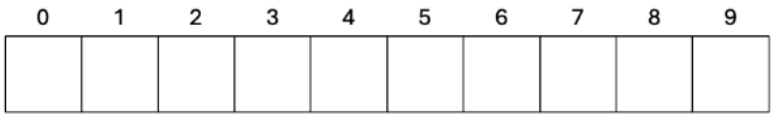
Array

一、核心概念

在連續的記憶體位置中，存儲相同資料類型的元素集合 。



1. 陣列的三大特性

- 連續記憶體（Contiguous Memory）：元素在記憶體中是一個接一個排隊的。

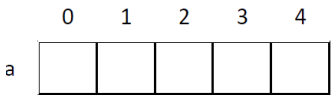
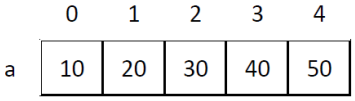
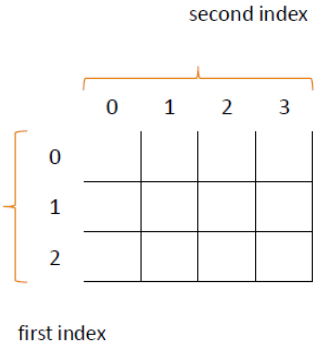
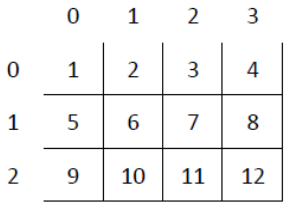
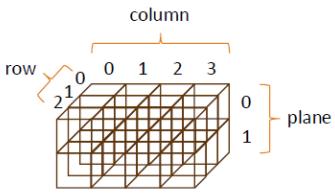
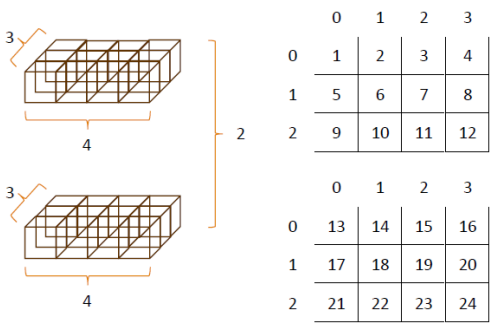
  *Array不適合頻繁的插入和刪除，因為要求記憶體連續，中間動一個就要牽動後面所有人(Frequent shifts)，導致成本極高（Costly shifts）。

- 相同型別：通常存儲相同資料型態（如全是 int 或全是 char）。

  *再C語言中字串本質上是以 \0 (Null character) 結尾的字元陣列，宣告 char s[6] = "Hello"; 時，Array Size 必須是 5 (字母) + 1 (\0) = 6 。

- 隨機存取（Random Access）：透過 Index（索引），可以在恆定時間 O(1) 內直接存取任何位置的元素。

2. 多維陣列的維度表示

維度	Declaration	Initialization	Access
1D	int array[5]; 	int array[5] = {10, 20, 30, 40, 50}; 	printf("%d", array[2]); // index[2]=30
2D	int array2d[3][4]; 	int array2d[3][4] = { {1, 2, 3, 4}, {5, 6, 7, 8}, {9,10,11,12}}; 	printf("%d", array2d[1][2]); // (row 1, col 2)=7
3D	int array3d[2][3][4]; 	int array3d[2][3][4] = { { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} }, { {13, 14, 15, 16}, {17, 18, 19, 20}, {21, 22, 23, 24} } }; 	printf("%d", array3d[1][2][3]); // first block (row 2, col 3)=24

二、 動態陣列（Dynamic Array）與記憶體管理

1. 靜態 vs. 動態比較表

特性	Static Array	Dynamic Array
大小決定時間	Compile time	Runtime
記憶體位置	usually Stack	usually Heap
記憶體用量	Minimal	Extra capacity buffer
彈性	固定長度，無法改變	可縮放大小
C++ STL	std::array	std::vector

2. C 語言核心函數

- malloc：配置指定大小的記憶體，若失敗會回傳 NULL 。

- realloc：重新調整已配置記憶體的大小。*重新配置後，起始地址可能會改變

- free：釋放記憶體，忘記呼叫會導致 Memory Leak（記憶體洩漏） 。

3.記憶體位址計算 (Address Calculation)

- 公式： Address of a[i] = Start_Address + (i * sizeof(DataType))

- (void*)&array[i] // Array memory location

三、 進階操作：排序與搜尋

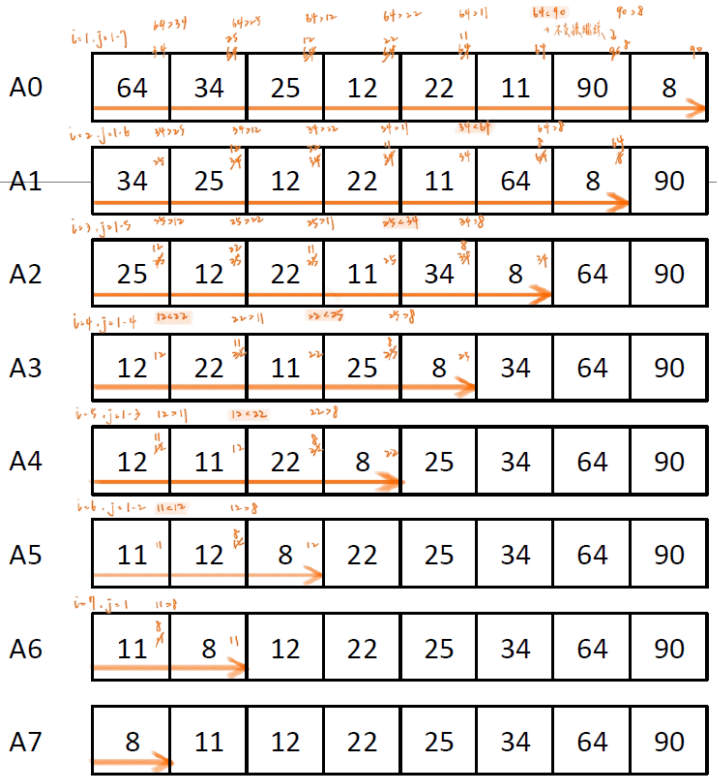
1. 排序演算法（Sorting）

目標為由小到大（Ascending）：

- **冒泡排序（Bubble Sort）：**重複比較相鄰元素，若順序錯誤則交換，較大的值會像氣泡一樣「浮」到右側。

Bubble Sort

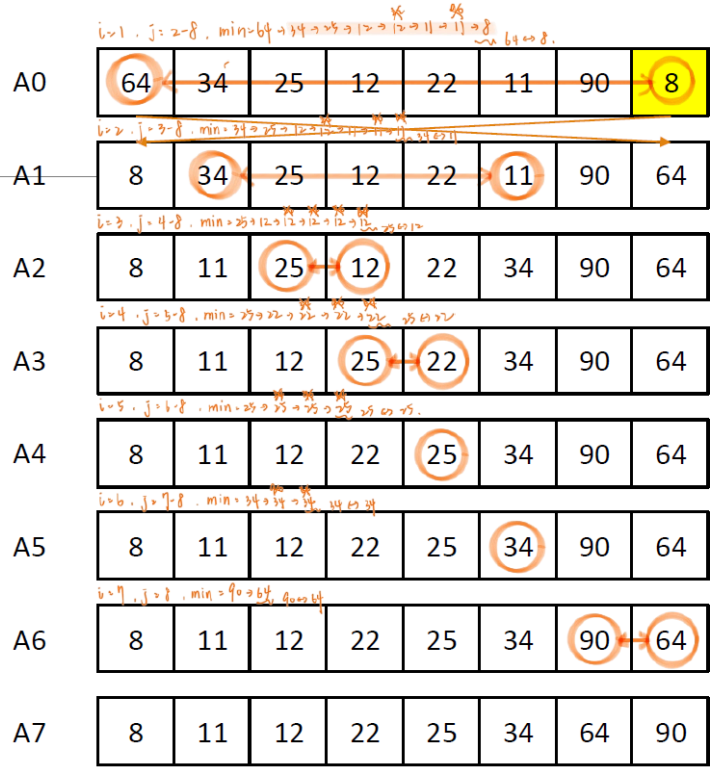
```
procedure bubbleSort(A[1..n]):
    for i from 1 to n-1:
        for j from 1 to n-i:
            if A[j] > A[j+1]:
                swap A[j] and A[j+1]
```



- **選擇排序（Selection Sort）**：每一輪從未排序的部分找出最小元素，將其交換到已排序部分的末尾。

Selection Sort

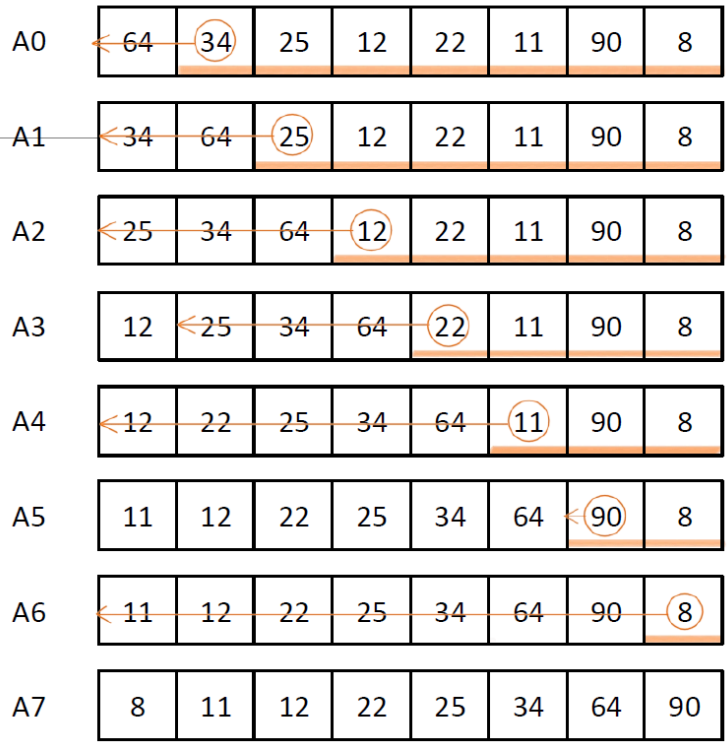
```
procedure selectionSort(A[1..n]):
    for i from 1 to n-1:
        minIndex = i
        for j from i+1 to n:
            if A[j] < A[minIndex]:
                minIndex = j
        swap A[i] and A[minIndex]
```



- **插入排序（Insertion Sort）**：將陣列分為「已排序」和「未排序」，每次取一個未排序元素插入到已排序序列的正確位置。

Insertion Sort

```
procedure insertionSort(A[1..n]):
    for i from 2 to n:
        key = A[i]
        j = i - 1
        while j > 0 and A[j] > key:
            A[j+1] = A[j]
            j = j - 1
        A[j+1] = key
```



2. 搜尋演算法（Searching）

- **線性搜尋（Linear Search）**：適用於未排序陣列，需遍歷整個陣列，時間複雜度 O(n)。
- **二分搜尋（Binary Search）**：僅適用於已排序陣列，透過每次砍半範圍提高效率，時間複雜度 O(log n)。

四、時間複雜度

操作	時間複雜度	原因
Access (讀取)	O(1)	直接用索引計算偏移量
Update (更新)	O(1)	直接覆蓋該位置
Insert (插入)	O(n)	需要搬移後方所有元素以騰出空間
Delete (刪除)	O(n)	刪除後需要將後方元素往前補位
Search (線性)	O(n)	最壞情況要看過所有元素
Search (二分)	O(log n)	每次排除一半的可能性