自傳

轉學動機

- 曾經，我以為藥學系就是我想要的未來工作科系.....

- 　高三那時，我對生物相關的知識非常感興趣，而學測成績能填到最好的三類科系就是藥學系，我滿心期待我將學到的各種知識，於是開始了我的藥學系大學人生，在大一到大二的過程中，我學到草藥與一些生物知識，然而糊裡糊塗的讀到大三後，我才發現，學習的內容都與化學有極大程度的相關，其中更是有複雜且難以背誦的各式藥物化學結構、藥物的機轉以及分析儀器，這些難以碰觸實踐且枯燥的理論知識，讓我對藥學系的熱忱逐漸消散，直到達到冰點。

- 　在一次選修的html課程中，我發現了自己對編寫程序這件事產生了興趣，在解決老師提的一些加分作業時，我絞盡腦汁想出作業的解決方法，最後成功的時候，那份滿足的快感是難以描述的。我是z 世代出身的小孩，從小就常接觸電腦，也喜愛使用電腦這個工具，此外我還對解決數學問題有興趣，且對解決數學問題的過程感到愉悅，尤其是成功解出答案的一刻，更是讓人能反覆回味的美果，我這才發現從小我就熱愛解決邏輯問題，而資訊系的程式語言學習正符合解決邏輯問題的過程，與我這愛好不謀而合。正因為如此，我相信轉入資訊系能使我再次找到熱忱，在熱愛中學習進步，使自己變的更好，也更加熱愛學習。

最拿手的一科

- 我最拿手的一科是英文，從我小學4年級時，我的母親就將我送去一個私人英語教室，這個英文補習班是採一對一教學，教我英文的大廖老師提倡多講英文來回答問題，即使講得不好也要用英文回答，因為只有在強迫你要講英文的環境中學習，你才能使英文能力快速進步，所以我們就採取全英文上課。

- 一開始有很多單字我都不會的時候，時常句子講到一半時，就會問某個中文對應的英文單字該怎麼說，久而久之詞彙量上升之後，我逐漸越來越擅長用英語回答問題及對話，我印象最深刻的是，她規定每次上課開始時，都要我用1分鐘，簡短介紹一下這週發生最讓我印象深刻的事情，就是這個規定讓我的口說能力進步突飛猛進，而英文的詞彙量則是靠英文文章的閱讀在遇到不會的單字時，先標出它的音標在查單字的含義寫在旁邊，不斷的反覆練習。假日時，我們會到英語教室，由助教考這週發的單字表以及文章的默背，和文法的小考。

- 而支持我不斷學習英文的動力來自YouTube的國外頻道，當時我是先由中文翻譯的影片找到這位主播的，我馬上愛上了他的影片，但之後翻譯的人停止了翻譯，這使得我開始困擾，而在尋求解決方法的時候，我發現最好的方法就是將英文學好，如此一來我就能直接聽原文觀看影片，而不需要再尋求他人翻譯，當時我就在補習班學習以及聽原文影片的過程中，將英文能力打磨了出來，英文補習班將我的文法基礎打的紮實，而英文影片則讓我的口說能力上升，如此交叉學習讓我在高三時多益考了900多分。直到現在英文的能力仍然保持不錯，唯一缺乏的還是學無止盡的單字量。

最不拿手的一科

- 歷史是我最不拿手的一科，從國中開始我就對歷史很不擅長。歷史這個科目注重背誦，其中有許多年代與事件，在國內與外國史中，中國史尤其困難。需要背誦的內容，從夏朝一路到民國，可以說是相當大的篇章，但內容廣大不是我對歷史不拿手的唯一原因，是各個事件的時間重疊與時代背景的聯繫，才是使我混淆的重大原因。當外國史也合進來時，年表上某個事件的途中，同時國外發生了什麼事，又或是18xx-19xx年同時在中國不同地方發生了不同事件，這些內容對我來說難以在腦中整合，即使寫下年表還是難以記憶。

- 當時我們的班導就是歷史老師，還是以講故事方式上課風趣的老師，因為由他教課我國中的歷史成績還不算差，但到了高中換了一個歷史老師後，我就不擅長記憶這些歷史事件了。

- 我對歷史的興趣不深也是其中一個原因，學習歷史的目的是記取教訓，以歷史為借鑒，但是身為現代學生的我很難再遇到什麼吳三桂還是燕王朱棣這種古代的戰亂了，除了記得一些古代故事外，對我個人最多也是增長見識，但無多少益處，只能說歷史事件雖然驚人或有趣，但對現代人又過於遙遠，對這些事件不感興趣又還要默背發生年月日就又更困難了，只能說我對歷史不拿手還是建立在對歷史興趣缺缺之上。

三支程式代表作

Html 九九乘法表

index.html M　JS 99.js M　# table.css　{} launch.json

JS 99.js > ...

```javascript
var HTML = "<table border=1 width=100%>";
for (j=1;j<10;j++){
    HTML += "<tr>";
    for (i=1;i<10;i++){
        HTML += "<td align=center>" + i*j + "</td>";
    }
    HTML += "</tr>";
}
HTML += "</table>";
document.getElementById("list").innerHTML = HTML;
```

index.html M　JS 99.js M　# table.css　{} launch.json

index.html > html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>9*9乘法表</title>
    <link rel="stylesheet" type="text/css" href="table.css">
</head>
<body>
    <p>九九乘法表</p>
    <div id="list"></div>
    <script src="99.js"></script>
</body>
</html>
```

這是寫出九九乘法表的程式

在javascript中，本程式透過將要寫入html的程式碼以字串的形式儲存，做出九九乘法表的表格

1.令HTML為一變數，將表格的html程式碼字串先存入

2.用雙重for迴圈逐次寫入1~9行，並在各個行中逐次寫入1~9列

3.在每列中放入第i個行*第j個列的數字

4.使用getElementById找到html中 的id，配合innerHTML，使 內的內容替換成HTML中存的字串

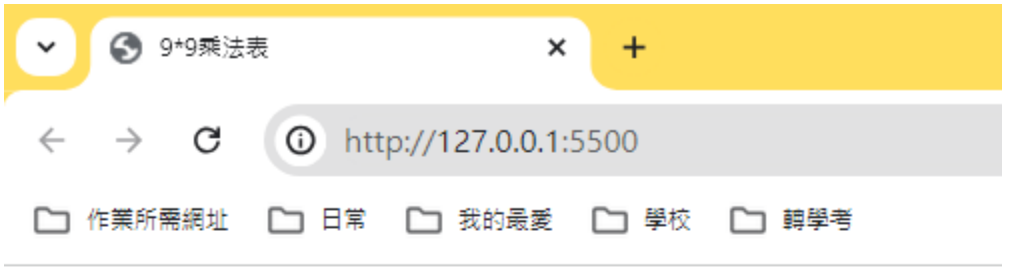
5.最後用<script src=>將同資料夾的javascript引入html程式中

6.在html接受到javascirpt的字串後，會自動將裡面的程式碼編譯成html的形式

中的替換的html編碼

```
<table border=1 width=100%><tr><td
align=center>1</td><td align=center>2</td><td
align=center>3</td><td align=center>4</td><td
align=center>5</td><td align=center>6</td><td
align=center>7</td><td align=center>8</td><td
align=center>9</td></tr><tr><td
align=center>2</td><td align=center>4</td><td
align=center>6</td><td align=center>8</td><td
align=center>10</td><td align=center>12</td><td
align=center>14</td><td align=center>16</td><td
align=center>18</td></tr>……<tr><td
align=center>9</td><td align=center>18</td><td
align=center>27</td><td align=center>36</td><td
align=center>45</td><td align=center>54</td><td
align=center>63</td><td align=center>72</td><td
align=center>81</td></tr></table>
```

這是表格的結果



九九乘法表

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

完整程式列表

index.html M　JS 99.js M　# table.css　{} launch.json

JS 99.js > ...

```
var HTML = "<table border=1 width=100%>";
for (j=1;j<10;j++){
    HTML += "<tr>";
    for (i=1;i<10;i++){
        HTML += "<td align=center>" + i*j + "</td>";
    }
    HTML += "</tr>";
}
HTML += "</table>";
document.getElementById("list").innerHTML = HTML;
```

index.html M　JS 99.js M　# table.css　{} launch.json

index.html > html

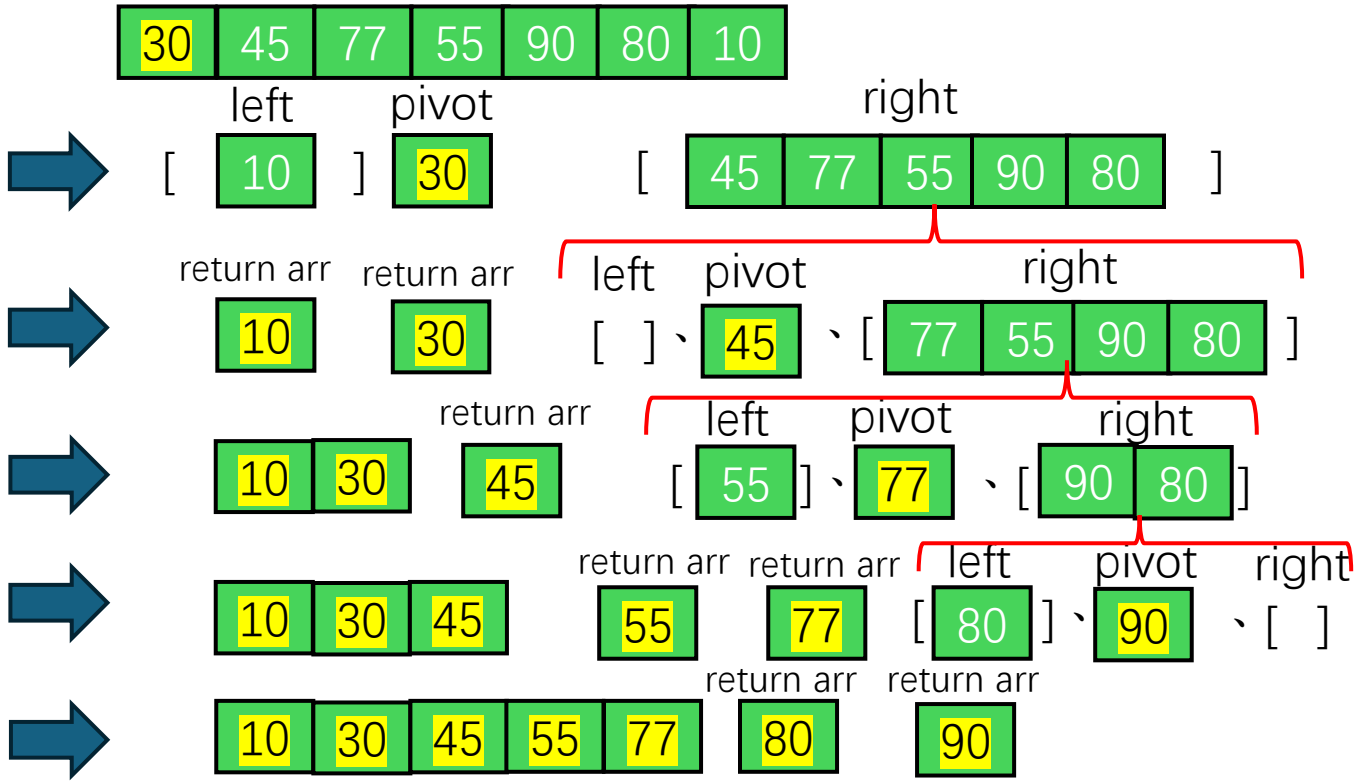
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>9*9乘法表</title>
    <link rel="stylesheet" type="text/css" href="table.css">
</head>
<body>
    <p>九九乘法表</p>
    <div id="list"></div>
    <script src="99.js"></script>
</body>
</html>
```

Javascript 快速排序法+二分搜尋法

```
var prompt = require('prompt');

prompt.start();

let data = [30,45,77,55,90,80,10];

function quicksort(arr) {
  if (arr.length <= 1) {
    return arr;
  }
  let left = [];
  let right = [];
  let pivot = arr[0];
  for (i = 1; i < arr.length; i++) {
    let num = arr[i];
    if (num < pivot){
      left.push(num);
    } else {
      right.push(num);
    }
  }
  return [...quicksort(left), pivot, ...quicksort(right)];
}
console.log(quicksort(data));

function binarysearch(arr,goal) {
  let head = 0;
  let end = arr.length - 1;
  let mid;
  while (head <= end) {
    mid = ((head + end) / 2) | 0;
    if (goal < arr[mid]) {
      end = mid - 1;
    } else if (goal > arr[mid]) {
      head = mid + 1;
    } else {
      return ("搜尋選項在第" + (mid + 1) + "項");
    }
  }
  return("無搜尋資料");
}

prompt.get(['number'],function(err,result){
    console.log(binarysearch(quicksort(data),result.number));
});
```

這是快速排序法+二分搜尋法的程式

快速排序法的程式是
1.先給予一資料陣列 data
2.在function quicksort中，本程式使用的方式是選定data第一個數字做為基準值，宣告兩個空陣列"左"和"右"
3.將接下來的各個數字小於基準值的放進左陣列，大於的則放進右陣列
4.之後用遞迴的方式使得左陣列、基準值、右陣列個別再跑一次function，前面的if有判斷陣列資料剩下小於等於1個數字時會直接回傳
5.在多次循環下數字就會由小到大排序傳回data中



```
JS quicksort.js ×
JS quicksort.js > prompt.get() callback
1  var prompt = require('prompt');
2
3  prompt.start();
4
5  let data = [30,45,77,55,90,80,10];
6
7  function quicksort(arr) {
8    if (arr.length <= 1) {
9      return arr;
10   }
11   let left = [];
12   let right = [];
13   let pivot = arr[0];
14   for (i = 1; i < arr.length; i++) {
15     let num = arr[i];
16     if (num < pivot){
17       left.push(num);
18     } else {
19       right.push(num);
20     }
21   }
22   return [...quicksort(left), pivot, ...quicksort(right)];
23 }
24 console.log(quicksort(data));
25
26 function binarysearch(arr,goal) {
27   let head = 0;
28   let end = arr.length - 1;
29   let mid;
30   while (head <= end) {
31     mid = ((head + end) / 2) | 0;
32     if (goal < arr[mid]) {
33       end = mid - 1;
34     } else if (goal > arr[mid]) {
35       head = mid + 1;
36     } else {
37       return ("搜尋選項在第" + (mid + 1) + "項");
38     }
39   }
40   return("無搜尋資料");
41 }
42
43 prompt.get(['number'],function(err,result){
44     console.log(binarysearch(quicksort(data),result.number));
45 });
```

二分搜尋法的程式是

1.輸入一個陣列與要在陣列中尋找的數字

2.數字用prompt javascript的寫法使得能輸入任意數字

3.在 function binarysearch 中，先宣告head=0、end=陣列長度-1和mid

4.本程式使用位元運算子的or方法配上0，使得(head+end)/2在存在小數點時，能捨去小數點後的數字變為整數

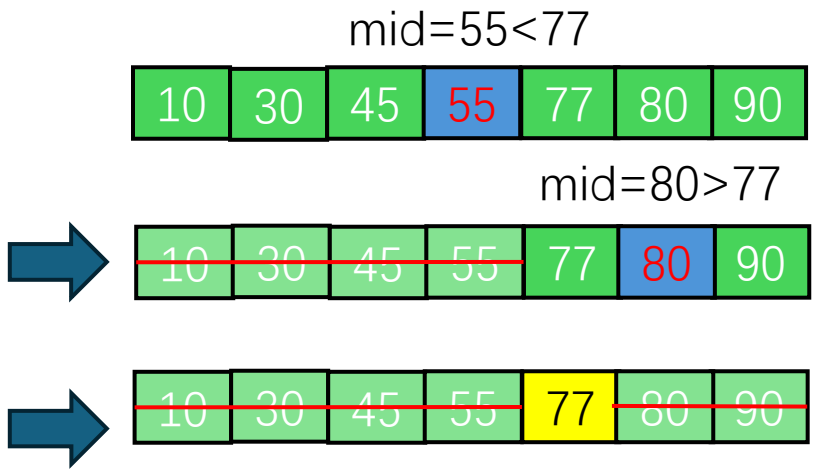
5.當mid位置的數字大於尋求的數字時，會使搜尋範圍從尾部往前縮至一半

6.反之，當mid位置的數字小於尋求的數字時，會使搜尋範圍從部頭往前縮至一半

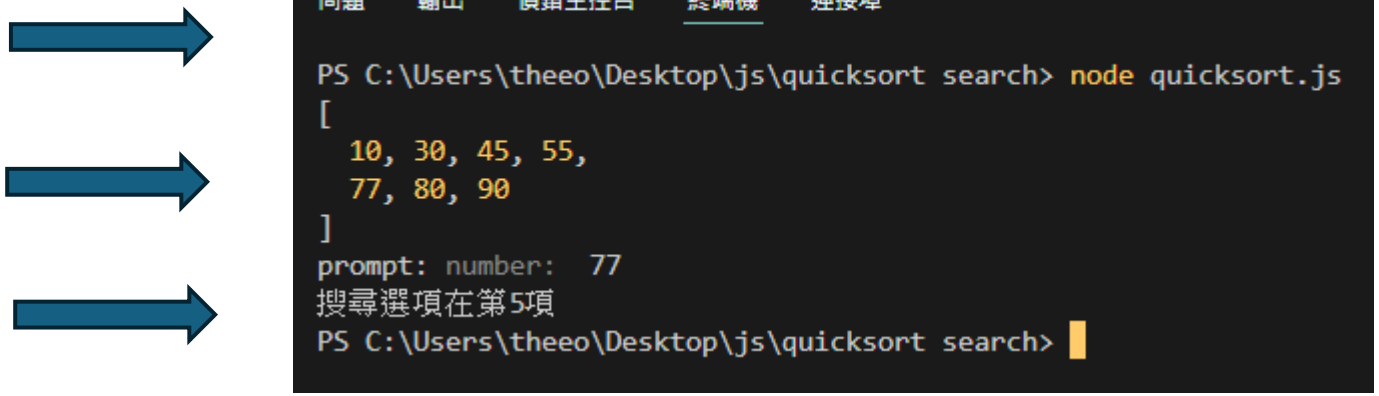
7.反覆進行直到搜尋範圍縮至1個數字時，搜尋資料就在mid+1項的位置

8.若無此資料，最後就會回"無搜尋資料"

以10,30,45,55,80,77,90 搜尋77為例



這是執行結果


問題 輸出 偵錯主控台 終端機 連接埠
PS C:\Users\theeo\Desktop\js\quicksort search> node quicksort.js
[
10, 30, 45, 55,
77, 80, 90
]
prompt: number: 77
搜尋選項在第5項
PS C:\Users\theeo\Desktop\js\quicksort search>

完整程式列表

```
var prompt = require('prompt');

prompt.start();

let data = [30,45,77,55,90,80,10];

function quicksort(arr) {
  if (arr.length <= 1) {
    return arr;
  }
  let left = [];
  let right = [];
  let pivot = arr[0];
  for (i = 1; i < arr.length; i++) {
    let num = arr[i];
    if (num < pivot){
      left.push(num);
    } else {
      right.push(num);
    }
  }
  return [...quicksort(left), pivot, ...quicksort(right)];
}
console.log(quicksort(data));

function binarysearch(arr,goal) {
  let head = 0;
  let end = arr.length - 1;
  let mid;
  while (head <= end) {
    mid = ((head + end) / 2) | 0;
```

```
32      if (goal < arr[mid]) {
33        end = mid - 1;
34      } else if (goal > arr[mid]) {
35        head = mid + 1;
36      } else {
37        return ("搜尋選項在第" + (mid + 1) + "項");
38      }
39    }
40    return("無搜尋資料");
41  }
42
43  prompt.get(['number'],function(err,result){
44      console.log(binarysearch(quicksort(data),result.number));
45  });
```

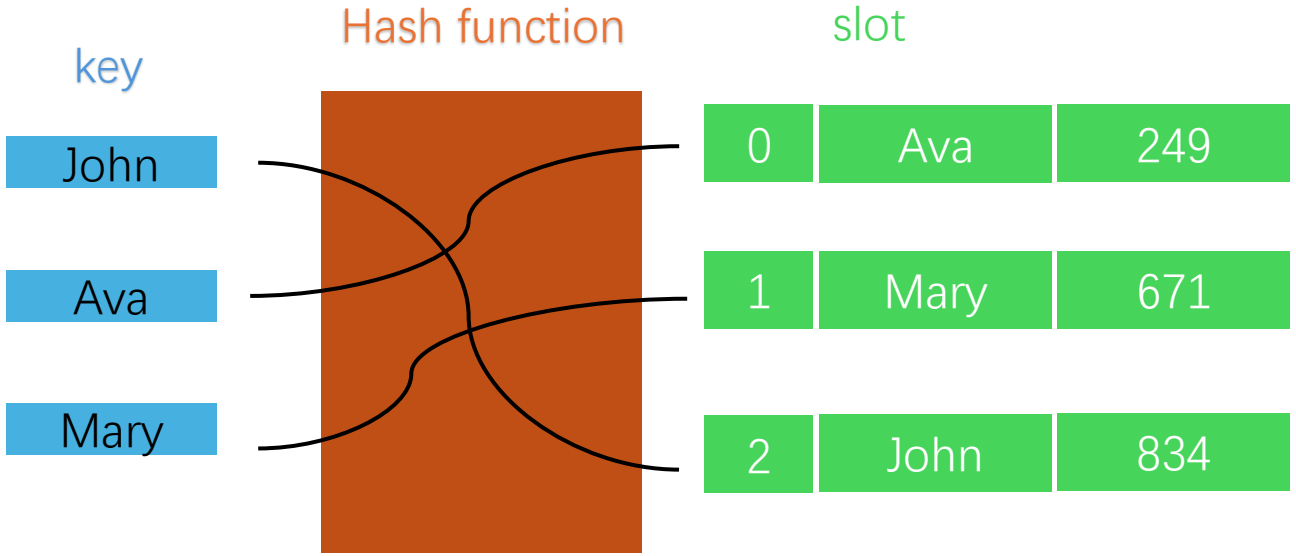
Javascript 雑湊表(HashTable)

JS hashtable.js > HashTable > search > slot

```
function hash(key, size) {
    let hashCode = 0;
    for (let index in key) {
      hashCode += key.charCodeAt(index);
    }
    return hashCode % size;
}

class Box{
    constructor(key,value){
        this.key=key;
        this.value=value;
    }
}

class HashTable {
    constructor() {
      this.size = 16;
      this.slots = new Array(this.size).fill(null);
      this.length = 0;
    }
```

這個程式是雜湊表(HashTable)

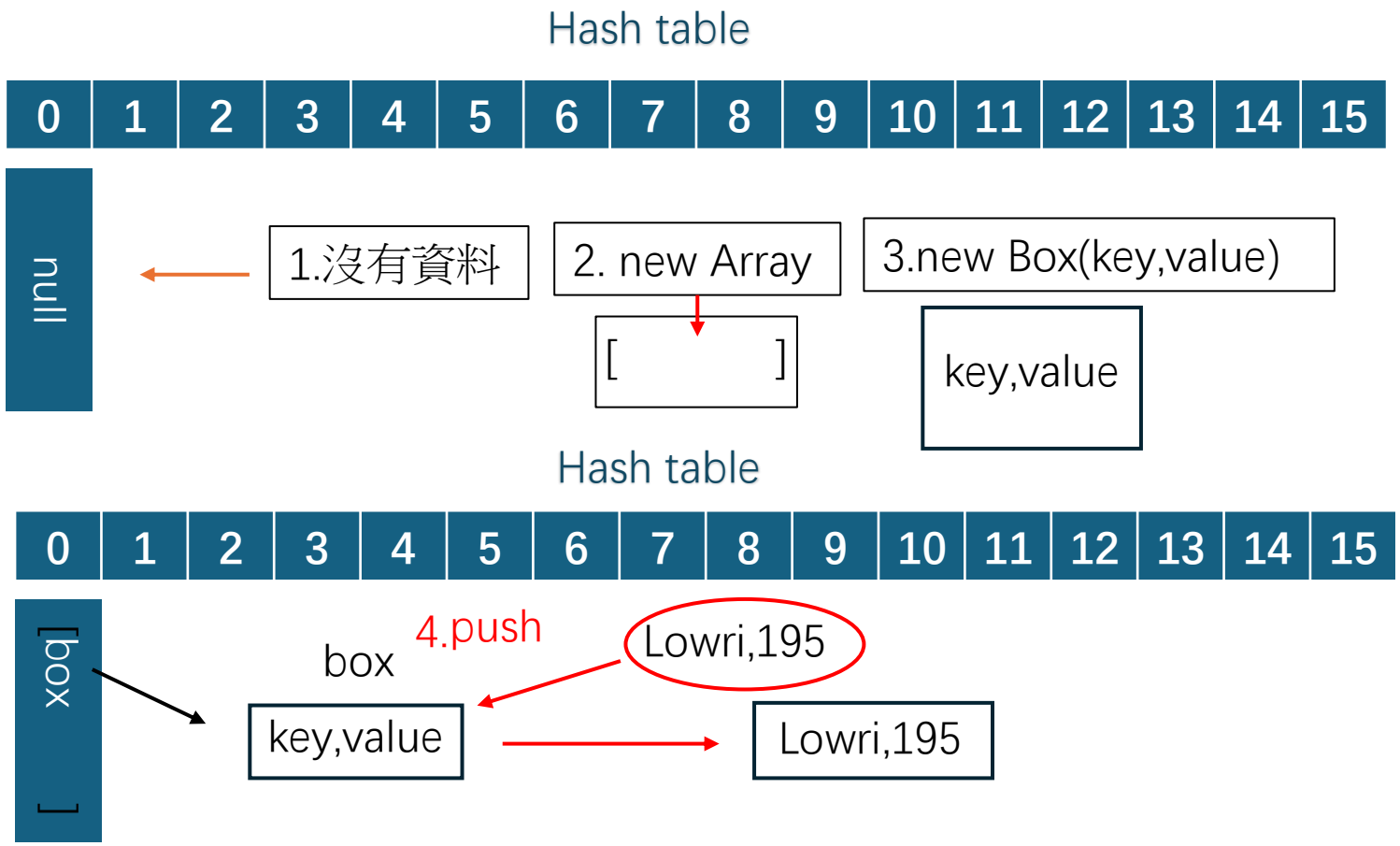
概念是透過雜湊函數計算出key所對應的一個slot，將該組key與value存入該slot之中，進而建立雜湊表格，如果不同的key找到相同對應的slot時，則在該slot內依序存進該slot中的陣列裡面。當要尋找資料時，key能快速找到其相對位置的slot，如果不同的key找到相同對應的slot時，則在該slot內的陣列中線性搜尋該相同的key，並取得其相對應的value，如此有加密與節省尋找時間的功能。



本程式使用的雜湊函數是以charCodeAt()，將key的字串先換成整數後除以slots的size取餘數來決定key與value放的slot位置，在本程式中的雜湊表size是16，所以就是餘16。
本程式用class 定義box 來存放key與value，定義HashTable 有16個slots預先填入null，並用length存入資料的數量。

```
22      add(key,value){
23          const hashIndex = hash(key, this.size);
24          const slot = this.slots[hashIndex];
25          if (!slot){
26              this.slots[hashIndex] = new Array();
27              let box = new Box(key,value);
28              this.slots[hashIndex].push(box);
29              this.length++;
30          } else {
31              let found=false;
32              for(let i=0;i<slot.length;i++){
33                  let box = slot[i];
34                  if(box.key==key){
35                      box.value=value;
36                      found=true;
37                      break;
38                  }
39              }
40              if(!found){
41                  let box = new Box(key,value);
42                  this.slots[hashIndex].push(box);
43                  this.length++;
44              }
45          }
46      }
```

add(key,value)是將key與value存入slot的方法，會先用if確認該slot內是否已經有陣列可以存放box，如果沒有就先創建一個陣列，再創建1個box放入陣列中，然後將box用push填入box陣列中，



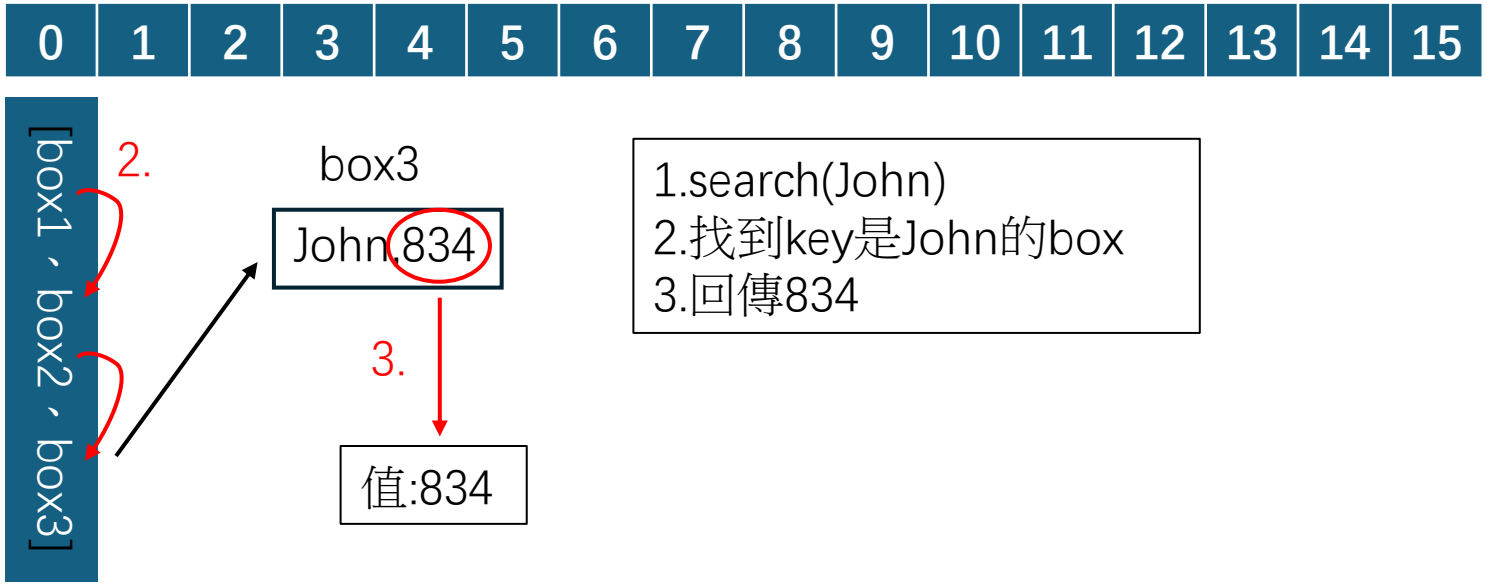
```
22      add(key,value){
23          const hashIndex = hash(key, this.size);
24          const slot = this.slots[hashIndex];
25          if (!slot){
26              this.slots[hashIndex] = new Array();
27              let box = new Box(key,value);
28              this.slots[hashIndex].push(box);
29              this.length++;
30          } else {
31              let found=false;
32              for(let i=0;i<slot.length;i++){
33                  let box = slot[i];
34                  if(box.key==key){
35                      box.value=value;
36                      found=true;
37                      break;
38                  }
39              }
40              if(!found){
41                  let box = new Box(key,value);
42                  this.slots[hashIndex].push(box);
43                  this.length++;
44              }
45          }
46      }
```

如果slot中已經有box陣列的情況下，如果有相同key不同value的資料加入時，本程式會用新資料替換掉舊資料，用一個found變數當flag，預設為false，然後用for迴圈在陣列中線性尋找是否有相同的key的box。如果有找到相同key的box，則將新的value替換進該box中，將found設為true，然後離開迴圈。這樣就可以將舊value換成新value了。而如果沒有找到相同的key，則該key為新的key，則創建1個新的box放入box陣列中，然後將資料用push填入box陣列中。

唯有替換資料的部分沒有放入**this.length++**，因為資料總數不變。

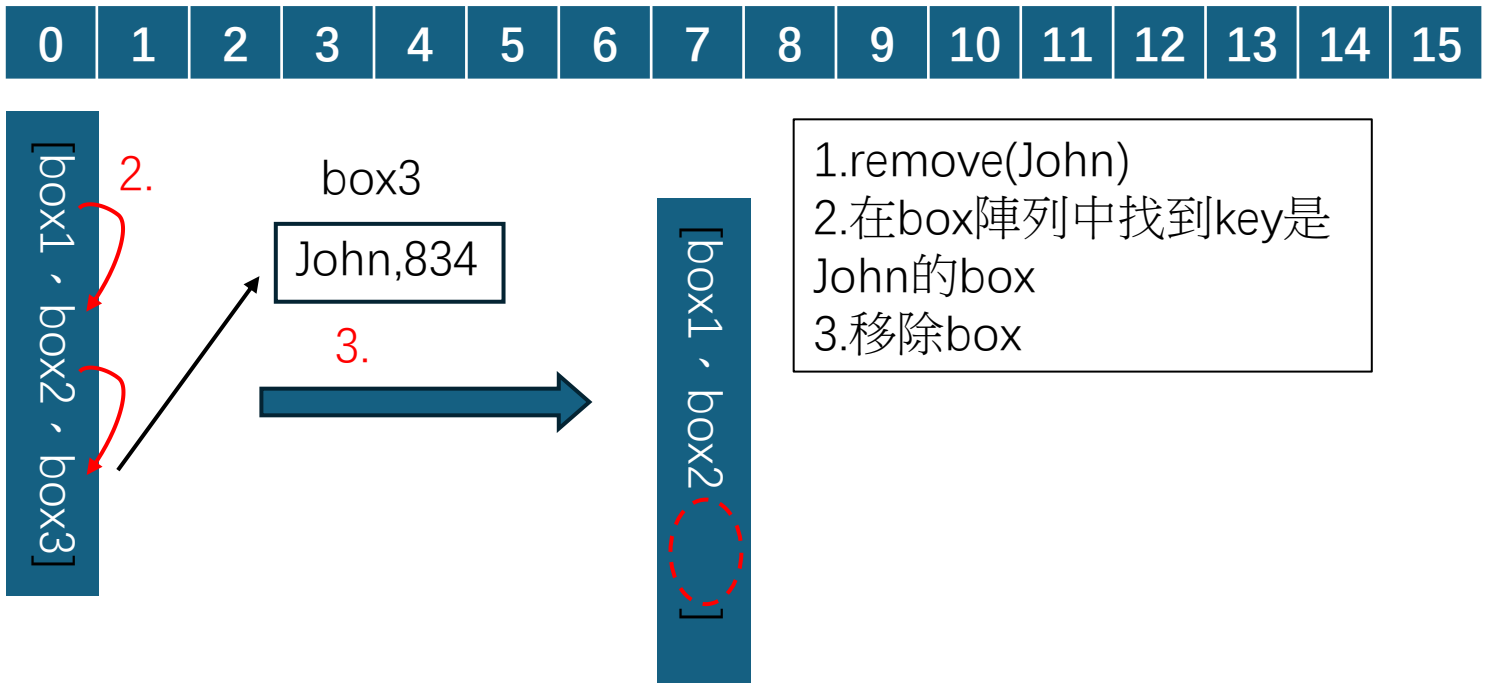
```
47      search(key){
48          const hashIndex = hash(key, this.size);
49          const slot = this.slots[hashIndex];
50          for(let i=0;i<slot.length;i++){
51              let box = slot[i];
52              if(box.key==key){
53                  return box.value;
54              }
55          }
56          return "查無此資料!";
57      }
```

search(key)是用key找到已存入資料的方法，透過hash函數找到slot，再利用for迴圈在box陣列內線性搜尋是否有相同的key的box，若有相同，則回傳該box中的value，否則回傳"查無此資料!"。



```
58      remove(key){
59          const hashIndex = hash(key, this.size);
60          const slot = this.slots[hashIndex];
61          for(let i=0;i<slot.length;i++){
62              let box = slot[i];
63              if(box.key==key){
64                  slot.splice(i,1);
65                  this.length--;
66              }
67          }
68          return "資料已清除!";
69      }
70  }
71
72  const ht = new HashTable();
73
74  ht.add("Louis", "767");
75  ht.add("Lowri", "195");
76  ht.add("Michelle", "162");
77  ht.add("Julie", "934");
78  ht.add("John", "329");
79  ht.add("Theresa", "331");
80  ht.add("John", "834");
81  ht.add("Eddie", "378");
82  ht.add("Robin", "77");
83  ht.add("Alfie", "532");
84  ht.add("Ava", "249");
85  ht.add("Owen", "208");
86  ht.add("Theodore", "539");
87  ht.add("Savannah", "54");
```

remove(key)是用來移除資料的方法，同樣透過透過hash函數找到slot，再利用for迴圈在box陣列內線性搜尋是否有相同的key的box。如果有，就使用splice將該位置的box移除。最後回報"資料已清除!"。



```
88    ht.add("Marc", "938");
89    ht.add("Sara", "420");
90    ht.add("Carmen", "599");
91    ht.add("Christine", "415");
92    ht.add("Mary", "671");
93    ht.add("Melissa", "171");
94    ht.add("Gethin", "483");
95    ht.add("Vanessa", "396");
96    ht.add("Eden", "638");
97    ht.add("Zoya", "72");
98    ht.add("Mae", "23");
99    ht.add("Edith", "609");
100   ht.add("Elizabeth", "660");
101   ht.add("Abdul", "619");
102   ht.add("Tabitha", "745");
103   ht.add("Jon", "957");
104   ht.add("Francis", "621");
105   ht.add("Millie", "767");
106
107   console.log(ht.search("John"));
108   console.log(ht.search("Ava"));
109   console.log(ht.remove("John"));
110   console.log(ht.search("John"));
```

這是實際程式表現出來的結果

完整程式列表

```
function hash(key, size) {
    let hashCode = 0;
    for (let index in key) {
      hashCode += key.charCodeAt(index);
    }
    return hashCode % size;
}

class Box{
    constructor(key,value){
        this.key=key;
        this.value=value;
    }
}

class HashTable {
    constructor() {
      this.size = 16;
      this.slots = new Array(this.size).fill(null);
      this.length = 0;
    }
    add(key,value){
        const hashIndex = hash(key, this.size);
        const slot = this.slots[hashIndex];
        if (!slot){
            this.slots[hashIndex] = new Array();
            let box = new Box(key,value);
            this.slots[hashIndex].push(box);
            this.length++;
        } else {
            let found=false;
```

```
            for(let i=0;i<slot.length;i++){
                let box = slot[i];
                if(box.key==key){
                    box.value=value;
                    found=true;
                    break;
                }
            }
            if(!found){
                let box = new Box(key,value);
                this.slots[hashIndex].push(box);
                this.length++;
            }
        }
    }
    search(key){
        const hashIndex = hash(key, this.size);
        const slot = this.slots[hashIndex];
        for(let i=0;i<slot.length;i++){
            let box = slot[i];
            if(box.key==key){
                return box.value;
            }
        }
        return "查無此資料!";
    }
    remove(key){
        const hashIndex = hash(key, this.size);
        const slot = this.slots[hashIndex];
        for(let i=0;i<slot.length;i++){
            let box = slot[i];
            if(box.key==key){
                slot.splice(i,1);
                this.length--;
            }
```

```
            }
            return "資料已清除!";
        }
    }

    const ht = new HashTable();

    ht.add("Louis", "767");
    ht.add("Lowri", "195");
    ht.add("Michelle", "162");
    ht.add("Julie", "934");
    ht.add("John", "329");
    ht.add("Theresa", "331");
    ht.add("John", "834");
    ht.add("Eddie", "378");
    ht.add("Robin", "77");
    ht.add("Alfie", "532");
    ht.add("Ava", "249");
    ht.add("Owen", "208");
    ht.add("Theodore", "539");
    ht.add("Savannah", "54");
    ht.add("Marc", "938");
    ht.add("Sara", "420");
    ht.add("Carmen", "599");
    ht.add("Christine", "415");
    ht.add("Mary", "671");
    ht.add("Melissa", "171");
    ht.add("Gethin", "483");
    ht.add("Vanessa", "396");
    ht.add("Eden", "638");
    ht.add("Zoya", "72");
    ht.add("Mae", "23");
    ht.add("Edith", "609");
    ht.add("Elizabeth", "660");
```

```
ht.add("Abdul", "619");
ht.add("Tabitha", "745");
ht.add("Jon", "957");
ht.add("Francis", "621");
ht.add("Millie", "767");

console.log(ht.search("John"));
console.log(ht.search("Ava"));
console.log(ht.remove("John"));
console.log(ht.search("John"));
```