Problem Set 5

Both theory and programming questions are due **Monday, October 31** at **11:59PM**. Please download the .zip archive for this problem set, and refer to the `README.txt` file for instructions on preparing your solutions.

We will provide the solutions to the problem set 10 hours after the problem set is due. You will have to read the solutions, and write a brief **grading explanation** to help your grader understand your write-up. You will need to submit the grading explanation by **Thursday, November 3rd, 11:59PM**. Your grade will be based on both your solutions and the grading explanation.

Problem 5-1. [40 points] **The Knight's Shield**

The optimized circuit verifier that you developed on your Amdtel internship was a huge success and got you on a sure track to landing a sweet offer. You also got transferred to a research group that is working on the *Knight's Shield (KS)*[1], a high-stakes project to develop a massive multi-core chip aimed at the exploding secure cloud computing market.

The KS chip packs 16,384 cores in a die that's the same size as a regular CPU die. However, each core is very small, and can only do arithmetic operations using 8-bit or 16-bit unsigned integers (see Table 1). Encryption algorithms typically use 2,048-bit integers, so the KS chip will ship with software that supports arithmetic on large integers. Your job is to help the KS team assess the efficiency of their software.

Operation	R1 size	R2 size	Result size	Result
`ZERO`			8 / 16	0 (zero)
`ONE`			8 / 16	1 (one)
`LSB R1`	16		8	R1 % 256 (least significant byte)
`MSB R1`	16		8	R1 / 256 (most significant byte)
`WORD R1`	8		16	R1 (expanded to 16-bits)
`ADD R1, R2`	8 / 16	8 / 16	16	$\texttt{R1} + \texttt{R2}$
`SUB R1, R2`	8 / 16	8 / 16	16	$\texttt{R1} - \texttt{R2} \mod 65536$
`MUL R1, R2`	8	8	16	$\texttt{R1} \cdot \texttt{R2}$
`DIV R1, R2`	16	8	8	$\texttt{R1} \div \texttt{R2} \mod 256$
`MOD R1, R2`	16	8	8	$\texttt{R1} \,\%\, \texttt{R2}$
`AND R1, R2`	8 / 16	8 / 16	8 / 16	R1 & R2 (bitwise AND)
`OR R1, R2`	8 / 16	8 / 16	8 / 16	R1 ‖ R2 (bitwise OR)
`XOR R1, R2`	8 / 16	8 / 16	8 / 16	R1 ^ R2 (bitwise XOR)

Table 1: Arithmetic operations supported by the KS chip. All sizes are in bits.

[1]The code name is Amdtel confidential information. Please refrain from leaking to TechCrunch.

The KS library supports arbitrarily large base-256 numbers. The base was chosen such that each digit is a byte, and two digits make up a 16-bit number. Numbers are stored as a little-endian sequence of bytes (the first byte of a number is the least significant digit, for example 65534 = 0xFFFE would be stored as [0xFE, 0xFF]). For the rest of the problem, assume all the input numbers have N digits.

Consider the following algorithm for computing $A + B$, assuming both inputs have N digits.

```
ADD(A, B, N)
1  C = ZERO(N + 1) // ZERO(k) creates a k-digit number, with all digits set to 0s.
2  carry = 0
3  for i = 1 to N
4       digit = WORD(A[i]) + WORD(B[i]) + WORD(carry)
5       C[i] = LSB(digit)
6       carry = MSB(digit)
7  C[N + 1] = carry
8  return C
```

(a) [1 point] What is the running time of ADD?

1. $\Theta(1)$
2. $\Theta(\log N)$
3. $\Theta(N)$
4. $\Theta(N^2)$
5. $\Theta(N^2 \log N)$
6. $\Theta(N^{\log_2 3})$
7. $\Theta(N^{\log_2 6})$
8. $\Theta(N^3)$

Answer: 3. The for loop's run time is $\Theta(N)$.

(b) [1 point] What is the size of ADD's output?

1. $\Theta(1)$
2. $\Theta(\log N)$
3. $\Theta(N)$
4. $\Theta(N^2)$
5. $\Theta(N^2 \log N)$
6. $\Theta(N^{\log_2 3})$
7. $\Theta(N^{\log_2 6})$
8. $\Theta(N^3)$

Answer: 3. The for loop is the first N digits ([N+1] is the N+1th digit).

(c) [1 point] ADD's output size suggests an easy lower bound for the subroutine. Does the running time of ADD match this lower bound?

1. Yes
2. No

Answer: Yes.
runtime $\approx$ output size $\implies$ algorithm is optimal

Consider the following brute-force algorithm for computing $A \cdot B$, assuming both inputs have N digits.

```
MULTIPLY(A, B, N)
1  C = ZERO(2N)
2  for i = 1 to N
3       carry = 0
4       for j = 1 to N
5            digit = A[i] · B[j] + WORD(C[i + j − 1]) + WORD(carry)
6            C[i + j − 1] = LSB(digit)
7            carry = MSB(digit)
8       C[i + N] = carry
9  return C
```

(d) [1 point] What is the running time of MULTIPLY?

1. $\Theta(1)$
2. $\Theta(\log N)$
3. $\Theta(N)$
4. $\Theta(N^2)$
5. $\Theta(N^2 \log N)$
6. $\Theta(N^{\log_2 3})$
7. $\Theta(N^{\log_2 6})$
8. $\Theta(N^3)$

Answer: 4. $\Theta(N^2)$ from the $\Theta(N)$ loop embedded within the $\Theta(N)$ loop.

(e) [1 point] What is the size of MULTIPLY's output?

1. $\Theta(1)$
2. $\Theta(\log N)$
3. $\Theta(N)$
4. $\Theta(N^2)$
5. $\Theta(N^2 \log N)$

6. $\Theta(N^{\log_2 3})$
7. $\Theta(N^{\log_2 6})$
8. $\Theta(N^3)$

Answer: 3. The number is 2N digits, so $\Theta(N)$.

(f) [1 point] MULTIPLY's output size suggests an easy lower bound for the subroutine. Does the running time of MULTIPLY match this lower bound?

1. Yes
2. No

Answer: No. $\Theta(N^2) > \Theta(2N)$

Consider the following brute-force algorithm for computing $A \div B$ and $A \mod B$, assuming both inputs have N digits. The algorithm uses a procedure COPY(A, N) that creates a copy of an N-digit number A, using $\Theta(N)$ time.

```
DIVMOD(A, B, N)
 1  Q = ZERO(N) // quotient
 2  R = COPY(A, N) // remainder
 3  S_0 = COPY(B, N) // S_i = B · 2^i
 4  i = 0
 5  repeat
 6      i = i + 1
 7      S_i = ADD(S_i-1, S_i-1, N)
 8  until S_i[N + 1] > 0 or CMP(S_i, A, N) == GREATER
 9  for j = i − 1 downto 0
10      Q = ADD(Q, Q, N)
11      if CMP(R, S_j, N) != SMALLER
12          R = SUBTRACT(R, S_j, N)
13          Q[0] = Q[0]||1 // Faster version of Q = Q + 1
14  return (Q, R)
```

(g) [1 point] CMP(A, B, N) returns GREATER if $A > B$, EQUAL if $A = B$, and SMALLER if $A < B$, assuming both A and B are N-digit numbers. What is the running time for an optimal CMP implementation?

1. $\Theta(1)$
2. $\Theta(\log N)$
3. $\Theta(N)$
4. $\Theta(N^2)$
5. $\Theta(N^2 \log N)$

6. $\Theta(N^{\log_2 3})$
7. $\Theta(N^{\log_2 6})$
8. $\Theta(N^3)$

Answer: 3. $\Theta(N)$. In the worst case each (corresponding) digit will be compared.

(h) [1 point] SUBTRACT(A, B, N) computes $A - B$, assuming A and B are N-digit numbers. What is the running time for an optimal SUBTRACT implementation?

1. $\Theta(1)$
2. $\Theta(\log N)$
3. $\Theta(N)$
4. $\Theta(N^2)$
5. $\Theta(N^2 \log N)$
6. $\Theta(N^{\log_2 3})$
7. $\Theta(N^{\log_2 6})$
8. $\Theta(N^3)$

Answer: 3. $\Theta(N)$. Same as add.

(i) [1 point] What is the running time of DIVMOD?

1. $\Theta(1)$
2. $\Theta(\log N)$
3. $\Theta(N)$
4. $\Theta(N^2)$
5. $\Theta(N^2 \log N)$
6. $\Theta(N^{\log_2 3})$
7. $\Theta(N^{\log_2 6})$
8. $\Theta(N^3)$

Answer: 4. $\Theta(N^2)$. The until loop runs $\Theta(N)$ times, and takes $\Theta(N)$ time (add) to run. So does the for loop.

The KS library does not use the DIVMOD implementation above. Instead, it uses Newton's method to implement DIV(A, B, N) which computes the division quotient $A \div B$, assuming both inputs have N digits. DIV relies on the subroutines defined above. For example, it uses MULTIPLY to perform large-number multiplication and ADD for large-number addition. MOD(A, B, N) is implemented using the identity $A \mod B = A - (A \div B) \cdot B$.

(j) [2 points] How many times does DIV call MULTIPLY?

1. $\Theta(1)$
2. $\Theta(\log N)$

3. $\Theta(N)$
4. $\Theta(N^2)$
5. $\Theta(N^2 \log N)$
6. $\Theta(N^{\log_2 3})$
7. $\Theta(N^{\log_2 6})$
8. $\Theta(N^3)$

Answer: 2. $\Theta(\log(N))$ since $\epsilon_{n+1} = -\epsilon_i^2 \propto \epsilon_n^2$. Quadratic convergence, number of correct digits doubles each step (lecture 12 notes). So for N digit numbers it will take $\log_2(N)$ steps.

(k) [2 points] What is the running time of MOD?

1. $\Theta(1)$
2. $\Theta(\log N)$
3. $\Theta(N)$
4. $\Theta(N^2)$
5. $\Theta(N^2 \log N)$
6. $\Theta(N^{\log_2 3})$
7. $\Theta(N^{\log_2 6})$
8. $\Theta(N^3)$

Answer: 4. $\Theta\left(N^2\right)$. From page 3 of lecture 12: Complexity of division = complexity of multiplication.

Consider the following brute-force algorithm for computing $B^E \mod M$, assuming all the input numbers have N digits.

```
POWMOD(B, E, M, N)
 1  R = ONE(N) // result
 2  X = COPY(B, N) // multiplier
 3  for i = 1 to N
 4      mask = 1
 5      for bit = 1 to 8
 6          if E[i] & mask != 0
 7              R = MOD(MULTIPLY(R, X, N), M, 2N)
 8          X = MOD(MULTIPLY(X, X, N), M, 2N)
 9          mask = LSB(mask · 2)
10  return R
```

(l) [2 points] What is the running time for POWMOD?

1. $\Theta(1)$

2. $\Theta(\log N)$
3. $\Theta(N)$
4. $\Theta(N^2)$
5. $\Theta(N^2 \log N)$
6. $\Theta(N^{\log_2 3})$
7. $\Theta(N^{\log_2 6})$
8. $\Theta(N^3)$

Answer: 8. $\Theta(N^3)$. The for loop runs N times, and inside it the MOD(MULTIPLY()) line is $\Theta(N^2)$.

Assume the KS library swaps out the brute-force MULTIPLY with an implementation of Karatsuba's algorithm.

(m) [1 point] What will the running time for MULTIPLY be after the optimization?

1. $\Theta(1)$
2. $\Theta(\log N)$
3. $\Theta(N)$
4. $\Theta(N^2)$
5. $\Theta(N^2 \log N)$
6. $\Theta(N^{\log_2 3})$
7. $\Theta(N^{\log_2 6})$
8. $\Theta(N^3)$

Answer: 6. $\Theta\left(N^{\log_2(3)}\right)$. For Karatsuba's algorithm Multiply goes from $\Theta(N^2) \rightarrow \Theta\left(N^{log_2(3)}\right)$ (so does Mod).

(n) [2 points] What will the running time for MOD be after the optimization?

1. $\Theta(1)$
2. $\Theta(\log N)$
3. $\Theta(N)$
4. $\Theta(N^2)$
5. $\Theta(N^2 \log N)$
6. $\Theta(N^{\log_2 3})$
7. $\Theta(N^{\log_2 6})$
8. $\Theta(N^3)$

Answer: 6. $\Theta\left(N^{\log_2(3)}\right)$. It should be the same as multiply.

(o) [2 points] What will the running time for POWMOD be after the optimization?

1. $\Theta(1)$
2. $\Theta(\log N)$
3. $\Theta(N)$
4. $\Theta(N^2)$
5. $\Theta(N^2 \log N)$
6. $\Theta(N^{\log_2 3})$
7. $\Theta(N^{\log_2 6})$
8. $\Theta(N^3)$

Answer: 7. $\Theta(N) \times \Theta\left(N^{\log_2(3)}\right) = \Theta\left(N^{1+\log_2(3)}\right) = \Theta\left(N^{\log_2(2\times 3)}\right) = \Theta\left(N^{\log_2(6)}\right)$

(p) [20 points] Write pseudo-code for KTHROOT(A, K, N), which computes $\lfloor \sqrt[K]{A} \rfloor$ using binary search, assuming that A and K are both N-digit numbers. The running time for KTHROOT(A, K, N) should be $\Theta(N^{2+\log_2 3})$.

Answer: Your pseudo-code in latex.

Problem 5-2. [18 points] **RSA Public-Key Encryption**

The RSA (Rivest-Shamir-Adelman) public-key cryptosystem is a cornerstone of Internet security. It provides the "S" (security) in the HTTPS sessions used for e-commerce and cloud services that handle private information, such as e-mail. RSA secures SSH sessions (used to connect to Athena, for example), and MIT certificates used to log into Stellar. You figure that the KS chip must perform RSA efficiently, since RSA plays such an important role in cloud security. This problem will acquaint you with the encryption and decryption algorithms in RSA.

RSA works as follows. Each user generates two large random primes p and q, and sets his public modulus $m = p \cdot q$. The user then chooses a small number[2] e that is co-prime with $(p-1)(q-1)$, and computes $d = e^{-1} \mod (p-1)(q-1)$. The user announces his public key (e, m) to the world, and keeps d private. In order to send an encrypted message to our user, another user would encode the message as a number smaller than n, and encrypt it as $c = E(n) = n^e \mod m$. Our user would decode the message using $D(c) = c^d \mod m$. Assume that keys can be generated reasonably fast and that $D(E(n)) = n$, for all but a negligible fraction of values of n.

(a) [1 point] What is the running time of an implementation of $D(n)$ that uses the KS library in Problem 1, with the optimized version of MULTIPLY (Karatsuba's algorithm), assuming that n, d and m are N-byte numbers?

1. $\Theta(1)$
2. $\Theta(\log N)$
3. $\Theta(N)$
4. $\Theta(N^2)$
5. $\Theta(N^2 \log N)$
6. $\Theta(N^{\log_2 3})$
7. $\Theta(N^{\log_2 6})$
8. $\Theta(N^3)$

Answer: 7. $\Theta\left(N^{\log_2(6)}\right)$. It just calls PowMod.

You're thinking of using RSA to encrypt important sensitive images, such as last night's picture of you doing a Keg stand. Formally, a picture has $R \times C$ pixels (R rows, C columns), and each pixel is represented as 3 bytes that are RGB color space coordinates[3]. The RSA key is (e, m), where m is an N-byte number. An inefficient encryption method would process each row of pixel data as follows:

1.Break the $3C$ bytes of pixel data into groups of $N-1$ bytes

2.Pad the last group with 0 bytes up to $N-1$ bytes

3.Encrypt each group of $N-1$ bytes to obtain an N-byte output

[2] 65,537 is a popular choice nowadays

[3] see `http://en.wikipedia.org/wiki/RGB_color_space`

4.Concatenate the N-byte outputs

(b) [1 point] How many calls to the RSA encryption function $E(n)$ are necessary to encrypt an $R \times C$-pixel image?

1. $\Theta(1)$
2. $\Theta(RC)$
3. $\Theta(\frac{RC}{N})$
4. $\Theta(\frac{RN}{C})$
5. $\Theta(\frac{CN}{R})$

Answer: 3. $\Theta\left(\frac{RC}{N}\right)$. The $\Theta(RC)$ pixels are encrypted in groups of $N-1 \to \Theta(N)$.

(c) [1 point] What is the running time for decrypting an $R \times C$-pixel image that was encrypted using the method above, using the KS library in Problem 1, with the optimized version of MULTIPLY (Karatsuba's algorithm)?

1. $\Theta(N)$
2. $\Theta(N^2)$
3. $\Theta(N^2 \log N)$
4. $\Theta(N^{\log_2 3})$
5. $\Theta(N^{\log_2 6})$
6. $\Theta(RCN)$
7. $\Theta(RCN^2)$
8. $\Theta(RCN^2 \log N)$
9. $\Theta(RCN^{\log_2 3})$
10. $\Theta(RCN^{\log_2 6})$
11. $\Theta(RN)$
12. $\Theta(RN^2)$
13. $\Theta(RN^2 \log N)$
14. $\Theta(RN^{\log_2 3})$
15. $\Theta(RN^{\log_2 6})$

Answer: 9. $\Theta\left(RCN^{\log_2(3)}\right)$. $D(n)$runtime$\times$num calls to $D(n) = \Theta\left(N^{\log_2(6)}\right)\Theta\left(\frac{RC}{N}\right) = \Theta\left(RCN^{\log_2(3)}\right)$

(d) [5 points] A fixed point under RSA is a number n such that $E(n) \equiv n \mod m$, so RSA does not encrypt the number at all. Which of the following numbers are fixed points under RSA? (True / False)

1. 0
2. 1
3. 2

4. 3
5. $m-2$
6. $m-1$

Answer: 0, 1, and $m-1$ are fixed points under RSA, the rest aren't.
$E(n) \equiv n \text{mod} m \implies n^e \text{mod} m = n \text{mod} m$
$(m-1)^e \text{mod} m = (-1 \times \text{mod} m)^e = -1 = (m-1) \text{mod} m$ (if e is odd, which it is).

(e) [5 points] What other weaknesses does the RSA algorithm have? (True / False)

1. $E(-n) \equiv -E(n) \mod m$
2. $E(n_1) + E(n_2) \equiv E(n_1 + n_2) \mod m$
3. $E(n_1) - E(n_2) \equiv E(n_1 - n_2) \mod m$
4. $E(n_1) \cdot E(n_2) \equiv E(n_1 \cdot n_2) \mod m$
5. $E(n_1)^{n_2} \equiv E(n_1^{n_2}) \mod m$

Answer: 1. True $E(-n) \equiv -E(n) \mod m$ since $E(-n) = (-n)^e \mod m = -(n^e \mod m) = -E(n) \mod m$.
2. False. $E(n_1) + E(n_2) = n_1^e + n_2^e \neq (n_1 + n_2)^e \mod m$.
3. False, following the same logic as 2.
4. True. $E(n_1)E(n_2) = n_1^e \mod (m) n_2^e \mod m = n_1 n_2^e (\mod m)^2 = E(n_1 n_2) \mod m$
5. True. $E(n_1)^{n_2} = (n_1^e \mod m)^{n_2} = (n_1^{e n_2} \mod m)^{\mod m} = E(n_1^{n_2}) \mod m$.

(f) [5 points] Amdtel plans to use RSA encryption to secretly tell Gopple when its latest smartphone CPU is ready to ship. Amdtel will send one message every day to Gopple, using Gopple's public key (e_G, m_G). The message will be NO (the number 20079 when using ASCII), until the day the CPU is ready, then the message will change to YES (the number 5858675 when using ASCII). You pointed out to your manager that this security scheme is broken, because an attacker could look at the encrypted messages, and know that the CPU is ready when the daily encrypted message changes. This is a problem of deterministic encryption. If $E(20079)$ always takes the same value, an attacker can distinguish $E(20079)$ from $E(5858675)$. How can the problem of deterministic encryption be fixed? (True / False)

1. Append the same long number (the equivalent of a string such as 'XXXPADDINGXXX') to each message, so the messages are bigger.
2. Append a random number to each message. All random numbers will have the same size, so the receiver can recognize and discard them.
3. Use a different encryption key to encrypt each message, and use Gopple's public exponent and modulus to encrypt the decryption key for each message.
4. Use an uncommon encoding, such as UTF-7, so that the attacker will not know the contents of the original messages.

5. Share a "secret" key with Gopple, so that the attacker can't use the knowledge on Gopple's public exponent and modulus.

Answer: 1. False. The message only changes once.
2. True. It changes the no's from day to day.
3. True. Same reason as 2.
4. False. It will still change only once.
5. False. Same reasons as 1 and 4.

Problem 5-3. [42 points] **Image Decryption**

Your manager wants to show off the power of the Knight's Shield chip by decrypting a live video stream directly using the RSA public-key crypto-system. RSA is quite resource-intensive, so most systems only use it to encrypt the key of a faster algorithm. Decrypting live video would be an impressive technical feat!

Unfortunately, the performance of the KS chip on RSA decryption doesn't come even close to what's needed for streaming video. The hardware engineers said the chip definitely has enough computing power, and blamed the problem on the RSA implementation. Your new manager has heard about your algorithmic chops, and has high hopes that you'll get the project back on track. The software engineers suggested that you benchmark the software using images because, after all, video is just a sequence of frames.

The code is in the `rsa` directory in the zip file for this problem set.

(a) [2 points] Run the code under the python profiler with the command below, and identify the method inside `bignum.py` that is most suitable for optimization. Look at the methods that take up the most CPU time, and choose the first method whose running time isn't proportional to the size of its output.

```
python -m cProfile -s time rsa.py < tests/1verdict_32.in
```

Warning: the command above can take 1-10 minutes to complete, and bring the CPU usage to 100% on one of your cores. Plan accordingly. If you have installed PyPy successfully, you should replace `python` with `pypy` in the command above for a 2-10x speed improvement.

What is the name of the method with the highest CPU usage?

Answer: fast_mul since __add__ is already optimized.

(b) [1 point] How many times is the method called?

Answer: 93496

(c) [1 point] The troublesome method is implementing a familiar arithmetic operation. What is the tightest asymptotic bound for the worst-case running time of the method that contains the bottleneck? Express your answer in terms of N, the number of digits in the input numbers.

1. $\Theta(N)$.
2. $\Theta(N \log n)$
3. $\Theta(N \log^2 n)$
4. $\Theta(N^{\log_2 3})$
5. $\Theta(N^2)$
6. $\Theta(N^{\log_2 7})$
7. $\Theta(N^3)$

Answer: $\Theta\left(n^{\log_2(3)}\right)$
Comparing the code to lecture 11 page 5, it is very clear that Karatsuba's Method is being used. For reference self_high = x_0, other_high = y_0, result_high = z_1,...

(d) [1 point] What is the tightest asymptotic bound for the worst-case running time of division? Express your answer in terms of N, the number of digits in the input numbers.

1. $\Theta(N)$.
2. $\Theta(N \log n)$
3. $\Theta(N \log^2 n)$
4. $\Theta(N^{\log_2 3})$
5. $\Theta(N^2)$
6. $\Theta(N^{\log_2 7})$
7. $\Theta(N^3)$

Answer: Newton's method is used, so the run time matches that of multiplication: $\Theta\left(n^{\log_2(3)}\right)$

We have implemented a visualizer for your image decryption output, to help you debug your code. The visualizer will also come in handy for answering the question below. To use the visualizer, first produce a trace.

```
TRACE=jsonp python rsa.py < tests/1verdict_32.in > trace.jsonp
```

On Windows, use the following command instead.

```
rsa_jsonp.bat < tests/1verdict_32.in > trace.jsonp
```

Then use Google Chrome to open `visualizer/bin/visualizer.html`

(e) [6 points] The test cases that we supply highlight the problems of RSA that we discussed above. Which of the following is true? (True / False)

1. Test `1verdict_32` shows that RSA has fixed points.
2. Test `1verdict_32` shows that RSA is deterministic.
3. Test `2logo_32` shows that RSA has fixed points.
4. Test `2logo_32` shows that RSA is deterministic.
5. Test `5future_1024` shows that RSA has fixed points.
6. Test `5future_1024` shows that RSA is deterministic.

Answer: 1. True. You can tell from the black areas.
2. True. It still has a face (not probabilistic).
3. True. MIT is readable encrypted.
4. False. Fuzzy edges
5. False. Looks like noise.
6. False. Looks like noise.
5future_ 1024 runs very slow (hours).

(f) [1 point] Read the code in `rsa.py`. Given a decrypted image of $R \times C$ pixels (R rows, C columns), where all the pixels are white (all the image data bytes are 255), how many times will `powmod` be called during the decryption operation in `decrypt_image`?

1. $\Theta(1)$
2. $\Theta(RC)$
3. $\Theta(\frac{RC}{N})$
4. $\Theta(\frac{RN}{C})$
5. $\Theta(\frac{CN}{R})$

Answer: $\Theta(1)$, it is chunked in the 'while i <len(hex_string):' loop in decrypt.

(g) [30 points] The multiplication and division operations in `big_num.py` are implemented using asymptotically efficient algorithms that we have discussed in class. However, the sizes of the numbers involved in RSA for typical key sizes aren't suitable for complex algorithms with high constant factors. Add new methods to `BigNum` implementing multiplication and division using straight-forward algorithms with low constant factors, and modify the main multiplication and division methods to use the simple algorithms if at least one of the inputs has 64 digits (bytes) or less. Please note that you are not allowed to import any additional Python libraries and our test will check this.
Answer: see big_num.py: specifically slow_mul and slow_divmod.

The KS software testing team has put together a few tests to help you check your code's correctness and speed. `big_num_test.py` contains unit tests with small inputs for all `BigNum` public methods. `rsa_test.py` runs the image decryption code on the test cases in the `tests/` directory.

You can use the following command to run all the image decryption tests.

```
python rsa_test.py
```

To work on a single test case, run the simulator on the test case with the following command.

```
python rsa.py < tests/1verdict_32.in > out
```

Then compare your output with the correct output for the test case.

```
diff out tests/1verdict_32.gold
```

For Windows, use `fc` to compare files.

```
fc out tests/1verdict_32.gold
```

While debugging your code, you should open a new Terminal window (Command Prompt in Windows), and set the `KS_DEBUG` environment variable (`export KS_DEBUG=true`; on Windows, use `set KS_DEBUG=true`) to use a slower version of our code that has more consistency checks.

When your code passes all tests, and runs reasonably fast (the tests should complete in less than 90 seconds on any reasonably recent computer using PyPy, or less than 600 seconds when using CPython), upload your modified `big_num.py` to the course submission site. Our automated grading code will use our versions of `test_rsa.py`, `rsa.py` and `ks_primitives.py` / `ks_primitives_unchecked.py`, so please do not modify these files.