資料結構

HW1

學號 : B103012002

姓名 : 林凡皓

1. **Programming environment :**

這次作業我使用Visual Studio Code做為開發環境和Python作為開發語言。要在Visual Studio Code中安裝Python的開發環境需先到Python的官方網站下載Python，並在Visual Studio Code中安裝好Python的延伸模組。由於本次作業在資料視覺化的部分有式用到Python套件matplotlib，因此需要到cmd輸入pip install matplotlib。

1. **Design of my program :**

* \_\_init\_\_ :

此函數的主要功能為接收一個coefficient list並將此list儲存到self.\_coeff中，以及根據此list計算出多項式的最高次方。

關於self.\_coeff的部分，由於coefficient list的前幾個位元如果為0將會被忽略，因此我使用while loop來將前面所有0移除後再將coefficient list assign給self.\_coeff。

關於最高次方的計算，我是透過計算self.\_coeff的長度減一得到。之所以要減一是因為多項式的常數項為x^0。

* \_\_add\_\_ :

此函數的主要功能為將兩個多項式相加。

由於兩個多項式的最高次方不一定相同，因此我將多項式相加想成兩個部分，第一為次方小於等於較小次方的部分，第二為次方大於較小次方的部分。

對於第一部份來說，相加後的結果為兩個多項是直接相加。

對於第二部分來說，相加後的結果其實就是次方數較高的多項式。

要注意的是這樣的思維還需要加上一些list index的操作，這是因為coefficient list的規則為最高次方最靠左。

* \_\_sub\_\_ :

此函數的主要功能為將兩個多項式相減。

解題想法和\_\_add\_\_相同，差別只有在次方小於等於較小次方的部分，原本的兩多項式相加要改為相減，以及要再額外判斷是較大的多項式減較小的多項式，還是較小的多項式減較大的多項式。如果為後者，在次方大於較小次方的部分，相減後的結果會是較大的多項式加上一個負號。

* \_\_mul\_\_ :

此函數的主要功能為將兩個多項式相乘。

本題可以利用巢狀for loop迭代self.\_coeff和other.\_coeff。這邊的for loop需要對enumerate(self.\_coeff)和enumerate(other.\_coeff)迭代，因為會需要同時取得元素的index和數值(self.\_coeff的index會用i表示，other.\_coeff的index會用j表示)。

相乘後的結果的第i + j元素的值會是self.\_coeff第i個元素的值乘上other.\_coeff第j個元素的值。

* \_\_neg\_\_ :

此函數的主要功能為將一個多項式取負號。

本題可以先創建一個空的list，之後利用for loop迭代self.\_coeff的每個元素，並將其加負號後append到剛才創建的空的list中。

1. **Time complexity analysis & benchmarking :**

* \_\_init\_\_ :

1. Big O notation :

一張含有 文字, 螢幕擷取畫面, 陳列, 字型 的圖片

自動產生的描述

Big O 分析結果為O(n)。

1. timeit 驗證 :

\_\_init\_\_中最壞情況為coefficients中的最前面有許多0，如此一來程式執行時會循環while迴圈很多次，因此我設計的實驗就是在coefficients最前面不段加入0。驗證結果與視覺化結果如下 :

一張含有 文字, 螢幕擷取畫面, 字型, 設計 的圖片

自動產生的描述

一張含有 文字, 行, 繪圖, 圖表 的圖片

自動產生的描述

由上圖可以看出來，執行時間隨著0的加入約為線性成長關係。

* \_\_add\_\_ :

1. Big O notation :

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述

Big O 分析結果為O(n+k)。其中n為self.\_coeff的長度，k為other.\_coeff的長度。

1. timeit 驗證 :

\_\_add\_\_時間複雜度主要會和self.\_coeff和other.\_coeff的長度相關，因此我選擇固定其中一個長度(a的長度)，並持續增加另外一個長度(b的長度)來驗證Big O分析結果，驗證結果如下 :

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述

一張含有 文字, 行, 圖表, 繪圖 的圖片

自動產生的描述

由上圖可以看出，不管是固定a或b的長度，隨著另外一個長度增加，執行時間約為線性成長，因此Big O notation應該為O(n+k)。

* \_\_sub\_\_ :

1. Big O notation :

一張含有 文字, 螢幕擷取畫面, 軟體 的圖片

自動產生的描述

Big O分析結果為O(n+k)。其中n為self.\_coeff的長度，k為other.\_coeff的長度。

1. Timeit驗證 :

\_\_sub\_\_時間複雜度主要會和self.\_coeff和other.\_coeff的長度相關，因此我選擇固定其中一個長度，並持續增加另外一個長度來驗證Big O分析結果，驗證結果如下 :

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述

一張含有 文字, 行, 圖表, 繪圖 的圖片

自動產生的描述

由上圖可以看出，不管是固定a或b的長度，隨著另外一個長度增加，執行時間約為線性成長，因此Big O notation應該為O(n+k)。

* \_\_mul\_\_ :

1. Big O notation :

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

Big O分析結果為O(nk)。其中n為self.\_coeff的長度，k為other.\_coeff的長度。

1. Timeit驗證 :

\_\_add\_\_時間複雜度主要會和self.\_coeff和other.\_coeff的長度相乘相關，因此我選擇持續增加self.\_coeff和other.\_coeff的長度來驗證Big O分析結果，驗證結果如下 :

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

一張含有 文字, 行, 螢幕擷取畫面, 圖表 的圖片

自動產生的描述

由上圖可以看出，隨著a, b長度增加，執行時間約為x^2成長，因此Big O notation應該為O(nk)。

* \_\_neg\_\_ :

1. Big O notation :

一張含有 文字, 螢幕擷取畫面, 軟體, 多媒體軟體 的圖片

自動產生的描述

Big O分析結果為O(n)。

1. Timeit 驗證 :

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

一張含有 文字, 行, 繪圖, 圖表 的圖片

自動產生的描述

由上圖可以看出來，執行時間隨著a的長度變長約為線性成長關係。

1. **總結與心得 :**

這次作業主要用的資料結構為list。

在本次作業中，我花了需多時間在熟悉Python class的語法和使用。我在做作業時有遇到一個問題就是假設我在\_\_neg\_\_函數中不是透過創建一個新的list，而是直接對self.\_coeff做修改並將修改的結果return，那我只要呼叫這個函數，例如x5 = - x2，那x2的self.\_coeff也會被修改掉。所以不可以直接對self.\_coeff做修改(除非本來就是要在呼叫函數時修改self.\_coeff)，而是應該要創建一個新的list並對這個新的list做修改。

除此之外，本次作業也讓我對list的操作更加熟悉，特別是利用for loop 迭代一個list，以及list index operation的部分，這都是在這次作業中大量使用到的技巧。

最後就是時間分析的部分，上課時聽教授講說timeit就是要先創間一個虛擬的執行環境，然後你的主程式為\_\_main\_\_，因此會需要用form \_\_main\_\_ import會用到參數。當時聽到只覺得這是什麼，好複雜，但是經過這次時作後發覺到這其實沒有那麼難理解，而且對於創建一個虛擬環境以及form \_\_main\_\_ import這兩個部分都有更深的認識。另外我也自己嘗試是用matplotlib來將我分析的結果視覺化，在這過程中我除了多了解到matplotlib一些基礎的使用方式之外，也透過視覺化的幫助讓我更加清楚我的code的時間複雜度。

1. **Reference :**

[1] OpenAI. (2023). ChatGPT (Mar 14 version) [Large language model]. <https://chat.openai.com/>

[2] “matplotlib.pyplot”

<https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html>

[3] Vivian Lo (2019) “[演算法]Big O and Time Complexity”. <https://medium.com/@yunyubee/%E6%BC%94%E7%AE%97%E6%B3%95-big-o-and-time-complexity-65f2dfafe9d1>