SJSU
Students

Question 1

Let k=6. Assume a cache with initial contents 2,3,4,5,6,7 and with at least one item not in the cache. Give an example sequence of length at least 7 of cache requests

$$1, 2, 3, 4, 5, 6, 7, 1, 2, 3, 4, 5$$

and a sequence of random choices by the Marker algorithm so that its competitiveness on this sequence with these choices would be greater than H_6.

Using the sequence above, the Marker algorithm goes as follows:

Cache before request	Marker bits before request	Request	Cache after request	Marker bits after request
234567	000000	1 (miss)	234167	000100
234167	000100	2, 3, 4 (hits)	234167	111100
234167	111100	5 (miss)	234157	111110
234157	111110	6 (miss)	234156	111111
Reset				
234156	000000	7 (miss)	734156	100000
734156	100000	1 (hit)	734156	100100
734156	100100	2 (miss)	734152	100101
734152	100101	3, 4, 5 (hits)	734152	111111

5 misses occurred for this request sequence using Marker algorithm.

The minimum number of misses can be achieved by MIN as follows:

Cache before request	Request	Cache after request
234567	1 (miss)	234561
234561	2, 3, 4, 5, 6 (hits)	234561
234561	7 (miss)	234571
234571	1, 2, 3, 4, 5 (hits)	234571

⇨ 2 misses occurred for the same request sequence for MIN

$H_6 = 1 / 1 + 1 / 2 + \ldots + 1/6 = 49/20$
$C_{Marker} = 5 / 2 = 50/20$
⇨ $C_{Marker} > H_6$ *(Q.E.D)*

Question 2.1

Use the extended Euclidean algorithm to find the multiplicative inverse of 14mod2145.

```
Extended-Euclid(2145, 14):
        if (14 == 0) then return (14, 1, 0)
        (d', x', y') = Extended-Euclid(14, 3)
        Extended-Euclid(14, 3):
                if (3 == 0) then return (14, 1, 0)
                (d', x', y') = Extended-Euclid(3, 2)
                Extended-Euclid(3, 2):
                        if (2 == 0) then return (3, 1, 0)
                        (d', x', y') = Extended-Euclid(2, 1)
                        Extended-Euclid(2, 1):
                                if (1 == 0) then return (2, 1, 0)
                                (d', x', y') = Extended-Euclid(1, 0)
                                Extended-Euclid(1, 0):
                                        if (0 == 0) then return (1, 1, 0)
                                (d', x', y') = (1, 1, 0)
                                (d, x, y) = (1, 0, 1 - 2 * 0)
                                return (1, 0, 1)
                        (d', x', y') = (1, 0, 1)
                        (d, x, y) = (1, 1, 0 - 1 * 1)
                        return (1, 1, -1)
                (d', x', y') = (1, 1, -1)
                (d, x, y) = (1, -1, 1 - 4 * -1)
                return (1, -1, 5)
        (d', x', y') = (1, -1, 5)
        (d, x, y) = (1, 5, -1 - 153 * 5)
        return (1, 5, -766)
```

Therefore, the multiplicative inverse for 14 mod 2145 is -766 (or 1379 mod 2145)

Yu-Tang Shen
015938611

Question 2.2

Solve $6x \equiv 9 \bmod 33$ for all solutions.

For equation $6x \equiv 9(mod33)$, there are either gcd(6, 33) = 3 solutions or none.

```
Modular-Linear-Equation-Solver(6, 9, 33):
        (d, x', y') = Extended-Euclid(6, 33) # see calulation at (*)
        (d, x', y') = (3, -5, 1)
        solution = []
        if (3 | 9):
                x_0 = -5 * (9/3) mod 33
                x_0 = 18
                for i in [0, 3):
                        solution.append(18 + i * (33/3)) mod n
                         i = 0:
                                solution = [18]
                         i = 1:
                                solution = [18, 29]
                         i = 2:
                                solution = [18, 29, (18+22) mod 33]
                                          = [18, 29, 7]

        return solution
```

Solution for $6x \equiv 9(mod33)$ is **[7, 18, 29]**

(*) Euclid(6, 33) calculation

```
Extended-Euclid(6, 33):
        if(33 == 0) return (6, 1, 0)
        (d', x', y') = Extended-Euclid(33, 6)
        Extended-Euclid(33, 6):
                if(6 == 0) return (33, 1, 0)
                (d', x', y') = Extended-Euclid(6, 3)
                Extended-Euclid(6, 3):
                        if (3 == 0) return (6, 1, 0)
                        (d', x', y') = Extended-Euclid(3, 0)
                        Extended-Euclid(3, 0):
                                if (0 == 0) return (3, 1, 0)
                        (d', x', y') = (3, 1, 0)
                        (d, x, y) = (3, 0, 1 - 2 * 0)
                        return (3, 0, 1)
                (d', x', y') = (3, 0, 1)
                (d, x, y) = (3, 1, 0 - 5 * 1)
                return (3, 1, -5)
        (d', x', y') = (3, 1, -5)
        (d, x, y) = (3, -5, 1 - 0 * -5)
        return (3, -5, 1)
```

Question 3

Using the Chinese Remainder theorem, determine a number $x \bmod 2145$ that satisfies $x \equiv 2 \bmod 3$, $x \equiv 3 \bmod 5$, and $x \equiv 4 \bmod 11$, and $x \equiv 5 \bmod 13$

n_i => [3, 5, 11, 13]
m_i => $[2145/3, 2145/5, 2145/11, 2145/13]$
= [715, 429, 195, 165]

Run Extended Euclid (m_i, n_i) *(denoted as EE)*

i = 1

EE(715, 3)
=> EE(3, 1)
=> EE(1, 0)
=> (1, 1, 0)
=> (1, 0, 1 – 3 * 0)
=> (1, 0, 1)
=> (1, 1, 0 – 238 * 1)
=> (1, 1, -238)
t_1 = 715^{-1} = 1 mod 3
c_1 = m_1 * t_1 = 715 * 1 = 715

i = 2

EE(429, 5)
=> EE(5, 4)
=> EE(4, 1)
=> EE(1, 0)
=>(1, 1, 0)
=> (1, 0, 1 – 4 * 0)
=> (1, 0, 1)
=> (1, 1, 0 – 5 // 4 * 1)
=> (1, 1, -1)
=> (1, -1, 1 - 429//5 * -1)
=> (1, -1, 86)
t_2 = 429^{-1} = -1 mod 5
c_2 = 429 * -1 = -429

i = 3 (EE was called similar to above cases, simplified to the inline format as follows)

EE(195, 11) => EE(11, 8) => EE(8, 3) => EE(3, 2) => EE(2, 1) => EE(1, 0)
(1, -4, 71) <= (1, 3, -4) <= (1, -1, 3) <= (1, 1, -1) <= (1, 0, 1) <= (1, 1, 0)
t_3 = 195^{-1} = -4 mod 11
c_3 = 195 * -4 = -780

<u>i = 4</u>

EE(165, 13) => EE(13, 9) => EE(9, 4) => EE(4, 1) => EE(1, 0)
(1, 3, -38) <= (1, -2, 3) <= (1, 1, -2) <= (1, 0, -1) <= (1, 1, 0)
t_4 = 165^{-1} = 3 mod 13
c_4 = 165 * 3 = 495

a_i => [2, 3, 4, 5]
c_i =>[715, -429, -780, 495]

x = 2 * 715 + 3 * -429 + 4 * -780 + 5 * 495
= 1430 – 1287 – 3120 + 2475
= -502 mod 2145
= **<u>1643 mod 2145</u>**

<u>Check:</u>
1643÷3 = 547…2
1643÷5 = 328…3
1643÷11=149…4
1643÷13=126…5

Question 4

Suppose p=11, q=17. If we choose e=3, what would be a the RSA public and private keys?

p = 11, q = 17, e = 3

n = pq = 11 * 17 = 187

phi(n) = (p-1)(q-1) = 160 was indeed relatively prime to 3

EE(160, 3) => EE(3, 1) => EE(1, 0)
(1, 1, -53) <= (1, 0, 1) <= (1, 1, 0)

160 * 1 + 3 * -53 = 1
3 * -53 = 1 mod 160, **d** = -53 mod 160 **= 107 mod 160**

pubkey = (3, 187)
secret = (107, 187)
M = 83

Show the result of encrypting with the private key, the message 83. Show the steps in decrypting it, to get the original number back.

Encrypt

Cipher_text = Message^e mod n

C = 83^{107} mod 187 = 162 (Using the *expMod* function in my Java program)

```java
static int expMod(int x, int y, int p){
        if (x == 0)
            return 0;
        if (y == 0)
            return 1;

        long tmp;
        if (y % 2 == 0){
            tmp = expMod(x, y / 2, p);
            tmp = (tmp * tmp) % p;
        }else{
            tmp = x % p;
            tmp = (tmp * expMod(x, y - 1, p) % p) % p;
        }

        return (int)((tmp + p) % p);
    }
```

Decrypt

Plain_text = Cipher_text^d mod n

P= 162^3 mod 187 = **83** (equals to original message!)

Yu-Tang Shen
015938611

Coding question

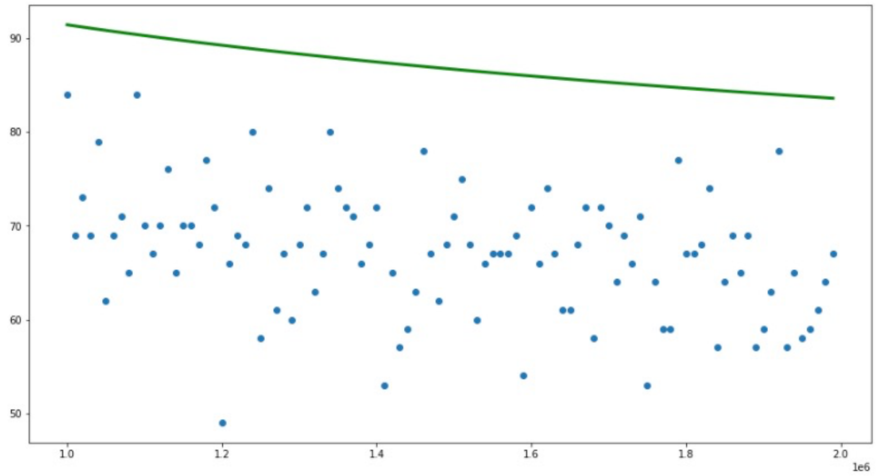
According to Brun's argument[1], the number of twin primes grow in $O(N/log^2(N))$, so I did linear regression on $N/log^2(N)$. In this experiment, the number of primes could be bounded by $\mathbf{2.04 \times N/log^2(N)}$ as follows.

```
In [33]: df = pd.read_csv('result10.csv')
         X = list(zip(df.nDividedByLognSquared, [0 for _ in range(100)]))
         y = df[df.predict == 0].prime
         reg = LinearRegression().fit(X, y)
         print(reg.coef_)
         df.predict = df.nDividedByLognSquared * reg.coef_[0]

         [2.03920101 0.        ]
```

```
In [34]: plt.figure(figsize = (15, 8))
         plt.scatter(x = df.bins, y = df.prime)
         plt.plot(df.bins, df.predict, linewidth = 3, c = 'green')

Out[34]: [<matplotlib.lines.Line2D at 0x7fd60b8f22b0>]
```



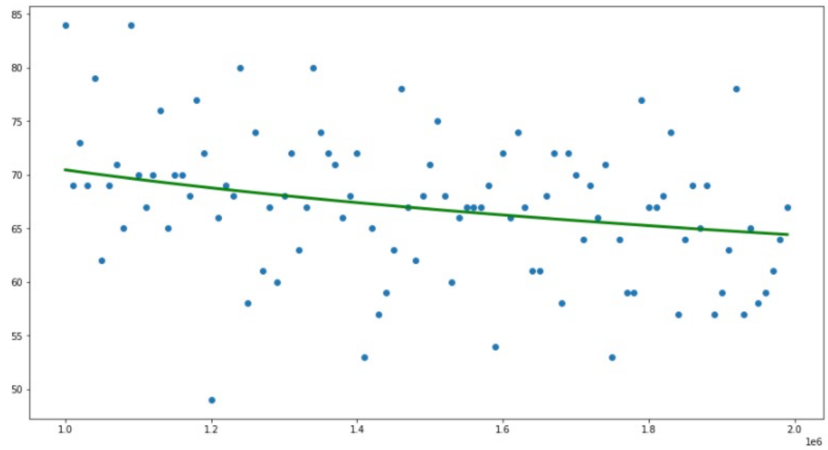
If we were to predict the number of primes in a range, $\mathbf{1.57 \times N/log^2(N)}$ could be a better fit. (the data was too scattered for a good function without large bias)

```
In [35]: df = pd.read_csv('result10.csv')
         X = list(zip(df.nDividedByLognSquared, [0 for _ in range(100)]))
         y = df[df.predict == 0].prime
         reg = LinearRegression(fit_intercept = False).fit(X, y)
         print(reg.coef_)
         df.predict = df.nDividedByLognSquared * reg.coef_[0]

         [1.57141764 0.        ]
```

```
In [36]: plt.figure(figsize = (15, 8))
         plt.scatter(x = df.bins, y = df.prime)
         plt.plot(df.bins, df.predict, linewidth = 3, c = 'green')

Out[36]: [<matplotlib.lines.Line2D at 0x7fd60baeb640>]
```



[1] https://en.wikipedia.org/wiki/Brun%27s_theorem