李驊祐 資工三 B10902078

For this homework, I used python to execute all my code. I imported the necessary library for this homework, including ecdsa, hashlib, random, and gmpy2.invert.

```
import ecdsa
import hashlib
import random
from gmpy2 import invert


G = ecdsa.SECP256k1.generator
N = ecdsa.SECP256k1.order

d = 2078
```

P1.
First I find the generator G and order N of ecdsa.SECP256k1. Then, I evaluate 4G.

```
p1_ans = 4 * G
print("G_x = {}\nG_y = {}".format(p1_ans.x(), p1_ans.y()))
```

```
G_x = 103388573995635080359749164254216598308788835304023601477803095234286494993683
G_y = 37057141145242123013015316630864329550140216928701153669873286428255828810018
```

P2. I evaluate 5G.

```
p2_ans = 5 * G
print("G_x = {}\nG_y = {}".format(p2_ans.x(), p2_ans.y()))
```

```
G_x = 21505829891763648114329055987619236494102133314575206970830385799158076338148
G_y = 98003708678762621233683240503080860129026887322874138805529884920309963580118
```

P3. I evaluate dG.

```
Q = d * G
print("G_x = {}\nG_Y = {}".format(Q.x(), Q.y()))
```

```
G_x = 105589526114650606199207717196912087758404270520796559587273649751971378262 62
G_Y = 995957544221242302681819508408070470951253757942360156741392845809931744277 39
```

P4.

First, I turn the last 4 digits of my student id d = 2078 into binary. Then, I use a for loop to check every bit (excluding MSB). The number of double operations is equal to the number of bits (excluding MSB), and the number of add operations is equal to the number of 1-bit (excluding MSB).

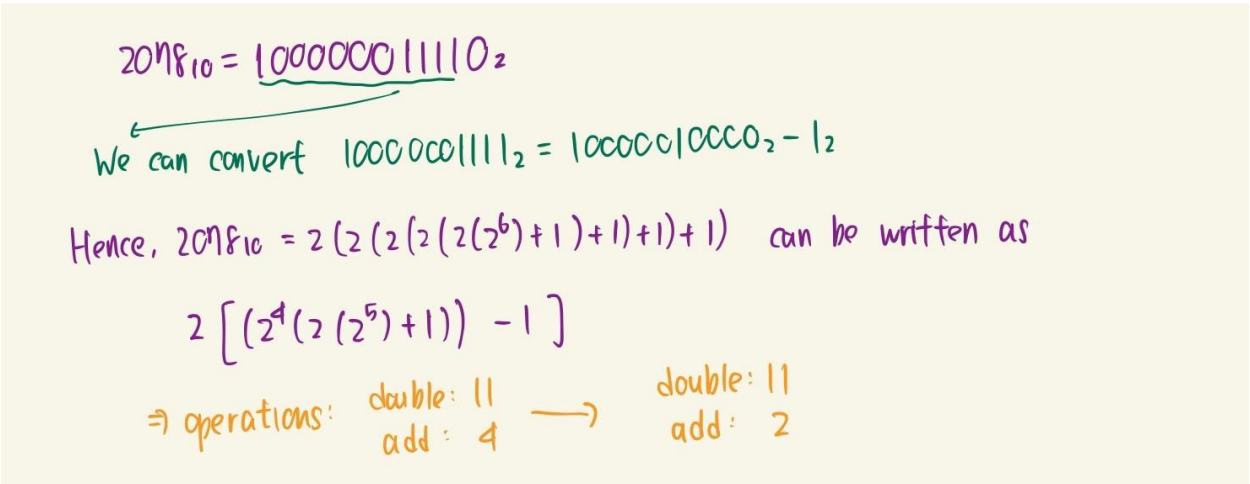
```
def count_bits(integer):
    double = 0
    add = 0
    binary_representation = bin(integer)[2:]
    print("bin = {}".format(binary_representation))
    for bit in binary_representation[1:]:
        double += 1
        if (bit == '1'):
            add += 1
    return double, add
double, add = count_bits(d)
print("double = {}, add = {}".format(double, add))
```

```
bin = 100000011110
double = 11, add = 4
```

double = number of bits (excluding MSB) = 12 - 1 = 11

add = number of 1 (excluding MSB) = 4

P5.



P6.

First, I find one transaction from blockchain.com and get its hash_id (message). Then, I use SHA-256 to encode the message. Then, I let z = Ln leftmost bit of e, where Ln is the bit length of group order N. Then, I have a while loop. In the while loop, I will randomly select an ephemeral key k, and use it to calculate r and s. I do this until both r and s are not 0.

```python
# this is the bitcoin transaction I used:
# https://www.blockchain.com/explorer/transactions/btc/a473c52545422e78f5ac9b46d97766d85a880cfa8acbc662ffd89a9bcae2a3ff

message = "a473c52545422e78f5ac9b46d97766d85a880cfa8acbc662ffd89a9bcae2a3ff"
# 1. Calculate e = HASH(m)
hash_func = hashlib.sha256()
hash_func.update(message.encode('utf-8'))
e = hash_func.hexdigest()
print("e = {}".format(e))

# 2. Let z be the Ln leftmost bits of e, where Ln is the bit length of the group order n.
Ln = N.bit_length()
Ln = min(Ln, len(message))
z = message[:Ln]
z = int(z, 16)
print("z = {}".format(z))

r = 0
s = 0
dA = 2078
while True:
    # 3. select a random integer k from [1, n-1]
    k = random.randint(1, N-1)
    # 4. Calculate the curve point (x1, y1) = k * G
    x1 = (k * G).x()
    y1 = (k * G).y()
    # 5. Calculate r = x1 mod n. If r = 0, go back to while loop
    r = x1 % N
    # 6. Calculate s = k^-1 (z + rdA) mod n. If s = 0, go back to while loop
    s = (invert(k, N) * (z + r * dA)) % N
    # 7. The signature is the pair (r, s)
    if r != 0 and s != 0:
        break
print("r = {}".format(r))
print("s = {}".format(s))
```

✓ 0.0s

```
e = 5b2cb9133cc060148fd6bfab62f99a64315cb5703861cfc60ac948abbc1c6113
z = 74383855228947319945519443418357011162203109683400000045254379739972964099071
r = 83748745981425700137858991025602091054163788315517283949451734993474396910701
s = 94425169214722955971837123327655141594272356123609848962749568756453437145168
```

P7. First, I calculate the public key that corresponds to my private key. Then, I ensure that both r and as are within [1, N-1]. Then, I calculate w, u1, and u2, and use it to evaluate x1 and y1. If x1 is equal to r under mod N, then the signature is valid.

```python
# public key QA = dA * G
QA = dA * G

# 1. Verify that r and s are integers in [1, n-1]. If not, the signature is invalid
assert r >= 1 and r <= N-1 and s >= 1 and s <= N-1, "signature is invalid"
# 2. Calculate e = HASH(m), where HASH is the same function used in the signature generation
# 3. Let z be the Ln leftmost bits of e.
# 4. Calculate w = s^-1 mod n.
w = (invert(s, N)) % N
print("w = {}".format(w))
# 5. Calculate u1 = zw mod n and u2 = rw mod n.
u1 = (z * w) % N
u2 = (r * w) % N
print("u1 = {}".format(u1))
print("u2 = {}".format(u2))
# 6. Calculate the curve point (x1, y1) = u1 * G + u2 * QA.
x1 = (u1 * G + u2 * QA).x()
y1 = (u1 * G + u2 * QA).y()
print("x1 = {}".format(x1))
print("y1 = {}".format(y1))
# 7. The signature is valid if r = x1 (mod n), invalid otherwise.
if ((r % N) == (x1 % N)):
    print("Signature is valid")
else:
    print("Signature is invalid")
```

✓ 0.0s

```
w = 23292357514322650997895796328501131175917510866493348638375504257518750780694
u1 = 81025228337971866432262071461746633967568130148770154303415668158114620120627
u2 = 10733349790388307994288080689704925650522906440154079844581668302998175257620 2
x1 = 74289610472057215889246045397740155592818111169389504114792202052545610211533
y1 = 10685685680891686572265086341341817471087184190383039599171422924503287572320 6
Signature is valid
```

P8.

$p(x) = ax^2 + bx + c$

$p(1) = a + b + c = 10 \dots ①$

$p(2) = 4a + 2b + c = 20 \dots ②$

$p(3) = 9a + 3b + c = 2018 \dots ③$

② − ①: $3a + b = 10 \dots ④$

③ − ①: $8a + 2b = 2068 \dots ⑤$

⑤ − 2 × ④: $2a = 2048 \Rightarrow a = \boxed{1024}$

④: $3 \cdot 1024 + b = 10 = 1001$

$\Rightarrow b = 1001 - 3072 = \boxed{6945}$

①: $1024 + 6945 + c = 10 = 1001$

$\Rightarrow c = 1001 - 1024 - 6945 = \boxed{2048}$

I also wrote a code for this part. I used lagrange and sympy.Poly to find the coefficient a, b, c.

```
from sympy import symbols, Poly, GF

x = symbols('x')

c_lagrange = ((10 * (x - 2) * (x - 3) * invert(1 - 2, 10007) * invert(1 - 3, 10007)) + (20 * (x - 1) * (x - 3) * invert(2 - 1, 10007) * invert(2 - 3, 10007)) +
              (2078 * (x - 1) * (x - 2) * invert(3 - 1, 10007) * invert(3 - 2, 10007)))

p = Poly(c_lagrange, domain=GF(10007))
print(p)
```

```
Poly(1024*x**2 - 3062*x + 2048, x, modulus=10007)
```