application will exhibit much spatial locality in its file accesses. The only way that spatial locality would occur would be if users tended to access records that have nearly-equal FIDs. But the FID is just a serial number for the record and bears no relation to the attributes of the geographic feature, and it does not seem likely that users would be curious about the FIDs themselves.

   Discuss whether, and why, the application is likely to exhibit temporal locality in its accesses of the data file.

   The application will exhibit temporal locality if users tend to repeat accesses to the same records. This is actually rather likely; for example, current events may spur widespread interest in certain geographic features, leading to the phenomenon that many users will request the same records.

6. [10 points] Why should a container implemented as a C++ template provide a `const` version of its `Find()` function? A sample of possible client code would be useful to illustrate your discussion.

   If an object is declared as `const`, then only `const` methods may be invoked on it.

   This situation is quite likely to arise when a container is used, because in C++ container objects are typically passed either by reference or by constant reference, in order to avoid the cost of copying the entire container (not to mention the data objects it organizes).

   Note: this has absolutely nothing to do with the issue of the client modifying data objects within the container. In almost every case, the data objects are not physical elements of the container itself, and `const` member functions are only forbidden to change non-mutable data members of the objects on which they are invoked. For example, a tree object probably has only one data member, a pointer to the root node, and that is the only thing that a `const` member function of the tree implementation is not allowed to modify. In particular, there is absolutely no reason that a `const Find()` function in a tree cannot return an entirely non-`const` pointer to a data object.

7. [10 points] Explain the rationale for using an inheritance hierarchy for the nodes when implementing a PR quadtree; address the issue of why one would not derive the leaf type from the internal type, or vice versa (but not only that issue).

Internal nodes only need to store pointers to other nodes, while leaf nodes only need to store single data objects. Since it would be wasteful of memory to use a single node type, this argues in favor of having two different kinds of nodes, say `Leaf` and `Internal`. (That, however, does not automatically require the use of inheritance.)

The pointers in an internal node may point to other internal nodes or to leaf nodes. If we disregard the somewhat silly and wasteful possibility of having 4 `Leaf` pointers and 4 `Internal` pointers in each internal node, we need a single pointer type that can target both kinds of nodes and provide us with some way to determine the actual kind of node at runtime. This implies that we need to put `Leaf` and `Internal` into an inheritance hierarchy so we can use base-type pointers and `dynamic_cast`.

However, since `Leaf` and `Internal` objects have no common data members, we cannot derive one from the other without adding unnecessary data members to the subtype.

8. [10 points] Recall the definition of the AVL tree, and the rotation operations that may be used to re-balance the tree after an insertion occurs.

If you have a large AVL tree, will it be always necessary to perform rotations after every insertion that increases the height of the tree. Explain.

In an AVL tree, rotations are needed after an insertion only if the insertion resulted in a node having subtrees whose heights differ by more than 1. That will not necessarily occur.

Exactly what do the AVL rotations do that results in an improvement over a simple BST?

The rotations after an insertion reduce the height of a subtree (relative to what it would have been in a plain BST). That is what leads to the AVL typically having fewer levels than a BST created by inserting the same elements in the same order.