Q1. (10 points)

Is $4^{1536} \equiv 9^{4824}$ mod 35? Show all work to support your answer. Hint: 35 is not a prime so you cannot apply Fermat's little theorem directly. However, 35 is a product of primes and you can apply the theorem to those prime factors. You can also use modular exponentiation approach (but show all steps).

A variation of Fermat's Little Theorem says $a^{(p-1)(q-1)} \equiv 1 \bmod p*q$. We can apply this to each of the numbers given in the problem.

For 4^{1536}:

$(4^{(7-1)(5-1)}) \equiv 1 \bmod 7*5 \rightarrow (4^{24}) \equiv 1 \bmod 35$.

Now we need to equate (4^{24}) with 4^{1536}. Luckily, 1536 is divisible by 24 and returns an answer of 64. Thus we can raise each side of the equation from above to the power of 64.

$(4^{24})^{64} \equiv 1^{64} \bmod 35 \rightarrow 4^{1536} \equiv 1 \bmod 35$.

Now, we repeat the process with 9^{4824}:

$9^{4824} = (3^2)^{4824} = 3^{9648}$. 9648 is also divisible by 24 and yields 402.

$(3^{24})^{402} \equiv 1 \bmod 35$.

$4^{1536} \equiv 9^{4824}$ mod 35 is thus true, because both yield the remainder 1 when divided by 35.

Q2. (10 points)

Solve $x^{86} \equiv 6 \bmod 29$, i.e., find the value x for which the equation is true. Hint: You can use fermat's little theorem.

Using the variation of Fermat's little theorem that says $a^{p-1} \equiv 1 \bmod p$: $X^{28} = 1 \bmod 29$.

$X^{28} * X^{28} * X^{28} * X^2 \equiv 6 \bmod 29 \rightarrow X^2 \equiv 6 \bmod 29$.

From here I just thought of numbers that when squared would give the remainder 6 when divided by 29.

X	**X^2**	**X^2 mod 29**
1	1	1
2	4	2
3	9	9
4	16	16
5	25	25
6	36	7
7	49	20
8	64	6

As we can see from the table, x = 8 satisfies the equation above.

Q3. (10 points)

Prove that gcd(F_{n+1},F_n) = 1, for n≥1, where F_n is the n-th Fibonacci element.

I will prove this with induction.

Base Case: n = 0. F_1 = 1. F_0 = 1. Gcd(0,1) = 1

Induction Case: Assume gcd(F_{n-1},F_n) = 1 and implies gcd(F_{n+1},F_n) = 1

gcd(F_{n+1},F_n)	= gcd(F_n,F_{n+1} mod F_n)	(We know that gcd(a,b) = gcd(b, a mod b)
	= gcd(F_n,(F_{n-1} + F_n) mod F_n)	(We know this given the definition of the Fibonacci series)
	= gcd(F_n,F_{n-1})	(Properties of modular arithmetic)
	= 1	(By induction hypothesis)

Thus gcd(F_{n-1},F_n) = 1 and implies gcd(F_{n+1},F_n). Combined with the base case of gcd(F_0,F_1)=1, this implies that gcd(Fn+1,Fn) = 1, for n≥1.

Q4. (10 points)

Assume that the cost to multiply a n-bit integer with a m-bit integer is O(nm). Given integers x and y with n-bits and m-bits, respectively, give an efficient algorithm to compute x^y. Show that the method is correct, and analyze its running time.

My algorithm:

```
def test(x,y):
   if (y==1):
      return x
   elif (y==2):
      return x**2;
   elif (y%2==0):
      return test(test(x,y/2),2);
   else:
      return x*test(test(x,(y-1)/2),2);
```

This bit of testing code in python:

```
for x in range(1,5):
      for y in range(1,5):
         print("x=",x,"y=",y,"f=",test(x,y))
```

Gives the following output:

```
x= 1 y= 1 f= 1
x= 1 y= 2 f= 1
x= 1 y= 3 f= 1
x= 1 y= 4 f= 1
x= 1 y= 5 f= 1
x= 2 y= 1 f= 2
x= 2 y= 2 f= 4
x= 2 y= 3 f= 8
x= 2 y= 4 f= 16
x= 2 y= 5 f= 32
x= 3 y= 1 f= 3
x= 3 y= 2 f= 9
x= 3 y= 3 f= 27
x= 3 y= 4 f= 81
x= 3 y= 5 f= 243
x= 4 y= 1 f= 4
x= 4 y= 2 f= 16
x= 4 y= 3 f= 64
x= 4 y= 4 f= 256
x= 4 y= 5 f= 1024
x= 5 y= 1 f= 5
x= 5 y= 2 f= 25
x= 5 y= 3 f= 125
x= 5 y= 4 f= 625
x= 5 y= 5 f= 3125
```

While the above output demonstrates that the code is probably correct, we can use some logical thinking to prove that the code will function correctly for all integers x and y.

The code works by rewriting powers depending on the parity of the power. Given x^y, if y is even, the formula can be written as $(x^{y/2})^2$ and if y is odd, the formula can be written as $x * (x^{(y-1)/2})^2$ (This is also verifiable given the properties of exponents). Applying the above alternative forms recursively means that any two positive integers – one a base and one a power -- can be written as a series of intermediate halves.

Because the problem is cut in half with every application of the recursive rule, we can say that this method requires log(m) multiplications. The problem also specifies that multiplication is O(n*m), so the total runtime of the method is O(nmlog(m) or just O(nm) if we're considering the biggest leading component.