The Midterm Exam was given in class on Wednesday, October 21, 2009.

1. [4 pts] When we define a (non-template) class, we generally use **two kinds** of files. Indicate what each of these files is called and what it is for.

   - **Header.** Holds the class definition and prototypes of associated global function. Primarily for the interface of the class.
   - **Source.** Holds definitions of member functions and global functions. Primarily for the implementation of the class.

2. [4 pts total] **SRP.**
   2a. [2 pts] What is the Single Responsibility Principle?

   Every module should have exactly one well defined responsibility.

   2b. [2 pts] Why is following SRP a good thing?

   It makes code easier to design, and easier to understand and maintain. It also improves error-handling capability.

3. [4 pts] What important **distinctions** between arrays and Linked Lists do we need to consider when analyzing the efficiency of algorithms that deal with them? (Give at least **two**.)

   - Arrays are random-access, while Linked Lists are not. That is, in an array one can quickly access an item by its index. In a Linked List, this is slow.
   - Inserting and deleting at a given position is much faster in a Linked List than in an array.
4. [6 pts total] **The Big Three.**
   4a. [3 pts] What does the C++ “Law of the Big Three” state?

   If you declare one of the Big Three (copy constructor, copy assignment operator, destructor), then you should probably declare all of them.

   4b. [3 pts] In what circumstances do we usually need to write “the Big Three”?

   We usually need to write the Big Three when we an object manages a resource.

5. [6 pts total] **Order of Functions.** Write the order of each function below, using big-$O$.
   5a. [3 pts]

   ```cpp
   void func_a(int arr[], int n) // n is size of arr
   {
     cout << "Item 0 = " << arr[0] << endl;
     cout << "Item 1 = " << arr[1] << endl;
     cout << "Item 2 = " << arr[2] << endl;
   }
   
   **Order (big-$O$):** $O(1)$.
   
   5b. [3 pts]

   ```cpp
   void func_b(int arr[], int n) // n is size of arr
   {
   for (int a = 0; a < n; ++a)
     for (int b = 0; b < a; ++b)
       for (int c = 0; c < b; ++c)
         arr[c] += arr[b];
   }
   ```

   **Order (big-$O$):** $O(n^3)$.

(exam page 3)
6. [6 pts total] **Resource Ownership.**
   6a. [3 pts] In the context of this class, what does it mean to “own” a resource?

   To *own* a resource is to be responsible for releasing it.

   6b. [3 pts] Give two examples of resources that might be owned, and what it means to release each one.

   *Here are three:*
   
   - Dynamically allocated memory. Freeing the memory.
   - A dynamic object. Destroying the object and freeing its memory.
   - An open file. Flushing and closing the file.

7. [6 pts total] **Recursion vs. Iteration.**
   7a. [4 pts] Recursion has a number of disadvantages, as contrasted with iteration. Give at least two of these.

   *Here are three:*
   
   - Recursion has more function-call overhead.
   - Recursion has poor error-handling in the case of stack overflow.
   - Recursion can easily (but not always!) lead to highly inefficient algorithms.

   7b. [2 pts] Under what circumstances is it very easy to convert recursive code to iterative form?

   When the code uses tail recursion.
8. [10 pts total] **Order of Operations.** In each part, indicate the order of the given operation, using big-O and, based on this, in words. Assume that a good algorithm is used, within the constraints given. Use n to denote the size of the input.

8a. [2 pts] Find an item with a given value in a sorted array.
**Big-O:** $O(\log n)$.

**In Words:** Logarithmic time.

8b. [2 pts] Print the middle item in an array of known size.
**Big-O:** $O(1)$.

**In Words:** Constant time.

8c. [2 pts] Print the middle item in a Linked List of known size.
**Big-O:** $O(n)$.

**In Words:** Linear time.

8d. [2 pts] Sort an array with Quicksort.
**Big-O:** $O(n^2)$.

**In Words:** Quadratic time.

8e. [2 pts] Sort a Linked List.
**Big-O:** $O(n \log n)$.

**In Words:** Log-linear time.

9. [4 pts] One strategy for dealing with a possible error condition in a function is to signal the client code that an error has occurred. What are two other strategies? **Hint:** These must **not** involve signaling the client code.

- Prevent the error, using preconditions.
- Contain the error, by fixing it inside the function.
10. [9 pts total] **Properties of Algorithms.**

10a. [3 pts] What does it mean for an algorithm to be “**scalable**”?

An algorithm is *scalable* if it works well with increasingly large problems.

10b. [3 pts] What does it mean for an algorithm to be “**in-place**”?

An algorithm is *in-place* if it does not require significant additional space, that is if it uses only constant additional space (beyond that required for its input).

10c. [3 pts] What does it mean for an algorithm to be “**stable**”?

An algorithm that rearranges a list (e.g., a sorting algorithm) is *stable* if it never changes the relative order of equivalent items.

11. [4 pts] The Introsort algorithm is, for many purposes, the fastest sort known. However, in certain situations, some other algorithm is faster. List **two** such situations. For each, say which *sorting algorithm* would be faster in that situation.

*Here are three:*

- Sorting a small list. Insertion Sort.
- Sorting a nearly sorted list. Insertion Sort.
- Sorting a Linked List. Merge Sort.
12. [12 pts total] **Impossible?** Your friend Egbert is known for making wild claims. In each part, Egbert claims he has a new algorithm. Indicate whether Egbert claim is possible or impossible (circle one). If it is possible, explain **how** to do it. If impossible, explain **why**.

12a. [4 pts] Egbert says, “I have a new algorithm that can find the largest number in any given unsorted array, in logarithmic time.”

**POSSIBLE**

Such an algorithm must read all of its input, which requires at least linear time.

12b. [4 pts] Egbert says, “I have a new algorithm that can sort a list in linear time, assuming that each item’s position is no more than 1000 away from its position in the final sorted list.”

**POSSIBLE**

This is nearly sorted data. Insertion Sort can do this.

12c. [4 pts] Egbert says, “I have a new algorithm that can sort any list in linear time, given only an appropriate comparison function.”

**POSSIBLE**

Egbert is saying that he has a linear-time general-purpose comparison sort. However, we know that such sorts always require at least log-linear time.