Faculty of Arts and Sciences
Department of Computer Science
CMPS 200 – Introduction to Programming
Exam 3

Name: **ID number:**

---

1. Polynomials. (40%)
A degree n polynomial is a function defined by a list of $n+1$ coefficients as follows:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots = \sum_{k=0}^{n} a_k x^k$$

Write a class `Polynom` for representing and operating on polynomials. A polynomial may be represented by an ordered list of the $n+1$ coefficients of the $n+1$ terms of the polynomial, ordered from degree 0 to degree n. Your class should implement the interface below, in a file `polynom.py`.

- `__init__()`, a constructor that takes in a list of tuples, each tuple containing two integers representing the coefficient and degree of one term of the polynomial, respectively.
  For example, the call `p1 = Polynom([(2, 0), (4, 2), (1, 3)])` constructs the polynomial object $2 + 4x^2 + x^3$. The internal representation will be the list `[2, 0, 4, 1]`.
- `__str__()`, which returns a string representation of a polynomial.
  For example, the call `print(p1)` prints: `<Polynomial, degree 3: 2 + 0x + 4x^2 + 1x^3>`
- `__eq__()`, to test for equality (==) of two polynomials.
- `__add__()`, an addition method (+) that returns the sum of two polynomials.
- `__mul__()`, a multiplication method ($*$) that returns the product of two polynomials.
  Note: Remember that the product of two polynomials is obtained by adding the product of every term of the first one by every term of the second one. Also remember that the product of two terms involves multiplying their coefficients while adding their degrees.

Below is a test function you may use to test your class:

```
def test():
  p1 = Polynom([(2, 0), (4, 2), (1, 3)])                    # 2 + 4x^2 + x^3
  p2 = Polynom([(1, 1), (1, 2)]                             # x + x^2
  p3 = p1 + p2
  p4 = p1 * p2
  answer1 = Polynom([(2, 0), (1, 1), (5, 2), (1, 3)])       # 2 + x + 5x^2 + x^3
  answer2 = Polynom([(2, 1), (2, 2), (4, 3), (5, 4), (1, 5)])  # 2x + 2x^2 + 4x^3 + 5x^4 + x^5

  print(p3 == answer1)   # should print True
  print(p4 == answer2)   # should print True
```

Answer the following questions about the methods you wrote:

- What is the asymptotic complexity of the addition method you wrote above?

- What is the asymptotic complexity of the multiplication method you wrote above?

2. Exponentiation. (30%)
We consider a function that raises a polynomial to the power d,

$$q(x) = p(x)^d$$

In particular we consider two versions of the function:

- the first version `power1(p, d)` works by multiplying p by itself $d-1$ times.
- the second version `power2(p, d)` works by observing that $p(x)^d = p(x)^{d/2} * p(x)^{d/2}$. Therefore, it may be computed by first computing $p(x)^{d/2}$ and multiplying that result by itself. $p(x)^{d/2}$ may be computed recursively using the same idea. You may assume that d is a power of two (i.e., 2, 4, 8, 16, 32, etc.) in this problem.

Write your functions `power1()` and `power2()` in a file `power.py`. You may test your functions using the following test code:

```
from polynom import Polynom           # import the Polynom class from polynom.py

YOUR CODE HERE

p1 = Polynom([(1, 0), (1, 1)]          # the polynomial 1 + x
print(power1(p1, 4))                   # should print
                                       # <Polynomial, degree 4: 1 + 4x + 6x^2 + 4x^3 + x^4>
print(power1(p1, 8) == power2(p1, 8))  # should print True
```

Answer the following questions about the functions you wrote above:

- What is the asymptotic complexity of `power1()` in terms of d? Explain very briefly.

- What is the the asymptotic complexity of `power2()` in terms of d? Explain very briefly.

3. Shuffle. (30%)
In this program we consider the following strategy for shuffling a list of n integers:

- shuffle the left half and the right half of the list separately. If the list has an odd length, the size of the two halves will differ by one.
- combine the two shuffled lists by choosing an element *randomly* either from the top of the first or the second list, and appending it to a result list to be returned, in a strategy similar to the merging step of mergesort.
  In other words, we loop n times and at each iteration we get a 0 or 1 random number (`random.randint(0, 1)`), and use this random value to choose the top element either from the first list or from the second list, to append it to the result. If one of the lists is empty we cannot, of course, append an element from it, and can simply append the remaining elements of the other list to the result.

The left and right halves of the list can be shuffled using the same strategy, recursively,

Write a recursive function `shuffle(lst)` that takes in a list of integers and returns a list containing the same intergers in a shuffled order. Your function should use the shuffling algorithm described above.

A test example of your function might be as follows:

```
lst1 = [i for i in range(20)]    # lst1 is [0, 1, 2, ...., 19]
print(shuffle(lst1))             # will print the numbers 0 through 19 in a shuffled order
```

Answer the following questions about the function you wrote above:

- Write the recurrence equation that describes the asymptotic complexity of the shuffling algorithm

- What is the solution of this recurrence equation?

Submission.

- Zip the three `.py` files above (`polynom.py`, `power.py`, and `shuffle.py`) in a single archive file `exam3_`***`netid_token`*** where ***`netid`*** is your AUBnet user name, and ***`token`*** is your individualized 4-letter string distributed to you. Submit to Moodle.
- Turn in your exam sheet and don't forget to write your name on it.