Programming Assignment 1

Part I (due 10/09/09)

1. Design from scratch your own function merge and add it to the C++ code stl_merging.cpp discussed in class. You need to call your function merge right after the call to STL merge and then output your results to compare them with the output of STL merge. The output of your program should look as follows:

   The result of merging by standard STL merge:
   *The result of merging by STL merge should be printed out here.*

   The result of merging by my implementation of merge algorithm:
   *The result of merging by your implementation of merge should be printed out here.*

2. Remove outputs from your solution of problem #1 and place instructions for timing the execution of both merge procedures. The new output of your program should look as follows:

   It took … milliseconds for STL merge to merge lists of sizes … and … .

   It took … milliseconds for my merge to merge lists of sizes … and … .

In problems 3,4 you will need to implement the quick-sort algorithm (textbook page 33) and the merge-sort algorithm based on linked lists discussed in section 2.8 of Foundations of Computer Science. In order to get the second algorithm adapt the C code available at:

http://www-db.stanford.edu/~ullman/fcsccode/fig2.31.txt

to C++.

3. Write a C++ program, which performs the following tasks:
   - Creates two copies of a random array of the size specified by the user.
   - Runs quick-sort on the first copy of the random array.
   - Runs merge-sort based on linked lists on the second copy.
   - Prints the results of sorting by each method. The output of your program should look as follows:

   The results of sorting:

   by quick-sort
   *…*,

   by merge-sort based on linked lists
   *…*

4. Remove outputs from your solution of problem #3 and place instructions for timing the execution of both sorting procedures. Modify your recursive functions in such a way that the number of recursive calls and the depth of the tree of
recursive calls are computed automatically (consider only recursive calls of sorting algorithms). The new output of your program should look as follows:

The results of sorting a random array of size … are:

… milliseconds by quick-sort sort,
The depth of the tree of recursive calls: …,
The number of recursive calls: ….

… milliseconds by merge-sort based on linked lists,
The depth of the tree of recursive calls: …,
The number of recursive calls: ….

Part II (due 10/16/09, may submit on 10/09/09)

In problems 5,6 you will need to implement a non-recursive version of merge-sort algorithm. Arrange two nested loops to accomplish this task. The outer loop should provide the size of segments for merging. The inner loop should take care of selecting positions of segments. The inner loop should start at the left edge and move your segments to the right. Arrange appropriate values of variables left, middle, right, so that sorting is accomplished just by iterating the call merge(a, left, middle, right).

5. Write a C++ program, which performs the following tasks:
   - Creates five copies of a random array of the size specified by the user.
   - Runs STL sort on the first copy of the random array.
   - Runs two versions of recursive merge-sort on copies number two and three. The first version of merge-sort should be based on the STL merge and the second one on your implementation of merge operation.
   - Runs two versions of non-recursive merge-sort on copies number four and five. Again the first version should be based on the STL merge and the second one on your implementation of merge operation.
   - Prints the results of sorting by each method. The output of your program should look as follows:

The results of sorting:

by STL sort
…,
by recursive merge-sort with STL merge
…,
by recursive merge-sort with my merge
…,
by non-recursive merge-sort with STL merge
..., by non-recursive merge-sort with my merge
...

6. Remove outputs from your solution of problem #5 and place instructions for timing the execution of all five sorting procedures. The new output of your program should look as follows:

The results of sorting a random array of size ... are:

... milliseconds by STL sort,
... milliseconds by recursive merge-sort with STL merge,
... milliseconds by recursive merge-sort with my merge,
... milliseconds by non-recursive merge-sort with STL merge,
... milliseconds by non-recursive merge-sort with my merge.

7. Continue experiments with random trees started in random_trees_tests. Substitute the generator of random numbers by a generator of random permutations. You may use STL algorithm random shuffle in order to generate random permutations. After insertion of all items compute the height of resulting tree, the total search time, the logarithm (base 2) of the number of inserted nodes and display them on the screen.

8. (Extra credit, 10 points) Rewrite the C++ code for evaluating arithmetic expressions from section 10.5 of Big C++ in Perl.

9. (Extra credit, 20 points) Remove recursion from the linked list implementation of merge sort algorithm of problems 3,4 and time the resulting code against the original recursive merge sort based on linked lists. You need to remove recursion from three functions merge, split and MergeSort.

Grading:
problems 1,2 together – 20 points,
problems 3,4 together – 25 points,
problems 5,6 together – 30 points,
problem 7 – 25 points.
Total: 100 points

Report:
Print out the code of your implementations of the following algorithms: merge operation, non-recursive merge-sort, quick-sort and merge-sort with automatic computation of the number of recursive calls and the depth of the tree of recursive calls. For each of the problems 2,4,6,7 report a series of representative experimental results in a form of a table, analyze them and draw conclusions. It might be helpful to review section 11.5 of Big C++ in order to get ideas on how numerical results should be displayed and analyzed.
Please keep in mind that the target of this assignment is to implement selected algorithms and then evaluate their relative performance.

**Presentation Rules:**
You need to submit your report before starting the presentation of the code. You will need two windows for each program (once you establish connection with the server it is enough to click the new terminal widow button to open another session). In one of them you will display the source code and in the other you will execute the code and display the output.