C++資料結構與程式設計

排序與搜尋

NTU CSIE

Outline

排序

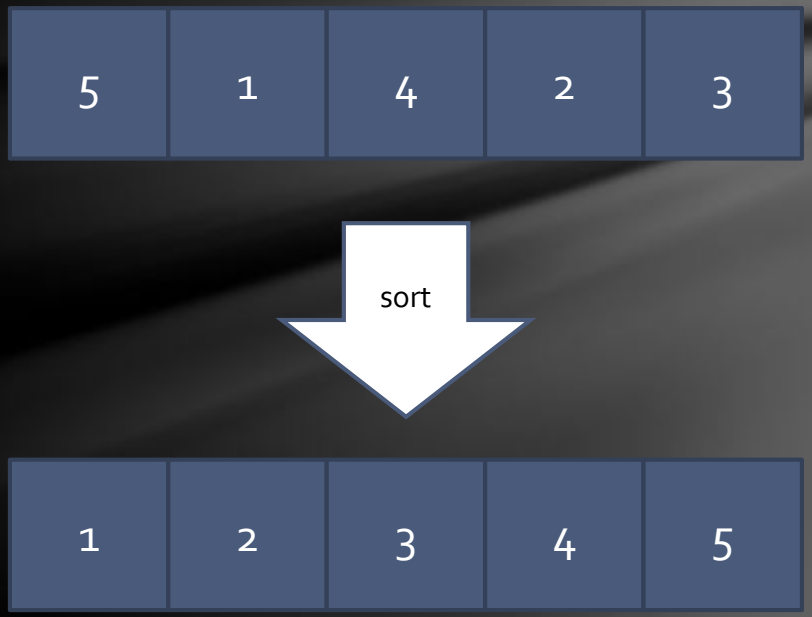
- 泡沫排序法（Bubble Sort）
- 選擇排序法（Selection Sort）
- 插入排序法（Insertion Sort）

搜尋

- 循序搜尋法（Linear Search）
- 二分搜尋法（Binary Search）

排序（Sorting)

- 簡介
  - 將一群資料按照某一規則排列，使其具有遞增（減）的關係
- 分類與比較
  - 執行效率
  - 記憶體空間
  - 穩定、不穩定
- 用途
  - 資料搜尋
  - 進階的分析與處理

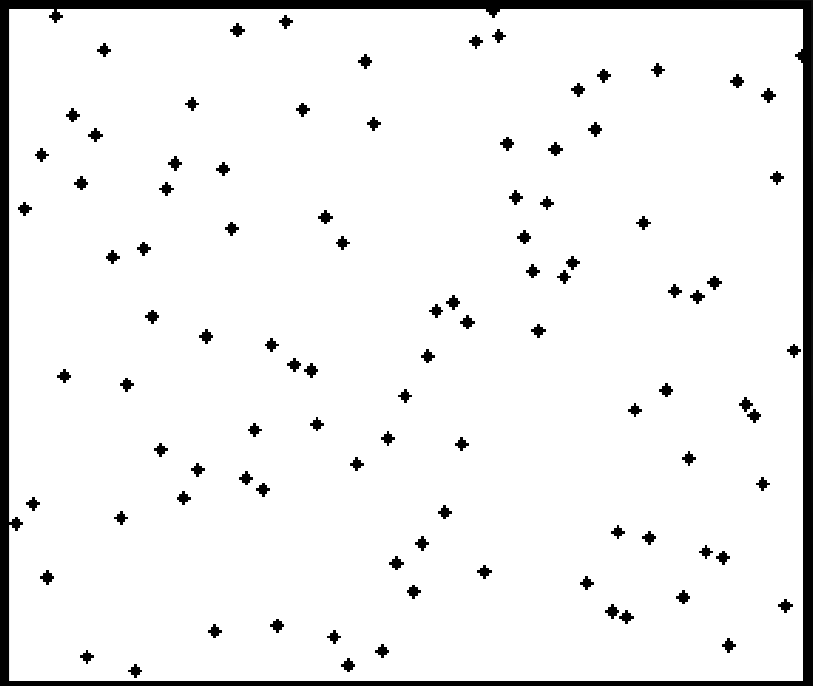


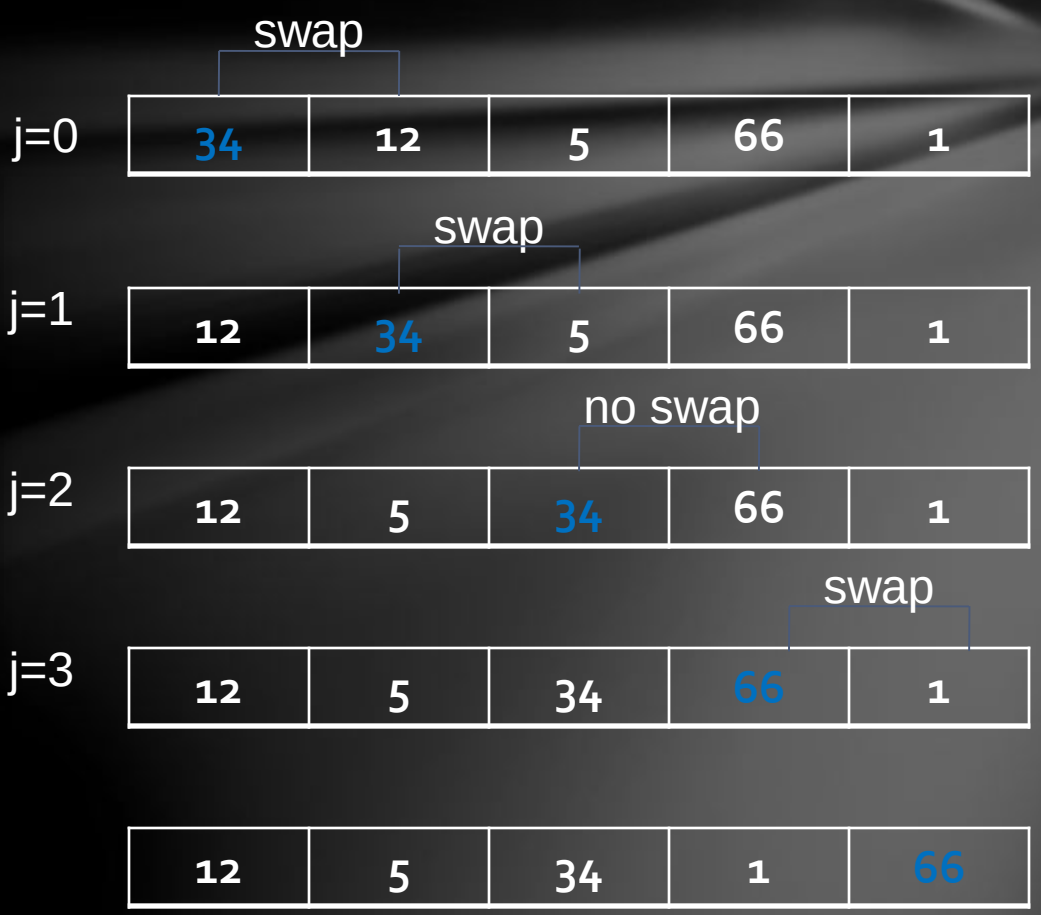
泡沫排序法（Bubble Sort）

使用一迴圈將最大值換到陣列最後

如果有n個數要判斷n-1組數值

- 5個數要判斷4次



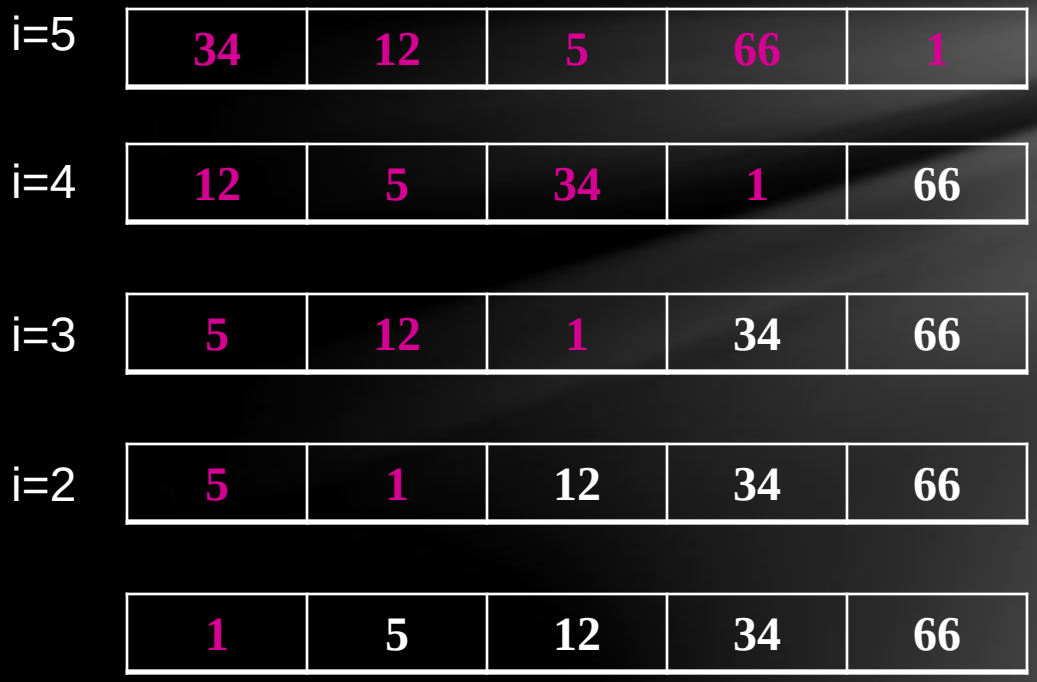


泡沫排序法（Bubble Sort）

再使用一迴圈連續對陣列把最大值放到最後

如果有n個數要做n-1次

- 5個數要做4次



泡沫排序法（Bubble Sort）

- 簡介
  - 將相鄰的資料兩兩比較大小，決定是否交換
- 範例程式碼（將資料由小排到大）

```
#include <stdio.h>
void swap(int *a, int *b)
{
   int temp;

   temp = *a;
   *a = *b;
   *b = temp;
}
void print(int n, int *p)
{
   int i;

   for(i=0; i<n; i++)
   {
      printf("%d ", p[i]);
   }
   printf("\n");
}
```

```
int main()
{
   int data[5] = {34,12,5,66,1};  // 欲排序的資料
   int i, j;
   int n=5;

   for(i=n; i>1; i--)
   {
      for(j=0; j<i-1; j++)
      {
         if(data[j+1] < data[j])
         {
            swap(&data[j+1], &data[j]);
         }
      }
   }
   print(n, data);
   return 0;
}
```

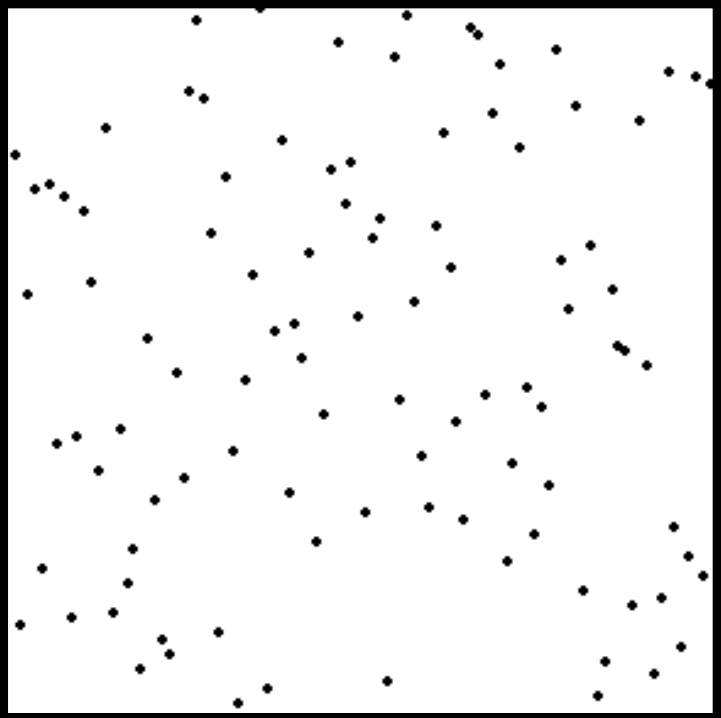
https://goo.gl/AmsHU3

選擇排序法（Selection Sort）

使用一迴圈將最小值換到陣列最前

如果有n個數要判斷n-1個數值

- 5個數要判斷4次
- 只需交換一次

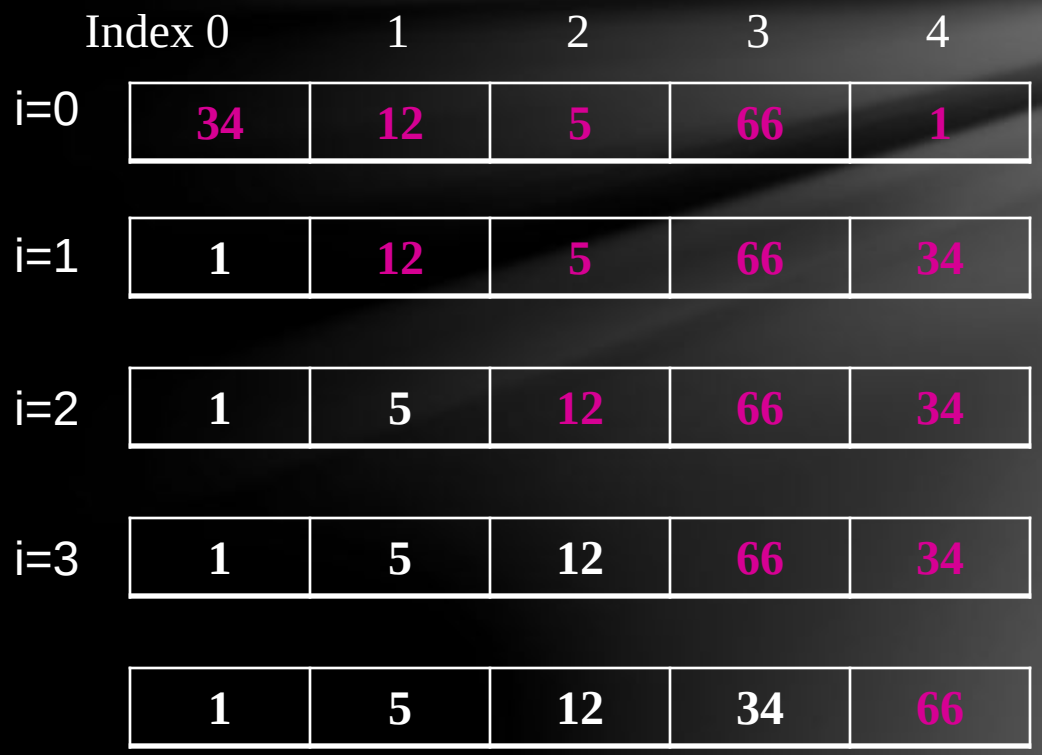


選擇排序法（Selection Sort）

再使用一迴圈連續對陣列把最小值放到第i個位置

如果有n個數要做n-1次

- 5個數要做4次



選擇排序法（Selection Sort）

- 簡介
  - 在一段資料中找出最大（小）值後，才做交換。
- 範例程式碼（將資料由小排到大）

```c
#include <stdio.h>
void swap(int *a, int *b)
{
   int temp;

   temp = *a;
   *a = *b;
   *b = temp;
}
void print(int n, int *p)
{
   int i;

   for(i=0; i<n; i++)
   {
      printf("%d ", p[i]);
   }
   printf("\n");
}
```

```c
int main()
{
   int data[5] = {34,12,5,66,1};  // 欲排序的資料
   int i, j, pos;  // pos: 紀錄目前最小值位置
   int n=5;
   for(i=0; i<n-1; i++)
   {
      pos = i;
      for(j=i+1; j<n; j++) // 找出最小值
      {
         if(data[j] < data[pos])
            pos = j;
      }
      // 把最小值跟第 i 個做交換
      swap(&data[i], &data[pos]);
   }
   print(n, data);
   return 0;
}
```

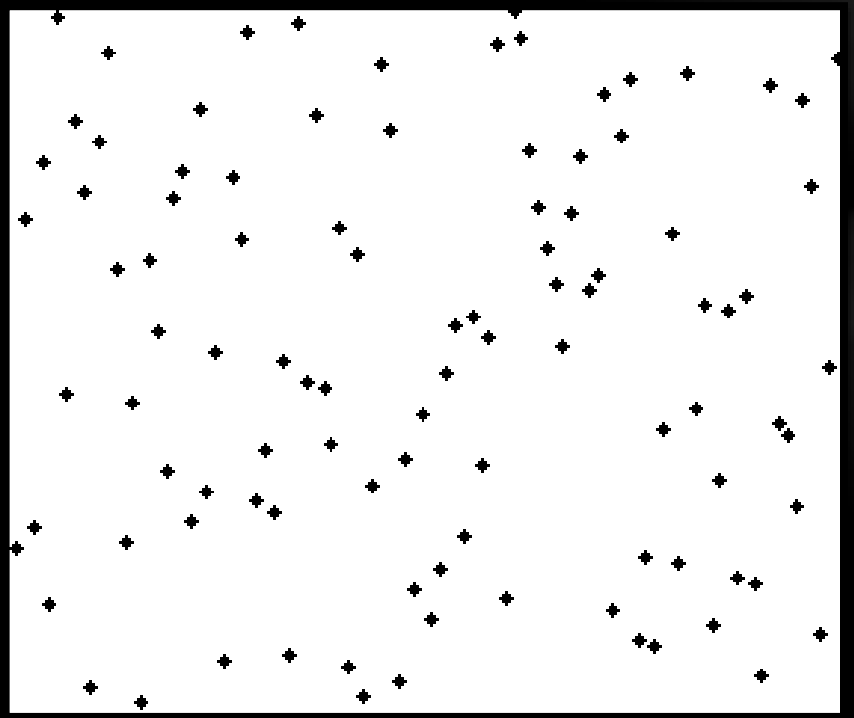
example
https://goo.gl/tbk7S5

插入排序法（Insertion Sort）

使用一迴圈將key插到適當的位置

如果有n個數要判斷n-1個數值

- 2個數要判斷1次
- 不一定需要交換



```
key=data[i];
for(j=i-1; j>=0 && data[j]>key; j--)  {
        data[j+1] = data[j];
}
data[j+1] = key;  //將key插入
```

要被插入
的值(key)

j=0

34	12	5	66	1

插入排序法（Insertion Sort）

再使用一迴圈連續對陣列的值做插入

如果有n個數要做n-1次

- 5個數要做4次

插入排序法（Insertion Sort）

- 簡介
  - 將一段資料中最右(左)邊的資料當作key，然後往左(右)塞入此資料中作排序。
- 範例程式碼（將資料由小排到大）

```
#include <stdio.h>
void print(int n, int *p)
{
   int i;

   for(i=0; i<n; i++)
   {
      printf("%d ", p[i]);
   }
   printf("\n");
}
```

```
int main()
{
   int data[5] = {34,12,5,66,1};  // 欲排序的資料
   int i, j;
   int key;  // 紀錄要被插入的值
   int n=5;
   for( i=1; i<n; i++)
   {
      key=data[i];
      for(j=i-1; j>=0 && data[j]>key; j--)
      {
         data[j+1] = data[j];
      }
      data[j+1] = key;  //將key插入
   }
   print(n, data);
   return 0;
}
```

example
https://goo.gl/7qdQhw
https://goo.gl/sx8vym

小練習(chap03_ex1_sort_num.c)

讓使用者任意輸入6個數字，將數字”由小到大”做排序並將每次排序的結果輸出。

並寫一個讓使用者輸入之介面，功能如下

- (1) 泡沫排序法
- (2) 選擇排序法
- (3) 插入排序法

Sample Input

```
33 5 22 4 88 6
1
```

Sample Output

```
5 22 4 33 6 88
5 4 22 6 33 88
4 5 6 22 33 88
4 5 6 22 33 88
4 5 6 22 33 88
4 5 6 22 33 88
```

https://jgirl.ddns.net/problem/0/2024

回家作業(chap03_ex2_sort_char.c)

寫一個讓使用者輸入之介面，功能如下

- (1) 泡沫排序法
- (2) 選擇排序法
- (3) 插入排序法

讓使用者任意輸入6個字串(字元陣列長度最大128)，將字串”**由英文字母順序大到小**”做排序並將每次排序的結果輸出。

提示: 使用strcmp, strcpy來實作 (**string.h**)

strcmp(data[0], data[1])

第一個字串大於第二個字串回傳正值，反之回傳負值。相等則為0

提示：char data[10][128]; // 欲排序的資料

傳進函式：void func(char p[][128]){printf("%s ", p[0]);}

(回家)小練習

- 試著比較這三種排序法的速度
- 以同一台電腦
- 計算從開始排序到結束所花費的時間 (計時方法)
- 重覆一千萬次後計算平均時間
- 或加總一千萬次所花費的時間
- ※可用兩陣列做復製 memcpy() 每次排序後還原成為排序

```c
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
    clock_t t1, t2;
    t1 = clock();
    Sleep(1234);
    t2 = clock();
    printf("%lf\n",
    (double)(t2-t1)/CLOCKS_PER_SEC);
    return 0;
}
```

- 比較這三種排序哪種較快？
- 陣列的內容對不同排序法的速度影響多大？

Visual C++ 的問題

```
'strcpy': This function or variable may be unsafe.
Consider using strcpy_s instead. To disable deprecation,
use _CRT_SECURE_NO_WARNINGS. See online help for details.
```

在 ProjectProperties -> Configuration Properties -> C/C++ -> Preprocessor -> Preprocessor Definitions

加入這兩行

_CRT_SECURE_NO_DEPRECATE

_CRT_NONSTDC_NO_DEPRECATE

大綱

排序

- 泡沫排序法（Bubble Sort）
- 選擇排序法（Selection Sort）
- 插入排序法（Insertion Sort）

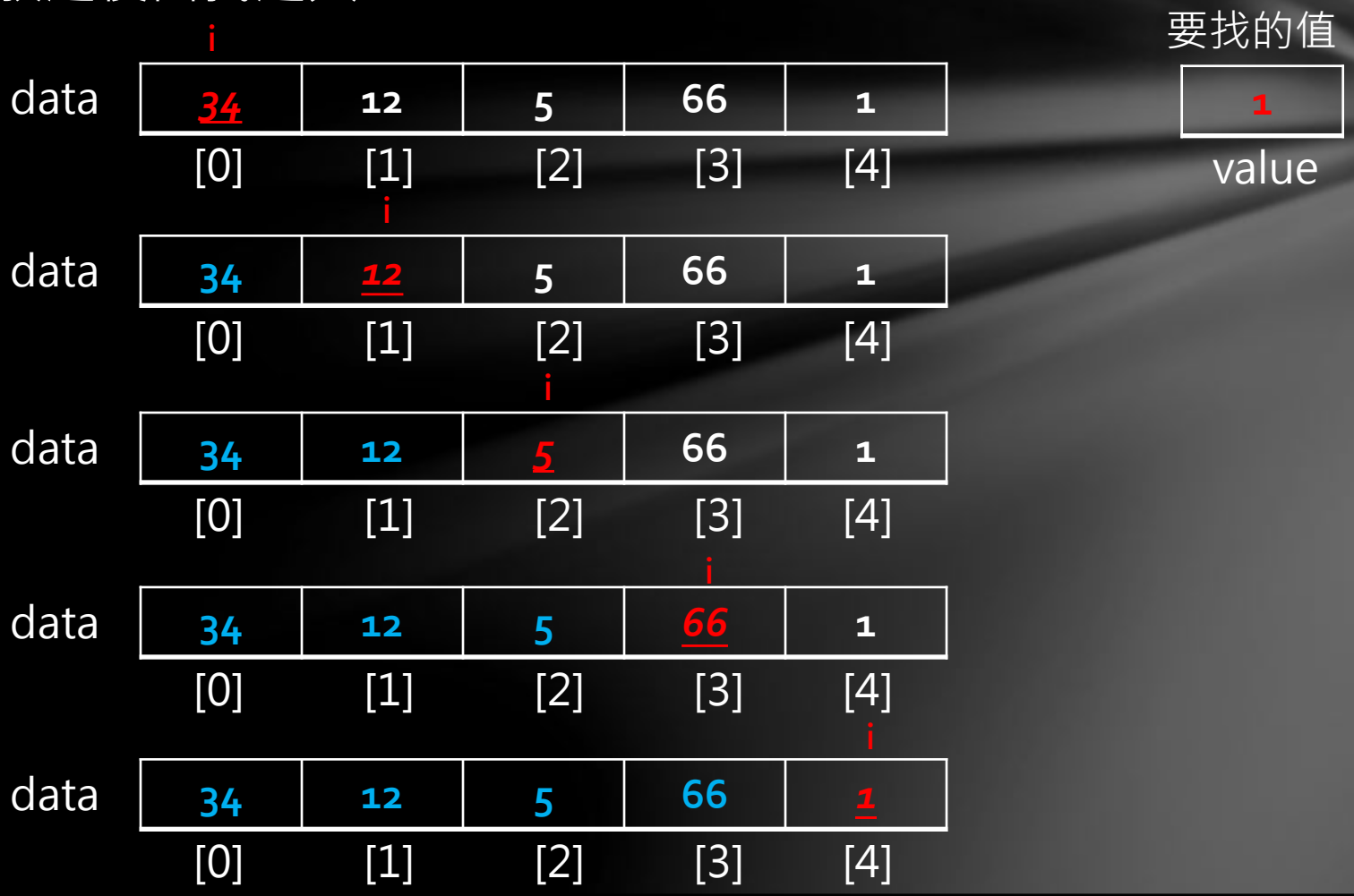
搜尋

- 循序搜尋法（Linear Search）
- 二分搜尋法（Binary Search）

循序搜尋法（Linear Search）

使用迴圈一個一個找資料存放之位置

- 資料越多找越久...
- 資料放越後面找越久...



循序搜尋法（Linear Search）

簡介

- 在一群資料中，從頭搜尋到尾直到找到資料為止。

範例程式碼

```
int LinearSearch(int n, int *p, int value)
{
   int i;
   for(i=0; i< n; i++)
   {
      if(p[i] == value)
      {
         return i;  // 找到: 傳回資料位置
      }
   }
   return -1;  // 找不到: 回傳-1
}
```

二元搜尋法（Binary Search）

使用二分法找資料

- 必須確保資料有經過排序

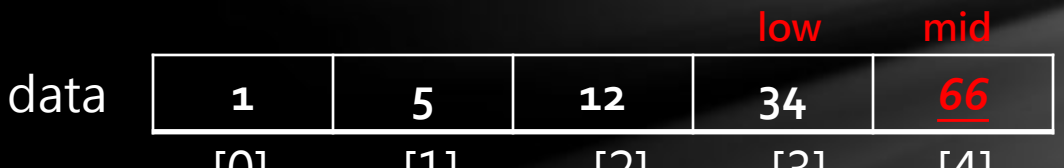
```
if(p[mid] < value)
    low = mid+1;
```

要找的值

34

value


```
mid = (low + high) / 2;
```




```
if(p[mid] > value)
    high = mid - 1;
```

二元搜尋法（Binary Search）

簡介

- 對一群排序過的資料，使用二分法的方式做搜尋
- 範例程式碼(在一群資料中找出變數key的索引值)

```
int BinarySearch(int n, int *p, int value)
{
   int low=0, high=n, mid;
   while(low <= high)
   {
      mid = (low + high) / 2;
      if(p[mid] > value)
         high = mid - 1;
      else if(p[mid] < value)
         low = mid+1;
      else
         return mid;
   }
   return -1;
}
```

Example

小練習(chap03_ex3_search_num.c)

寫一個讓使用者輸入之介面，功能如下

- (1) 插入一整數於陣列(提示: 使用插入排序法)
- (2) 尋找一整數並印出 (使用線性搜尋法)
- (3) 尋找一整數並印出 (使用二元搜尋法)
- (4) 印出目前資料(由小到大)
- (5) 離開
- ※須印出搜尋(比對)的過程如，要找55，線性搜尋從index 0開始找 其值為33到 index 2 值55才找到，如下：

```
0: 33
1: 44
2: 55
```

```
2: 55
```

https://jgirl.ddns.net/problem/0/2025

```
1 insert
2 linear search
3 bin search
4 print
5 exit
4
1 5 12 34 66
請按任意鍵繼續 .
```

```
1 insert
2 linear search
3 bin search
4 print
5 exit
2
34
0: 1
1: 5
2: 12
3: 34
found 34
請按任意鍵繼續 . . .
```

```
1 insert
2 linear search
3 bin search
4 print
5 exit
3
34
2: 12
4: 66
3: 34
found 34
請按任意鍵繼續 . . .
```

回家作業(chap03_ex4_search_char.c)

將上一程式之輸入資料改為字串(字元陣列長度最大128)

提示：char data[10][128]; // 欲排序的資料

傳進函式：

```
void func(char p[][128])
{
        printf("%s ", p[0]);
}
```

(回家)小練習

- 試著比較這二種搜尋法的速度
- 以同一台電腦
- 用固定的陣列內容
- 計算從開始搜尋到結束所花費的時間 (計時方法)
- 重覆1000次後計算平均時間
- 或加總1000次所花費的時間

```c
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
    clock_t t1, t2;
    t1 = clock();
    Sleep(1234);
    t2 = clock();
    printf("%lf\n",
    (double)(t2-t1)/CLOCKS_PER_SEC);
    return 0;
}
```

- 比較這二種搜尋法哪種較快？
- 搜尋的內容所在位置對不同搜尋法的速度影響多大？
- 搜尋內容的長度對兩種搜尋法的影響？

動動腦

- 長度1000的陣列，反覆搜尋1000次

```
LinearSearch() => 1.15
BinarySearch() => 0.01
```

- 長度10000的陣列，反覆搜尋100次

```
LinearSearch() => 11.30
BinarySearch() => 0.01
```

- 長度200000的陣列，反覆搜尋1次

```
LinearSearch() => 43.68500
BinarySearch() => 0.00100
```

- 速度差異明顯
- 但，鏈結串列能用嗎？