This assignment consists of 5 problems:

1. basic exponentiation
2. modular exponentiation
3. generate a Diffie-Hellman keypair, given an order
4. implement a parameter checker that takes as input a candidate set of Diffie-Hellman parameters, and returns a boolean indicating whether paramters are valid
5. given a public key, modulus, and base as inputs, return a generated public key and the shared secret

Here is the template for solution. The solution needs to be in python 3.10 or higher

import typing

def problem1(b: int, e: int) -> int:

 """

 Return base `b` raised to the exponent `e`

 >>> problem1(2, 3)

 8

 """

def problem2(b: int, e: int, m: int) -> int:

 """

 Return base `b` raised to the exponent `e` modulo prime modulus `m`

 >>> problem2(2, 3, 5)

 3

 """

class DHKeyPair(typing.TypedDict):

 """

 A wrapper type representing a Diffie-Hellman keypair, consisting of public

 key `A` and private exponent `a`.

 >>> DHKeyPair(a=1, A=2)

 {'a': 1, 'A': 2}

 >>> DHKeyPair({'a': 1, 'A': 2})

 {'a': 1, 'A': 2}

 """

 A: int

 a: int

def problem3(g: int, p: int) -> DHKeyPair:

 """

 Given a generator `g` and prime modulus `p`, return a valid Diffie-Hellman

 keypair under `p` and `g`. The keypair should be returned a dict with the

 private exponent `a` keyed by `'a'` and the public key `A` keyed by `'A'`.

 Recall that private exopnent `a` is computed as a random integer, and that

 public key `A` is computed as `g^a mod p`.

 # not doctest as output is random

 > problem3(7, 17)

 {'a': 8, 'A': 16}

 > problem3(7, 17)

 {'a': 12, 'A': 13}

 """

def problem4(g: int, p: int, a: int, A: int) -> bool:

 """

 Given a generator `g`, prime modulus `p`, private exponent `a`, and Alice's

 public key `A`, return a boolean indicating whether the parameter set is

 valid.

 Recall that:

 - trivial exponents (i.e. 0, 1) are invalid

 - the generator must me less than the modulus

 - private exponent `a` must be greater than generator `g` and less than

 prime modulus `p`: `g < a < p`.

 - because the public key is computed modulo `p`, it must be less than

 `p`

 - `A` must be computed as `g ^ a mod p`

 >>> problem4(5, 17, 0, 6)

 False

 >>> problem4(20, 17, 3, 6)

 False

 >>> problem4(5, 17, 3, 20)

 False

 >>> problem4(7, 17, 12, 13)

 True

 """

class DHNegotiatedSecret(typing.TypedDict):

 """

 A wrapper type representing a Diffie-Hellman secret, consisting of secret

 `s` and public key `A`.

 >>> DHNegotiatedSecret(s=1, A=2)

 {'s': 1, 'A': 2}

 >>> DHNegotiatedSecret({'s': 1, 'A': 2})

 {'s': 1, 'A': 2}

 """

 s: int

 A: int

def problem5(

 g: int, p: int, B: int, b: typing.Optional[int] = None

) -> DHNegotiatedSecret:

 """

 Given a generator `g`, prime modulus `p`, and Bob's public key `B`, first

 compute a valid Diffie-Hellman keypair for Alice consisting of public key

 `A` and private exponent `a`, using `g` and `p`. Then, using your private

 exponent `a`, compute the shared secret `s`. Return a DHNegotiatedSecret

 dict with your public key `A` keyed by `'A'` and the shared secret `s`

 keyed by `'s'`.

 Recall that Alice computes the shared secret `s` by raising Bob's public

 key `B` to their (Alice's) private exponent `a`, all modulo `p`. As an

 equation, this looks like `s = B^a mod p`.

 Please note that the optional parameter `b` is \*\*not required for your

 solution\*\*, and is only there for use by the auto-grader.

 # not doctest as output is random

 > problem5(5, 17, 9)

 {'A': 4, 's': 16}

 > problem5(5, 17, 9)

 {'A': 10, 's': 2}

 """