CIS 291 – Data Structures in C++ - Spring 2005
Week 4 Lab Exercise

Week 4 Lab Exercise due: Tuesday, February 8th, END of lab

Purpose: thinking/experimentation related to sequential and binary search

Consider the number of comparisons in sequential search and binary search. Obviously, for large numbers of elements, binary search is the winner in terms of number of comparisons. But, for smaller numbers of elements, this is not nearly as clear.

Answer the following questions on a piece of paper individually. Then, compare and discuss your answers with at least one other class member. Then, write your name on the "Next:" list to get your work checked over.

1. Consider the (woefully under-documented) **count_seql.cpp** available from the course web page. (It is woefully under-documented to show that it would be nice if better documentation were there, and so as to force you to read the actual code, for class purposes (which you normally would not force a "normal" user of your code to do, of course).)

(a) How does this differ from the approach described in lecture?

(b) What is being done to/with the last parameter?

Run **test_count_seql.cpp**, and look at the results.

(c) Within **test_count_seql**, why is the **count** variable reset before each call to **count_seql**?

2. Now consider the (woefully undocumented) **count_bin.cpp**, also available from the course web page.
(a) How does this differ from the implementation described in lecture?

(b) Run **test_count_bin.cpp**, and look at the results. Are these what you expected? Why or why not?

3. Now consider the (woefully undocumented) **count_combo.cpp**, also available from the course web page.

How does this differ from **count_bin.sql**?

4. Run **test_count_combo.cpp**; note that you will need to type both **count_seql.cpp** and **count_combo.cpp** (or their **.o** versions) in the linking call of **g++** that creates the executable **test_count_combo**.

Experiment with different values of **SMALL_ENOUGH**; make a small table recording, for each value that you try:
* the value that you tried for SMALL_ENOUGH,
* what value you searched for (that was NOT in the array),
* how many comparisons resulted in that search,
* what value you searched for (that WAS in the array, but not in the middle), and
* how many comparisons resulted in that search.

Keep experimenting until you have found what you believe to be the optimal value for **SMALL_ENOUGH** (to minimize the number of comparisons needed); I'll expect to see at least 3 different SMALL_ENOUGH values attempted, and probably more.

5. Recall the "shortcuts" possible for sequential search (as described in lecture).
(a) Considering how we are using **count_seql** within **count_combo**, would adding those shortcuts reduce the number of comparison overall in **count_combo**? Why or why not? (Remember, if you were to add comparisons, you would need to increment the comps_so_far accordingly.)

(b) Assume that you were to be using **count_seql** by itself to search possibly thousands of items. Would adding the shortcuts be likely to reduce the number of comparisons overall then?

To receive credit for this lab exercise, the above must be completed by the end of the lab period.