

Coursework: Divide and Conquer

Software Engineering - Algorithms
Department of Computing
Imperial College London

27 February 2014

Objectives

The aim of this coursework is to enhance your algorithmic skills by mastering the divide and conquer strategy. You are asked to show that you can

- solve recurrences with different methods introduced in the course,
- implement divide and conquer solutions for given problems,
- compare naive and advanced implementations of algorithms solving the same problem.

All implementations will be done in C++ and skeleton files are provided which contain placeholders where you are supposed to insert your code. Solving recurrences will require you to write down some math, and the comparisons of different algorithms involve textual descriptions and graphical plots. For those tasks you will have to insert your solutions in a provided Latex skeleton file. From the latex file you will have to generate a PDF document.

Latex provides a convenient way for typesetting mathematical equations. In case you are not familiar with Latex, examples are provided in the skeleton file. There is plenty of information online about how to use Latex. Starters might find this one useful: <http://www.maths.tcd.ie/~dwilkins/LaTeXPrimer/>.

In some tasks, you are asked to use `gnuplot` to generate performance analysis plots from measurements. If you are not familiar with `gnuplot` you can find information about how to plot data on <http://www.gnuplotting.org/plotting-data/>. Information on how to export plots into graphic files suitable for embedding into Latex can be found on <http://www.gnuplotting.org/output-terminals/>.

Submission

The submission deadline is **24.00 on Monday 17 March 2014**.

You will be provided with a Git repository on Gitlab initialised with some skeleton files for C++ and Latex. Your submission will be comprised of two parts. You have to submit the modified C++ and Latex skeleton files via Git, where you will also add and submit the generated PDF document. In addition, a text file with the SHA-1 hash of the final commit needs to be submitted to CATE.

Your workflow for this coursework might be as follows:

1. Clone your repository to a local machine to work on the coursework

```
git clone git@gitlab.doc.ic.ac.uk:mygroup/algorithms.git
```

2. Make changes to the files as instructed. Generate a PDF from the latex file as follows

```
latex report.tex && dvipdf report.dvi
```

3. Add files that you are happy with to Git's staging area, for example

```
git add MergeSort.cpp
git add KaratsubaMultiplication.cpp
git add report.tex
git add report.pdf
git add ...
```

and commit the results including some comment about your changes, for example

```
git commit -m 'bugfix in function MergeSort::merge() and added figure to report'
```

4. The changes are now in your local branch. Try running `git log` to see this.
5. Push your changes to the server:

```
git push origin master
```

Note that you can do all those steps as often as you want. It is sometimes useful to commit and even push intermediate results and partial solutions (also for backup reasons). Once you are happy with your files record the SHA-1 hash of your final commit into a text file called `submission_sha.txt`. One way of doing this is to run

```
git rev-parse HEAD >> submission_sha.txt
```

just after you have done the final commit. You can also get the information directly from the Gitlab interface accessible via <http://gitlab.doc.ic.ac.uk>. Upload the file `submission_sha.txt` to CATE in the usual way to record what you want to be considered for marking.

Assessment

Your submission will be marked according to the correctness and style of the code submitted, as well as the solutions presented in your PDF document. The marks for this coursework are allocated as follows:

Recurrences	20%
Sorting	40%
Integer Multiplication	40%

Code that does not compile using the current lab-installed Linux C++ compiler will be limited to a maximum of 60% of the task's marks.

Feedback for this coursework will be returned by Friday 31 March 2014.

Task 1: Recurrences

The first task is about solving recurrences. You are given a couple of recurrence equations below and you have to derive their solutions in terms of running time complexity using the specified methods.

You are provided a Latex document `report.tex`. Insert each solution right after the recurrence definition. An example of how to write mathematical expressions in Latex is provided in the document.

Substitution Method Recursion trees allow you to analyse recurrences to obtain a guess for the substitution method. A closely related method is to expand out the recurrence a few times, until a pattern emerges. We call this the expansion method. An example of how to use the expansion method is given in the Latex document.

Use the expansion method to guess an upper bound and the substitution method to verify your guess:

- $T(n) = 3T(n/2) + n$
- $T(n) = T(n/2) + n^2$
- $T(n) = 4T(n/2 + 2) + n$
- $T(n) = 2T(n - 1) + 1$
- $T(n) = T(n - 1) + T(n/2) + n$

Master Method Use the master method to give tight asymptotic Θ bounds:

- $T(n) = 2T(n/4) + 1$
- $T(n) = 2T(n/4) + n$
- $T(n) = 2T(n/2 + 17) + n$
- $T(n) = 4T(n/2) + 2^n$
- $T(n) = T(3n/4) + \sqrt{n}$

Task 2: Sorting

In this task you are asked to implement `InsertionSort` and `MergeSort`. You need to perform an experimental analysis of their running time. Based on your analysis, you should implement a third sorting algorithm, `HybridSort`, which is similar to `MergeSort` but uses `InsertionSort` for the base case. The problem size for which the base case is invoked has to be inferred from the running time analysis.

Implementation

You are provided with C++ skeleton files for all three sorting algorithms.

- Implement `InsertionSort::sortpart()`.
- Implement `MergeSort::sortpart()` and `MergeSort::merge()`.
- Implement `HybridSort::sortpart()` after performing the running time analysis for the two algorithms above. Reuse functions that you have already implemented above.

Hint: In `MergeSort::merge()` you can use `INT_MAX` as the sentinel for the subsequences. You can use the provided test functions `test_insertion.cpp`, `test_merge.cpp` and `test_hybrid.cpp` to check if your implementations perform correct sortings. You can pass integers as command line arguments.

Analysis

You are provided with routines `analyse_insertion.cpp`, `analyse_merge.cpp` and `analyse_hybrid.cpp`. Once you have finished the implementations, run those routines to collect performance measurements automatically saved in TXT files. Use `gnuplot` to generate plots from those measurements. Fill in the corresponding sections in the provided Latex document `report.tex` for the following sub-tasks:

- In section 2.1, embed the performance analysis plots of `InsertionSort` and `MergeSort`.
- In section 2.2, discuss in one paragraph what you can observe from those plots. Infer the turning point for which `MergeSort` becomes much more efficient.
- In section 2.1, add the performance analysis plot of `HybridSort`.
- In section 2.2, add one paragraph of discussion on what you can observe from the last plot compared to the previous plots.

Hint: If you copy the measurements from the three TXT files into a single file where each dataset is separated by a blank line, you can generate a single plot containing all three performance curves. This allows you to better compare the varying performance.

Task 3: Integer Multiplication

In this task you are asked to implement the two divide and conquer algorithms for base 10 integer multiplication introduced in the lecture and compare their performance in terms of running time. The first algorithm is called `RecursiveMultiplication` which recursively solves the multiplication of n -digit integers by dividing the original problem into four subproblems. The second algorithm is called `KaratsubaMultiplication` in which the multiplication is solved recursively as well, but by dividing the original problem in three subproblems, only.

Implementation

You are provided with C++ skeleton files for both algorithms. Additionally, you will find a `Helper` class which contains functions that are shared by the two algorithms.

- Implement the two functions `Helper::digits()` and `Helper::split()`. The first should give the number of digits of an integer, the second one splits an integer into two parts at a given digit.
- Implement `RecursiveMultiplication::multiply()`.
- Implement `KaratsubaMultiplication::multiply()`.

Hint: Use the provided test functions `test_recursive.cpp` and `test_karatsuba.cpp` to check if your implementations perform correct multiplications. You can pass integers as command line arguments.

Analysis

You are provided with two analysis routines `analyse_recursive.cpp` and `analyse_karatsuba.cpp`. Once you have finished the implementations, run those routines to collect performance measurements automatically saved in TXT files. Use `gnuplot` to generate plots from those measurements. Fill in the corresponding sections in the provided Latex document `report.tex` for the following sub-tasks:

- In section 3.1, embed the performance analysis plots.
- In section 3.2, discuss in one paragraph what you can observe from those plots. Do you find any unexpected behaviour? If yes, what could be the reasons for that?
- In section 3.3, define the recurrences for both algorithms.
- In section 3.3, add solutions to the recurrences using the master method for deriving tight bounds.

Hint: Similar to the analysis of the sorting algorithms, you can produce a single plot with both curves by copying the measurements into a single file. This allows better comparison of the varying performance.

CO202 – Software Engineering – Algorithms

Coursework 2: Divide and Conquer

Submission deadline:
24.00 Monday 17 March 2014

1. Recurrences

substitution method, master method

2. Sorting

insertion sort vs. merge sort

3. Integer Multiplication

recursive, karatsuba multiplication

C++ and LaTeX

Coursework: Integer Multiplication

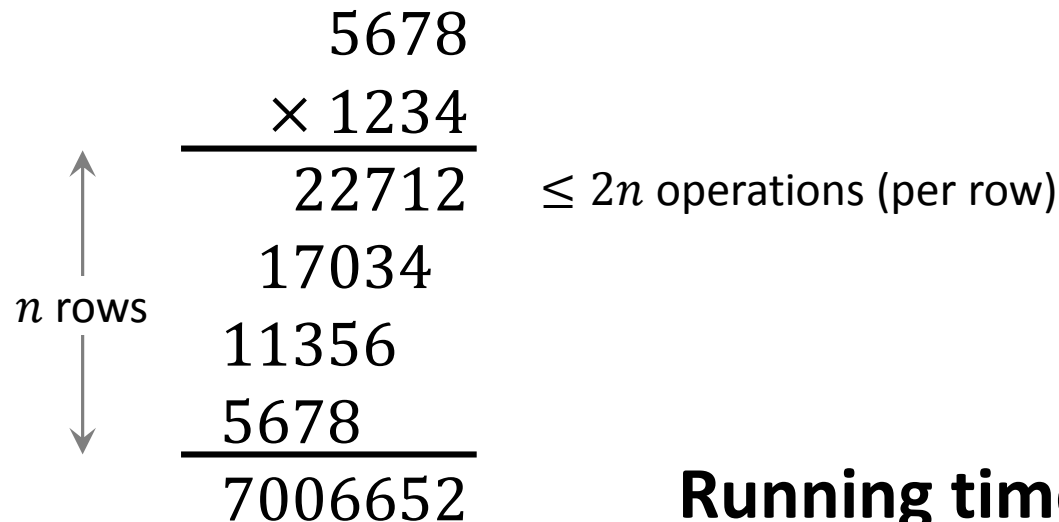
- **Input:** Two n -digit numbers x and y
- **Output:** The product $x \times y$

Example: $x = 5678$ $y = 1234$

Coursework: Integer Multiplication

- **Input:** Two n -digit numbers x and y
- **Output:** The product $x \times y$

Example: $x = 5678$ $y = 1234$



The diagram illustrates the standard multiplication of two 4-digit numbers, 5678 and 1234. It shows four rows of partial products, each shifted one position to the left. A vertical double-headed arrow on the left, labeled "n rows", indicates the number of rows. To the right of the partial products, the text " $\leq 2n$ operations (per row)" is written. The final product, 7006652, is shown at the bottom.

$$\begin{array}{r} 5678 \\ \times 1234 \\ \hline 22712 \\ 17034 \\ 11356 \\ 5678 \\ \hline 7006652 \end{array} \leq 2n \text{ operations (per row)}$$

Running time: $\Theta(n^2)$

Coursework: Recursive Integer Multiplication

Example: $x = \begin{array}{|c|c|} \hline 56 & 78 \\ \hline a & b \\ \hline \end{array} \quad y = \begin{array}{|c|c|} \hline 12 & 34 \\ \hline c & d \\ \hline \end{array}$ Split numbers at digit $\lfloor \frac{n}{2} \rfloor$

$$x = 10^{\lfloor \frac{n}{2} \rfloor} a + b \qquad y = 10^{\lfloor \frac{n}{2} \rfloor} c + d$$

$$x \times y = (10^{\lfloor \frac{n}{2} \rfloor} a + b) \times (10^{\lfloor \frac{n}{2} \rfloor} c + d)$$

$$x \times y = 10^{2\lfloor \frac{n}{2} \rfloor} ac + 10^{\lfloor \frac{n}{2} \rfloor} (ad + bc) + bd$$

Idea: Recursively compute ac , ad , bc , bd

Coursework: Karatsuba Multiplication

$$x \times y = 10^{2\lfloor \frac{n}{2} \rfloor} ac + 10^{\lfloor \frac{n}{2} \rfloor} (ad + bc) + bd$$

Rewrite: $(ad + bc) = (a + b)(c + d) - ac - bd$

Idea: Recursively compute ac , bd , $(a + b)(c + d)$

Only **three** recursive subproblems!