Lab 2 Multiplying two long binary numbers

C S 372 / 469 Data Structures and Algorithms

August 27, 2019

In this lab, you will develop code to multiply two long binary numbers. You will understand how the asymptotic runtime is related to the observed runtime.

1 Iterative multiplication of two binary numbers by definition

Write an iterative C++ program to multiply two binary numbers by definition of multiplication. Please define your function using C++ vector template as follows:

```
vector<bool> multiply_itr(vector<bool> & x, vector<bool> & y)
{
  vector<bool> z;
  //
  // your code here
  //
  return z;
}
```

We use the `vector` class to handle dynamic memory allocation. Unlike pointers, a vector is created and deleted dynamically without calling the `new` or `delete` functions of C++, greatly reducing the chance of memory leak.

The binary number is saved in a vector of boolean type in little-endian, with the least significant bit saved at index 0. For example, if you want to save the 7-bit binary number 1000111 in `x`, then the value of `x` is

```
x[0]=1, x[1]=1, x[2]=1, x[3]=0, x[4]=0, x[5]=0, x[6]=1
```

If `y` represents a 6-bit binary number 111011, then the function should return the product of `x` and `y`, an 13-bit number 1,0000,0101,1101 saved in `z`, a boolean vector of length 13, i.e.,

```
z[0]=1, z[1]=0, z[2]=1, z[3]=1,
z[4]=1, z[5]=0, z[6]=1, z[7]=0,
z[8]=0, z[9]=0, z[10]=0, z[11]=0,
z[12]=1
```

If your answer has more zeros in `z[13]` and above, it is also correct.

Your code should perform the standard multiplication

```
                            1000111
x                            111011
---------------------------------------
                            1000111
                           1000111
                          0000000
                         1000111
                        1000111
+                      1000111
---------------------------------------
                     1000001011101
```

2 Recursive multiplication of two n-bit numbers by divide-and-conquer

Write a C++ program to multiply two binary numbers using the divide-and-conquer algorithm described in class and also in the textbook (page 47, Figure 2.1 `multiply`). Please define your function using C++ vector template as follows:

```
vector<bool> multiply(vector<bool> & x, vector<bool> & y) {
  vector<bool> z;
  // your code here
  return z;
}
```

The two input vectors may not have the same length. You can append zeros to the end of the shorter one to make them the same length before you divide them

into left and right parts. You should not assume the number of bits is always even. You will need to take care of this carefully in implementing the algorithm outlined in the book. One strategy is to pad zeros to the input numbers to make them having the power of two (2^k) bits.

3 Test the two functions

Develop a C/C++ test function to include five examples to check your multiplication functions. If a function passes the tests, minimal output should be displayed on the screen to indicate success; otherwise, point out which example failed the function. This test function should take a function parameter `mul` so that it can test both of your multiplication functions. Your code should thus include

```
bool test(vector<bool> (* mul)(vector<bool> &x, vector<bool> &y))
{
  // Example 1

  // TO-DO: if failed, print out error message

  // Example 2

  // TO-DO: if failed, print out error message

  ...

  // Example 5

  // TO-DO: if failed, print out error message
}

bool testall()
{
  // test the iterative solution:
  bool passed = test(multiply_itr);

  // TO-DO: print out success / failure messages

  // test the divide-and-conquer solution:
  passed = test(multiply);
```

```
  // TO-DO: print out success / failure messages

  return passed;
}
```

Your C++ program must include a main() function that calls the `testall()` function. When the program is compiled by a C++ compiler it generates a binary executable file that will run when invoked from the command line.

4 Visualize the runtime of the two methods

Develop R code inside your C++ source code file to generate runtime report on the two functions you developed. You will generate two plots as shown in Figure 1. It shows both empirical runtime (solid lines) and estimated theoretical runtime (dotted lines), in addition to a diagonal line (dashed lines). Your code will estimate the coefficients c_1 and c_2 so that you can plot the theoretical runtime (dotted lines).

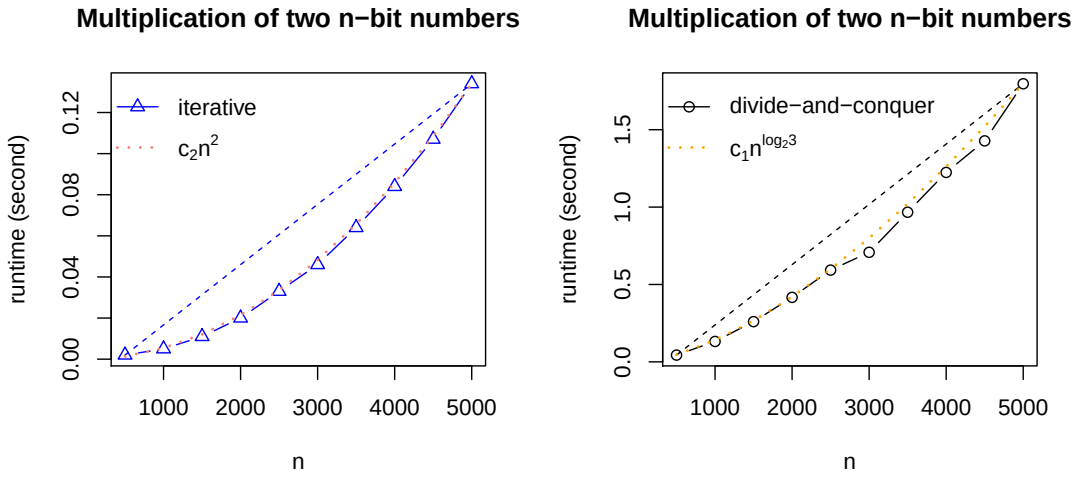

Figure 1: Required output from your runtime observations and analysis.

You need to call the `testall()` function first. If the test failed, visualization should not proceed and you must fix the code first.

Using the following R code, you can generate n-bit random binary numbers and use them to time your functions:

```
n <- 10
x <- sample(c(0,1), n, replace=TRUE)
y <- sample(c(0,1), n, replace=TRUE)
system.time(multiply(x, y))[["user.self"]]
system.time(multiply_itr(x, y))[["user.self"]]
```

5 Recommendations on which method to use

Determine at what input size the two methods will take approximately the same amount of time to finish. How long is the estimated runtime for this input size on your computer. Then formulate a recommendation on how the two functions should be used.

These questions must be answered for a specific computer. Please report your computer processor information including processor type, clock frequency and memory size.

6 Submission

Write a lab report to describe your work done in the following sections:

1. Introduction (define the background, motivation, and the problem),
2. Methods (provide the solutions),

   You will describe the input, output, and steps taken to convert input to output. It should include pseudocode with sufficient detail such that it can be implemented by another programmer with no additional communication.

   This is the core part of the report that explains the technical detail.
3. Results
   (a) show numbers, tables, or figures.
   (b) report the estimated c_1 and c_2 for the runtime of the two functions on your computer.

(c) answer questions posted in section 5.

4. Discussion (general implications and issues),
5. Conclusions (summarize the lab and point to a future direction).

Submit the source code files and your lab report online.