郭桐維教授

資料結構

作業二

國貿二 郭許謙 110301034

目錄

一、六種排序方法的比較實驗

(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Insertion sort 、Merge Sort、Counting sort、Quick sort (Lomuto's partition)、Quick sort (Hoare's partition)、Quick sort (Three way partition) 排序二的K次方(K＝1、2、、、30)所花費時間的實驗。結束後再用 Polynominal Rregression Analysis的方式來預測排列更長的 Array 所會花費的時間。

實驗流程

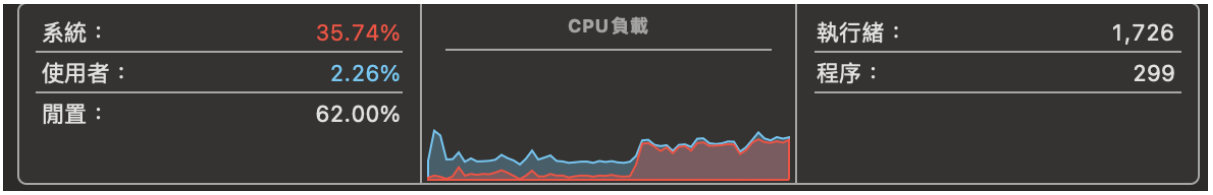
1) 實驗六種排序方法排序不同長度的array所花費的時間。
2) 用Python 的 Scikit-Learn 的套件來做 Polynominal Rregression Analysis的方式來做預測分析
3) 製作各項分析數據、訓練Polynominal Regression、時間複雜度的圖表
4) 利用上述圖表以及時間複雜度來分析實驗結果

各項排序實驗能測到的最大的 Array

排序方法	測試最大數據	原因
Insertion sort	2的18次方	排序時間超過一小時
Merge sort	2的26次方	記憶體不足
Counting sort	2的26次方	記憶體不足
Lomuto's partition	2的23次方	排序時間超過一小時
Hoare's partition	2的27次方	記憶體不足
Three way partition	2的27次方	記憶體不足

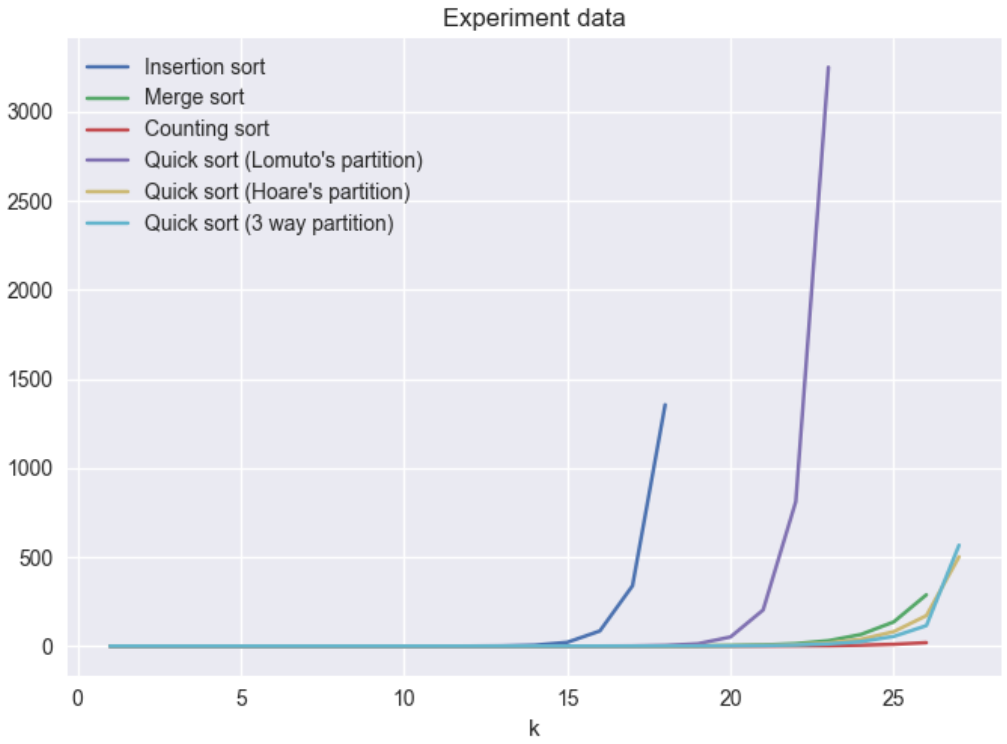
在這次的作業二中我發現，除了可以透過記憶體壓力圖表來看有沒有發生Ram 記憶體不足的情況，我做實驗時觀察到記憶體不足時，作業系統的CPU用量也會突然暴增，我推論這是作業系統在把 Python所需的記憶體寫進Disk所耗費大量CPU的情況。

（圖一）Ram不足時的記憶體壓力（黃色部份）



（圖二）Ram不足時的CPU負載

（二）實驗結果



（圖三）實驗測得的平均時間

從上述的圖表我們可以觀察消耗時間:

Insertion sort >> Lomuto's partition > Merge sort ≈ Hoare's partition ≈ Three way partition > Counting sort

各排序法的時間複雜度

排序方法	平均時間複雜度	Stability
Insertion sort	O(n-square)	Yes
Merge sort	O(nlogn)	Yes
Counting sort	O(nlogn)	Yes
Lomuto's partition	O(nlogn)	No
Hoare's partition	O(nlogn)	No
Three way partition	O(n+k)	No

上述圖表大致上可以表示我們的實驗結果，不過對於Quicksort (Lomuto's partition) 的廢時為何會高於其他的排序法似乎看不出一個合理的解釋，因此下方圖表會比較 Lomuto's partition 和 Hoare's partition的差異，以便解釋Lomuto's partition會有較高耗時的原因。

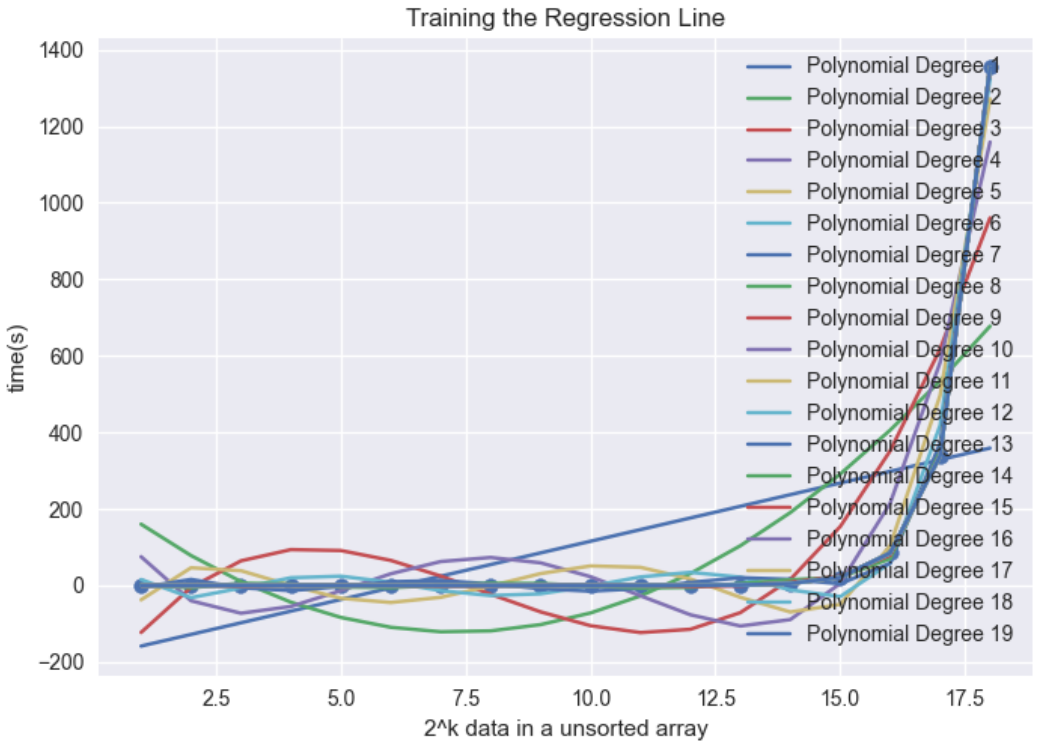
Lomuto's partition vs Hoare's partition

Basis	Lomuto's	Hoare's
Average time complexity	O(nlogn)	O(nlogn)
Efficiency	Less efficient	comparatively more efficient
Numbers of swaps and partitioning	Three times more swaps than Hoare's	Three times less swaps than Hoare's
Time complexity when all elements are equal	O(n-square)	O(nlogn)
Time complexity when array is already sorted	O(n-square)	O(n-square)

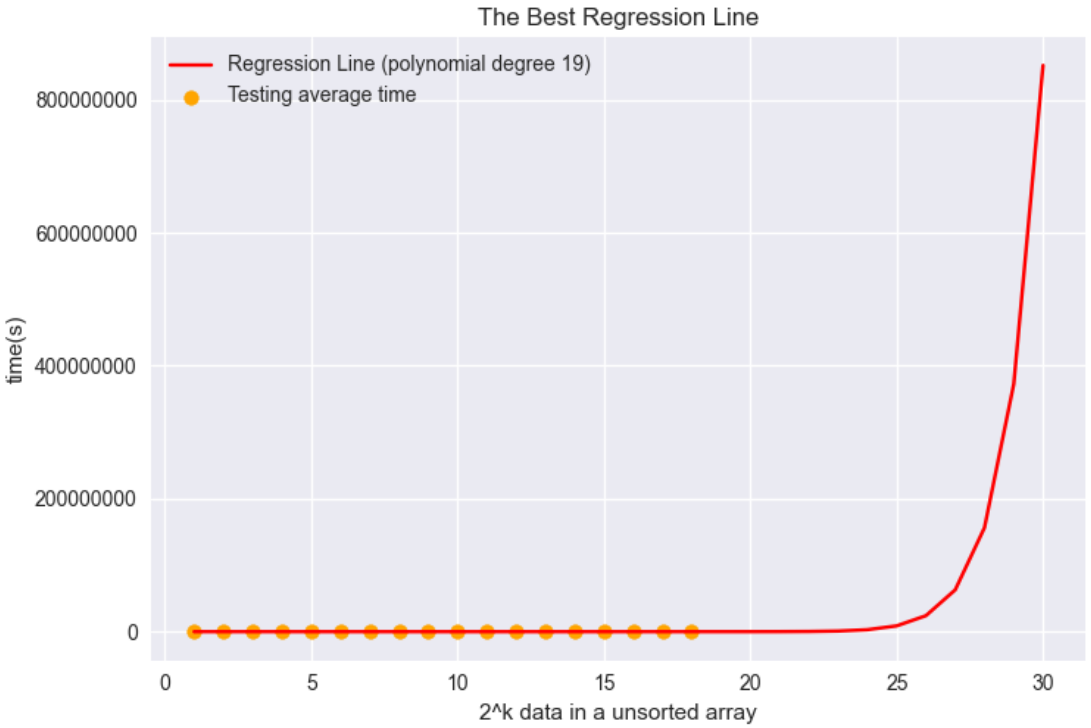
從上述圖表可以得知Lomuto'a partition 會比 Hoare's partition 多了3倍以上的swap 次數，而這種係數的差異並不影響時間複雜度，不過隨著n變大，這種小差異也會造成好時有很大的差距。

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，用此方法測試零次方到19次方的Regression line，並採分數最高的Regression line 當作我預測基準。(以下圖表將以Insertion sort 為例)

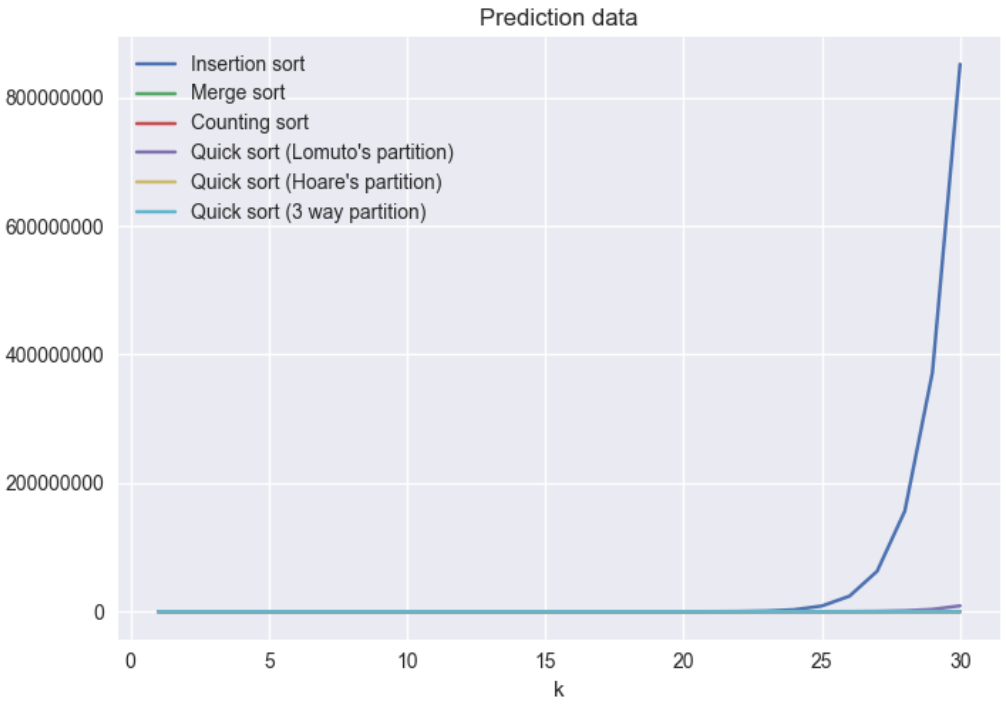


（圖四）Insertion sort 的 Regression training

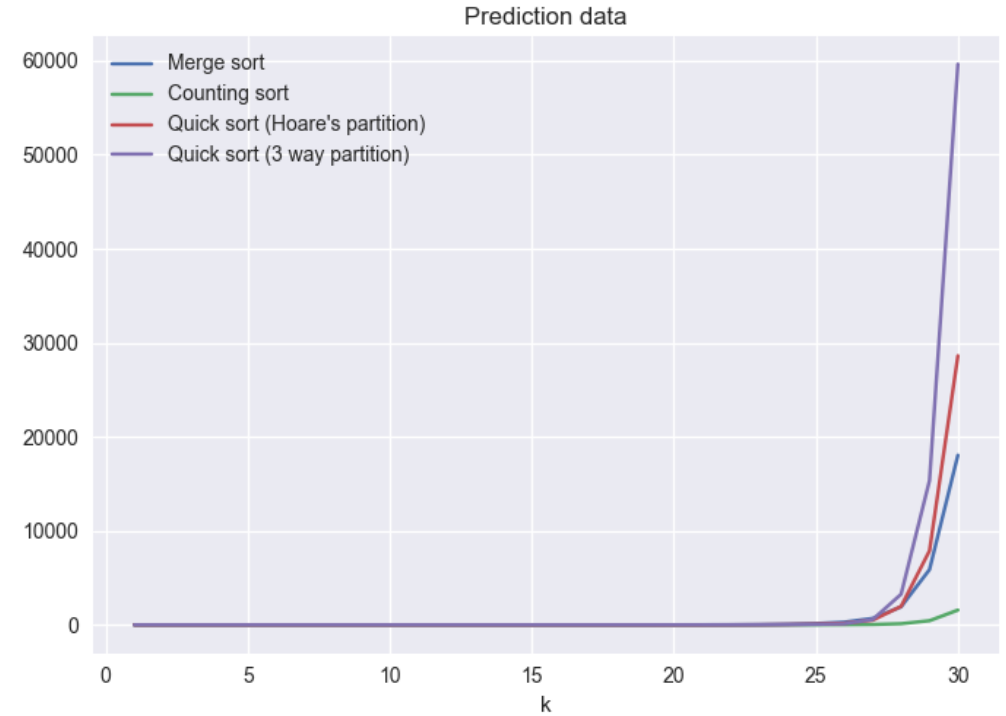


（圖五）Insertion sort 的 Best regression line

預測結果分析



（圖六）六種排序法的Predition line

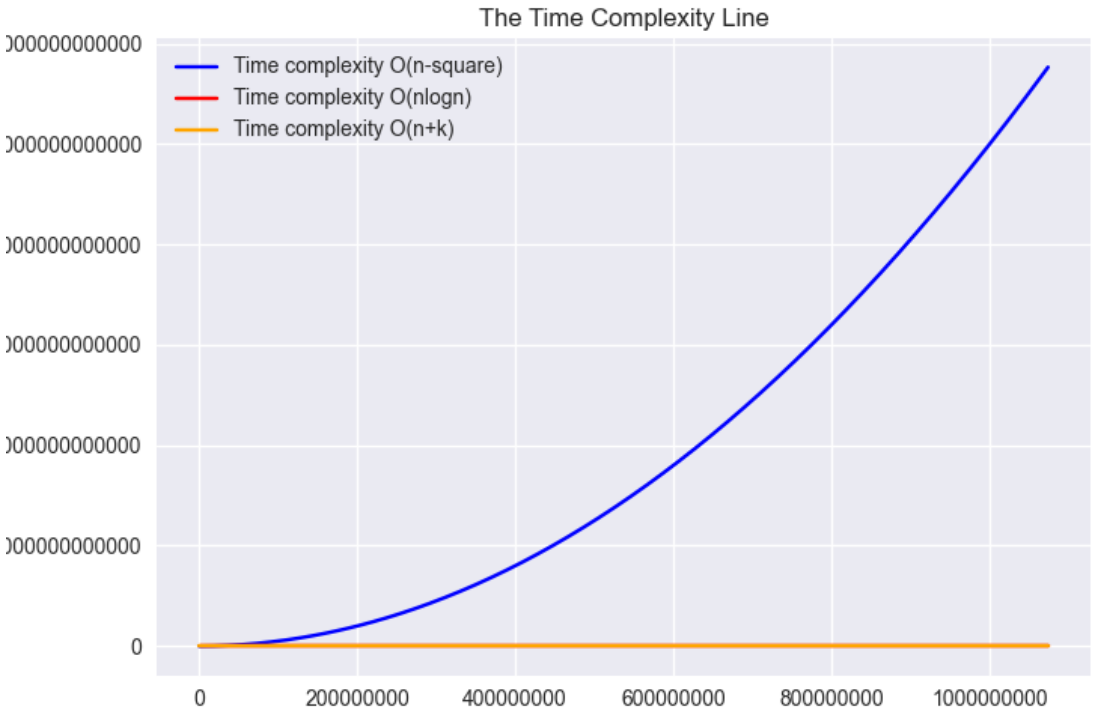


（圖六）四種排序法的Predition line (除去成長最快的兩個)

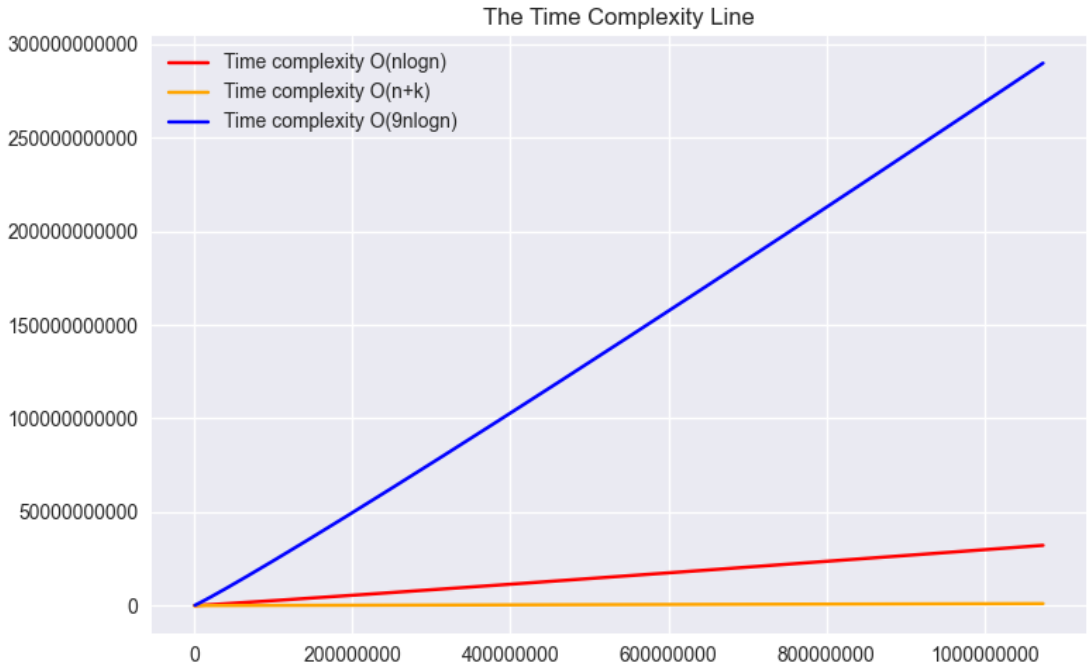
從上述的圖表我們可以觀察預測消耗時間:

Insertion sort >> Lomuto's partition > Merge sort ≈ Hoare's partition ≈ Three way partition > Counting sort（與實驗結果大致符合，但在Merge sort 、Hoare's partition、Three way partition 還是有差異）

雖然這樣已足證明預測是符合實驗結果，下圖還是會用時間複雜度的關係圖再比對一次。



（圖七）時間複雜度比較圖



（圖八）時間複雜度比較圖（n-square除外）

從（圖七）與（圖八）中我們可以觀察到時間複雜度的圖與實驗數據的圖和預測數據的圖都符合。而我們更可以從圖八中得知，細數的差異確實是會造成相同時間複雜度的演算法產生差異很大的數據 。而我認為這也是造成 Merge sort、Hoare's partition、Three way partition 的預測分析圖與實驗數據圖有些微差異的原因。

二、每個Sorting演算法程式碼的來源 / 實作程式碼

（一）Insertion sort

資料來源：https://www.programiz.com/dsa/insertion-sort

```
# Insertion sort
# https://www.programiz.com/dsa/insertion-sort
# Average time complexity : O(n^2)
# Space complexity : O(1)

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

def insertionSort(array):

   for step in range(1, len(array)):
       key = array[step]
       j = step - 1

        # Compare key with each element on the left of it until an element smaller
than it is found
       # For descending order, change key<array[j] to key>array[j].

       while j >= 0 and key < array[j]:
           array[j + 1] = array[j]
           j = j - 1

       # Place key at after the element just smaller than it.
       array[j + 1] = key


# 輸入 Dataframe
insertion       =       pd.read_csv("/Users/stevenkuo/College       Course/大二/Data
Structure/Assignment 2_1113/Insertion sort.csv")
hw2_data        =       pd.read_csv("/Users/stevenkuo/College       Course/大二/Data
Structure/Assignment 2_1113/Assignment2_data.csv")
hw2_pred_data      =      pd.read_csv("/Users/stevenkuo/College      Course/大二/Data
Structure/Assignment 2_1113/Assignment2_pred_data.csv")



# Experiment
'''
timelist = list()


for i in range(0,10):
   print(i)
   data = list()
   for j in range(0,2**1):
       data.append(random.randint(1,1000))
```

```
    start = time.time()
    insertionSort(data)
    end = time.time()
    timelist.append(end-start)


insertion["k = 1"]  = timelist
insertion.to_csv("Insertion sort.csv", index=False)
'''


# 平均資料

insertion_avg = hw2_data[0:18]
x_insertion = insertion_avg["k"].values.reshape(-1,1)
y_insertion = insertion_avg["Insertion sort"].values.reshape(-1,1)



## 訓練資料

insertion_train = list()
k_train = list()

for i in range(0,18):
    for j in range(0,10):
        insertion_train.append(insertion.iloc[j,i])
        k_train.append(hw2_data["k"].iloc[i])

train_dict = {"train_k" : k_train,"train_insertion" : insertion_train}
train_data = pd.DataFrame(train_dict)


x_insertion_train = train_data["train_k"].values.reshape(-1,1)
y_insertion_train = train_data["train_insertion"].values.reshape(-1,1)

## 預測資料
test_df = hw2_data[0:30]
test_data1 = test_df["k"].values.reshape(-1,1)

test_list = list()
for i in range(0,2**18,100):
    test_list.append(i)
test_dict = {"testing":test_list}
test_data2 = pd.DataFrame(test_dict)
```

```
## 訓練不同次方多項式的迴歸模型

## 裝不同次方多項式的分數
scores = list()

## 設定欲實驗的次方項
polynomial_degree = list()
for i in range(1,20):
   polynomial_degree.append(i)

for index, num in enumerate(polynomial_degree):
  ## 訓練多項式迴歸模型
            regressor            =            make_pipeline(PolynomialFeatures(num),
LinearRegression(fit_intercept=False))
 regressor.fit(x_insertion_train,y_insertion_train)
 ## 裝進精準度
 scores.append(regressor.score(x_insertion,y_insertion))


 ## 預測數據
 pred = regressor.predict(x_insertion)
  ## 視覺化多項式迴歸模型線
 plt.style.use("seaborn")
 plt.plot(x_insertion, regressor.predict(x_insertion), label = 'Polynomial Degree '
+ str(polynomial_degree[index]))


## 印出分數
max_index = scores.index(max(scores))
print('Polynomial    Degree    '    +    str(max_index+1)    +    '    Score:    '    +
str(scores[max_index]))

## 繪製數據集資料點
plt.scatter(x_insertion,y_insertion)
plt.ticklabel_format(style="plain")
plt.style.use("seaborn")
plt.legend()
plt.xlabel("2^k data in a unsorted array")
plt.title("Training the Regression Line")
plt.ylabel("time(s)")
plt.show()
```

```
## 最佳預測迴歸
top_regression = make_pipeline(PolynomialFeatures(max_index+1),
LinearRegression(fit_intercept=False))
## 訓練最佳預測迴歸模型
top_regression.fit(x_insertion_train,y_insertion_train)

## 預測數據
pred_data = regressor.predict(test_data1)

# 儲存預測數據
'''
for i in range(0,30):
   hw2_pred_data["Insertion sort"].iloc[i] = pred_data[i][0]
hw2_pred_data.to_csv("Assignment2_pred_data.csv", index=False)
'''
print(pred_data)


## 視覺化多項式迴歸模型線
plt.plot(test_data1,  regressor.predict(test_data1),  color  =  "red",  label  =
'Regression Line (polynomial degree '+str(max_index+1)+")")

## 繪製數據集資料點
plt.scatter(x_insertion,y_insertion,  color  =  "orange",  label  =  "Testing  average
time")
plt.style.use("seaborn")
plt.xlabel("2^k data in a unsorted array")
plt.ylabel("time(s)")
plt.ticklabel_format(style="plain")
plt.legend()
plt.title("The Best Regression Line")
plt.show()


## Time complexity line
x = np.linspace(0, 30, 1000)
y = (2**x)**2
plt.plot(x,y, color = "blue", label = "Time complexity O(n-square)")
plt.style.use("seaborn")
plt.title("The Time Complexity Line -- Insertion Sort")
plt.legend()
plt.ticklabel_format(style="plain")
plt.show()
```

（二）Merge sort

資料來源：https://www.programiz.com/dsa/merge-sort

```
# MergeSort in Python
# https://www.programiz.com/dsa/merge-sort
# Average time complexity : O(nlogn)
# Space complexity : O(n)



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


def mergeSort(array):
    if len(array) > 1:

        #  r is the point where the array is divided into two subarrays
        r = len(array)//2
        L = array[:r]
        M = array[r:]

        # Sort the two halves
        mergeSort(L)
        mergeSort(M)

        i = j = k = 0

        # Until we reach either end of either L or M, pick larger among
        # elements L and M and place them in the correct position at A[p..r]
        while i < len(L) and j < len(M):
            if L[i] < M[j]:
                array[k] = L[i]
                i += 1
            else:
                array[k] = M[j]
                j += 1
```

```
            k += 1

        # When we run out of elements in either L or M,
        # pick up the remaining elements and put in A[p..r]
        while i < len(L):
            array[k] = L[i]
            i += 1
            k += 1

        while j < len(M):
            array[k] = M[j]
            j += 1
            k += 1


# 輸入 Dataframe
merge = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment 2_1113/Merge sort.csv")
hw2_data = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment 2_1113/Assignment2_data.csv")
hw2_pred_data = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment 2_1113/Assignment2_pred_data.csv")



# Experiment
'''
merge = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment 2_1113/Merge sort.csv")

timelist = list()


for i in range(0,10):
   print(i)
   data = list()
   for j in range(0,2**1):
       data.append(random.randint(1,1000))
   start = time.time()
   mergeSort(data)
   end = time.time()
   timelist.append(end-start)

merge["k = 1"]  = timelist
```

```
merge.to_csv("Merge sort.csv", index=False)
'''
# 平均資料

merge_avg = hw2_data[0:26]
x_merge = merge_avg["k"].values.reshape(-1,1)
y_merge = merge_avg["Merge sort"].values.reshape(-1,1)

## 訓練資料

merge_train = list()
k_train = list()

for i in range(0,26):
   for j in range(0,10):
       merge_train.append(merge.iloc[j,i])
       k_train.append(hw2_data["k"].iloc[i])

train_dict = {"train_k" : k_train,"train_counting" : merge_train}
train_data = pd.DataFrame(train_dict)


x_merge_train = train_data["train_k"].values.reshape(-1,1)
y_merge_train = train_data["train_counting"].values.reshape(-1,1)

## 預測資料
test_df = hw2_data[0:30]
test_data1 = test_df["k"].values.reshape(-1,1)


## 訓練不同次方多項式的迴歸模型

## 裝不同次方多項式的分數
scores = list()

## 設定欲實驗的次方項
polynomial_degree = list()
for i in range(1,20):
   polynomial_degree.append(i)

for index, num in enumerate(polynomial_degree):
  ## 訓練多項式迴歸模型
           regressor           =           make_pipeline(PolynomialFeatures(num),
LinearRegression(fit_intercept=False))
 regressor.fit(x_merge_train,y_merge_train)
```

```
 ## 裝進精準度
 scores.append(regressor.score(x_merge,y_merge))


 ## 預測數據
 pred = regressor.predict(x_merge)
  ## 視覺化多項式迴歸模型線
  plt.plot(x_merge, regressor.predict(x_merge), label = 'Polynomial Degree ' +
str(polynomial_degree[index]))


## 印出分數

max_index = scores.index(max(scores))
print('Polynomial Degree ' + str(max_index+1) + ' Score: ' +
str(scores[max_index]))
print(scores)

## 繪製數據集資料點
plt.scatter(x_merge,y_merge)
plt.style.use("seaborn")
plt.ticklabel_format(style="plain")
plt.legend()
plt.xlabel("2^k data in a unsorted array")
plt.ylabel("time(s)")
plt.title("Training the Regression Line")
plt.show()


## 最佳預測迴歸
top_regression = make_pipeline(PolynomialFeatures(max_index+1),
LinearRegression(fit_intercept=False))
## 訓練最佳預測迴歸模型
top_regression.fit(x_merge_train,y_merge_train)

## 預測數據
pred_data = regressor.predict(test_data1)

# 儲存預測數據
'''
for i in range(0,30):
   hw2_pred_data["Merge sort"].iloc[i] = pred_data[i][0]
hw2_pred_data.to_csv("Assignment2_pred_data.csv", index=False)
'''
print(pred_data)
```

```
## 視覺化多項式迴歸模型線
plt.style.use("seaborn")
plt.plot(test_data1,  regressor.predict(test_data1),  color  =  "red",  label  =
'Regression Line (polynomial degree '+str(max_index+1)+")")

## 繪製數據集資料點
plt.scatter(x_merge,y_merge, color = "orange", label = "Testing average time")
plt.style.use("seaborn")
plt.xlabel("2^k data in a unsorted array")
plt.ylabel("time(s)")
plt.ticklabel_format(style="plain")
plt.title("The Best Regression Line")
plt.legend()
plt.show()

## Time complexity line
x = np.linspace(0, 30, 1000)
y = np.log(2**x)*(2**x)
plt.plot(x,y, color = "blue", label = "Time complexity O(nlogn)")
plt.style.use("seaborn")
plt.legend()
plt.title("The Time Complexity Line -- Merge Sort")
plt.ticklabel_format(style="plain")
plt.show()
```

（三）Counting sort

資料來源：https://www.programiz.com/dsa/counting-sort

```
# Counting sort in Python programming
# https://www.programiz.com/dsa/counting-sort
# Average time complexity : O(n+k)
# Space complexity : O(max)

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

def countingSort(array):
    size = len(array)
    output = [0] * size

    # Initialize count array
    count = [0] * 1001

    # Store the count of each elements in count array
    for i in range(0, size):
        count[array[i]] += 1

    # Store the cummulative count
    for i in range(1, 1001):
        count[i] += count[i - 1]

    # Find the index of each element of the original array in count array
    # place the elements in output array
    i = size - 1
    while i >= 0:
        output[count[array[i]] - 1] = array[i]
        count[array[i]] -= 1
        i -= 1

    # Copy the sorted elements into original array
    for i in range(0, size):
        array[i] = output[i]


# 輸入 Dataframe
counting = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment 2_1113/Counting sort.csv")
hw2_data = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment 2_1113/Assignment2_data.csv")
hw2_pred_data = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment 2_1113/Assignment2_pred_data.csv")
```

```
# Experiment
'''

timelist = list()


for i in range(0,10):
   print(i)
   data = list()
   for j in range(0,2**14):
       data.append(random.randint(1,1000))
   start = time.time()
   countingSort(data)
   end = time.time()
   timelist.append(end-start)


counting["k = 14"]  = timelist

counting.to_csv("Counting sort.csv", index=False)
'''




# 平均資料

counting_avg = hw2_data[0:26]
x_counting = counting_avg["k"].values.reshape(-1,1)
y_counting = counting_avg["Counting sort"].values.reshape(-1,1)

## 訓練資料

counting_train = list()
k_train = list()

for i in range(0,26):
   for j in range(0,10):
       counting_train.append(counting.iloc[j,i])
       k_train.append(hw2_data["k"].iloc[i])

train_dict = {"train_k" : k_train,"train_counting" : counting_train}
train_data = pd.DataFrame(train_dict)
```

```
x_counting_train = train_data["train_k"].values.reshape(-1,1)
y_counting_train = train_data["train_counting"].values.reshape(-1,1)

## 預測資料
test_df = hw2_data[0:30]
test_data1 = test_df["k"].values.reshape(-1,1)



## 訓練不同次方多項式的迴歸模型

## 裝不同次方多項式的分數
scores = list()

## 設定欲實驗的次方項
polynomial_degree = list()
for i in range(1,20):
   polynomial_degree.append(i)

for index, num in enumerate(polynomial_degree):
  ## 訓練多項式迴歸模型
            regressor            =            make_pipeline(PolynomialFeatures(num),
LinearRegression(fit_intercept=False))
 regressor.fit(x_counting_train,y_counting_train)
 ## 裝進精準度
 scores.append(regressor.score(x_counting,y_counting))


 ## 預測數據
 pred = regressor.predict(x_counting)
  ## 視覺化多項式迴歸模型線
 plt.plot(x_counting, regressor.predict(x_counting), label = 'Polynomial Degree ' +
str(polynomial_degree[index]))


## 印出分數
max_index = scores.index(max(scores))
print('Polynomial    Degree    '    +    str(max_index+1)    +    '    Score:    '    +
str(scores[max_index]))

## 繪製數據集資料點
plt.scatter(x_counting,y_counting)
plt.style.use("seaborn")
plt.ticklabel_format(style="plain")
```

```
plt.legend()
plt.xlabel("2^k data in a unsorted array")
plt.ylabel("time(s)")
plt.title("Training the Regression Line")
plt.show()


## 最佳預測迴歸
top_regression = make_pipeline(PolynomialFeatures(max_index+1),
LinearRegression(fit_intercept=False))
## 訓練最佳預測迴歸模型
top_regression.fit(x_counting_train,y_counting_train)

## 預測數據
pred_data = regressor.predict(test_data1)

# 儲存預測數據
'''
for i in range(0,30):
   hw2_pred_data["Counting sort"].iloc[i] = pred_data[i][0]
hw2_pred_data.to_csv("Assignment2_pred_data.csv", index=False)
'''
print(pred_data)


## 視覺化多項式迴歸模型線
plt.style.use("seaborn")
plt.plot(test_data1, regressor.predict(test_data1), color = "red", label =
'Regression Line (polynomial degree '+str(max_index+1)+")")

## 繪製數據集資料點
plt.scatter(x_counting,y_counting, color = "orange", label = "Testing average
time")
plt.style.use("seaborn")
plt.xlabel("2^k data in a unsorted array")
plt.ylabel("time(s)")
plt.ticklabel_format(style="plain")
plt.title("The Best Regression Line")
plt.legend()
plt.show()

## Time complexity line
x = np.linspace(0, 30, 1000)
y = 2**x+1000
plt.plot(x,y, color = "blue", label = "Time complexity O(n+k)")
```

```
plt.style.use("seaborn")
plt.legend()
plt.title("The Time Complexity Line -- Counting Sort")
plt.ticklabel_format(style="plain")
plt.show()
```

（四）Quick sort (Lomuto’s partition)

資料來源：https://www.geeksforgeeks.org/hoares-vs-lomuto-partition-scheme-quicksort/

```
# Quick sort(Lomuto's partition) in Python programming
# https://www.geeksforgeeks.org/hoares-vs-lomuto-partition-scheme-quicksort/


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




def partition(arr, low, high):

    # pivot
    pivot = arr[high]

    # Index of smaller element
    i = (low - 1)
    for j in range(low, high):

        # If current element is smaller than or
        # equal to pivot
        if (arr[j] <= pivot):
```

```
            # increment index of smaller element
            i += 1
            arr[i], arr[j] = arr[j], arr[i]
    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    return (i + 1)

''' The main function that implements QuickSort
arr --> Array to be sorted,
low --> Starting index,
high --> Ending index '''

def quickSort(arr, low, high):
    if (low < high):

        ''' pi is partitioning index, arr[p] is now
        at right place '''
        pi = partition(arr, low, high)

        # Separately sort elements before
        # partition and after partition
        quickSort(arr, low, pi - 1)
        quickSort(arr, pi + 1, high)

# 輸入 Dataframe
lomuto = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment
2_1113/Quick sort_Lomuto's partition.csv")
hw2_data = pd.read_csv("/Users/stevenkuo/College Course/大二/Data
Structure/Assignment 2_1113/Assignment2_data.csv")
hw2_pred_data = pd.read_csv("/Users/stevenkuo/College Course/大二/Data
Structure/Assignment 2_1113/Assignment2_pred_data.csv")



# Experiment
'''
Lomuto = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment
2_1113/Quick sort_Lomuto's partition.csv")

timelist = list()


for i in range(0,10):
    print(i)
    data = list()
    for j in range(0,2**12):
```

```
        data.append(random.randint(1,1000))
    n = len(data)
    start = time.time()
    quickSort(data,0,n-1)
    end = time.time()
    timelist.append(end-start)


Lomuto["k = 12"]  = timelist

Lomuto.to_csv("Quick sort_Lomuto's partition.csv", index=False)
'''

# 平均資料

lomuto_avg = hw2_data[0:23]
x_lomuto = lomuto_avg["k"].values.reshape(-1,1)
y_lomuto = lomuto_avg["Quick sort (Lomuto's partition)"].values.reshape(-1,1)

## 訓練資料

lomuto_train = list()
k_train = list()

for i in range(0,23):
    for j in range(0,10):
        lomuto_train.append(lomuto.iloc[j,i])
        k_train.append(hw2_data["k"].iloc[i])

train_dict = {"train_k" : k_train,"train_lomuto" : lomuto_train}
train_data = pd.DataFrame(train_dict)


x_lomuto_train = train_data["train_k"].values.reshape(-1,1)
y_lomuto_train = train_data["train_lomuto"].values.reshape(-1,1)

## 預測資料
test_df = hw2_data[0:30]
test_data1 = test_df["k"].values.reshape(-1,1)



## 訓練不同次方多項式的迴歸模型

## 裝不同次方多項式的分數
```

```
scores = list()

## 設定欲實驗的次方項
polynomial_degree = list()
for i in range(1,20):
   polynomial_degree.append(i)

for index, num in enumerate(polynomial_degree):
  ## 訓練多項式迴歸模型
 regressor = make_pipeline(PolynomialFeatures(num),
LinearRegression(fit_intercept=False))
 regressor.fit(x_lomuto_train,y_lomuto_train)
 ## 裝進精準度
 scores.append(regressor.score(x_lomuto,y_lomuto))


 ## 預測數據
 pred = regressor.predict(x_lomuto)
  ## 視覺化多項式迴歸模型線
 plt.plot(x_lomuto, regressor.predict(x_lomuto), label = 'Polynomial Degree ' +
str(polynomial_degree[index]))


## 印出分數
max_index = scores.index(max(scores))
print('Polynomial Degree ' + str(max_index+1) + ' Score: ' +
str(scores[max_index]))

## 繪製數據集資料點
plt.scatter(x_lomuto,y_lomuto)
plt.style.use("seaborn")
plt.ticklabel_format(style="plain")
plt.legend()
plt.xlabel("2^k data in a unsorted array")
plt.ylabel("time(s)")
plt.title("Training the Regression Line")
plt.show()


## 最佳預測迴歸
top_regression = make_pipeline(PolynomialFeatures(max_index+1),
LinearRegression(fit_intercept=False))
## 訓練最佳預測迴歸模型
top_regression.fit(x_lomuto_train,y_lomuto_train)
```

```
## 預測數據
pred_data = regressor.predict(test_data1)

# 儲存預測數據
'''
for i in range(0,30):
   hw2_pred_data["Quick sort (Lomuto's partition)"].iloc[i] = pred_data[i][0]
hw2_pred_data.to_csv("Assignment2_pred_data.csv", index=False)
'''
print(pred_data)


## 視覺化多項式迴歸模型線
plt.style.use("seaborn")
plt.plot(test_data1, regressor.predict(test_data1), color = "red", label =
'Regression Line (polynomial degree '+str(max_index+1)+")")

## 繪製數據集資料點
plt.scatter(x_lomuto,y_lomuto, color = "orange", label = "Testing average time")
plt.style.use("seaborn")
plt.xlabel("2^k data in a unsorted array")
plt.ylabel("time(s)")
plt.ticklabel_format(style="plain")
plt.title("The Best Regression Line")
plt.legend()
plt.show()




# This code is contributed by SHUBHAMSINGH10
```

（五）Quick sort (Hoare's partition)

資料來源：https://www.geeksforgeeks.org/hoares-vs-lomuto-partition-scheme-quicksort/

```
# Quick sort(Hoare's partition) in Python programming
# https://www.geeksforgeeks.org/hoares-vs-hoare-partition-scheme-quicksort/


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



def partition(arr, low, high):

   pivot = arr[low]
   i = low - 1
   j = high + 1

   while (True):

       # Find leftmost element greater than
       # or equal to pivot
       i += 1
       while (arr[i] < pivot):
           i += 1

       # Find rightmost element smaller than
       # or equal to pivot
       j -= 1
       while (arr[j] > pivot):
           j -= 1

       # If two pointers met.
       if (i >= j):
           return j

       arr[i], arr[j] = arr[j], arr[i]
```

```
''' The main function that implements QuickSort
arr --> Array to be sorted,
low --> Starting index,
high --> Ending index '''


def quickSort(arr, low, high):
    ''' pi is partitioning index, arr[p] is now
    at right place '''
    if (low < high):

        pi = partition(arr, low, high)

        # Separately sort elements before
        # partition and after partition
        quickSort(arr, low, pi)
        quickSort(arr, pi + 1, high)


''' Function to print an array '''

# 輸入 Dataframe
hoare = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment
2_1113/Quick sort_Hoare's partition.csv")
hw2_data = pd.read_csv("/Users/stevenkuo/College Course/大二/Data
Structure/Assignment 2_1113/Assignment2_data.csv")
hw2_pred_data = pd.read_csv("/Users/stevenkuo/College Course/大二/Data
Structure/Assignment 2_1113/Assignment2_pred_data.csv")


# Experiment
'''
Hoare = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment
2_1113/Quick sort_Hoare's partition.csv")

timelist = list()


for i in range(0,10):
    print(i)
    data = list()
    for j in range(0,2**14):
        data.append(random.randint(1,1000))
    n = len(data)
```

```
    start = time.time()
    quickSort(data,0,n-1)
    end = time.time()
    timelist.append(end-start)


Hoare["k = 14"]  = timelist

Hoare.to_csv("Quick sort_Hoare's partition.csv", index=False)
'''

# 平均資料

hoare_avg = hw2_data[0:27]
x_hoare = hoare_avg["k"].values.reshape(-1,1)
y_hoare = hoare_avg["Quick sort (Hoare's partition)"].values.reshape(-1,1)

## 訓練資料

hoare_train = list()
k_train = list()

for i in range(0,27):
    for j in range(0,10):
        hoare_train.append(hoare.iloc[j,i])
        k_train.append(hw2_data["k"].iloc[i])

train_dict = {"train_k" : k_train,"train_hoare" : hoare_train}
train_data = pd.DataFrame(train_dict)


x_hoare_train = train_data["train_k"].values.reshape(-1,1)
y_hoare_train = train_data["train_hoare"].values.reshape(-1,1)

## 預測資料
test_df = hw2_data[0:30]
test_data1 = test_df["k"].values.reshape(-1,1)



## 訓練不同次方多項式的迴歸模型

## 裝不同次方多項式的分數
scores = list()
```

```
## 設定欲實驗的次方項
polynomial_degree = list()
for i in range(1,20):
   polynomial_degree.append(i)

for index, num in enumerate(polynomial_degree):
  ## 訓練多項式迴歸模型
 regressor = make_pipeline(PolynomialFeatures(num),
LinearRegression(fit_intercept=False))
 regressor.fit(x_hoare_train,y_hoare_train)
 ## 裝進精準度
 scores.append(regressor.score(x_hoare,y_hoare))


 ## 預測數據
 pred = regressor.predict(x_hoare)
  ## 視覺化多項式迴歸模型線
 plt.plot(x_hoare, regressor.predict(x_hoare), label = 'Polynomial Degree ' +
str(polynomial_degree[index]))


## 印出分數
max_index = scores.index(max(scores))
print('Polynomial Degree ' + str(max_index+1) + ' Score: ' +
str(scores[max_index]))

## 繪製數據集資料點
plt.scatter(x_hoare,y_hoare)
plt.style.use("seaborn")
plt.ticklabel_format(style="plain")
plt.legend()
plt.xlabel("2^k data in a unsorted array")
plt.ylabel("time(s)")
plt.title("Training the Regression Line")
plt.show()


## 最佳預測迴歸
top_regression = make_pipeline(PolynomialFeatures(max_index+1),
LinearRegression(fit_intercept=False))
## 訓練最佳預測迴歸模型
top_regression.fit(x_hoare_train,y_hoare_train)

## 預測數據
pred_data = regressor.predict(test_data1)
```

```
# 儲存預測數據
'''
for i in range(0,30):
   hw2_pred_data["Quick sort (Hoare's partition)"].iloc[i] = pred_data[i][0]
hw2_pred_data.to_csv("Assignment2_pred_data.csv", index=False)
'''
print(pred_data)


## 視覺化多項式迴歸模型線
plt.style.use("seaborn")
plt.plot(test_data1, regressor.predict(test_data1), color = "red", label =
'Regression Line (polynomial degree '+str(max_index+1)+")")

## 繪製數據集資料點
plt.scatter(x_hoare,y_hoare, color = "orange", label = "Testing average time")
plt.style.use("seaborn")
plt.xlabel("2^k data in a unsorted array")
plt.ylabel("time(s)")
plt.ticklabel_format(style="plain")
plt.title("The Best Regression Line")
plt.legend()
plt.show()




# This code is contributed by shubhamsingh10
```

（六）Quick sort (three way partition)

資料來源：https://gist.github.com/adonese/4bf34d5b57ee0358626c

```
# python3 program for 3-way quick sort
# https://gist.github.com/adonese/4bf34d5b57ee0358626c

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

def partition3(A, l, r):

   lt = l  # We initiate lt to be the part that is less than the pivot
   i = l   # We scan the array from left to right
   gt = r  # The part that is greater than the pivot
   pivot = A[l]     # The pivot, chosen to be the first element of the array, that
why we'll randomize the first elements position
                  # in the quick_sort function.
   while i <= gt:      # Starting from the first element.
       if A[i] < pivot:
           A[lt], A[i] = A[i], A[lt]
           lt += 1
           i += 1
       elif A[i] > pivot:
           A[i], A[gt] = A[gt], A[i]
           gt -= 1
       else:
           i += 1

   return lt, gt



def quick_sort(A, l, r):
   """
   quick_sort: One of the most used sorting algorithm.
    It makes to recursive calls. One to sort the left part separately, other for
sorting the right part.
     The partition key is chosen randomly via ``random.randint(l, r)`` and it's
between the ``l,  r``.

   PARAMETERS:
   -----------
   A: Array or the sequence that we want to sort.
   l: The lower bound of the array that we want to sort. It's not very important we
might replace it by a wrapper function
    that only takes in an array as input. In this case it's the first element in the
left part of the array.
   r: It's the same as l, only differs as it's the first element from the end.
```

```
    RETURNS:
    -------
    Sorted list A.
    """
    if l >= r:
        return
    k = random.randint(l, r)
    A[k], A[l] = A[l], A[k]

    lt, gt = partition3(A, l, r)
    quick_sort(A, l, lt - 1)
    quick_sort(A, gt + 1, r)

# 輸入 Dataframe
three_way       =       pd.read_csv("/Users/stevenkuo/College       Course/大二/Data
Structure/Assignment 2_1113/Quick sort_3 way partition.csv")
hw2_data        =       pd.read_csv("/Users/stevenkuo/College       Course/大二/Data
Structure/Assignment 2_1113/Assignment2_data.csv")
hw2_pred_data       =       pd.read_csv("/Users/stevenkuo/College       Course/大二/Data
Structure/Assignment 2_1113/Assignment2_pred_data.csv")



# Experiment
'''
Three_way       =       pd.read_csv("/Users/stevenkuo/College       Course/大二/Data
Structure/Assignment 2_1113/Quick sort_3 way partition.csv")

timelist = list()


for i in range(0,10):
    print(i)
    data = list()
    for j in range(0,2**14):
        data.append(random.randint(1,1000))
    n = len(data)
    start = time.time()
    quick_sort(data,0,n-1)
    end = time.time()
    timelist.append(end-start)


Three_way["k = 14"]  = timelist
```

```
Three_way.to_csv("Quick sort_3 way partition.csv", index=False)
'''


# 平均資料

three_way_avg = hw2_data[0:27]
x_three_way = three_way_avg["k"].values.reshape(-1,1)
y_three_way = three_way_avg["Quick sort (3 way partition)"].values.reshape(-1,1)

## 訓練資料

three_way_train = list()
k_train = list()

for i in range(0,27):
   for j in range(0,10):
       three_way_train.append(three_way.iloc[j,i])
       k_train.append(hw2_data["k"].iloc[i])

train_dict = {"train_k" : k_train,"train_three_way" : three_way_train}
train_data = pd.DataFrame(train_dict)


x_three_way_train = train_data["train_k"].values.reshape(-1,1)
y_three_way_train = train_data["train_three_way"].values.reshape(-1,1)

## 預測資料
test_df = hw2_data[0:30]
test_data1 = test_df["k"].values.reshape(-1,1)



## 訓練不同次方多項式的迴歸模型

## 裝不同次方多項式的分數
scores = list()

## 設定欲實驗的次方項
polynomial_degree = list()
for i in range(1,10):
   polynomial_degree.append(i)

for index, num in enumerate(polynomial_degree):
  ## 訓練多項式迴歸模型
```

```
            regressor            =            make_pipeline(PolynomialFeatures(num),
LinearRegression(fit_intercept=False))
 regressor.fit(x_three_way_train,y_three_way_train)
 ## 裝進精準度
 scores.append(regressor.score(x_three_way,y_three_way))


 ## 預測數據
 pred = regressor.predict(x_three_way)
  ## 視覺化多項式迴歸模型線
 plt.plot(x_three_way, regressor.predict(x_three_way), label = 'Polynomial Degree '
+ str(polynomial_degree[index]))


## 印出分數
max_index = scores.index(max(scores))
print('Polynomial    Degree    '    +    str(max_index+1)    +    '    Score:    '    +
str(scores[max_index]))

## 繪製數據集資料點
plt.scatter(x_three_way,y_three_way)
plt.style.use("seaborn")
plt.ticklabel_format(style="plain")
plt.legend()
plt.xlabel("2^k data in a unsorted array")
plt.ylabel("time(s)")
plt.title("Training the Regression Line")
plt.show()


## 最佳預測迴歸
top_regression            =            make_pipeline(PolynomialFeatures(max_index+1),
LinearRegression(fit_intercept=False))
## 訓練最佳預測迴歸模型
top_regression.fit(x_three_way_train,y_three_way_train)

## 預測數據
pred_data = regressor.predict(test_data1)

# 儲存預測數據
'''
for i in range(0,30):
   hw2_pred_data["Quick sort (3 way partition)"].iloc[i] = pred_data[i][0]
hw2_pred_data.to_csv("Assignment2_pred_data.csv", index=False)
'''
```

```
print(pred_data)



## 視覺化多項式迴歸模型線
plt.style.use("seaborn")
plt.plot(test_data1,   regressor.predict(test_data1),   color   =   "red",   label   =
'Regression Line (polynomial degree '+str(max_index+1)+")")

## 繪製數據集資料點
plt.scatter(x_three_way,y_three_way,  color  =  "orange",  label  =  "Testing  average
time")
plt.style.use("seaborn")
plt.xlabel("2^k data in a unsorted array")
plt.ylabel("time(s)")
plt.ticklabel_format(style="plain")
plt.title("The Best Regression Line")
plt.legend()
plt.show()
```

（七）Graphing code

```
import pandas as pd
import numpy as np
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




hw2_pred_data = pd.read_csv("/Users/stevenkuo/College Course/大二/Data
Structure/Assignment 2_1113/Assignment2_pred_data.csv")
hw2_data = pd.read_csv("/Users/stevenkuo/College Course/大二/Data
Structure/Assignment 2_1113/Assignment2_data.csv")

#1~18
```

```
insertion = pd.read_csv("/Users/stevenkuo/College Course/大二/Data
Structure/Assignment 2_1113/Insertion sort.csv")
#1~26
merge = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment
2_1113/Merge sort.csv")
#1~26
counting = pd.read_csv("/Users/stevenkuo/College Course/大二/Data
Structure/Assignment 2_1113/Counting sort.csv")
#1~23
lomuto = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment
2_1113/Quick sort_Lomuto's partition.csv")
#1~27
hoare = pd.read_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment
2_1113/Quick sort_Hoare's partition.csv")
#1~27
three_way = pd.read_csv("/Users/stevenkuo/College Course/大二/Data
Structure/Assignment 2_1113/Quick sort_3 way partition.csv")


# raw data plot
plt.style.use("seaborn")
hw2_data.plot("k",["Insertion sort","Merge sort","Counting sort","Quick sort
(Lomuto's partition)","Quick sort (Hoare's partition)","Quick sort (3 way
partition)"])
plt.ticklabel_format(style="plain")
plt.title("Experiment data")
plt.legend()
plt.show()

# raw prediction data
plt.style.use("seaborn")
hw2_pred_data.plot("k",["Insertion sort","Merge sort","Counting sort","Quick sort
(Lomuto's partition)","Quick sort (Hoare's partition)","Quick sort (3 way
partition)"])
plt.ticklabel_format(style="plain")
plt.title("Prediction data")
plt.legend()
plt.show()

# raw prediction data (without the top 2)
plt.style.use("seaborn")
hw2_pred_data.plot("k",["Merge sort","Counting sort","Quick sort (Hoare's
partition)","Quick sort (3 way partition)"])
plt.ticklabel_format(style="plain")
plt.legend()
```

```
plt.title("Prediction data")
plt.show()




# raw data with units
plt.style.use("seaborn")
hw2_data.plot("units",["Insertion sort","Merge sort","Counting sort","Quick sort
(Lomuto's partition)","Quick sort (Hoare's partition)","Quick sort (3 way
partition)"])
plt.ticklabel_format(style="plain")
plt.title("Experiment Data")
plt.legend()
plt.show()



# time complexity plot
x1 = np.linspace(0, 2**30, 1000)
y1 = x1**2
plt.plot(x1,y1, color = "blue", label = "Time complexity O(n-square)")
x2 = np.linspace(0, 2**30, 1000)
y2 = x2*np.log2(x2)
plt.plot(x2,y2, color = "red", label = "Time complexity O(nlogn)")
x3 = np.linspace(0, 2**30, 1000)
y3 = x3+1000
x4 = np.linspace(0, 2**30, 1000)
y4 = x4*np.log2(x4)*9
plt.plot(x4,y4, color = "blue", label = "Time complexity O(9nlogn)")
plt.plot(x3,y3, color = "orange", label = "Time complexity O(n+k)")
plt.style.use("seaborn")
plt.title("The Time Complexity Line")
plt.legend()
plt.ticklabel_format(style="plain")
plt.show()


# time complexity plot (without n^2)

x2 = np.linspace(0, 2**30, 1000)
y2 = x2*np.log2(x2)
plt.plot(x2,y2, color = "red", label = "Time complexity O(nlogn)")
x3 = np.linspace(0, 2**30, 1000)
y3 = x3+1000
```

```
plt.plot(x3,y3, color = "orange", label = "Time complexity O(n+k)")
x4 = np.linspace(0, 2**30, 1000)
y4 = x4*np.log2(x4)*9
plt.plot(x4,y4, color = "blue", label = "Time complexity O(9nlogn)")
plt.style.use("seaborn")
plt.title("The Time Complexity Line")
plt.legend()
plt.ticklabel_format(style="plain")
plt.show()
```

三、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發生在做實驗的時候，因為我是一個次方直接讓程式自己跑十次，所以有時遇到高次方的時候就會跑5小時以上，記得我第一天讓程式自己在我睡覺時跑時都很正常。不過當我把筆電帶出門時，那個數據永遠都長得很奇怪，後來翻了很多資料知道，作業系統在為充電狀態的時候會自動調整CPU使用，以調整電池用量。了解之後，我的實驗就都在充電的環境下進行。

心得

以下二點是我這次實驗收穫最大的事：

1) 解讀數據的方法

與其說這是我學到的事情，將直接一點就是我認為自己還做得不足的地方，關於這次預測數據的手法，我自認為我沒有找到一個最佳的方式去預測數據，因為一時間複雜度來看的話，這次的實驗除了Counting sort之外都是Exponential，這也可能是導致我預測數據與實驗數據有些微微差的原因，而我認為我自己做得最好的地方是我很知道自己的預測模型是怎麼運作的而非只是把數據丟到一個模型裡讓他自己跑出來。

2) 自我解決問題的能力

我在這次的作業和上一個作業非常不一樣，在作業一我都透過提問的方式去找到很多人然後得到答案，而這次的作業大部分的問題都是我自己查資料解決的，包括上述提到的作業系統在未充電狀態時

會自動調整C P U, 自己查資料去調整python的recursion limitation, 解決Quicksort跑不出來的問題, 甚至這次預測資料的套件也是我自己上網查學的，雖然自己解決問題的速度會比提問還來得慢，不過這個過程很紮實也很有成就感, 也是我在做作業二的時候最開心的事情。