Course COMPSCI.220FT – (2002)

Assignment 1 (Algorithm Analysis)

Due date: Tuesday, 26th March 2002, 8:00 pm.

Objectives

- Learn how to use the "Big-Oh" notation and basic recurrences to analyse algorithm performance.
- Experimentally estimate performance of sorting algorithms.

The course webpage is at `http://www.cs.auckland.ac.nz/compsci220ft`

Requirements

1. (*15% of marks*) You have developed a divide-and-conquer algorithm and found that its running time $T(n)$ for n data items is described by the following recurrence relation:

   $$T(n) = k \cdot T\left(\frac{n}{k}\right) + c \cdot k \cdot n; \;\; T(1) = 0.$$

   Use a telescoping technique to find a closed form formula for $T(n)$ in terms of c, n, and k. What is the computational complexity of this algorithm in a "Big-Oh" sense? Explain using the derived formula.

   Hint: To have the recurrence well-defined, assume that $n = k^m$ with the variable integer m and constant integer k (note that $m = \log_k n$).

   The recurrence $T(k^m) = k \cdot T(k^{m-1}) + c \cdot k^{m+1}$ telescopes as follows:

   $$\begin{array}{rcl} \frac{T(k^m)}{k^{m+1}} & = & \frac{T(k^{m-1})}{k^m} + c \\ \frac{T(k^{m-1})}{k^m} & = & \frac{T(k^{m-2})}{k^{m-1}} + c \\ \cdots & \cdots & \cdots \\ \frac{T(k)}{k^2} & = & \frac{T(1)}{k} + c \end{array}$$

   Therefore, $\frac{T(k^m)}{k^{m+1}} = c \cdot m$, or $T(k^m) = c \cdot k^{m+1} \cdot m$, or $T(n) = c \cdot k \cdot n \cdot \log_k n$. The complexity is $O(n \log n)$ because k is constant and the base of logarithm is of no importance.

2. (*5% of marks*) What value of $k = 2$, 3, or 4 should you choose to get the fastest processing?

   Hint: note that $\log_k n = \frac{\ln n}{\ln k}$ where ln denotes a natural logarithm (with the base $e = 2.71828\ldots$).

   The processing time $T(n) = c \cdot k \cdot n \cdot \log_k n$ can be easily rewritten as $T(n) = c\frac{k}{\ln k} \cdot n \cdot \ln n$ to give an explicit dependence of k. Because $\frac{2}{\ln 2} = 2.8854$, $\frac{3}{\ln 3} = 2.7307$, and $\frac{4}{\ln 4} = 2.8854$, the fastest processing is obtained for $k = 3$.

3. (*15% of marks*) Suppose you are a programmer in a big company which constantly needs to process very big data bases, containing each up to $n = 10^9$ records. Your manager asks you to suggest which one of the two software packages, **A** or **B**, of the price \$10,000 and \$20,000, respectively, should be bought to ensure the fastest processing.

   You have analysed processing algorithms implemented in the packages **A** and **B** and you have found that they are of the "Big-Oh" complexity $O(n \log n)$ and $O(n)$, respectively. But you are not going to recommend your manager to choose the package **B** before you conduct your own tests. You have found that the average time of processing $n = 10^4$ data items with the package **A** is 100 milliseconds and with the package **B** is 500 milliseconds. Work out exact conditions (in terms of the database size n) when one package actually outperforms the other one and recommend the best choice to your manager.

   The processing time of each package is $T_{\mathbf{A}}(n) = c_{\mathbf{A}} n \log n$ *and* $T_{\mathbf{B}}(n) = c_{\mathbf{B}} n$. *The tests allow us to derive the numerical values of the coefficients:*

$$\begin{aligned} c_{\mathbf{A}} &= \frac{100}{10^4 \log 10^4} &= \frac{1}{400 \log 10} \\ c_{\mathbf{B}} &= \frac{500}{10^4} &= \frac{1}{20} \end{aligned}$$

   To find out when the package **B** *begins to outperform* **A**, *we must estimate the data size* n_0 *that ensures* $T_{\mathbf{B}}(n) \le T_{\mathbf{A}}(n)$, *that is,* $\frac{n}{20} \le \frac{n \log n}{400 \log 10}$ *for* $n \ge n_0$. *The desired data size such that for all the larger sizes* $T_{\mathbf{B}}(n) \le T_{\mathbf{A}}(n)$ *is as follows:* $\log n_0 = \frac{400}{20} \log 10$, *or* $n_0 = 10^{20}$. *Thus for processing data bases of sizes* $n \le 10^9$, *the package of choice is* **A** *in spite of its larger complexity in the "Big-Oh" sense.*

4. (*5% of marks*) Work out the computational complexity of the following piece of code assuming that $n = 2^m$:

```
for( int i = n;  i > 0;  i-- ) {
  for( int j = 1;  j < n;  j *= 2 ) {
    for( int k = 0;  k < j;  k++ ) {
         ... // constant number C of operations with i, j, k
    }
  }
}
```

   The outer `for`*-loop goes round* n *times. For each* `i`, *the next loop goes round* $m = \log_2 n$ *times, because of doubling the variable* `j`. *For each* `j`, *the innermost loop by* `k` *goes round* `j` *times, so that the two inner loops together go round* $1+2+4+\ldots+2^{m-1} = 2^m - 1 \approx n$ *times. Loops are nested, so the bounds may be multiplied to give that the algorithm is* $O\left(n^2\right)$.

5. (*45% of marks*) Write a program `benchmark.java` that performs a series of benchmarking tests to see which sorting method, `mergeSort`, `heapSort`, or `quickSort`, is faster. The methods are given in the COMPSCI.220FT coursebook (Chapter 1, Section 1.2) and can be downloaded from the course web page ("Lectures", Part 1: Georgy's lecture notes). But you may choose any other available implementation.

   Your tests should include arrays of very "random" integers in the range $[0 \ldots 1,000,000]$. The size n_{test} of array should be an input parameter of your program, and you have to choose an appropriate size of the array for testing all three methods on the available PCs such that the average time of sorting one array of the chosen size was about 0.05 - 0.1 second.

The program should repeat sorting several times (with different randomly filled arrays of the same size n_{test}) in order to find and output for each method the average time, the standard deviation, and the range of the sorting time.

Let k denote the serial number of a test. Let K be the total number of the tests. Let $T_{\text{sort},k}$ be the sorting time for the test k using a particular sort. Then the average time A_{ps}, standard deviation S_{ps}, and the range of the time $[T_{\text{ps,min}}, T_{\text{ps,max}}]$ are computed as follows

$$\begin{aligned}
A_{\text{ps}} &= \frac{1}{K}\sum_{k=1}^{K} T_k \\
S_{\text{ps}} &= \sqrt{\frac{1}{K}\sum_{k=1}^{K} \left(T_{\text{ps},k} - A_{\text{ps}}\right)^2} \\
T_{\text{ps,min}} &= \min\{T_{\text{ps},1}, \dots, T_{\text{ps},K}\} \\
T_{\text{ps,max}} &= \max\{T_{\text{ps},1}, \dots, T_{\text{ps},K}\}
\end{aligned}$$

To fill arrays, you may use the random generator `java.lang.Math.random()`. It provides `double` output values that you should convert into integers in the range $[0 \dots 1,000,000]$.

To measure processing time you can use the `System.currentTimeMillis()` method, for instance,

```
long start = System.currentTimeMillis();
sortingMethod( arrayToSort );
long runTime = System.currentTimeMillis() - start;
```

The actual times depend on a computer and OS. Typically, the average time for Quicksort, $T_{\text{quicksort}}$, is about 2–2.5 times less than $T_{\text{mergesort}}$ which is only a bit less than T_{heapsort}. But depending on implementation, the time-increasing order of "quick – heap – mergesort" may be encountered, too. Typical standard deviations are less than 5–10% of the average time.

6. (*15% of marks*) Using the above array size n_{test} and average times of its sorting with mergesort, heapsort, and quicksort, $\{A_{\text{ps}} \ : \ \text{ps} = \text{mergesort, heapsort, quicksort}\}$, derive explicit formulas for computing the expected average time of sorting an array of a given size by each method. Compare the expected values to the experimental results obtained by running the program `benchmark.java` for the array of the size $n = 2 \cdot n_{\text{test}}$.

The sorting methods under consideration are of $O(n \log n)$ complexity. The above average times allow to derive the following formulas for the processing time of these methods:

$$T_{\text{ps}}(n) = \frac{A_{\text{ps}}}{n_{\text{test}} \log n_{\text{test}}} n \log n$$

so that for $n = 2 \cdot n_{\text{test}}$ the expected time is:

$$T_{\text{ps}}(2 \cdot n_{\text{test}}) = A_{\text{ps}} \frac{2 \cdot n_{\text{test}} \log(2 \cdot n_{\text{test}})}{n_{\text{test}} \log n_{\text{test}}} = 2 \cdot A_{\text{ps}} \cdot \left(1 + \frac{2}{\log n_{\text{test}}}\right)$$

Dates and Marks

Worth:	8.33% of your total course grade
Handout:	Friday, March 8, 2002
Due:	Tuesday, March 26, 2002, 8:00pm

Bonus and penalty scheme

- Submitted before 8:00pm Sunday March 24: 5% bonus
- Submitted before 8:00pm Monday March 25: 2% bonus
- Submitted before 8:00pm Tuesday March 26: no bonus
- Submitted before 8:00pm Wednesday March 27: 20% penalty
- Submitted before 8:00pm Thursday March 28: 50% penalty
- Submitted after 8:00pm Thursday March 28: your assignment will not be accepted

If you need an extension you will need to talk to your course administrator **BEFORE** the due date.

Feel free to make multiple submissions before the due date. Only the last assignment submitted will be marked.

Submit your assignment using the web-based assignment dropbox system. You are to electronically hand in

- an ASCII text file called `answers1.txt` which should contain the answers to the questions. The printout of your program `benchmark.java` should be also included to this file as the part of answer to Questions 5 and 6. You can prepare this file in any text editor, e.g. Notepad.
- the source file of your Java application program `benchmark.java` and the resulting byte code file `benchmark.class`

KEEP YOUR SUBMISSION RECEIPT. IT IS YOUR ONLY PROOF OF SUBMISSION.