For this assignment, you will use C++ to implement two different algorithms for multiplying very large integers. The purpose of the assignment is to give you some experience translating a non-trivial algorithm from pseudocode to a working implementation.

Your task

Write a program that does these steps, in this order:

1. Read two digit strings $a$ and $b$, separated by a newline character, from standard input (that is, from `cin`). Store each as an array of digits. The numbers $a$ and $b$ may be very large, potentially thousands of digits each. Either or both of the numbers may be zero.

2. Multiply $a$ and $b$ using the $\Theta(n^2)$ time brute force algorithm alluded to in class. The textbook mentions this algorithm on page 187 as the “pencil and paper” algorithm. It’s also the method that is usually taught to small children.

3. Output the text “Brute Force: ”, followed by the answer from Step (2).

4. Multiply $a$ and $b$ again using the divide-and-conquer Karatsuba multiplication algorithm.

5. Output the text “Karatsuba: ”, followed by the answer from Step (4).

Here’s an example run, with the user’s input shown in italics:

```
1234
5678
Brute force: 7006652
Karatsuba: 7006652
```

That’s it. Just multiply the two numbers using both algorithms. Notice that there are no extra prompts or inputs required, so the program could also be run, for example, by redirecting the input from an input file. I will grade your programs in this way, using a Linux workstation functionally identical to the department’s Linux lab machines.

Some additional comments:

- The most common error I’ve seen for this kind of program is the try to use an `int` variable to store the input, the output, or values computed along the way. Remember that most C++ compilers use 32-bit numbers to store `int` variables. This means that for numbers larger than $2^{32} = 4,294,967,296$, things will begin to fail (without any error message) because of overflow. Because you want a program that works for numbers much larger that $2^{32}$, you must store numbers as vectors of digits at every step along the way.

- You may use (and indeed, probably should) use the STL `vector` class for the “array” that holds the digits in each number you manipulate. This simplifies some aspects of the problem because it is much easier to change the size of a vector than of an C++ array.
• You may generate additional debugging output for each algorithm if you like, as long as the two lines required in Steps (3) and (5) appear eventually. Such extra outputs may be useful in awarding partial credit if your solution is not fully correct.

• Watch out for the case that \( a \) and \( b \) have different numbers of digits. The easiest way to handle this is to “pad” the shorter number with leading 0’s to equalize the lengths.

• For calibration purposes: My solution took about 250 lines of C++ code, including generous amounts of comments.

**What to Submit**: You should submit, using the department’s dropbox website, a single C++ source file named containing all of the code for your program. I will compile this program using this command line:

```
g++ -Wall -std=c++11 yourfile.cpp
```