String Pattern Recognition

Project 1

Theoretical and Experimental Comparisons of Sorting Algorithms

Paco Gómez Martín

DEADLINE: October 23rd 2009, 23:59 p.m.

1 Introduction

This project consists of comparing several sorting algorithms both from a theoretical and experimental point of view. Goals of this project include:

- Acquisition of knowledge about sorting algorithms.
- Practice with recursive and iterative algorithms.
- Actual programming of sorting algorithms.
- Conducting a computational experiment.
- Comparison of theoretical and experimental results.
- Interpretation of results and presentation of conclusions.
- Writing an academic paper.

2 Sorting Algorithms

The following algorithms must be implemented:

1. **Minimum-selection sort.** Given an array of $M[1..n]$, at step j this algorithms selects the minimum element of $M[j..n]$ and places it in position j. After j steps the array is sorted.
2. **Insertion sort**. Seen in class.
3. **Quicksort**, recursive version. Check the references given in the course notes or simply check the web for further information.
4. **Quicksort**, iterative version.
5. **Mergesort**, recursive version. Seen in class.
6. **Mergesort**, iterative version. Seen in class.

3 Time and Space Complexities

For each of the algorithms, time and space complexities have to be deduced. This comprises setting up the equations and solving them. Bounds in terms of O-notation should be exhibited and, where possible, Ω-bounds, too. Insight in the analysis of algorithms will be much appreciated.

4 Experiments

Experiments have to be carried out according to the following directions.

1. Each algorithm has to be run on test data. Data will be generated in a random way.
2. Each algorithm has to sort n data, where n varies from 100 to $10,000$ and increases by 100 each time (that is, $n = 100, 200, \ldots, 10,000$).
3. The range of the data is the same as the number of data. For example, the first test data consist of 100 data in the range or $[1, 100]$; the second test data is composed of 200 data in the interval $[1, 200]$, and so forth.
4. If necessary, an algorithm may be run on test data a fixed number of times so that the running times obtained are meaningful (see Exercise 1 in the course notes).
5. Running times will be plotted against input size (again, see Exercise 1).
6. From the plots, the input size for which an algorithm is faster than another will be estimated.
7. Functional fits will be computed as done in Exercise 1 in the course notes. This can be accomplished by using a mathematical software such as StatGraphics, Derive or Maple. Should students need help at this point, they should contact me.
8. From the fitting functions estimate the constants associated with the complexity. Compare with the theoretical complexities obtained above.

5 Programming

Implementation of algorithms may be done in any language of student's choice. However, the language and its compiler should support certain features in order to be able to run the experiments properly. The choice of C, C++, Maple or the like should be enough. Source code and a .exe file have to be handed over.

6 Written Paper

A paper describing the following points must be handed over.

- Brief explanation of the algorithms.
- Brief explanation of the implementations. It can be done by including sufficiently detailed comments in the code.
- Brief description of the experiment.
- Interpretation of experimental data. Comparison of experimental data with theoretical complexities.
- Conclusions. In this section students must answer the following questions:
  1. What algorithm is best and under what circumstances?
  2. Classify the algorithms in terms of input size.
  3. Draw your own conclusions (be creative, but not extravagant or too showy).

The paper has to be written in correct Spanish or English; it also has to possess clarity of thought. Show me what you know; do not force to search for it through a poorly written paper.

7 Grading

The whole project counts 25% (2.5 points out of 10) of your final grade. I will take points off when:

- There is spelling mistakes (either in Spanish or in English, but I will be tougher if they occur in Spanish).
- It is plenty of irrelevant material. Down with the irrelevant!
- It lacks clarity of thought.
- It is lengthy, long-winded or poor in content.
- Code is not properly commented.
- Code is not properly structured.
- Variables have absurd names.
- There are run-time errors.

8 Questions and Office Hours

I am willing to answer your questions about algorithms, complexity or the experiment. I will not answer questions about coding errors as it is my feeling that, at this point, writing error-free code is your responsibility.