郭桐維教授

資料結構

作業一

國貿二 郭許謙 110301034

一、Dynamic array 與 Linked list 比較實驗

(一)實驗方法

實驗簡介

這次的實驗是在 Python 的環境下完成，分別打造了Dynamic array 和 Linked list，而在分別利用這兩個資料結構完成兩場實驗，其一儲存資料並記錄其時間；其二，加總實驗一所儲存的資料，並記錄其時間。而兩場實驗分別都做五次，最後圖表的數據是五次數據後的平均。

實驗流程

Dynamic array :

1) 建造一個長度為1、裡面儲存0的array。
2) 設一個長度為2的26次方的 for 迴圈，讓dynamic array 存2的26次方個電腦當下的時間點。
3) 在每一圈迴圈都會判斷當前是否為2的整數次方的迴圈，若為2的整數次方的迴圈，儲存當下實驗流程進行時間。
4) 儲存完資料後，在用另外一個迴圈去加總所有儲存資料，並在2的整數次方時紀錄時間。

Linked list :

1) 建造一個長度為1、沒有儲存東西的 Linked list。
2) 設一個長度為2的26次方的 for 迴圈，讓linked list 存2的26次方個電腦當下的時間點。
3) 在每一圈迴圈都會判斷當前是否為2的整數次方的迴圈，若為2的整數次方的迴圈，儲存當下實驗流程進行時間。
4) 儲存完資料後，在用另外一個迴圈去加總所有儲存資料，並在2的整數次方時紀錄時間。

實驗控制

這次的實驗都只進行到2的26次方，其中很大的原因是電腦Ram 的儲存空間，我的電腦在跑完2的26次方後過很久(約一小時)都沒有跑完2的27次方，雖然我的電腦在並沒有直接顯示記憶體不足，不過我用以下兩點觀察推論是記憶體空間不足所至：

1. 作業系統系統監視器

我的電腦Ram 的空間大小為8GB，在實驗開始之前我的電腦的記憶體壓力都屬於正常，不過當實驗開始後記憶體用量就急速上升，記憶體壓力也開始變成警告(紅色)，實驗約莫過2的26次方後記憶體壓力及記憶體就停止成長，進入高原期，只有python 的虛擬記憶體持續增加。

(圖一)執行程式時的記憶體壓力--最左邊為啟動時點

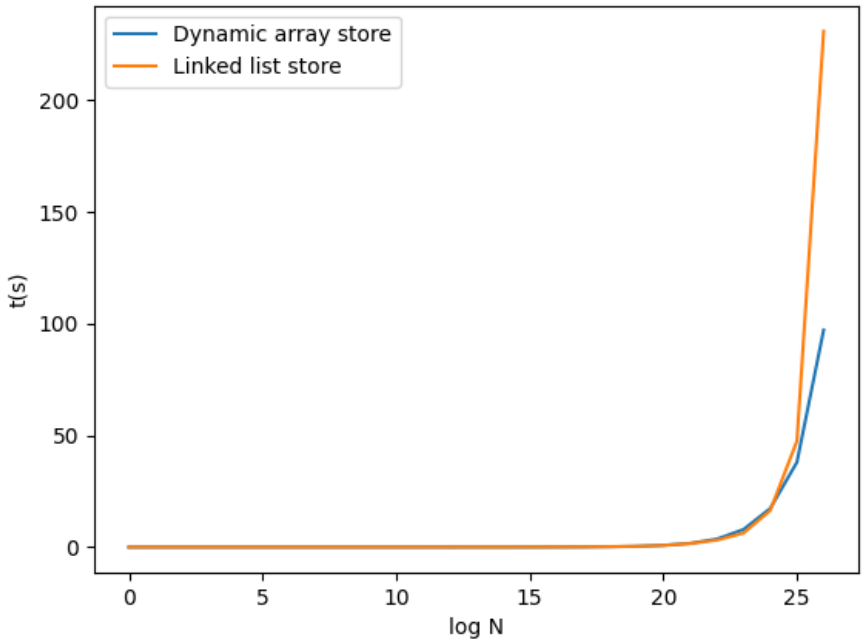
在(圖一)中我們可以觀察記憶體在程式執行時的變化，我認為記憶體壓力的兩次下修應該是作業系統在釋放其他程式閒置記憶體空間給Python 使用，因此在下修後都又快速的往上衝。

(圖二)結束程式時的記憶體壓力--黃綠交接處為結束時點

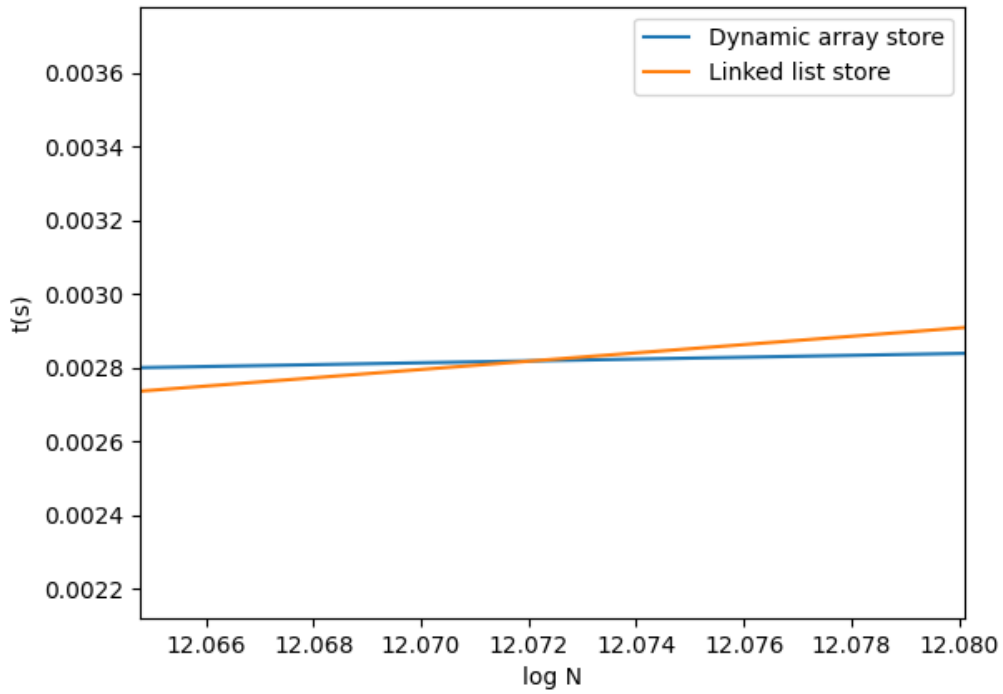
在(圖二)中我們可以發現記憶體在結束之前已經要不到更多的空間，另外在結束後記憶體壓力下修到6.65GB，而我們可以觀察程式結束後和(圖一)程式開始之前的記憶體壓力有很大的落差，而此正呼應前段所提，在(圖一)所觀察到的兩次下修，應該是作業系統在為Python 清出閒置程式或App的記憶體空間。

2. 電腦出現明顯卡頓
    a. 在跑第26次方之後，由於因為等太久，我想直接把程式關掉(control ^C)，不過我按了兩三次都沒有反應，直到過五分鐘程式才結束。當時的ram 依舊維持在7.34 / 8 GB，不過虛擬記憶體已經衝到19.5GB，我認為此卡頓是因Python 大到吃掉作業系統的效能所致，而在這個情況下應該沒辦法在繼續跑2的27次方以上。

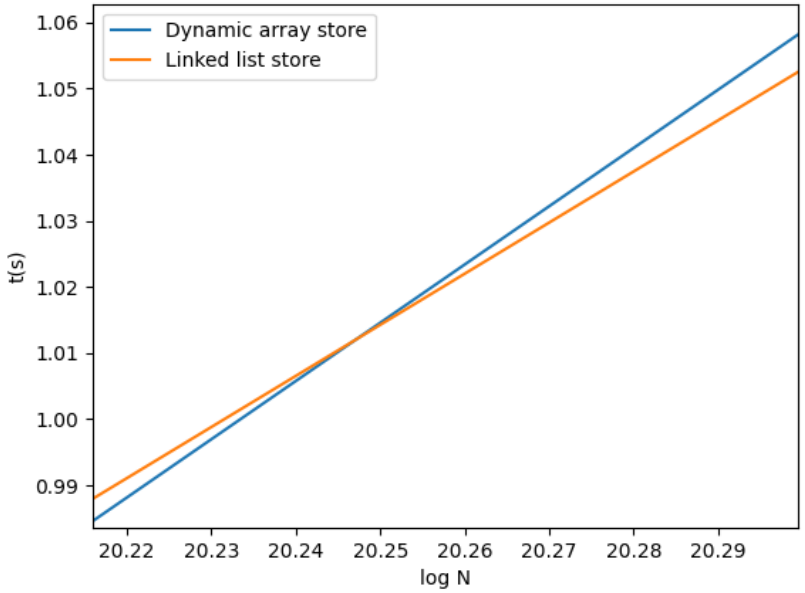
（二）Dynamic array vs Linked list traverse 耗時比較



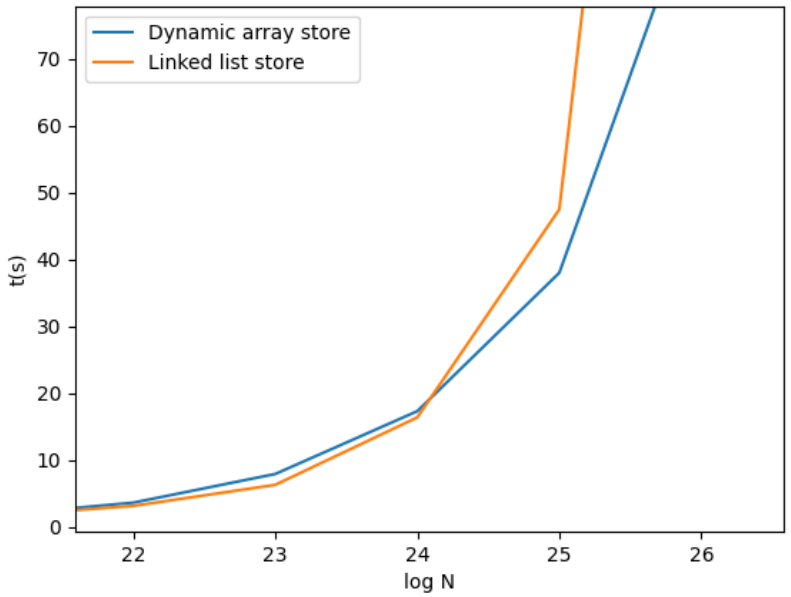
（圖三）Dynamic array 和 Linked list 儲存時間比較



（圖四）第一次交叉



（圖五）第二次交叉



（圖六）第三次交叉

實驗觀察

在 Dynamic array 和 Linked list 的耗時比較中，我們可以從(圖三)到(圖六)觀察到四個區段的現象：

1) 0 ~ 2^12.072 : Dynamic array 比 Linked list 慢
2) 2^12.072 ~ 2^20.25 : Dynamic array 比 Linked list 快
3) 2^20.25 ~ 2^24 : Dynamic array 比 Linked list 慢
4) 2^24 ~ 2^26 : Dynamic array 比 Linked list 快

問題觀察

1) 實驗結果不合理

   在這個實驗中，若依dynamic array 和linked list 的特性來判斷的話，我們預期到的結果應該是dynamic array 花的時間會大於linked list ，因為dynamic array 儲存的空間越來越大時，會花越多時間來找一段連續的閒置空間。另外若在前期linked list 花比較多時間來複製空間的話，那兩條線至多也只能交錯一次(程式穩定執行的情況下)，而實驗結果卻呈現3次的交錯，這表示電腦在程式執行期間有發生一些事情。

2) 發生 Flash 現象

   在(圖三)中，我們可以發現linked list 在2的26次方時，耗時突然暴增，我推測這是儲存記憶體過大而發生的 Flash 現象，因為在我的程式裡，dynamic array 一次是存24byte，而linked list 則是 48byte，因為儲存空間差兩倍，導致同樣存2的26筆資料，dynamic array 的第2的26次方時的耗時卻沒有突然暴增。

3) 實驗方法不合理

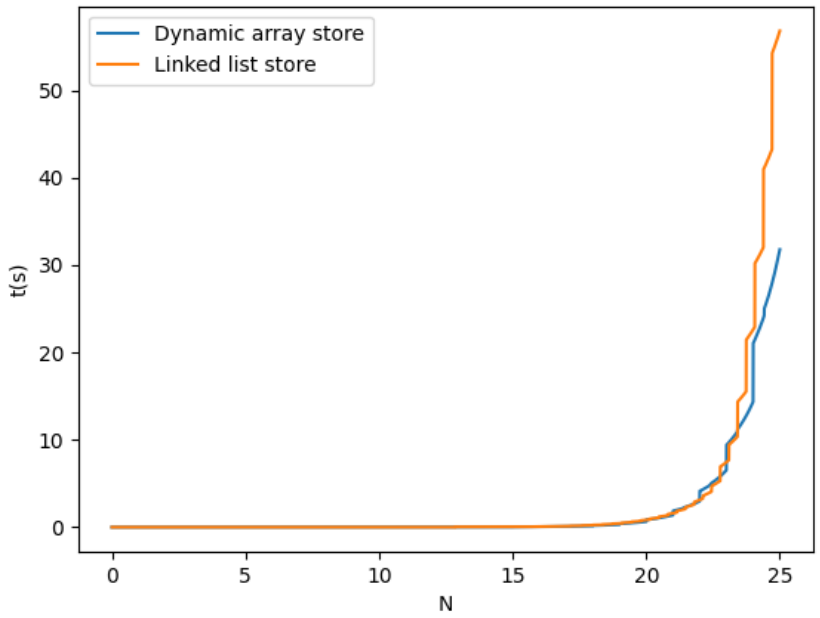
   我認為這種無法解釋的交錯很有可能是因為圖表呈現的數據是取平均的結果所至

問題修正

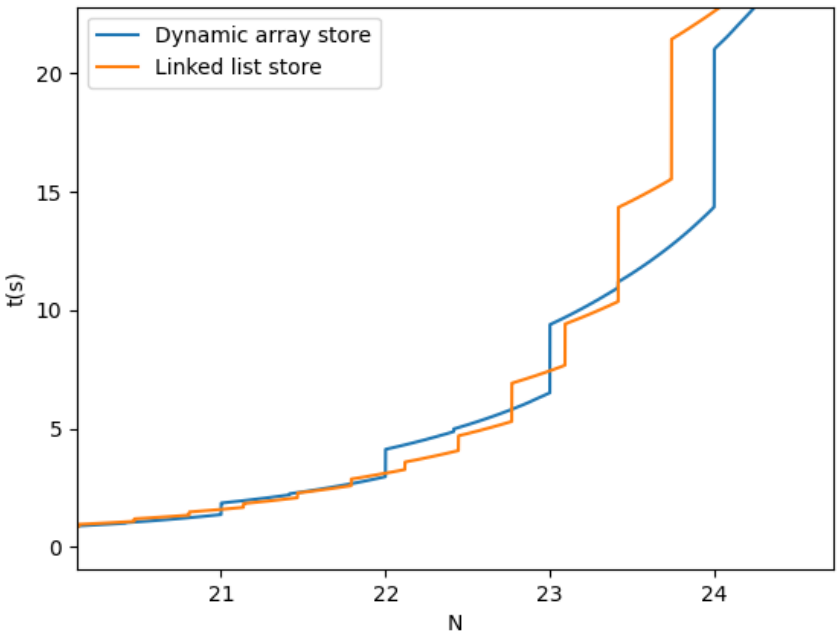
為了找出這些電腦在這期間發生的事情，我提出以下幾點修正，幫助我能更正確的找到產生上述實驗結果真正的原因。

1) 實驗規模改為2的25次方
2) 不採用平均數來製圖
3) 紀錄每新增一個資料的時點並點在圖上

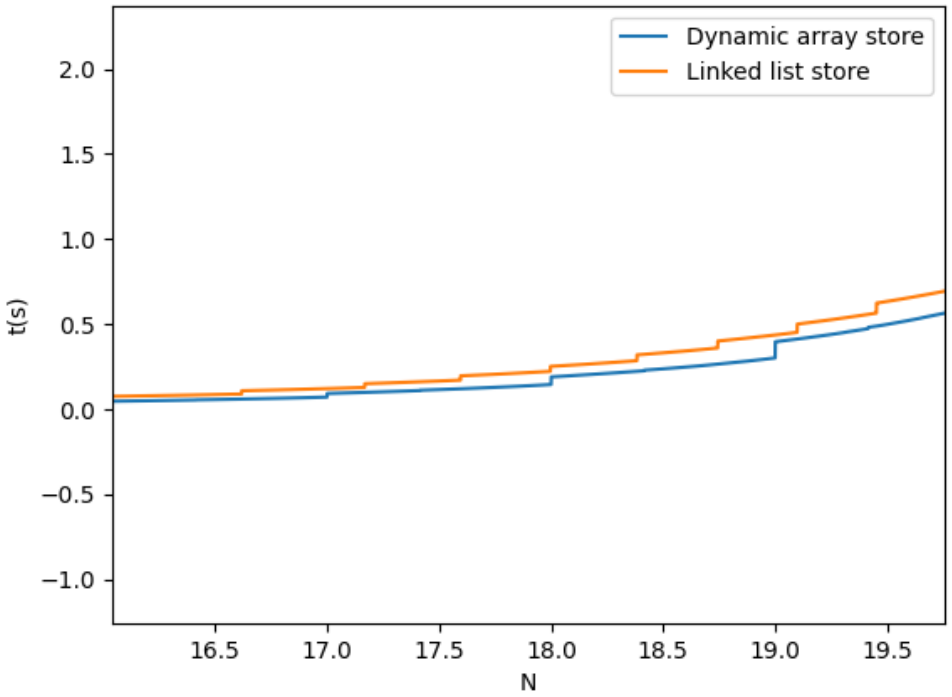
修正後的實驗結果



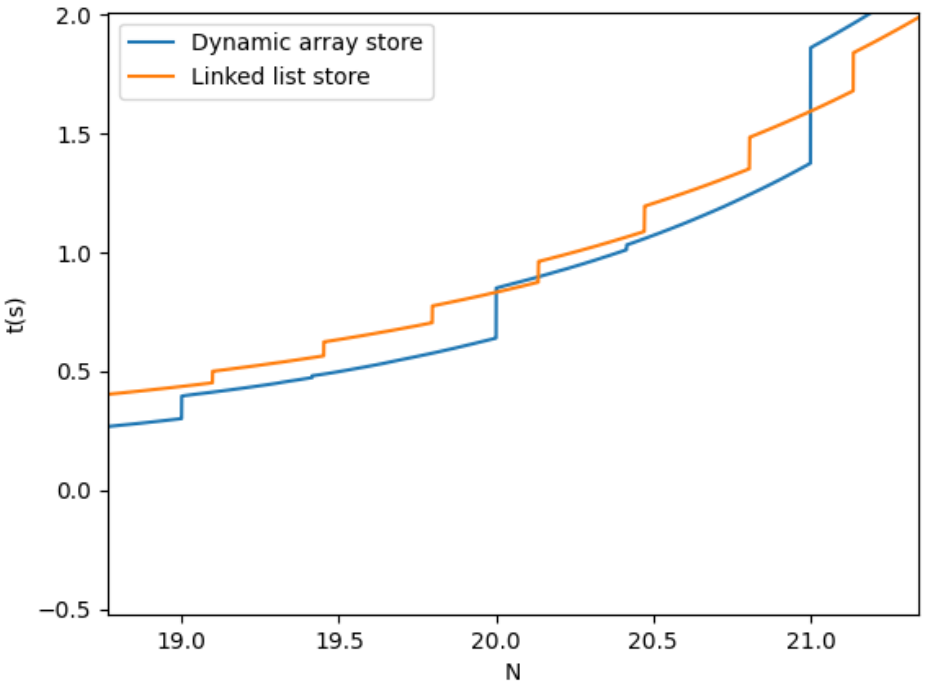
（圖七）Dynamic array 和 Linked list 儲存時間比較（修正後）



（圖八）多次交叉（修正後）



（圖九）明顯階梯狀（修正後）



（圖十）交叉，對應(圖五)

修正後的實驗結果解讀

1) Dynamic array 的階梯狀

從(圖九)和（圖十）中，我們都可以觀察到 Dynamic array 在程式執行的時候是呈現階梯狀的，而階梯上突然垂直上升的點都是落在2的整數次方上面，而我的dynamic array 裝滿要擴充時的resize 倍數剛好是兩倍，因此我判斷會造成圖形階梯狀的原因是dynamic array 在找更大的連續空間和複製資料的耗時。

2) Linked list 的階梯狀

若依正常的程式碼來看 linked list 照理不應該會出現階梯狀，不過我們還是可以從上圖中觀察出這種現象，我推論那些階梯狀所呈現的耗時應該是電腦ram 的空間不足，導致電腦需要花時間去壓縮其他閒置程式的記憶體空間，我們可以從(圖一)、(圖二)觀察出程式執行前的記憶體使用量是6.65GB ，而程式執行期間最多的記憶體使用量是 7.12 GB ，也就是說我們的程式所耗費的記憶體用量為0.47GB(504658657.28 byte)，而linked list 儲存一筆資料的空間為48 byte，也就是說0.47GB只夠我們存約1051萬筆資料（約2的23次方），也就是說在發生flash 之前，電腦必須壓縮其他閒置程式的記憶體空間，而電腦壓縮其他記憶體空間的耗時應該就是階梯狀的主因。

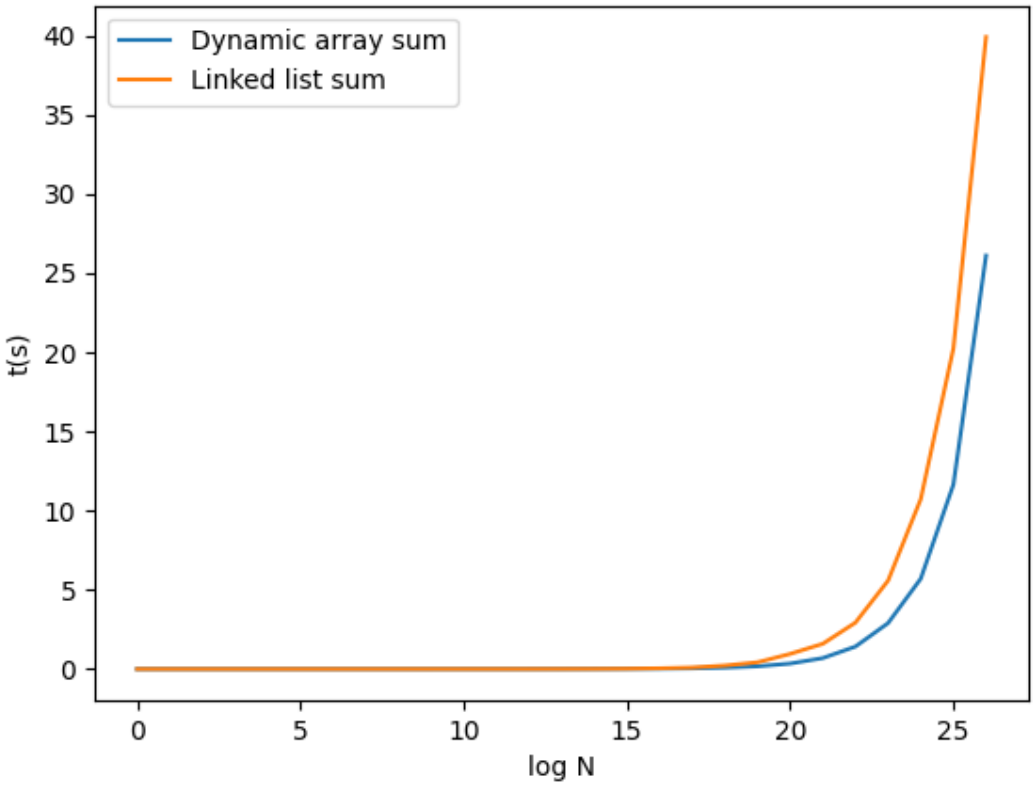
3) 交叉原因

就上述兩點的推測而言，在電腦記憶體不足的情況下迫使linked list 也發生要去壓縮記憶體的情況，而當兩個都是階梯狀的圖更容易產生交錯。

實驗預測

若在記憶體充足的情況下，linked list 的實驗應該會比dynamic array 快許多，且實驗數據越大，差距也會越大。

（三）Dynamic array vs Linked list sum 耗時比較



（圖七）Dynamic array 和 Linked list 加總時間比較

實驗解讀

在這個加總實驗中，我們可以發現linked list 所花費的時間比dynamic array 還要多，我認為這是合理的，因為linked list 相對要花比較多時間去搜尋下一筆資料。

不過我們也可以發現在2的20次方之後兩者的時間都有明顯的增長，而這表示在這個區間的資料，很有可能是被存到硬碟裏面。

二、實作程式碼解釋

（一）Dynamic array source code

建造一個dynamic array的程式碼是我在網路上找的，他是用Python 的<ctype >library 裡面的array，這個array 和c的array一樣，都必須先指定大小，並儲存在一個連續的空間。這個程式的append function 也符合dynamic array 的原理，新複製一個兩倍大的空間，再把資料抄過去。

資料來源：

https://www.geeksforgeeks.org/implementation-of-dynamic-array-in-python/

```
'''
Dynamic array reference

https://www.geeksforgeeks.org/implementation-of-dynamic-array-in-python/
'''

class DynamicArray(object):
    '''
    DYNAMIC ARRAY CLASS (Similar to Python List)
    '''

    def __init__(self):
        self.n = 0 # Count actual elements (Default is 0)
        self.capacity = 1 # Default Capacity
        self.A = self.make_array(self.capacity)



    def __len__(self):
        """
        Return number of elements sorted in array
        """
        return self.n

    def __getitem__(self, k):
        """
        Return element at index k
        """
        if not 0 <= k <self.n:
            # Check it k index is in bounds of array
            return IndexError('K is out of bounds !')

        return self.A[k] # Retrieve from the array at index k
```

```
  def append(self, ele):
      """
      Add element to end of the array
      """
      if self.n == self.capacity:

          # Double capacity if not enough room
          self._resize(2 * self.capacity)


      self.A[self.n] = ele # Set self.n index to element
      self.n += 1



  def _resize(self, new_cap):
      """
      Resize internal array to capacity new_cap
      """
      B = self.make_array(new_cap) # New bigger array

      for k in range(self.n): # Reference all existing values
          B[k] = self.A[k]

       del self.A
      self.A = B # Call A the new bigger array
      self.capacity = new_cap    # Reset the capacity




  def make_array(self, new_cap):
      """
      Returns a new array with new_cap capacity
      """
      return (new_cap*c.py_object)()
```

（二）Linked list source code

Link list 的source code 也是在網路上找的，他是分別建造 node 和 single link list 的 class，append 方式就是去建一個新node 並連結上一個node 。

資料來源：

https://lovedrinkcafe.com/python-single-linked-list/

```
'''
Linked list reference

https://lovedrinkcafe.com/python-single-linked-list/
'''



class Node:
    def __init__(self ,data=None, next=None):
        self.data = data
        self.next = next

class SingleLinkedList:
    def __init__(self):
        self.head = None
        self.tail = None

    def append(self, data):
        new_node = Node(data)


        if self.head == None:
            self.head = new_node
            self.tail = new_node
```

以下實驗的方法都很單純，我都是用一個for 迴圈去來去儲存當下的時點，並且在的整數次方時的時點存的我要畫圖的list 裡面。

（三）Dynamic array store experiment

```
#Dynamic array
DyArr_time = DynamicArray()
start_time_DA = time.time()

for i in range(1,2**26+1):
    DyArr_time.append(time.time())
    if math.log2(i)%1 ==0:
        dyarr_tra_logn.append(DyArr_time[i-1]-DyArr_time[0])

end_time_DA = time.time()
```

```
print("Dynamic array tranversing total spending time:",(end_time_DA -
start_time_DA),"s")
```

（四）Dynamic array sum experiment

```
#Dynamic array (sum)
dyarr_sum = 0

start_time_dasum = time.time()
for i in range(0,2**26):
   dyarr_sum = DyArr_time[i]+dyarr_sum
   if math.log2(i+1)%1 ==0:
       dyarr_sum_logn.append(time.time()-start_time_dasum)
end_time_dasum = time.time()
print("Dynamic array addition total time",end_time_dasum - start_time_dasum,"s")
```

（五）Linked list store experiment

```
#Linked list (transverse)
Linked_list_time = SingleLinkedList()
start_time_LL = time.time()

for i in range(1,2**26+1):
   Linked_list_time.append(time.time())
   if math.log2(i)%1 ==0:
       ll_tra_logn.append(Linked_list_time.tail.data-Linked_list_time.head.data)

end_time_LL = time.time()
print("Linked list traversing total spending time:",(end_time_LL -
start_time_LL),"s")
```

（六）Linked list sum experiment

```
#Linked list (Sum)
ll_sum = Linked_list_time.head.data
temp = Linked_list_time.head
start_time_llsum = time.time()
for i in range(1,2**26):
   temp = temp.next
   ll_sum = ll_sum + temp.data
   if math.log2(i)%1 ==0:
       ll_sum_logn.append(time.time()-start_time_llsum)
```

```
end_time_llsum = time.time()
ll_sum_logn.append(end_time_llsum - start_time_llsum)
print("Linked list addition total time",end_time_llsum - start_time_llsum,"s")
```

三、程式碼

```
from ast import Index
from faulthandler import dump_traceback_later
from turtle import st
import numpy as np
import sys
import ctypes as c
import random as rd
import time
import pandas as pd
import matplotlib.pyplot as plt
import math
import psutil
import os
from pympler.tracker import SummaryTracker
import gc
import csv


'''
   Time list
'''
dyarr_tra_logn =list()
dyarr_sum_logn =list()

ll_tra_logn =list()
ll_sum_logn =list()




'''
Dynamic array reference

https://www.geeksforgeeks.org/implementation-of-dynamic-array-in-python/
'''
```

```
class DynamicArray(object):
    '''
    DYNAMIC ARRAY CLASS (Similar to Python List)
    '''

    def __init__(self):
        self.n = 0 # Count actual elements (Default is 0)
        self.capacity = 1 # Default Capacity
        self.A = self.make_array(self.capacity)



    def __len__(self):
        """
        Return number of elements sorted in array
        """
        return self.n

    def __getitem__(self, k):
        """
        Return element at index k
        """
        if not 0 <= k <self.n:
            # Check it k index is in bounds of array
            return IndexError('K is out of bounds !')

        return self.A[k] # Retrieve from the array at index k

    def append(self, ele):
        """
        Add element to end of the array
        """
        if self.n == self.capacity:

            # Double capacity if not enough room
            self._resize(2 * self.capacity)


        self.A[self.n] = ele # Set self.n index to element
        self.n += 1



    def _resize(self, new_cap):
        """
```

```
        Resize internal array to capacity new_cap
        """
        B = self.make_array(new_cap) # New bigger array

        for k in range(self.n): # Reference all existing values
            B[k] = self.A[k]

         del self.A
        self.A = B # Call A the new bigger array
        self.capacity = new_cap    # Reset the capacity




    def make_array(self, new_cap):
        """
        Returns a new array with new_cap capacity
        """
        return (new_cap*c.py_object)()



'''
Linked list reference

https://lovedrinkcafe.com/python-single-linked-list/
'''


class Node:
    def __init__(self ,data=None, next=None):
        self.data = data
        self.next = next

class SingleLinkedList:
    def __init__(self):
        self.head = None
        self.tail = None

    def append(self, data):
        new_node = Node(data)


        if self.head == None:
            self.head = new_node
```

```
            self.tail = new_node


        else:
            self.tail.next = new_node
            self.tail = self.tail.next




    def delete(self):
        if self.head == None:
            return print("This list is empty")
        else:
            if len(self) == 1:
                self.head = None
            else:
                current_node = self.head
                while current_node.next != None:
                    self.tail = current_node
                    current_node = current_node.next
                self.tail.next = None

    def insert(self, index , data):
        '''
        To insert the data to the specific index
        '''
        if self.head == None:
            print("You can only insert the data to a not empty list")
        if not 1 <= index <= len(self):
            print("{} is not in range".format(index))

        elif index == 1:
            new_node = Node(data)
            new_node.next = self.head
            self.head = new_node
        else:
            cur_idx = 1
            new_node = Node(data)
            current_node = self.head
            while cur_idx+1 != index:
                current_node = current_node.next
                cur_idx += 1
            new_node.next = current_node.next
```

```
            current_node.next = new_node
    def delete_node_by_index(self, index):
        '''
        To delete the specific node
        '''
        if self.head == None:
            print("You can only delete the data from a not empty list")

        if index == 1 and len(self) > 1:
            self.head = self.head.next

        elif index == 1 and len(self) == 1:
            self.head = None
            self.tail = None

        elif 1 < index < len(self):
            cur_idx = 1
            current_node = self.head
            while cur_idx != index:
                previous_node = current_node
                current_node = current_node.next
                cur_idx += 1
            previous_node.next = current_node.next

        elif index == len(self):
            cur_idx = 1
            current_node = self.head
            while cur_idx != index:
                previous_node = current_node
                current_node = current_node.next
                cur_idx += 1
            previous_node.next = None
            self.tail = previous_node

        else:
            print("index out of range")

    def reverse(self):

        # reverse the order of the list
        previous_node = None
        current_node = self.head
        self.tail = current_node
        next_node = current_node.next
```

```
        while next_node is not None:
            current_node.next = previous_node
            previous_node = current_node
            current_node = next_node
            next_node = next_node.next

        current_node.next = previous_node
        self.head = current_node
        return


    def __len__(self):
        length = 0
        current_node = self.head
        while current_node != None:
            length += 1
            current_node = current_node.next
        return length

    def __str__(self):
        current_node = self.head
        chain = []
        index = 1
        while current_node != None:
            chain.append("["+str(index)+"]"+str(current_node.data))
            index += 1
            current_node = current_node.next
        return " --> ".join(chain)


'''
    Read csv
'''
logk_csv = pd.read_csv("Dyarr LL logk time.csv")



'''
    Experiment
'''

#Dynamic array (tranverse)
DyArr_time = DynamicArray()
start_time_DA = time.time()
```

```
for i in range(1,2**26+1):
   DyArr_time.append(time.time())
   if math.log2(i)%1 ==0:
       dyarr_tra_logn.append(DyArr_time[i-1]-DyArr_time[0])

end_time_DA = time.time()
print("Dynamic array tranversing total spending time:",(end_time_DA -
start_time_DA),"s")



#Dynamic array (sum)
dyarr_sum = 0

start_time_dasum = time.time()
for i in range(0,2**26):
   dyarr_sum = DyArr_time[i]+dyarr_sum
   if math.log2(i+1)%1 ==0:
       dyarr_sum_logn.append(time.time()-start_time_dasum)
end_time_dasum = time.time()
print("Dynamic array addition total time",end_time_dasum - start_time_dasum,"s")




#Linked list (tranverse)
Linked_list_time = SingleLinkedList()
start_time_LL = time.time()

for i in range(1,2**26+1):
   Linked_list_time.append(time.time())
   if math.log2(i)%1 ==0:
       ll_tra_logn.append(Linked_list_time.tail.data-Linked_list_time.head.data)

end_time_LL = time.time()
print("Linked list tranversing total spending time:",(end_time_LL -
start_time_LL),"s")



#Linked list (Sum)
ll_sum = Linked_list_time.head.data
temp = Linked_list_time.head
```

```
start_time_llsum = time.time()
for i in range(1,2**26):
   temp = temp.next
   ll_sum = ll_sum + temp.data
   if math.log2(i)%1 ==0:
       ll_sum_logn.append(time.time()-start_time_llsum)
end_time_llsum = time.time()
ll_sum_logn.append(end_time_llsum - start_time_llsum)
print("Linked list addition total time",end_time_llsum - start_time_llsum,"s")




'''
   Print the plot (dynamic array)
'''


#dyarr vs linked list (traverse)

vs_tra = logk_csv[["Dynamic array store","Linked list store"]]
vs_tra.plot(xlabel="log N",ylabel="t(s)")
plt.show()

#dyarr vs linked list (traverse)

vs_sum = logk_csv[["Dynamic array sum","Linked list sum"]]

vs_sum.plot(xlabel="log N",ylabel="t(s)")
plt.show()




'''
   see the size
'''

a = DynamicArray()
a.append(time.time())
print(sys.getsizeof(a[0]))
```

```
b = SingleLinkedList()
b.append(time.time())
b.append(time.time())
print(sys.getsizeof(b.head.data))
print(sys.getsizeof(b.head.next))
```

四、心得、疑問、遇到的困難

遇到的困難

這份作業遇到最大的困難就是當我要做修正的實驗時，我發現我的電腦的記憶體空間真的不太夠用，我把所有資料儲存到dataframe 後，要把圖畫出來就真的是一大考驗，因為整個csv 檔就要2.多GB 而且又要把兩條線畫在同一張圖上，後來才發現我的電腦的記憶體根本不夠讓我一次在圖上畫接近4億個點，所以後來才去瞭解記憶體不足的原因，並把數據量調低之後才開始變得比較順利。

疑問

其實在做實驗時，因為記憶體不足會發生電腦壓縮記憶體的現象，若記憶體真的不足後還會發生flash 的現象，我想問有沒有辦法去做實驗而得知哪時候發生壓縮記憶體、哪時候發生flash ？我原本想試著把這些場外因素排除，畫出單純的兩個資料結構的耗時比較圖，不過我發現我根本不了解作業系統的原理，導致我做不出來。

心得

以下三點是我這次實驗收穫最大的事:

1) 解讀數據的能力

這次的作業讓我深深的認知到解讀數據最重要的是domain knowledge ，因為我本身不是資科系，所以對於很多電腦和作業系統的基本原理，我都需要靠問老師、我爸爸，我才有辦法去靠他們給我的關鍵字找無數個網站去瞭解ram 大致的運行原理，瞭解後也才有辦法去做出最後的解讀。

2) 認識資料結構的重要性

我認為我們去學資料結構和跟去學經濟學的初衷都是很類似的，就是資源有限，這次的實驗對我來說困難重重，因為記憶體的限制所以我自己之後在做圖和讀dataframe 的時候，都遇到了不少困難，不過也幸好有遇到這些困難，也才認知到資料結構對於電腦有限資源的重要性。

3) 做實驗的流程

這點是我覺得這次作業我做不好的地方，這次作業我相信我花的時間一定超過60個小時，到最後會發現自己好像有一點沒有效率，除了花很多時間去瞭解電腦原理之外，花更多時間去處理當Python 做圖因資料太多跑不出來時的問題，簡單來說就是一直去找解決方法，可是又一直不斷發現問題並不是出在這裡，我覺得這樣子很沒有效率，但我也從中學到經驗，我才瞭解做實驗不適合去天馬行空的去想像說哪裡出問題，而是必須要花很多時間去瞭解這個問題的特性，去找出更多線索，在有條理的去推論，才不會花很多時間去處理根本不存在的問題。