

嗨～大家晚安

若 messages 數量是會變動的，那 task1 的時間複雜度？

先想想這個題目裡除了收到的 messages 數量，還有哪些可能會變動的因子：

1. 使用的捷運路線：主線站數、支線站數、不重複總站數

2. messages 中平均 message 的字數

那，我們先定義如下：

- 捷運線主線站數  $x$
- 支線站數  $y$
- 不重複總站數  $s$
- 訊息字數平均長度  $m$
- messages 數量  $n$

# 先大概說一下時間複雜度

BigO Notation 符號  $O()$

1. 在一個函式運作完成的過程，隨著數據量增加，執行時間受影響而**增加的趨勢**(比如：線性增加、指數增加...)

2. 考慮最糟情形：worst case，也就是在比較不同的運行方法時，在意的是：當**極大量輸入**時，比較執行時間如何增加，才能顯出方法的效能差異

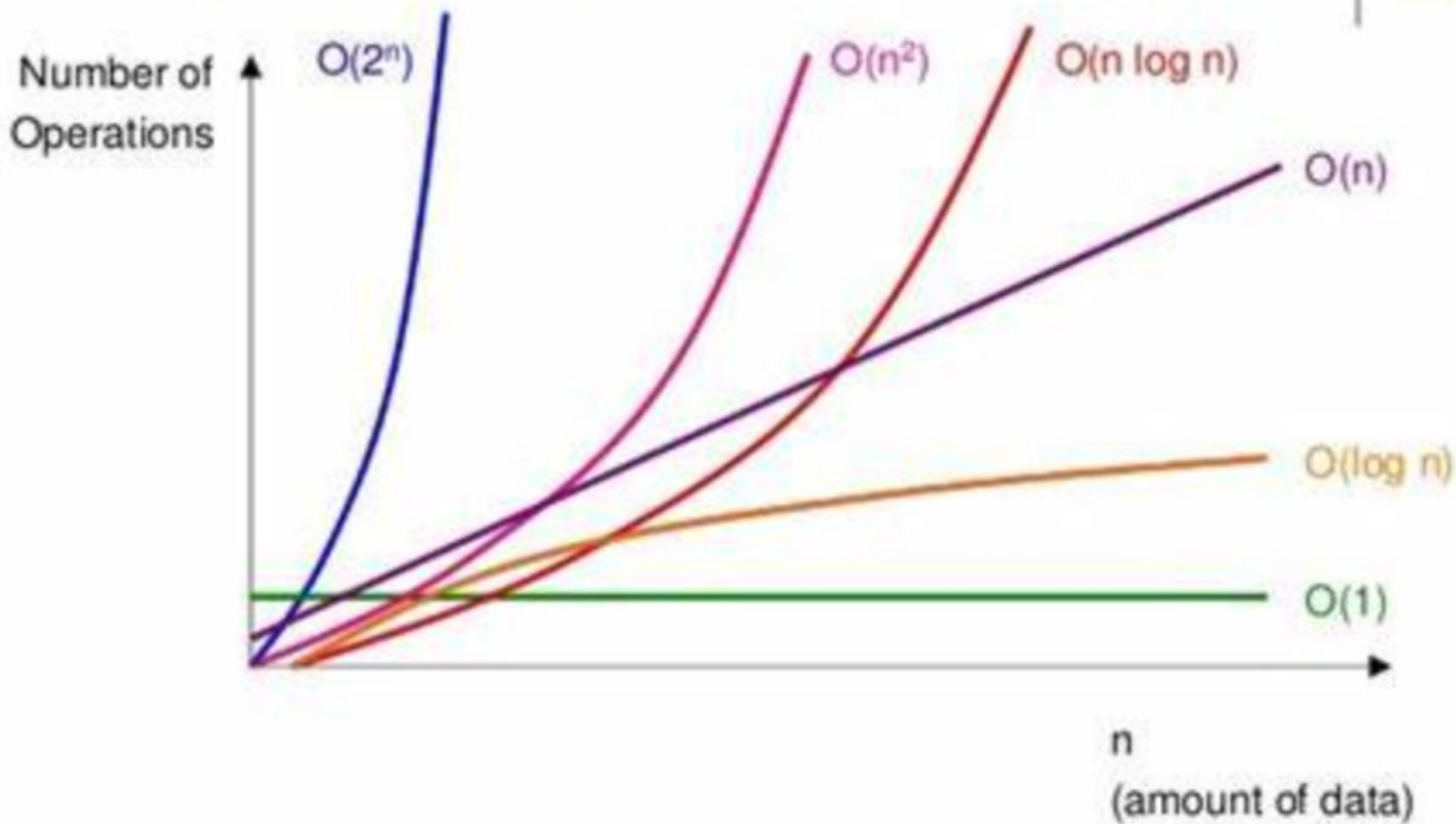
EX:

$O(1)$ : 常數時間複雜度，代表執行時間不會隨輸入數據量的變化而變化

$O(n)$ : 線性時間複雜度，執行時間跟輸入數據量成正比

$O(n^2)$ : 平方時間複雜度，常見於雙重迴圈

# Comparing Big O Functions



## 常數時間複雜度 $O(1)$

1. 無論輸入數據量多大，執行時間都保持不變

2. 舉例：access 陣列中的某個元素

以下函式，無論陣列 `arr` 有多長，`getFirstElement` 函數都只進行一次操作即可取得第一個元素。

JavaScript ▾

```
function getFirstElement(arr) {  
    return arr[0];  
}
```

## 線性時間複雜度 $O(n)$

1. 運行時間與輸入數據量成正比時，具有線性時間複雜度

2. 舉例：計算陣列所有元素的和

這個函式需要遍歷陣列 `arr` 的每一個元素來計算總和，所以隨著陣列長度的增加，所需的執行時間也會增加。

JavaScript ▾

```
function sumArray(arr) {  
  let total = 0;  
  for (let num of arr) {  
    total += num;  
  }  
  return total;  
}
```

## 平方時間複雜度 $O(n^2)$

1. 若有一個運算包含兩層嵌套迴圈，且每一層迴圈都遍歷輸入數據時，通常具有 $O(n^2)$ 的時間複雜度。

2. 若今天 `matrix = [ [1,2,3], [3,2,1] ]`，要知道 `matrix` 中所有元素其值總和  
外迴圈 => 有幾個子陣列 (假設 `i`)

```
[ [1,2,3],  
  [3,2,1] ]
```

內迴圈 => 每個子陣列有幾個元素 (假設 `j`)

那若想讓 `i`、`j` 一樣變這樣：

`matrix = [ [1,2,3], [3,2,1], [3,2,1] ]`

若 `i` 變大 ---> `n`

```
[ [1,2,3],  
  [3,2,1],  
  [0,0,0] ]
```

若 `j` 變大 ---> `n`

外迴圈 `n`、內迴圈 `n` =>  $O(n^2)$

```
function sumMatrix(matrix) {  
  let total = 0;  
  for (let i = 0; i < matrix.length; i++) {  
    for (let j = 0; j < matrix[i].length; j++) {  
      total += matrix[i][j];  
    }  
  }  
  return total;  
}
```

# 先上捷運表

三個物件: stationsMap(用來對照訊息中有沒有符合這條捷運的站名)、stationsMain(找站名在主線上位置)、stationsBranch(找站名在支線上位置)

```
const stationsMap = [  
  "Songshan",  
  "Nanjing Sanmin",  
  "Taipei Arena",  
  "Nanjing Fuxing",  
  "Songjiang Nanjing",  
  "Zhongshan",  
  "Beimen",  
  "Ximen",  
  "Xiaonanmen",  
  "Chiang Kai-Shek Memorial Hall",  
  "Guting",  
  "Taipower Building",  
  "Gongguan",  
  "Wanlong",  
  "Jingmei",  
  "Dapinglin",  
  "Qizhang",  
  "Xiaobitan",  
  "Xindian City Hall",  
  "Xindian",  
];
```

```
const mainStations = [  
  "Songshan",  
  "Nanjing Sanmin",  
  "Taipei Arena",  
  "Nanjing Fuxing",  
  "Songjiang Nanjing",  
  "Zhongshan",  
  "Beimen",  
  "Ximen",  
  "Xiaonanmen",  
  "Chiang Kai-Shek Memorial Hall",  
  "Guting",  
  "Taipower Building",  
  "Gongguan",  
  "Wanlong",  
  "Jingmei",  
  "Dapinglin",  
  "Qizhang",  
  "Xindian City Hall",  
  "Xindian",  
];
```

```
const branchStations = [  
  "Songshan",  
  "Nanjing Sanmin",  
  "Taipei Arena",  
  "Nanjing Fuxing",  
  "Songjiang Nanjing",  
  "Zhongshan",  
  "Beimen",  
  "Ximen",  
  "Xiaonanmen",  
  "Chiang Kai-Shek Memorial Hall",  
  "Guting",  
  "Taipower Building",  
  "Gongguan",  
  "Wanlong",  
  "Jingmei",  
  "Dapinglin",  
  "Qizhang",  
  "Xiaobitan",  
];
```



# 進入 findAndPrint(messages,currentStation)函式

迴圈外做的事：

```
function findAndPrint(messages, currentStation) {  
  const messagesPeople = Object.keys(messages);  
  
  let currentIndex = mainStations.indexOf(currentStation);  
  const XiaobitanIndex = branchStations.indexOf("Xiaobitan");  
  if (currentStation === "Xiaobitan") {  
    currentIndex = XiaobitanIndex;  
  }  
  if (currentIndex === -1) {  
    console.log("Station not found in the array.");  
    return;  
  }  
}
```

目前的捷運分支上就只有小碧潭站，所以分支直接找小碧潭位置，currentStation就三種可能：主線某站、小碧潭站、不在松山新店線

a. 將傳送 messages 的朋友 (messages 的 key 值) 先蒐集成一個陣列  $\Rightarrow$  時間複雜度  $O(n)$

b. 依據 findAndPrint 函式接收到的變數之一 currentStation 去捷運主線上查找位置  $\Rightarrow$  時間複雜度  $O(x)$

c. 找小碧潭站在捷運支線上的位置  $\Rightarrow$  時間複雜度  $O(y)$

```
let minDistance = Infinity;
let nearestFriend = "";
```

```
for (let i = 0; i < messagesPeople.length; i++) {
  const message = messages[messagesPeople[i]];
  const stationName = extractStationName(message);
  let stationIndex;
  if (stationName && stationName !== "Xiaobitan") {
    stationIndex = mainStations.indexOf(stationName);
  } else if (stationName && stationName === "Xiaobitan") {
    stationIndex = branchStations.indexOf(stationName);
  }

  if (stationIndex !== -1) {
    let distance;
```

每次迴圈處理一則輸入訊息，n則訊息則迴圈跑 n次，以迴圈本身時間複雜度為 $O(n)$

接著看迴圈內部做的事

## 進入迴圈

```
    if (
      (currentStation === "Xiaobitan" &&
        (stationName === "Xindian City Hall" || stationName === "Xindian")) ||
      (currentStation === "Xindian City Hall" &&
        stationName === "Xiaobitan") ||
      (currentStation === "Xindian" && stationName === "Xiaobitan")
    ) {
      distance = Math.abs(currentIndex - stationIndex) + 2;
    } else {
      distance = Math.abs(currentIndex - stationIndex);
    }

    if (distance < minDistance) {
      minDistance = distance;
      nearestFriend = messagesPeople[i];
    }
  }
}

console.log(nearestFriend);
}
```

# 內部主要做了兩件主要會影響整體時間複雜度的事：

1. 將每則輸入的訊息截取出符合我們使用捷運線的站名
2. 查找截取出的站名在捷運主線或支線的位置

第 1 點用了一個extractStationName函式：

```
function extractStationName(message) {  
  const regex = new RegExp(stationsMap.join("|"), "g");  
  const match = regex.exec(message);  
  
  if (match) {  
    return match[0];  
  } else {  
    return null;  
  }  
}
```

```
[  
  'Ximen',  
  index: 7,  
  input: "I'm at Ximen MRT station.",  
  groups: undefined  
]
```

正則表達式的執行時間複雜度會依賴於多個因素：輸入文本的長度、正則表達式的結構等等，這邊先簡化影響因子只有輸入文本的長度

extractStationName函式內部：使用正則表達式從訊息截取出符合站名

=>像拿著一個對照圖在每條訊息的字裡蒐尋，因此跟一則訊息字數量有關

時間複雜度 $O(m)$

# 正則表達式

可用來識別字符串中的特定模式

例如，可以用它來檢查一段文字是否符合電子郵件地址的格式、是否含有有效的電話號碼，或者提取文字檔中的特定訊息等。

有一個陣列，其中包含了多個單詞，創建一個正則表達式，可用 `join('|')` 方法，其會將陣列中的元素使用 `|` (正則表達式中的“或”運算符) 連接成一個新的字符串，然後將這個字符串轉換為正則表達式。

例如：

```
let words = ['apple', 'banana', 'cherry'];  
let regex = new RegExp(words.join('|'), 'g');
```

(stationsMap.join("|"), "g")生成的字串看起來像這樣  
:"Songshan|Nanjing Sanmin|Taipei Arena|...|Xindian"

`words.join('|')` 會產生字符串  
"apple|banana|cherry"

`new RegExp` 則將其轉換成正則表達式  
'/apple|banana|cherry/g'

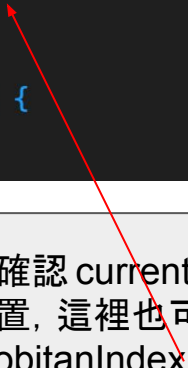
`g` 是全局標記，每次執行正則表達式查找後會在內部用 `lastIndex` 記錄上一次匹配完成後的索引，以利下次執行

`g` 搭配多次 `exec()` 能得到 `message` 中所有的匹配

# 內部主要做了兩件主要會影響整體時間複雜度的事：

## 第2點：查找截取出的站名在捷運主線或支線的位置

```
let stationIndex;  
if (stationName && stationName !== "Xiaobitan") {  
  stationIndex = mainStations.indexOf(stationName);  
} else if (stationName && stationName === "Xiaobitan") {  
  stationIndex = branchStations.indexOf(stationName);  
}  
  
if (stationIndex !== -1) {  
  let distance;
```



剛開始在迴圈外要確認 currentStation 時，就有找過小碧潭站位置，這裡也可簡化成 `stationIndex = XiaobitanIndex`  
時間複雜度： $O(\max(x, 1))$

假設剛剛那步驟有找到符合的，那接著要找它在主線或支線的位置在哪？  
先在主線找，主線找不到，再去支線找

時間複雜度： $O(\max(x, y))$

```
if (
  (currentStation === "Xiaobitan" &&
    (stationName === "Xindian City Hall" || stationName === "Xindian")) ||
  (currentStation === "Xindian City Hall" &&
    stationName === "Xiaobitan") ||
  (currentStation === "Xindian" && stationName === "Xiaobitan")
) {
  distance = Math.abs(currentIndex - stationIndex) + 2;
} else {
  distance = Math.abs(currentIndex - stationIndex);
}

if (distance < minDistance) {
  minDistance = distance;
  nearestFriend = messagesPeople[i];
}
}

console.log(nearestFriend);
}
```

後續賦值動作、比較距離等等，在迴圈內都是一次性動作，時間複雜度為  $O(1)$

# 總體時間

$O(n) + O(x) + O(y) + O(n * m) + O(n * \max(x,y)) \Rightarrow$  迴圈內動作要  $* n$

好的, 若將捷運線固定為松山新店線, 主線: 19 站, 支線: 18 站

1. 但是, 訊息量如果爆增到兩萬條..., 相對的, 這個不會變的 19 跟 18 的重要性, 就跟常數差不多, 可乎略不計  $\Rightarrow$  時間複雜度簡化為  $O(n) + O(n * m) + O(n) \Rightarrow$  可再簡化加法部份  $O(n * m)$
2. 那如果規定每則訊息字數固定( $m$ ), 且相對於會變動的訊息量( $n$ )小很多  $\Rightarrow$  時間複雜度可再簡化加法部份  $O(n)$

即 messages 輸入量會跟整體執行時間呈正比關係

# 空間複雜度

算法隨輸入數據量增加時所需空間資源的**增長趨勢**

currentIndex、XiaobitanIndex、minDistance、nearestFriend、stationIndex、distance等這些變數，其數量或大小跟輸入的messages數量無關，因此占用的空間可視為 $O(1)$

在findAndPrint函式內用Object.keys存取輸入訊息的人們，這會是一個陣列空間，空間大小跟輸入的messages有關，若定義messages數量為n，整體空間複雜度為 $O(n)$

```
const messagesPeople = Object.keys(messages);
```