Problem Set 6

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 6-1. [30 points] **I Can Haz Moar Frendz?**

Answer:

We preform a variety of breadth first search. We start the search at point S, and we stop after k steps. The mainly different things here is that we do not prevent to visit a node. we start from s, in the first step, we update all neighbors to ER(u_0,u_i) where (u_0,u_i) $\in E$ and next update each two-unit far node to ER(u_0,u_j) $\times$ ER(u_j,u_i), if ER(u_0,u_j) $\times$ ER(u_j,u_i) $>$ ER(u_0,u_i) or u_i hasn't been visited, then repeat the process. Finally after k steps, we will get the right answer.

```
MAX-EDGERANKS(s, G, K)
 1  INITIALIZZE-SINGLE-SOURCE(G,s)
 2  front = [s]
 3  tmp = [ ]
 4  for i = 1 to K
 5      for current In front
 6          for next In current.adj
 7              if d[next] > d[current] × w(current, next)
 8                  d[next] = d[current] × w(current, next)
 9                  if next not in tmp
10                      ADD(tmp, next)
11      front = tmp
12      tmp = [ ]
13  return d
```

Analysis:

INITIALIZZE-SINGLE-SOURCE cost $O(V)$, the loop in line 4 execute K times. The inner loop(line 7 10) execute $|E|$ times at most, because there at most $|E|$ edges in the graph, so we know the complexity is $O(V + kE)$.

Problem 6-2. [30 points] **RenBook Competitor**

Answer:

(a) We keep using DFS method on all vertices in the whole graph built by the problem(Libraries as Nodes, dependencies as directional edgs) without repating visiting, and after each vertex is finished, memory the finish time of DFS on each node then we know that the earier the node finished, the laster the libraries is installed, and that is the installation order.

(b) If we perform the algothrim in (a) and just produce the installation order ingore if a library had been installed or not, we will fail to meet the complexity. To fix that, we need to figure out how we can get the right answer without 'dive into' the installed libraries.

DFS-VISITED(n, G, F, P, s, end)

```
1  if ISINSTALLED(n) OR n IN F
2      return FALSE
3  if n IN P OR n == end
4      return TURE
5  later = FALSE
6  for neighbor in n.adj
7      if DFS-VISITED(n,G,F,P,s,end) ==TURE
8      later = TURE
9  if later
10     ADD(P,n)
11     return TURE
12 else
13     ADD(F,n)
14     return FALSE
```

INSTALL-ORDER-PRODUCER(t, G)

```
1  INITIALIZZE-SINGLE-SOURCE(G,s)
2  front = [s]
3  tmp = [ ]
4  for i = 1 to K
5      for current In front
6          for next In current.adj
7              if d[next] > d[current] × w(current, next)
8                  d[next] = d[current] × w(current, next)
9                  if next not in tmp
10                     ADD(tmp, next)
11     front = tmp
12     tmp = [ ]
13 return d
```

Problem 6-3. [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

(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: 4

(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: 4

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 2 3

(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: 1

(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.

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.