郭桐維教授

資料結構

作業三

國貿二 郭許謙 110301034

目錄

一、五種資料結構的比較實驗

(一)實驗方法

實驗簡介

這次的實驗是在 Python 的環境下完成，分別做了 BTree、Treap、Skip List、Hash Table、Array of sorted array、儲存2的k次方（k＝15、16、17、30）筆資料，並各搜尋100000筆資料所花費時間的實驗。結束後再用 Python matplotlib pyplot 的套件將數據視覺化、並檢視實驗數據的合理性。

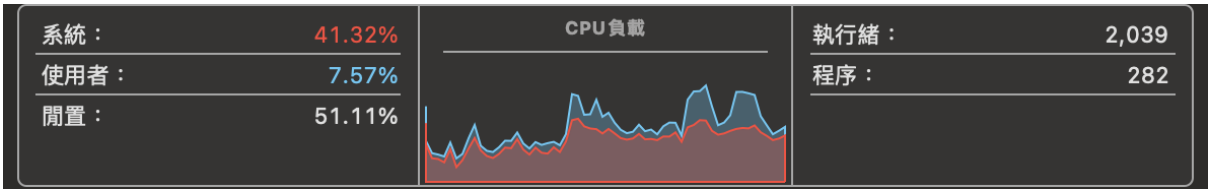
實驗流程

1) 實驗五種資料結構「儲存」2的k次方筆資料所花費的時間。
2) 實驗五種資料結構「搜尋」2的k次方筆資料所花費的時間。
3) 製作各項實驗結果、時間複雜度的圖表。
4) 利用上述圖表以及時間複雜度來分析實驗結果。

各項資料結構實驗能儲存的最大數據

資料結構	測試最大數據	原因
BTree	2的24次方	記憶體不足
Treap	2的23次方	記憶體不足
Skip List	2的23次方	記憶體不足
Hash Table	2的25次方	記憶體不足
Array of sorted array	2的25次方	記憶體不足

（圖一）Ram不足時的記憶體壓力



（圖二）Ram不足時的CPU負載

各項資料結構實驗的實作程式碼來源

資料結構	實作程式碼來源
BTree	使用Python BTrees 套件自行實作
Treap	使用Python Treap 套件自行實作
Skip List	網路上搜尋 Source Code
Hash Table	Python「內建」資料結構 Dictionary
Array of sorted array	自行設計實作

（圖表一）實作程式碼來源

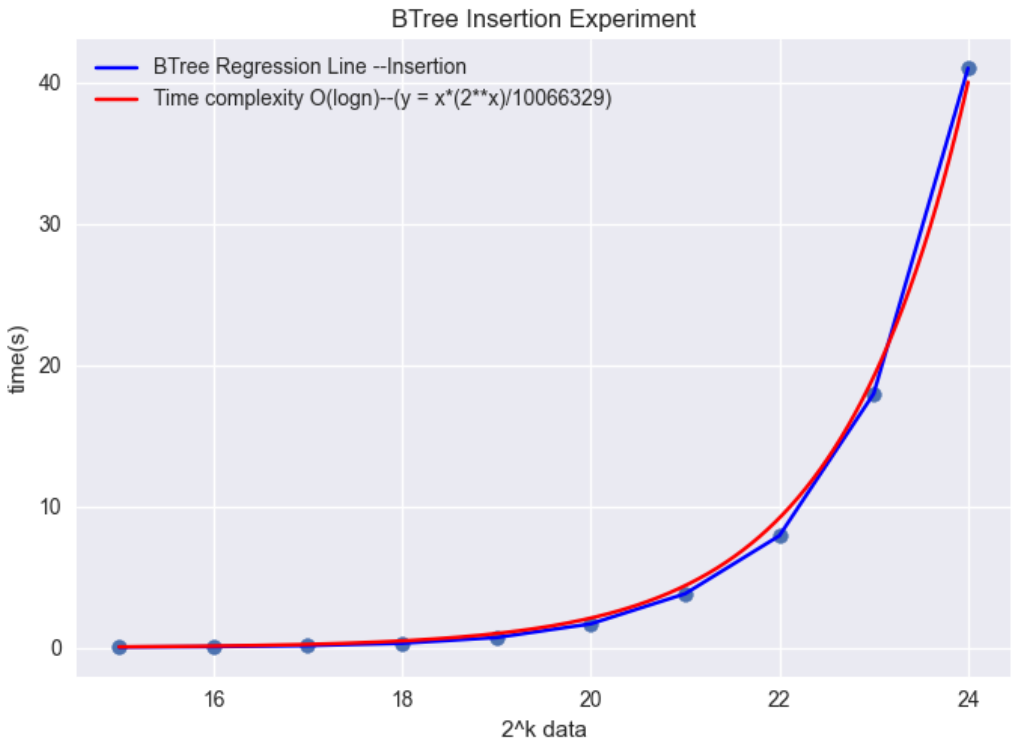
（二）實驗結果

驗證實驗數據「合理性」

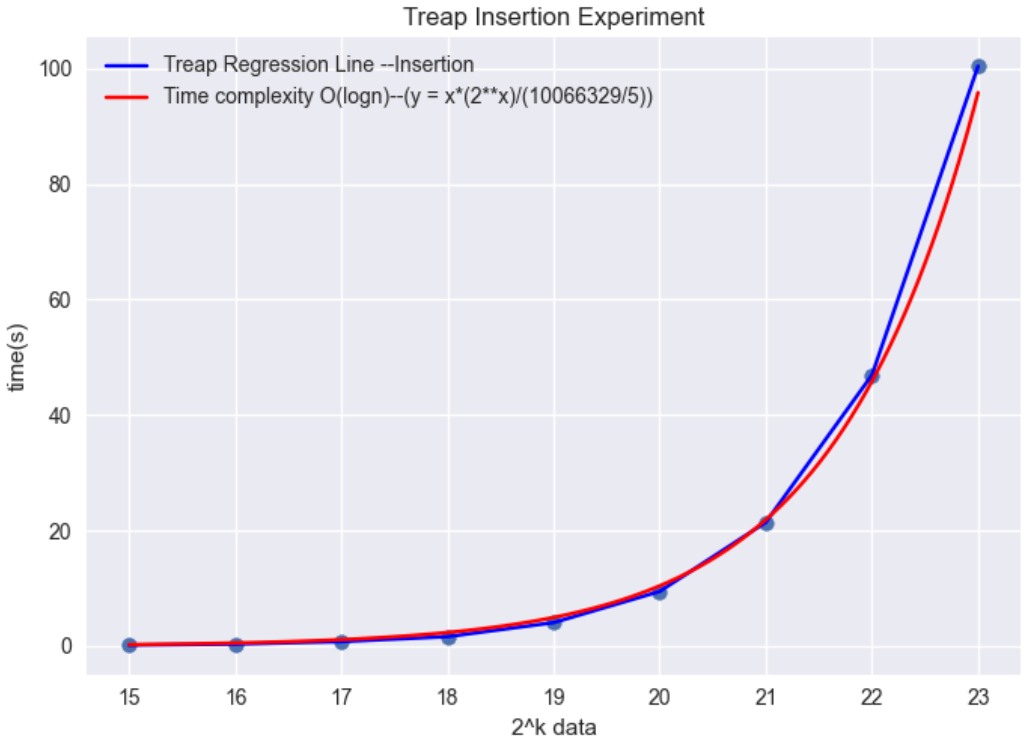
資料結構	Insertion 時間複雜度	Search 時間複雜度
BTree	O(logn)	O(logn)
Treap	O(logn)	O(logn)

Skip List	O(logn)	O(logn)
Hash Table	O(1)	O(1)
Array of sorted array	O(logn)	O(logn*logn)

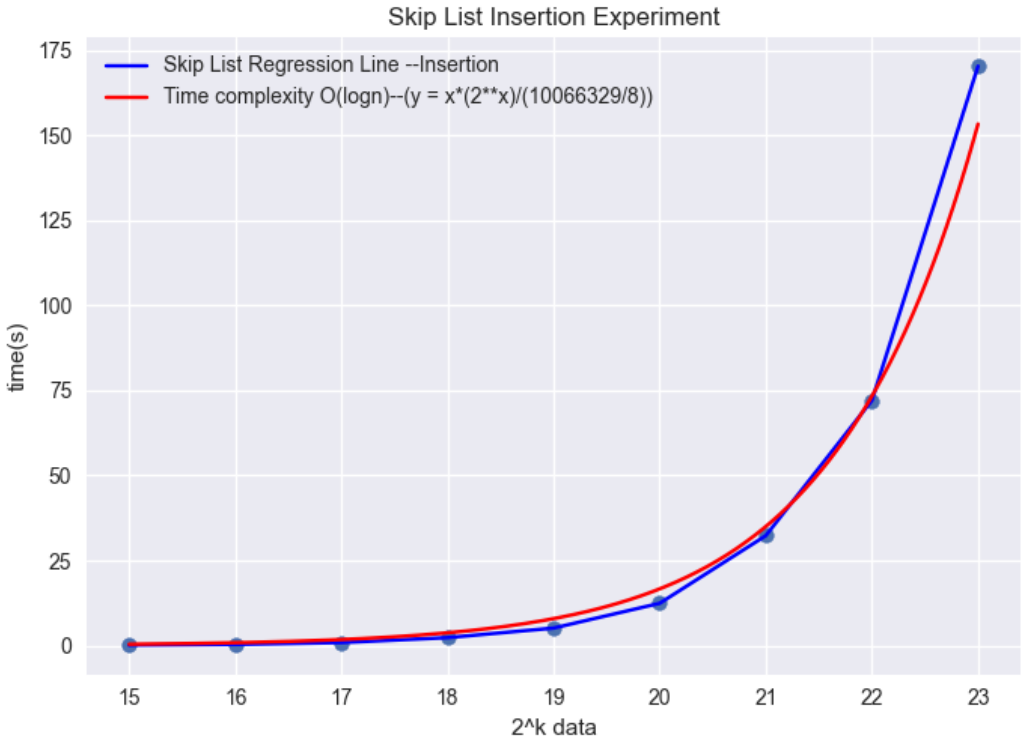
（圖表二）各資料結構 Insert & Search 的時間複雜度



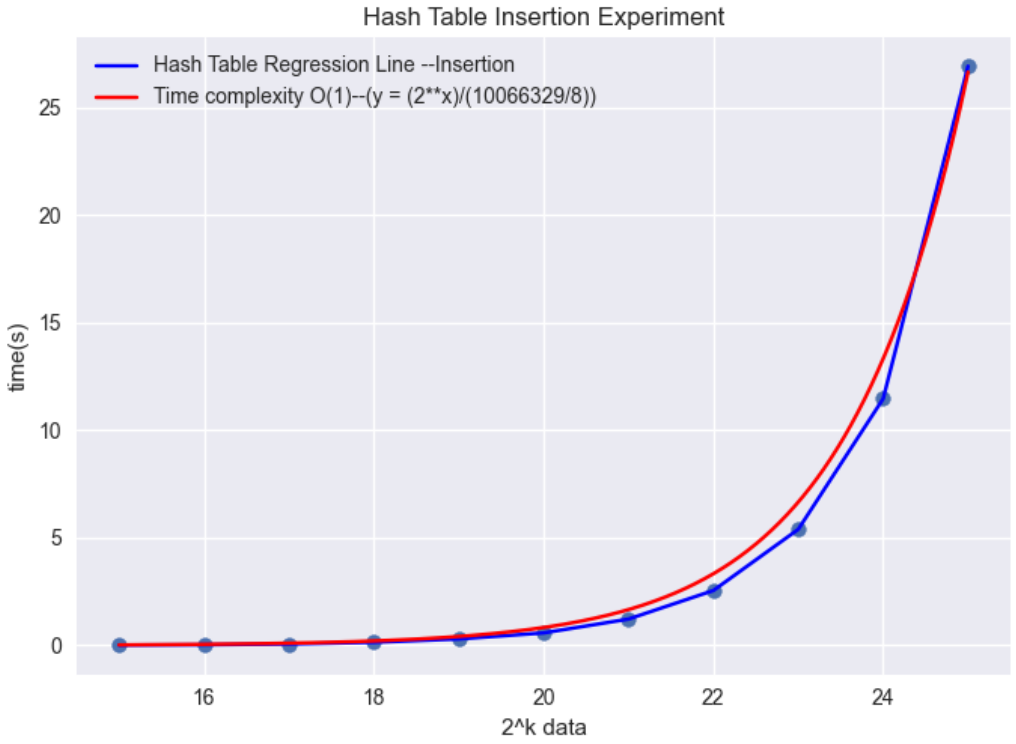
（圖三）BTree Insertion 實作



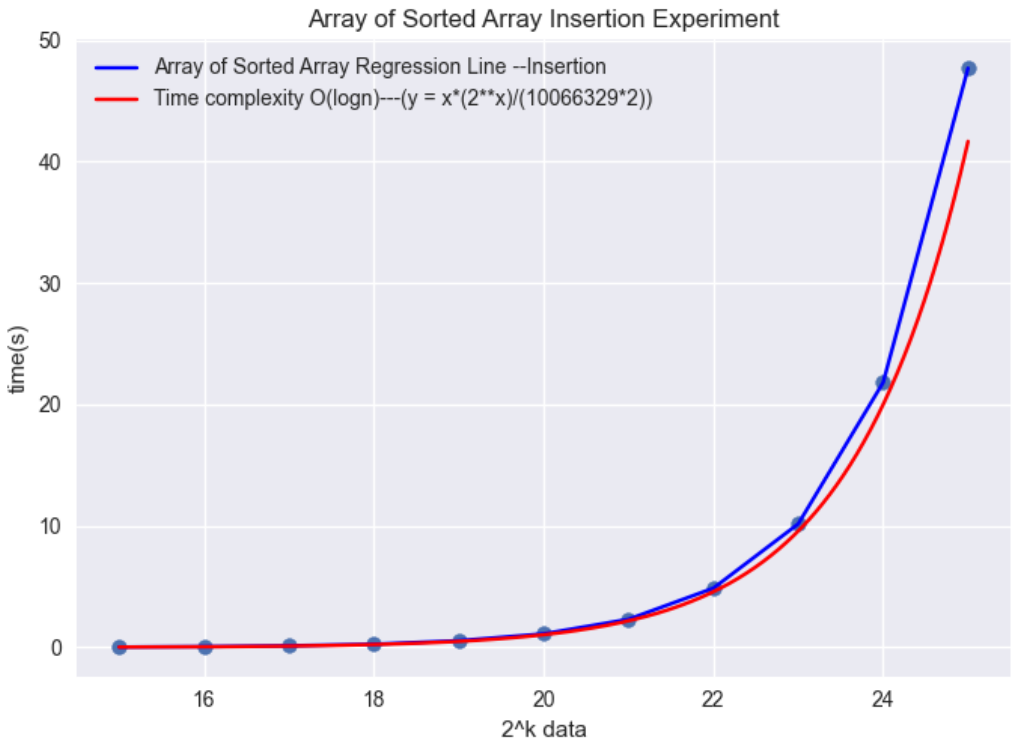
（圖四）Treap Insertion 實作



（圖五）Skip List Insertion 實作



（圖六）Hash Table Insertion 實作

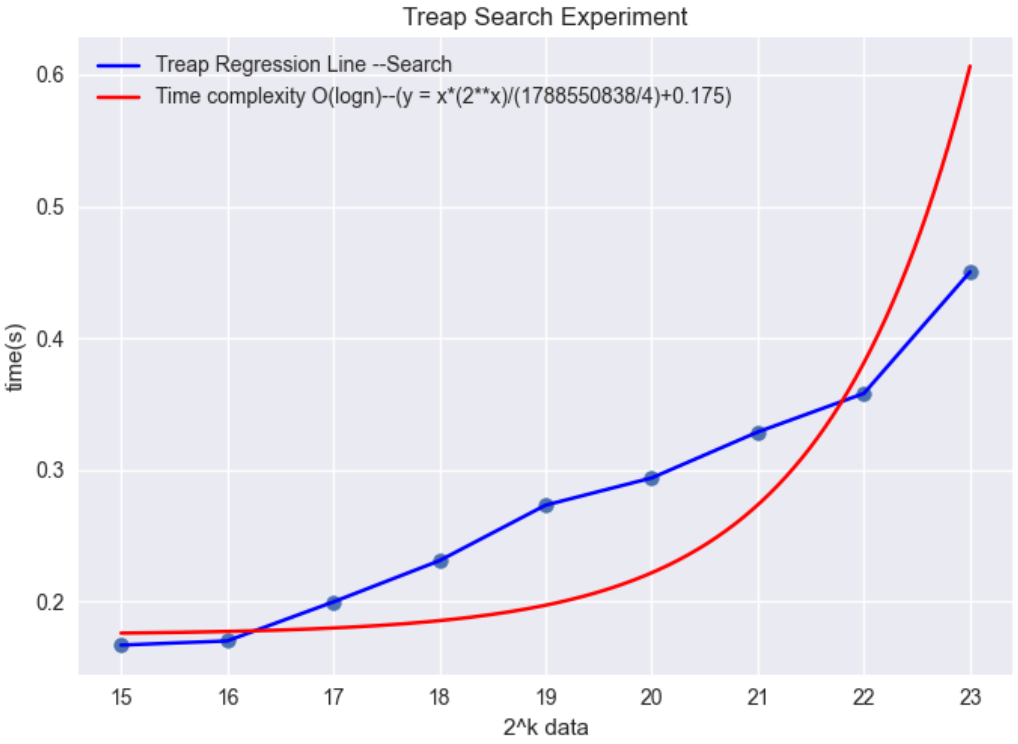


（圖七）Array of Sorted Array Insertion 實作

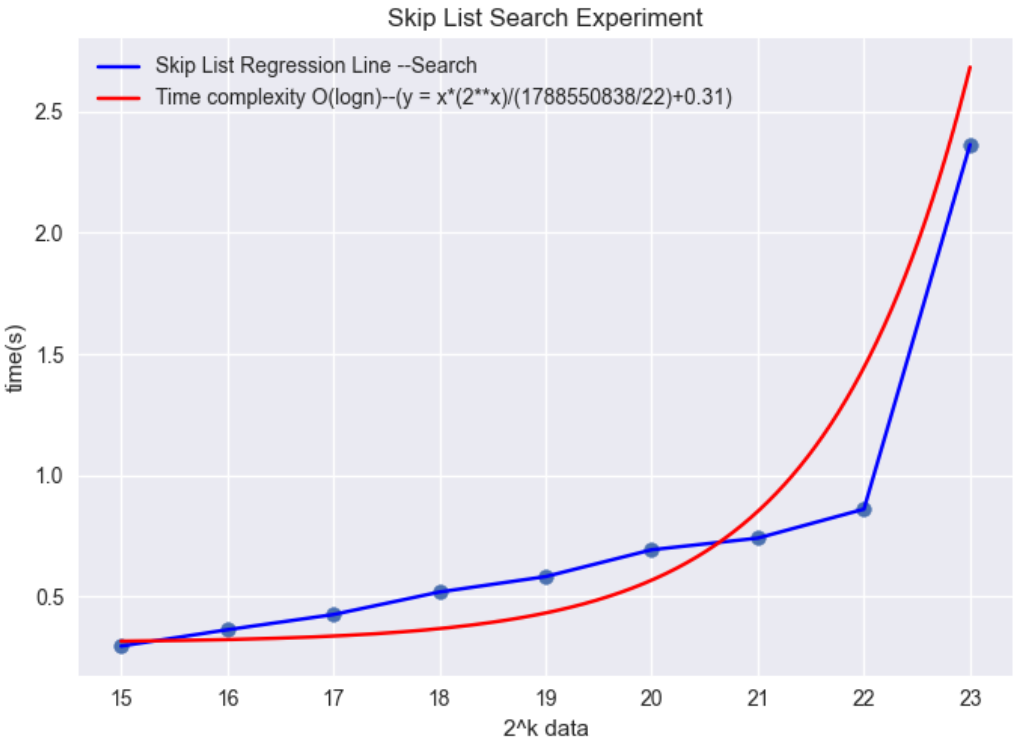
上述圖表(圖一)到(圖七)是各項「資料結構實驗」與其「時間複雜度」的疊圖，為了能達到更有效率的資料視覺化，我在不影響時間複雜度的前提下，在時間複雜度的線加入配合各項實驗數據加入常數、常數係數，使讀者能有效率的辨別其趨勢。

而我們可以發現五種的資料結構的 Insertion 實作都非常地貼近理論上的時間複雜度。

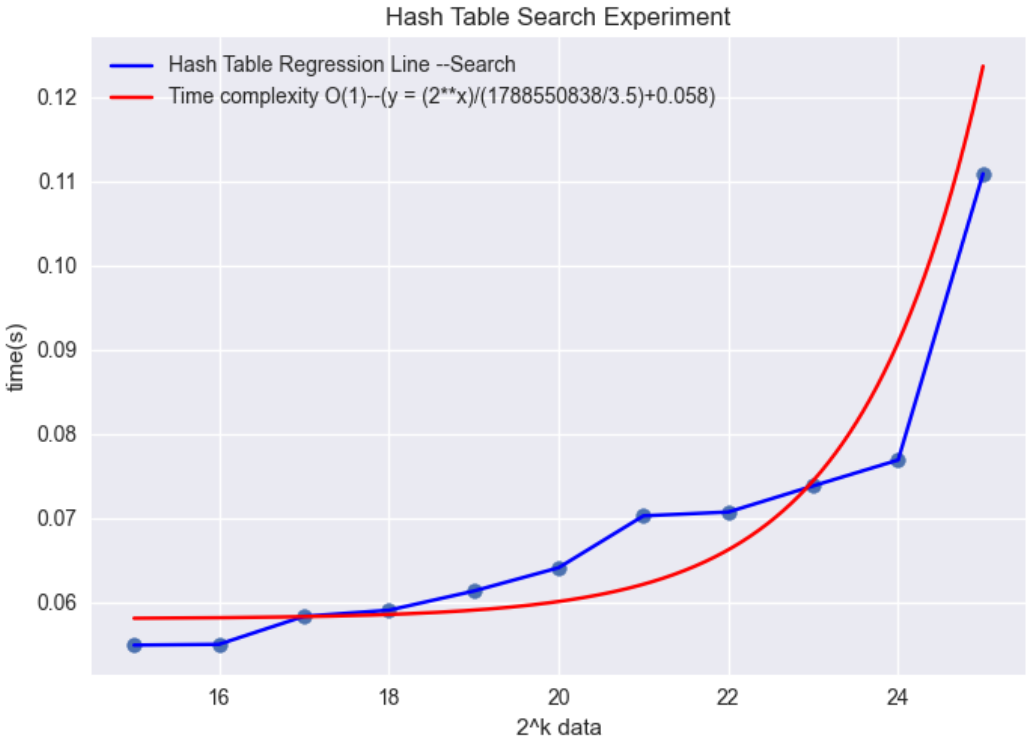
(圖八)BTree Search 實作



（圖九）Treap Search 實作



（圖十）Skip List Search 實作



（圖十一）Hash Table Search 實作

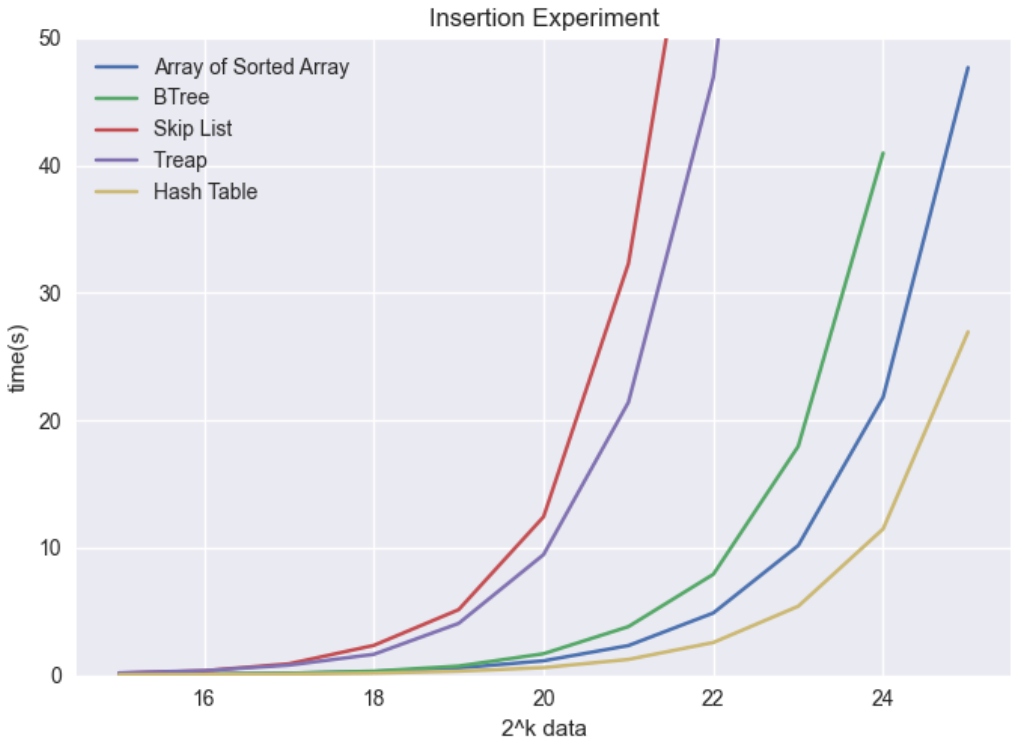
（圖十二）Array of Sorted Array Search 實作

上述圖表(圖八)到(圖十二)也是各項「資料結構實驗」與其「時間複雜度」的疊圖，為了能達到更有效率的資料視覺化，我也在不影響時間複雜度的前提下，在時間複雜度的線加入配合各項實驗數據加入常數、常數係數，使讀者能有效率的辨別其趨勢。

而我們可以發現五種的資料結構的「 Search 實作」相較於「 Insertion 實作 」並沒有那麼貼近時間複雜度的線，不過整體的趨勢的確是和時間複雜度相同的。

由上述圖表判斷，我認為我的實驗數據相當貼近理論值，且具有高度合理性。

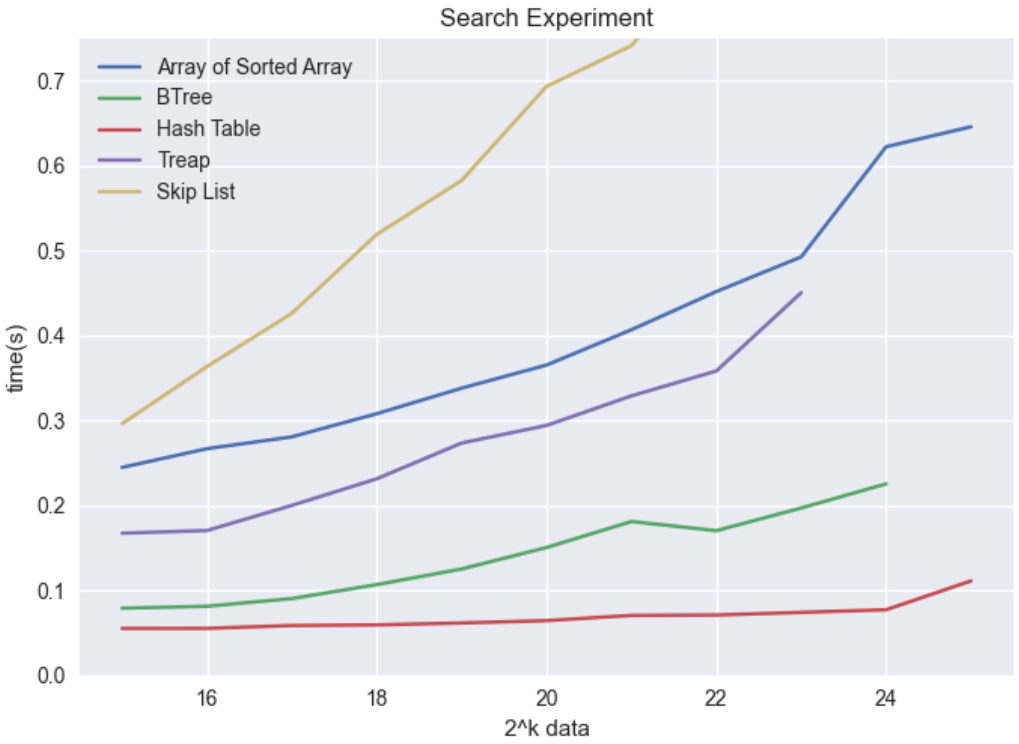
實驗結果分析



（圖十三）Insertion 實作結果

資料結構	Insertion 時間複雜度	實驗結果
BTree	O(logn)	第三快
Treap	O(logn)	第四快
Skip List	O(logn)	最慢
Hash Table	O(1)	最快
Array of sorted array	O(logn)	第二快

（圖表三）各資料結構 Insertion 實作比較



（圖十四）Insertion 實作結果

資料結構	Search 時間複雜度	實驗結果
BTree	O(logn)	第二快
Treap	O(logn)	第三快
Skip List	O(logn)	最慢
Hash Table	O(1)	最快
Array of sorted array	O(logn*logn)	第四快

（圖表四）各資料結構 Insertion 實作比較

以下我將以議題式的方式來探討我的實驗數據。

探討 Insertion 實驗中 BTree、Treap、Skip List、Array of Sorted Array 的耗時

這題的探討意義在於在相同理論時間複雜度度之間的耗時差異。首先我想先解釋為什麼我的 Array of Sorted Array 這之中最快的原因。我是使用 Python 的內建資料結構 Dictionary（類似 Hash Table）來儲存我在 每一個 Level 的 List，這會讓我把資料寫進List、讀List 的時間都變O(1)，遠比存在二維陣列中花更少時間及空間。我相信這就是我的Array of Sorted Array 最快的最主要的原因。

第二，我認為在這之中Skip List 會最慢是因為Source Code 來源的不同，由（圖表五）實作程式碼來源得知，Treap 和 BTree 的實作都是使用 Python 認證，且有持續維護、更新的套件，而 Skip List 則是網路上找的 Source Code，可想而知，相比之下，官網認證過的套件有很高的機率是比較有效率的，而這是我認為 Skip List 在 Python 環境中最慢的原因。

第三，我認造成 BTree 、Treap 之間的差異應該就是很直觀的 Treap 比 BTree 多一個 Rotate 的動作。

我認為以上推論足以合理的解釋實驗結果。

資料結構	實作程式碼來源
BTree	使用Python BTrees 套件自行實作
Treap	使用Python Treap 套件自行實作
Skip List	網路上搜尋 Source Code
Hash Table	Python「內建」資料結構 Dictionary
Array of sorted array	自行設計實作

(圖表五)實作程式碼來源

探討 Search 實驗中 Array of Sorted Array 的耗時

這題的探討意義在於明明 Array of Sorted Array 的時間複雜度最大，為何在此此實驗中的耗時卻比時間複雜度較小的資料結構還要小？

上述就有提到的我是使用一個類似 Hash Table 的資料結構來建構的外，另外我發現因為老師要求我們實驗的數據是2的整數次方，而在這種情況下會導致我只有一條長度為n的Array 在我的資料結構裡，而這時我所耗費的時間就會變成使用Binary Search 的 logn。而我又試著把實驗數據改成(2的k次方)減1，讓我的資料結構是有很多小array的，這時我發現，我所測到的時間會比原本的實驗還快上許多。

由此我發現其實我的 Array of Sorted Array的時間複雜度應該是logn，且大部分的時候會比logn還要快。

預測資料

時間預測上，我會使用 Python 的 Scikit-learn 套件來做 Polynominal Regression Analysis，我會取平均前的實驗數據來訓練我的 Regressiion line，並用實驗平均數據來當作測試Regrsssion line 分數的數據，從上述的(圖三)到(圖十二)就可以看出每個資料結構的趨勢。

二、每個資料結構程式碼的來源 / 實作程式碼

(一)BTree

```
# BTree Implementattion

# k = 24 (記憶體不足)



import treap
import random
import pandas as pd
import numpy as np
import time
from BTrees._OOBTree import OOBTree


"""************************************************************ Import dataframe
*********************************************************************"""

add = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment
3_1225/HW3_add_data.csv")

search = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment
3_1225/HW3_search_data.csv")



"""************************************************************ Experiment add
*********************************************************************"""
```

```
#test variable
x = 25



B = OOBTree()



start_add = time.time()

for i in range(0,2**x):
 B.update({random.randint(1,2**30+1) : i})

end_add = time.time()

add_time =  end_add - start_add
add["BTree"].iloc[x-15] = add_time







"""************************************************************ Experiment
(search) ********************************************************************"""

start_search = time.time()

for i in range(0,100000):
 temp = B.has_key(random.randint(1,2**30+1))

end_search = time.time()

search_time = (end_search - start_search)
search["BTree"].iloc[x-15] = search_time



print("add time:",add_time)
print("search time:",search_time)


"""************************************************************ Save data
****************************************************************"""

add.to_csv("HW3_add_data.csv", index=False)
```

```
search.to_csv("HW3_search_data.csv", index=False)
```

（二）Treap

```
# Treap Inplementation


# k = 23 (記憶體不足)



import treap
import random
import pandas as pd
import numpy as np
import time

"""************************************************************ Import dataframe
*****************************************************************"""

add = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment
3_1225/HW3_add_data.csv")

search = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment
3_1225/HW3_search_data.csv")


"""************************************************************ Experiment (add)
*****************************************************************"""

#test variable
x = 24


t = treap.treap()

start_add = time.time()

for i in range(0,2**x):
```

```
    t[random.randint(1,2**30+1)] = i

end_add = time.time()

add_time =  end_add - start_add
add["Treap"].iloc[x-15] = add_time


"""************************************************************ Experiment
(search) ******************************************************************"""
start_search = time.time()

for i in range(0,100000):
    try:
        temp = t.find(random.randint(1,2**30+1))
    except:
        pass

end_search = time.time()

search_time = (end_search - start_search)
search["Treap"].iloc[x-15] = search_time




print("add time:",add_time)
print("search time:",search_time)

"""************************************************************ Save data
******************************************************************"""

add.to_csv("HW3_add_data.csv", index=False)
search.to_csv("HW3_search_data.csv", index=False)
```

（三）Skip List

資料來源：https://www.geeksforgeeks.org/skip-list-set-2-insertion/

```
# Skip List Implementattion
# SourceCode : https://www.geeksforgeeks.org/skip-list-set-2-insertion/

# k = 23 (記憶體不足)


import treap
import random
import pandas as pd
import numpy as np
import time


"""************************************************************ Source Code
*****************************************************************"""


# Python3 code for searching and deleting element in skip list

import random

class Node(object):
    '''
    Class to implement node
    '''
    def __init__(self, key, level):
        self.key = key

        # list to hold references to node of different level
        self.forward = [None]*(level+1)

class SkipList(object):
    '''
    Class for Skip list
    '''
    def __init__(self, max_lvl, P):
        # Maximum level for this skip list
        self.MAXLVL = max_lvl

        # P is the fraction of the nodes with level
        # i references also having level i+1 references
        self.P = P

        # create header node and initialize key to -1
        self.header = self.createNode(self.MAXLVL, -1)
```

```
        # current level of skip list
        self.level = 0

    # create new node
    def createNode(self, lvl, key):
        n = Node(key, lvl)
        return n

    # create random level for node
    def randomLevel(self):
        lvl = 0
        while random.random()<self.P and \
            lvl<self.MAXLVL:lvl += 1
        return lvl

    # insert given key in skip list
    def insertElement(self, key):
        # create update array and initialize it
        update = [None]*(self.MAXLVL+1)
        current = self.header

        '''
        start from highest level of skip list
        move the current reference forward while key
        is greater than key of node next to current
        Otherwise inserted current in update and
        move one level down and continue search
        '''
        for i in range(self.level, -1, -1):
            while current.forward[i] and \
                current.forward[i].key < key:
                current = current.forward[i]
            update[i] = current

        '''
        reached level 0 and forward reference to
        right, which is desired position to
        insert key.
        '''
        current = current.forward[0]

        '''
        if current is NULL that means we have reached
        to end of the level or current's key is not equal
```

```
        to key to insert that means we have to insert
        node between update[0] and current node
    '''
        if current == None or current.key != key:
            # Generate a random level for node
            rlevel = self.randomLevel()

            '''
            If random level is greater than list's current
            level (node with highest level inserted in
            list so far), initialize update value with reference
            to header for further use
            '''
            if rlevel > self.level:
                for i in range(self.level+1, rlevel+1):
                    update[i] = self.header
                self.level = rlevel

            # create new node with random level generated
            n = self.createNode(rlevel, key)

            # insert node by rearranging references
            for i in range(rlevel+1):
                n.forward[i] = update[i].forward[i]
                update[i].forward[i] = n



    def deleteElement(self, search_key):

        # create update array and initialize it
        update = [None]*(self.MAXLVL+1)
        current = self.header

        '''
        start from highest level of skip list
        move the current reference forward while key
        is greater than key of node next to current
        Otherwise inserted current in update and
        move one level down and continue search
        '''
        for i in range(self.level, -1, -1):
            while(current.forward[i] and \
                current.forward[i].key < search_key):
                current = current.forward[i]
```

```
            update[i] = current

        '''
        reached level 0 and advance reference to
        right, which is possibly our desired node
        '''
        current = current.forward[0]

        # If current node is target node
        if current != None and current.key == search_key:

            '''
            start from lowest level and rearrange references
            just like we do in singly linked list
            to remove target node
            '''
            for i in range(self.level+1):

                '''
                If at level i, next node is not target
                node, break the loop, no need to move
                further level
                '''
                if update[i].forward[i] != current:
                    break
                update[i].forward[i] = current.forward[i]

            # Remove levels having no elements
            while(self.level>0 and\
                self.header.forward[self.level] == None):
                self.level -= 1
            print("Successfully deleted {}".format(search_key))

    def searchElement(self, key):
        current = self.header

        '''
        start from highest level of skip list
        move the current reference forward while key
        is greater than key of node next to current
        Otherwise inserted current in update and
        move one level down and continue search
        '''
        for i in range(self.level, -1, -1):
            while(current.forward[i] and\
```

```
                current.forward[i].key < key):
                current = current.forward[i]

        # reached level 0 and advance reference to
        # right, which is possibly our desired node
        current = current.forward[0]

        # If current node have key equal to
        # search key, we have found our target node



    # Display skip list level wise
    def displayList(self):
        print("\n*****Skip List******")
        head = self.header
        for lvl in range(self.level+1):
            print("Level {}: ".format(lvl), end=" ")
            node = head.forward[lvl]
            while(node != None):
                print(node.key, end=" ")
                node = node.forward[lvl]
            print("")




"""************************************************************ Import dataframe
*********************************************************************"""

add = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment
3_1225/HW3_add_data.csv")

search = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment
3_1225/HW3_search_data.csv")



"""************************************************************ Experiment add
*********************************************************************"""

#test variable
x = 24
```

```
lst = SkipList(x+3, 0.5)


start_add = time.time()

for i in range(0,2**x):
   lst.insertElement(random.randint(1,2**30+1))

end_add = time.time()

add_time =  end_add - start_add
add["Skip List"].iloc[x-15] = add_time




"""************************************************************ Experiment
(search) ********************************************************************"""

start_search = time.time()

for i in range(0,100000):
   lst.searchElement(random.randint(1,2**30+1))

end_search = time.time()

search_time = (end_search - start_search)
search["Skip List"].iloc[x-15] = search_time


print("add time:",add_time)
print("search time:",search_time)


"""************************************************************ Save data
********************************************************************"""

add.to_csv("HW3_add_data.csv", index=False)
search.to_csv("HW3_search_data.csv", index=False)
```

```

```

（四）Hash Table（Python Dictionary）

```
import treap
import random
import pandas as pd
import numpy as np
import time

"""************************************************************ Import dataframe
*****************************************************************"""

add = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment
3_1225/HW3_add_data.csv")

search = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment
3_1225/HW3_search_data.csv")


"""************************************************************ Experiment (add)
*****************************************************************"""

#test variable
x = 26


h = dict()

start_add = time.time()

for i in range(0,2**x):
   h[random.randint(1,2**30+1)] = i

end_add = time.time()

add_time =  end_add - start_add
add["Hash Table"].iloc[x-15] = add_time
```

```
"""************************************************************ Experiment
(search) ******************************************************************"""
start_search = time.time()

for i in range(0,100000):
    try:
        temp = h[random.randint(1,2**30+1)]
    except:
        pass

end_search = time.time()

search_time = (end_search - start_search)
search["Hash Table"].iloc[x-15] = search_time




print("add time:",add_time)
print("search time:",search_time)

"""************************************************************ Save data
******************************************************************"""

add.to_csv("HW3_add_data.csv", index=False)
search.to_csv("HW3_search_data.csv", index=False)
```

（五）Array of Sorted Array

```
# Array of Sorted Array Implementation

# k = 25 (記憶體不足)


import random
```

```
import pandas as pd
import numpy as np
import time


"""************************************************************ Import dataframe
*********************************************************************"""

add = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment
3_1225/HW3_add_data.csv")

search = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment
3_1225/HW3_search_data.csv")




"""************************************************************ Array of Sorted
Array Inplementation
*********************************************************************"""


Array_of_sorted_Array = dict()


def BinarySearch(array, x, low, high):

   # Repeat until the pointers low and high meet each other
   while low <= high:

       mid = low + (high - low)//2

       if array[mid] == x:
           return mid

       elif array[mid] < x:
           low = mid + 1

       else:
           high = mid - 1

   return -1


def ASA_Insert(n):
```

```
    p_empty = 1

    temp = [n]
    while True:            #建造一個 while loop 從第一層往下看，只要是滿的就跟input merge ，
遇到空的就放進去並跳出迴圈。


        try:
            temp = Array_of_sorted_Array[p_empty] + temp

        except:
            Array_of_sorted_Array[p_empty] = temp
            p_empty = 1
            break

        temp = sorted(temp)
        del Array_of_sorted_Array[p_empty]
        p_empty += 1


def ASA_Search(n):
    for i in sorted(list(Array_of_sorted_Array)):
        search_result = BinarySearch(Array_of_sorted_Array[i], n, 0,
len(Array_of_sorted_Array[i])-1)
        if search_result >= 0:
            return 1
        else:
            pass
    return 0



"""************************************************************** Experiment add
**********************************************************************"""


#test variable
x = 24




start_add = time.time()
```

```
for i in range(0,2**x-1):
 ASA_Insert(random.randint(1,2**30))

end_add = time.time()

add_time =  end_add - start_add
add["Array of sorted array"].iloc[x-15] = add_time




"""************************************************************ Experiment
(search) ********************************************************************"""

start_search = time.time()

for i in range(0,100000):
 temp = ASA_Search(random.randint(1,2**30+1))

end_search = time.time()

search_time = (end_search - start_search)
search["Array of sorted array"].iloc[x-15] = search_time

print(Array_of_sorted_Array)
print("add time:",add_time)
print("search time:",search_time)


"""************************************************************ Save data
********************************************************************"""

#add.to_csv("HW3_add_data.csv", index=False)
#search.to_csv("HW3_search_data.csv", index=False)
```

（六）Graphing Code

```
import random
import pandas as pd
import numpy as np
import time
import math
```

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split


"""************************************************************ Import dataframe
*****************************************************************"""

add = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment
3_1225/HW3_add_data.csv")

search = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment
3_1225/HW3_search_data.csv")









"""************************************************************* Time Complexity
Line  *****************************************************************"""

plt.style.use("seaborn")
plt.legend()
plt.ticklabel_format(style="plain")
"""


#
## Time complexity line
x = np.linspace(15, 24, 1000)
y = x*(2**x)/10066329
plt.plot(x,y, color = "red", label = "Time complexity O(logn)")

plt.show()

"""

"""************************************************************* Search Regression
Line ( BTree )
*****************************************************************"""
```

```
# 畫圖資料


x_btree_search = search["k (2**k)"].iloc[0:10]
y_btree_search = search["BTree"].iloc[0:10]

 plt.plot(x_btree_search, y_btree_search,color = "blue", label =  'BTree Regression
Line --Search')



## 繪製數據集資料點
plt.scatter(x_btree_search,y_btree_search)
plt.ticklabel_format(style="plain")
plt.style.use("seaborn")
plt.legend()
plt.xlabel("2^k data")
plt.title("BTree Search Experiment")
plt.ylabel("time(s)")


#
## Time complexity line
x = np.linspace(15, 24, 1000)
y = x*(2**x)/(1788550838*1.1)+0.078
plt.plot(x,y, color = "red", label = "Time complexity O(logn)--(y =
x*(2**x)/(1788550838*1.1)+0.078)")
plt.style.use("seaborn")
plt.legend()
plt.ticklabel_format(style="plain")
plt.show()


"""************************************************************ Insertion
Regression Line ( BTree )
************************************************************"""

# 畫圖資料


x_btree_add = add["k (2**k)"].iloc[0:10]
y_btree_add = add["BTree"].iloc[0:10]

 plt.plot(x_btree_add, y_btree_add, color = "blue", label =  'BTree Regression Line
--Insertion')
```

```
## 繪製數據集資料點
plt.scatter(x_btree_add,y_btree_add)
plt.ticklabel_format(style="plain")
plt.style.use("seaborn")
plt.legend()
plt.xlabel("2^k data")
plt.title("BTree Insertion Experiment")
plt.ylabel("time(s)")


#
## Time complexity line
x = np.linspace(15, 24, 1000)
y = x*(2**x)/10066329
plt.plot(x,y, color = "red", label = "Time complexity O(logn)--(y =
x*(2**x)/10066329)")
plt.style.use("seaborn")
plt.legend()
plt.ticklabel_format(style="plain")
plt.show()




"""************************************************************ Search Regression
Line ( Skip List )
************************************************************"""

# 畫圖資料


x_skiplist_search = search["k (2**k)"].iloc[0:9]
y_skiplist_search = search["Skip List"].iloc[0:9]

 plt.plot(x_skiplist_search, y_skiplist_search,color = "blue", label =  'Skip List
Regression Line --Search')



## 繪製數據集資料點
plt.scatter(x_skiplist_search,y_skiplist_search)
plt.ticklabel_format(style="plain")
```

```
plt.style.use("seaborn")
plt.legend()
plt.xlabel("2^k data")
plt.title("Skip List Search Experiment")
plt.ylabel("time(s)")



#
## Time complexity line
x = np.linspace(15, 23, 1000)
y = x*(2**x)/(1788550838/22)+0.31
plt.plot(x,y, color = "red", label = "Time complexity O(logn)--(y =
x*(2**x)/(1788550838/22)+0.31)")
plt.style.use("seaborn")
plt.legend()
plt.ticklabel_format(style="plain")
plt.show()


"""************************************************************ Insertion
Regression Line ( Skip List )
**********************************************************************"""

# 畫圖資料


x_skiplist_add = add["k (2**k)"].iloc[0:9]
y_skiplist_add = add["Skip List"].iloc[0:9]

 plt.plot(x_skiplist_add, y_skiplist_add, color = "blue", label =  'Skip List
Regression Line --Insertion')



## 繪製數據集資料點
plt.scatter(x_skiplist_add,y_skiplist_add)
plt.ticklabel_format(style="plain")
plt.style.use("seaborn")
plt.legend()
plt.xlabel("2^k data")
plt.title("Skip List Insertion Experiment")
plt.ylabel("time(s)")


#
```

```
## Time complexity line
x = np.linspace(15, 23, 1000)
y = x*(2**x)/(10066329/8)
plt.plot(x,y, color = "red", label = "Time complexity O(logn)--(y =
x*(2**x)/(10066329/8))")
plt.style.use("seaborn")
plt.legend()
plt.ticklabel_format(style="plain")
plt.show()




"""************************************************************ Search Regression
Line ( Treap )
**********************************************************************"""

# 畫圖資料


x_treap_search = search["k (2**k)"].iloc[0:9]
y_treap_search = search["Treap"].iloc[0:9]

 plt.plot(x_treap_search, y_treap_search,color = "blue", label =  'Treap Regression
Line --Search')



## 繪製數據集資料點
plt.scatter(x_treap_search,y_treap_search)
plt.ticklabel_format(style="plain")
plt.style.use("seaborn")
plt.legend()
plt.xlabel("2^k data")
plt.title("Treap Search Experiment")
plt.ylabel("time(s)")


#
## Time complexity line
x = np.linspace(15, 23, 1000)
y = x*(2**x)/(1788550838/4)+0.175
```

```
plt.plot(x,y, color = "red", label = "Time complexity O(logn)--(y =
x*(2**x)/(1788550838/4)+0.175)")
plt.style.use("seaborn")
plt.legend()
plt.ticklabel_format(style="plain")
plt.show()


"""************************************************************ Insertion
Regression Line ( Treap )
************************************************************"""

# 畫圖資料


x_treap_add = add["k (2**k)"].iloc[0:9]
y_treap_add = add["Treap"].iloc[0:9]

 plt.plot(x_treap_add, y_treap_add, color = "blue", label =  'Treap Regression Line
--Insertion')



## 繪製數據集資料點
plt.scatter(x_treap_add,y_treap_add)
plt.ticklabel_format(style="plain")
plt.style.use("seaborn")
plt.legend()
plt.xlabel("2^k data")
plt.title("Treap Insertion Experiment")
plt.ylabel("time(s)")


#
## Time complexity line
x = np.linspace(15, 23, 1000)
y = x*(2**x)/(10066329/5)
plt.plot(x,y, color = "red", label = "Time complexity O(logn)--(y =
x*(2**x)/(10066329/5))")
plt.style.use("seaborn")
plt.legend()
plt.ticklabel_format(style="plain")
plt.show()
```

```
"""************************************************************ Search Regression
Line ( Hash Table )
**********************************************************************"""

# 畫圖資料


x_hashtable_search = search["k (2**k)"].iloc[0:11]
y_hashtable_search = search["Hash Table"].iloc[0:11]

 plt.plot(x_hashtable_search, y_hashtable_search,color = "blue", label =  'Hash
Table Regression Line --Search')



## 繪製數據集資料點
plt.scatter(x_hashtable_search,y_hashtable_search)
plt.ticklabel_format(style="plain")
plt.style.use("seaborn")
plt.legend()
plt.xlabel("2^k data")
plt.title("Hash Table Search Experiment")
plt.ylabel("time(s)")


#
## Time complexity line
x = np.linspace(15, 25, 1000)
y = (2**x)/(1788550838/3.5)+0.058
plt.plot(x,y, color = "red", label = "Time complexity O(1)--(y =
(2**x)/(1788550838/3.5)+0.058)")
plt.style.use("seaborn")
plt.legend()
plt.ticklabel_format(style="plain")
plt.show()


"""************************************************************ Insertion
Regression Line ( Hash Table )
**********************************************************************"""

# 畫圖資料


x_hashtable_add = add["k (2**k)"].iloc[0:11]
```

```
y_hashtable_add = add["Hash Table"].iloc[0:11]

 plt.plot(x_hashtable_add, y_hashtable_add, color = "blue", label =  'Hash Table
Regression Line --Insertion')



## 繪製數據集資料點
plt.scatter(x_hashtable_add,y_hashtable_add)
plt.ticklabel_format(style="plain")
plt.style.use("seaborn")
plt.legend()
plt.xlabel("2^k data")
plt.title("Hash Table Insertion Experiment")
plt.ylabel("time(s)")


#
## Time complexity line
x = np.linspace(15, 25, 1000)
y = (2**x)/(10066329/8)
plt.plot(x,y, color = "red", label = "Time complexity O(1)--(y =
(2**x)/(10066329/8))")
plt.style.use("seaborn")
plt.legend()
plt.ticklabel_format(style="plain")
plt.show()


"""************************************************************ Search Regression
Line ( Array of Sorted Array )
************************************************************"""

# 畫圖資料


x_asa_search = search["k (2**k)"].iloc[0:11]
y_asa_search = search["Array of sorted array"].iloc[0:11]

 plt.plot(x_asa_search, y_asa_search,color = "blue", label =  'Array of Sorted
Array Regression Line --Search')



## 繪製數據集資料點
```

```
plt.scatter(x_asa_search,y_asa_search)
plt.ticklabel_format(style="plain")
plt.style.use("seaborn")
plt.legend()
plt.xlabel("2^k data")
plt.title("Array of Sorted Array Search Experiment")
plt.ylabel("time(s)")


#
## Time complexity line
x = np.linspace(15, 25, 1000)
y = x*x/(1000)
plt.plot(x,y, color = "red", label = "Time complexity O(logn*logn)--(y =
x*x/(1000))")
plt.style.use("seaborn")
plt.legend()
plt.ticklabel_format(style="plain")
plt.show()


"""************************************************************ Insertion
Regression Line ( Skip List )
************************************************************"""

# 畫圖資料


x_asa_add = add["k (2**k)"].iloc[0:11]
y_asa_add = add["Array of sorted array"].iloc[0:11]

 plt.plot(x_asa_add, y_asa_add, color = "blue", label =  'Array of Sorted Array
Regression Line --Insertion')



## 繪製數據集資料點
plt.scatter(x_asa_add,y_asa_add)
plt.ticklabel_format(style="plain")
plt.style.use("seaborn")
plt.legend()
plt.xlabel("2^k data")
plt.title("Array of Sorted Array Insertion Experiment")
plt.ylabel("time(s)")
```

```
#
## Time complexity line
x = np.linspace(15, 25, 1000)
y = x*(2**x)/(10066329*2)
plt.plot(x,y, color = "red", label = "Time complexity O(logn)---(y =
x*(2**x)/(10066329*2))")
plt.style.use("seaborn")
plt.legend()
plt.ticklabel_format(style="plain")
plt.show()


"""************************************************************ Search Graph
*****************************************************************"""

# 畫圖資料


plt.ticklabel_format(style="plain")
plt.style.use("seaborn")


 plt.plot(x_asa_search, y_asa_search, label =  'Array of Sorted Array')
plt.plot(x_btree_search, y_btree_search, label =  'BTree')
plt.plot(x_hashtable_search, y_hashtable_search, label =  'Hash Table ')
plt.plot(x_treap_search, y_treap_search, label =  'Treap')
plt.plot(x_skiplist_search, y_skiplist_search, label =  'Skip List')



## 繪製數據集資料點

plt.xlabel("2^k data")
plt.title("Search Experiment")
plt.ylabel("time(s)")
plt.ylim(0,0.75)
plt.legend()
plt.show()


"""************************************************************ Insertion Graph
*****************************************************************"""

# 畫圖資料
```

```
plt.ticklabel_format(style="plain")
plt.style.use("seaborn")


 plt.plot(x_asa_add, y_asa_add,  label =  'Array of Sorted Array ')
plt.plot(x_btree_add, y_btree_add,  label =  'BTree')
plt.plot(x_skiplist_add, y_skiplist_add,  label =  'Skip List ')
plt.plot(x_treap_add, y_treap_add, label =  'Treap ')
plt.plot(x_hashtable_add, y_hashtable_add,  label =  'Hash Table')



## 繪製數據集資料點


plt.xlabel("2^k data")
plt.title("Insertion Experiment")
plt.ylabel("time(s)")
plt.ylim(0,50)
plt.legend()



plt.show()
```

三、心得、疑問、遇到的困難

對於一個外系的學生而言，我完全沒有過利用物件導向來寫程式的經驗，雖然之前在作業一和作業二都有看過許多用Class和Function來建造一個演算法的Source code，不過讀懂Source code，和自己實作仍然有非常大的差距，我自己試過大概做了整整兩個禮拜，透過慢慢地摸索跟嘗試 ，除了到最後成功實作之外，我最開心的是我建造了一個很符合python環境的Array of sorted array，而且我還利用了Python 高階語言的優勢，基本上我自己寫的只有兩個Function，就完成這次的實作，我只能說花了很多時間，但是很值得。