Lab 2: Integer Multiplication

In this lab you will compare the three different versions of integer multiplication.

- **Method1**: Implement the grade-school method as described in the beginning of section 1.1.2. You must use only bit-shift operations to multiply two numbers (use their "internal" binary representations). However, you can use regular addition operations on the resulting values.
- **Method2**: Implement the recursive algorithm in Fig 1.1. You must use bit shift operations for multiplying and dividing by two, but can use regular integer addition.
- **Method3**: Implement the divide-and-conquer algorithm in Fig 2.1. Once again use only bit-wise operations for division and shifting, and also for splitting the input numbers into 2 parts. Use regular additions/subtractions on integers for the rest. Do **not** use any other bitvector/bitarray module. Note that this method recurses on the **number of bits** which reduce in half each time. You can use the python x.bit_length() routine to figure out the number of bits n it takes to represent a given integer x. Also, you may want to make n an input parameter to the algorithm, so the base case check becomes easy.

BTW, look at the errata for Fig 2.1, listed at: http://cseweb.ucsd.edu/~dasgupta/book/errata.pdf

---

What to turn in during lab

Your python script should take d, the number of digits, as a command line parameter.

You will need to perform r=10 runs to record the average time to multiply two d-digit integers in decimal using the three methods above.

For each run generate a random pair of integers x and y, that are d digits long, by using random.randint() or similar python function from the random module. (You can generate d random digits, concatenate them and convert into int). Call Method1, Method2 and Method3 on these inputs and record their running time on each pair.

Finally, print the average time for each method over the r=10 runs.

Answer the following questions:

1. Which method is the fastest? Why? Try different values of d, such as 100, 1000, 10000 and so on to determine the behavior with increasing d.
2. What is the largest number of digits d you can use to multiply using each method? Record if one or more method fails to run beyond a certain number of digits and reason why?