코드를 분석하여 시간 복잡도 및 점근적 상한 함수(big-oh, O)를 구현해야함. 점근적 상한함수는 “최악의 상황”을 고려한다는 것에 유의한다.

- “complexity.py”에 각 문제에 대한 시간 복잡도 및 점근적 상한 함수를 구현한다. 파일명 및 함수명은 대소문자를 구분하여 본 문서에 정의된 대로 작성해야함
- 각 문제에 대한 problemxx() 함수를 구현한다.
- 문제에서 f(n)과 g(n)은 각각 시간 복잡도 함수 및 점근적 상한 함수이다.
• f(n): Time complexity function
• g(n): Asymptotically upper bound function
- 각 문제에서 Ti 는 시간 비용을 나타내며, Ti 를 합하여 f(n)을 정확하게 구현해야 한다. 즉, g(n)은 다양한 정답이 존재할 수 있지만, f(n)은 정답이 정해져 있다.
- c와 n0는 점근적 상한 함수를 정의할 때 임의로 정하는 상수이며, c와 n0는 각각 실수 및 정수로서 0보다 크고 11보다 작다 ($1 \leq c, n0 \leq 10$).
- problemxx() 함수는 분석한 코드에 대한 f, g, c, n0를 반환한다. 함수 f와 g를 problemxx()의 내부에 정의하도록 한다 (내재 함수, nested function).
- 테스트는 c • g(n)이 f(n)의 밀착 상한(tight upper bound)인지 확인하기 위하여 n에 n0보다 큰 수를 대입하여 c • g(n)과 f(n)의 비율, c • g(n)/f(n)을 검사하는 방식으로 이루어진다. c • g(n)/f(n)은 20보다 작아야 한다.

$$\lim_{n\to\infty} \frac{c \cdot g(n)}{f(n)} \leq 20$$

(1) 아래 maximum 함수(seq의 길이는 n)의 시간 복잡도를 분석하여 f(n), g(n), c, n0를 반환하는 함수를 구현하시오.

```
➔ def problem01() -> (f, g, c, n0)
  def maximum(seq):
      maxval = -1                  T1 = 1
      for elem in seq:             T2 = 1
          if elem > maxval:        T3 = 1
              maxval = elem        T4 = 1
      # end of for
      return maxval                T5 = 1
```

$$\begin{aligned} f(n) &= T_1 + (T_2 + T_3 + T_4)n + T_5 \\ &= (T_2 + T_3 + T_4)n + (T_1 + T_5) \\ &= 3n + 2 \\ &\leq c \cdot n, \text{ for } c = 4, n \geq n_0 = 2. \\ g(n) &= n. \\ f(n) &\in O(g(n)). \end{aligned}$$

(2) 아래 multiply_allpairs 함수(seq의 길이는 n)의 시간 복잡도를 분석하여 f(n), g(n), c, n0를 반환하는 함수를 구현하시오.

```
➔ def problem02() -> (f, g, c, n0)
  def multiply_allpairs(seq):
      allpairs = []                T1 = 1
      for x in seq:                T2 = 1
          pairs = []               T3 = 1
          for y in seq:            T4 = 1
              pairs.append(x*y)    T5 = 1
          # end of for
          allpairs.append(pairs)   T6 = 1
      # end of for
      return allpairs              T7 = 1
```

(3) 아래 multiply_square_matrices 함수(a, b는 n×n 행렬)의 시간 복잡도를 분석하여 f(n), g(n), c, n0를 반환하는 함수를 구현하시오.

```
➔ def problem03() -> (f, g, c, n0)

  def multiply_square_matrices(a, b):
      n = len(a)                                    T01 = 1
      c = []                                        T02 = 1
      for i in range(n):                            T03 = 1
          c.append([])                              T04 = 1
          for _ in range(n):                        T05 = 1
              c[i].append(0.0)                      T06 = 1

      for i in range(n):                            T07 = 1
          for j in range(n):                        T08 = 1
              for k in range(n):                    T09 = 1
                  c[i][j] += (a[i][k] * b[k][j])    T10 = 1

      return c                                      T11 = 1
```

(4) 아래 DynamicArray.append 함수(self._length가 n을 의미)의 시간 복잡도를 분석하여 f(n), g(n), c, n0를 반환하는 함수를 구현하시오. (점근적 상한이 최악의 상황을 가정한다는 것에 유의하여 resize의 매개변수 capacity가 n에 따라 어떻게 변하는지 분석할 필요가 있다.)

```
➔ def problem04() -> (f, g, c, n0)


  class DynamicArray:
      def __init__(self):
          self._capacity = 1
          self._length = 0
          self._arr = self._capacity * [None]

      def __str__(self):
          return str(self._arr)

      def __len__(self):
          return self._length

      def __getitem__(self, i):
          return self._arr[i]

      def resize(self, capacity):
          arr_new = capacity * [None]       T01 = capacity
          for i in range(self._length):     T02 = 1
              arr_new[i] = self._arr[i]     T03 = 1

          self._arr = arr_new               T04 = 1
          self._capacity = capacity         T05 = 1

      def append(self, elem):
          if self._length == self._capacity: T06 = 1
              self.resize(2 * self._length)

          self._arr[self._length] = elem    T07 = 1
          self._length += 1                 T08 = 1

      @property
      def capacity(self):
          return self._capacity
```

(5) 아래 repeat_duplication 함수의 시간 복잡도를 분석하여 f(n), g(n), c, n0를 반환하는 함수를 구현하시오. (x0는 정수형 매개변수로 가정한다.)

```
➔ def problem05() -> (f, g, c, n0)

    def repeat_duplication(n, x0):
        x = [x0]                        T01 = 1
        for i in range(n):              T02 = 1
            y = []                      T03 = 1
            for elem in x:              T04 = 1
                y.append(elem)          T05 = 1
                y.append(elem + 1)      T06 = 1
                y.append(elem + 1)      T07 = 1
            x = y                       T08 = 1
        return x                        T09 = 1
```