

# Project 2

CS 3343 – Fall 2006  
Tom Bylander, Instructor

assigned October 2, 2006  
due October 31, 2006

Your code and report must be submitted in a zip file to WebCT. All code and analysis should be your own. Late reports are accepted with 50% off. Waiting until the last minute and encountering a technical problem with your computer or WebCT means that your report will be late even if you emailed to me or placed it under my door. Plan accordingly.

The goal of this project is to perform an empirical analysis of three  $\Theta(n \log n)$  sorting algorithms: mergesort, quicksort, and heapsort. Your program can be in C, C++, or Java, and should be straightforward translations from the pseudocode in the book except that you may modify mergesort to be more space-efficient. Check with me in advance if you want to use some other programming language.

I would prefer that your report be submitted as a PDF or Postscript file, or a group of linked HTML files. Submitting as a Word or OpenOffice document is ok though. I want to see the report as a single document. I do not want to open plots separately.

The input to your programs should be arrays of random numbers. Your code needs to count the number of basic operations (comparisons) and to record the time used to perform the sorting. To get reasonable averages for a given number of elements, you should run an algorithm several times for that number of elements, making sure you use different random values each time. You need to choose many different numbers of elements (at least 10, better to have 20).

Your code should also test that the algorithms are correct. For sorting, the numbers should be in ascending order and should be the same numbers you started with. Testing the algorithms should be excluded in the timings.

Your report should summarize your code: what files are included, what each method or function does, and how to invoke your program.

Your report should include two plots: the number of comparisons by the algorithms, and the timings for the algorithms. Do not produce different plots for different algorithms; it's very hard to compare across different plots. Each plot should include an  $n$ -log- $n$  curve for each algorithm generated from the data. You should be able to modify the code that performs a quadratic fit into code that performs an  $n$ -log- $n$  fit.

Your report should describe each plot and the conclusions from the results. For example, which algorithm is better? What simple quadratic function appears to correspond to the basic operation count? Does the basic operation count correspond to the timings?

## **Addendum: How to Efficiently Implement Testing**

In this addition, I will describe how to use hash tables to test whether the sorted array contains the same numbers you started with.

- Insert each of the original numbers into a hash table. Be sure that duplicate numbers result in duplicate numbers in the hash table.
- Create an additional Boolean array the same size as the hash table.
- After sorting, for each number in the sorted array, search for it in the hash table. If found, mark the Boolean array, else the sorting algorithm has an error. In case of duplicate numbers, be sure to search for an unmarked number in the hash table.
- Unmark the whole Boolean array so the same hash table can be used for multiple sorting algorithms.

This should result in an  $O(n)$  algorithm for testing a sorted array.

## **Addendum: Random Numbers**

I am not sure why some of you want to generate random numbers in a small range such as from 1 to 100. This will affect the result for larger sorts. If you are using Java, use a Random object and the values from the nextInt or nextDouble method without modification. If you are using C, use the values from rand without modification.