資料庫管理 HW01

B12508026 戴偉璿

1. (a) 使用 python 統計結果如下：

Workday	Registered	Casual
0	683,537	316,732
1	1,989,125	303,285

Table 1: 依工作日與否統計會員與非會員的總租借數量

由 Table 1可得知，會員在工作日的需求量遠大於非工作日；非會員在非工作日的需求量則略高於工作日。但考量到可能有極端數據的影響，我也同時也統計了「會員在上班日租借大於非上班日之平均/中位數」、「非會員在非上班日租借大於上班日之平均/中位數」的天數：(上班日 500 天，非上班日 231 天)

CMP base	Registered (workday > non-workday)	Casual (non-workday > workday)
Mean	379 (75.8%)	173 (74.8%)
Median	381 (76.2%)	172 (74.5%)

Table 2: 使用平均數與中位數比較假說成立的天數

由 Table 2可得知，無論是使用平均數或中位數來比較，皆有約七成五的天數符合假說，綜合 Table 1 和 Table 2的結果，因此「會員需求在上班日比非上班日高、非會員需求在非上班日比上班日高」假說成立。

以下是程式碼：

```python
import pandas as pd

data = pd.read_csv('Bike.csv')

summary = data.groupby("workday")[["registered", "casual"]].sum()
print(summary)

print("sum of workday", (data["workday"] == 1).sum())
print("sum of non-workday", (data["workday"] == 0).sum())
```

```
10
11          # not a workday, registered mean
12          avg_registered_nonwork = data.loc[data["workday"]==0, "registered"].mean()
13
14          # not a workday, registered median
15          med_registered_nonwork = data.loc[data["workday"]==0, "registered"].median()
16
17          # a workday, casual mean
18          avg_casual_work = data.loc[data["workday"]==1, "casual"].mean()
19
20          # a workday, casual median
21          med_casual_work = data.loc[data["workday"]==1, "casual"].median()
22
23          # cmp by mean
24          registered_support_days = (data.loc[data["workday"]==1, "registered"] >
            ↪  avg_registered_nonwork).sum()
25          casual_support_days = (data.loc[data["workday"]==0, "casual"] >
            ↪  avg_casual_work).sum()
26
27          print("sum of registered support work > no work: ", registered_support_days)
28          print("sum of casual support no work > work: ", casual_support_days)
29
30          # cmp by median
31          registered_support_days = (data.loc[data["workday"]==1, "registered"] >
            ↪  med_registered_nonwork).sum()
32          casual_support_days = (data.loc[data["workday"]==0, "casual"] >
            ↪  med_casual_work).sum()
33          print("sum of registered support work > no work: ", registered_support_days)
34          print("sum of casual support no work > work: ", casual_support_days)
```

(b) 使用 python 統計結果如下：

Month	2011		2012	
	Registered	Casual	Registered	Casual
1	35,116	3,073	87,775	8,969
2	41,973	6,242	94,416	8,721
3	51,219	12,826	133,257	31,618
4	72,524	22,346	135,768	38,456
5	104,771	31,050	151,630	44,235
6	112,900	30,612	159,536	43,294
7	104,889	36,452	161,902	41,705
8	107,849	28,842	171,306	43,197
9	100,873	26,545	174,795	43,778
10	98,289	25,222	164,303	34,538
11	86,573	15,594	131,655	21,009
12	78,875	8,448	110,468	13,245

Table 3: 2011 與 2012 年各月會員與非會員租借總數比較

由 Table 3可知，會員在每個月的租借量皆大於非會員，因此「不論哪一個月，會員總需求總是比非會員總需求高」假說成立。

以下是程式碼：

```
import pandas as pd

data = pd.read_csv('Bike.csv')

monthly = data.groupby(["year", "month"])[["registered", "casual"]].sum()
monthly = monthly.sort_index()
print(monthly)

```

(c) 使用 python 統計，2011 年標準差約為 1378.75；2012 年則是 1788.67，因此「第二年的需求變異程度高於第一年」假說成立。

以下是程式碼：

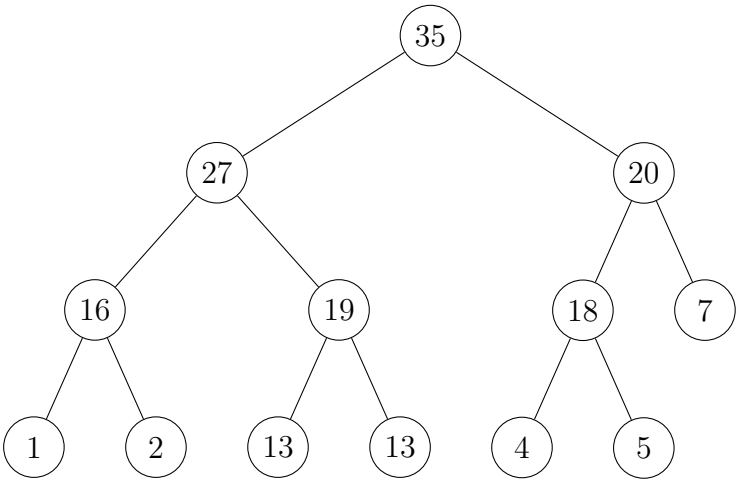
```
import pandas as pd

data = pd.read_csv('Bike.csv')

print("std of 2011 : ", data.loc[data["year"]==2011, "cnt"].std())
print("std of 2012 : ", data.loc[data["year"]==2012, "cnt"].std())
```

2. (a) 就複雜度分析的層面，在比較字串時最糟糕的狀況皆是比較到最後一個字元，因此交換指標相較於交換字串而言並不會有複雜度層面的改善。但就實際操作層面，每次交換指標時僅須交換指標的值，而不需要實際複製字串內容，這樣可以減少不必要的記憶體操作，提高效率。

(b) 答案如下：



(c) 可以。在一個樹狀結構中，每一個節點會有至多一個唯一的「最左子節點」及「最近右兄弟節點」；此外，在一個 Left-child-right-sibling 表示法的二元樹狀結構中，每個節點的「左子節點」以及「右子節點」也是唯一的，因此必然能在此結構中找到對應的樹狀結構的節點。此時，二元樹中的「左子節點」對應到樹狀結構中的「最左子節點」，而二元樹中的「右子節點」則對應到樹狀結構中的「最近右兄弟節點」。

3. (a) 作業系統通常包含了：程序與執行序、CPU 排程、記憶體管理、檔案系統、I/O 系統管理、網路管理、安全性與權限管理等模組。

(b)「哲學家用餐問題」是指，有幾位哲學家坐在圓桌要吃飯，每兩個人之間放一隻筷子，要同時取得兩隻才能就餐。哲學家們可能會「思考」或是「就餐」，當他們選擇就餐時就會去拿筷子。但由於筷子的數量有限，因此可能會發生「deadlock」的情況：每個哲學家都拿到一隻筷子，等不到另一隻。這個問題的本質是資源分配，不同程序會競爭有限的資源，若沒有適當分配，可能會導致系統無法繼續運作。其中一種解法為「服務生解法」，亦即在桌邊放一個服務生，當哲學家要拿筷子時必須先向服務生取得許可，服務生會確保不會讓所有哲學家同時拿到一隻筷子，這樣就能避免死結的發生。這個解法的概念和通訊協定中的令牌傳遞（Token Passing）類似，都是透過一個中介來控制資源的分配，確保系統能夠順利運作。

(c) 連續分配的原則是，在 HDD 中，一個檔案的資料儲存在連續的區域中。優點是在讀取時指針只要尋找一次起始位置，可以加快 I/O 速度以及減少磨損；此外，將資料放在連續的位置也可以支援同一個檔案中隨機存取，因為檔案是連續存放，起始位置加上偏移量就能直接找到目標區塊。但缺點也很顯然，長時間使用後，雖然總空間足夠，但可能沒有「足夠大的連續空間」，導致檔案無法配置。而且如果檔案會成長，事先要預留額外空間，否則超出時可能需要整個搬移到另一個更大的連續區塊，這使空間利用率不佳。在 SSD 中，檔案不需要連續存放也可以快速的查找，連續分配失去其優勢但劣勢依然存在，因此就沒必要連續分配了。

(d) 虛擬記憶體的目的是讓每個程式都能使用一個獨立的、連續的邏輯位址空間，而不需要完全受限於實體記憶體的大小。當程式切換或同時執行多個程式時，作業系統不必整個搬移程序，只需在需要的時候將對應的頁面從磁碟的 swap 區載入到記憶體；不活躍的頁面則可先換出。如此即可有效支援多工，並提升記憶體利用率。