程式設計
Ch14. Basic Sorting Algorithms

Chuan-Chi Lai 賴傳淇

Department of Communications Engineering
National Chung Cheng University

Spring Semester, 2024

Outline

簡介
Introduction

簡介 (Introduction)

- 我們在處理資料時，常常需要將資料進行排序，以方便查找、比較和進行其他操作。排序 (Sorting) 是一種將資料按照特定順序排列的演算法，被廣泛應用於各種領域，例如搜索、數據庫和統計分析等。
- 排序算法通常按照時間複雜度、空間複雜度和穩定性等特性進行分類。因為不同的排序算法在不同的應用場景中有其優缺點，所以需要根據實際情況進行選擇。
- 本章節將介紹常見的幾種基礎排序算法，為了方便比較，後續的算法介紹都以由小到大排序的方式呈現。

穩定性 (Stability)

- 鏈結串列的主要缺點如下:
  - 一個數列在經過排序後，假如相同數值的元素在排序後的相對位置是不變的，那麼我們會說這個排序算法是穩定的 (stable)。
  - 反之，稱為不穩定的 (unstable)。

氣泡排序
Bubble Sort

氣泡排序

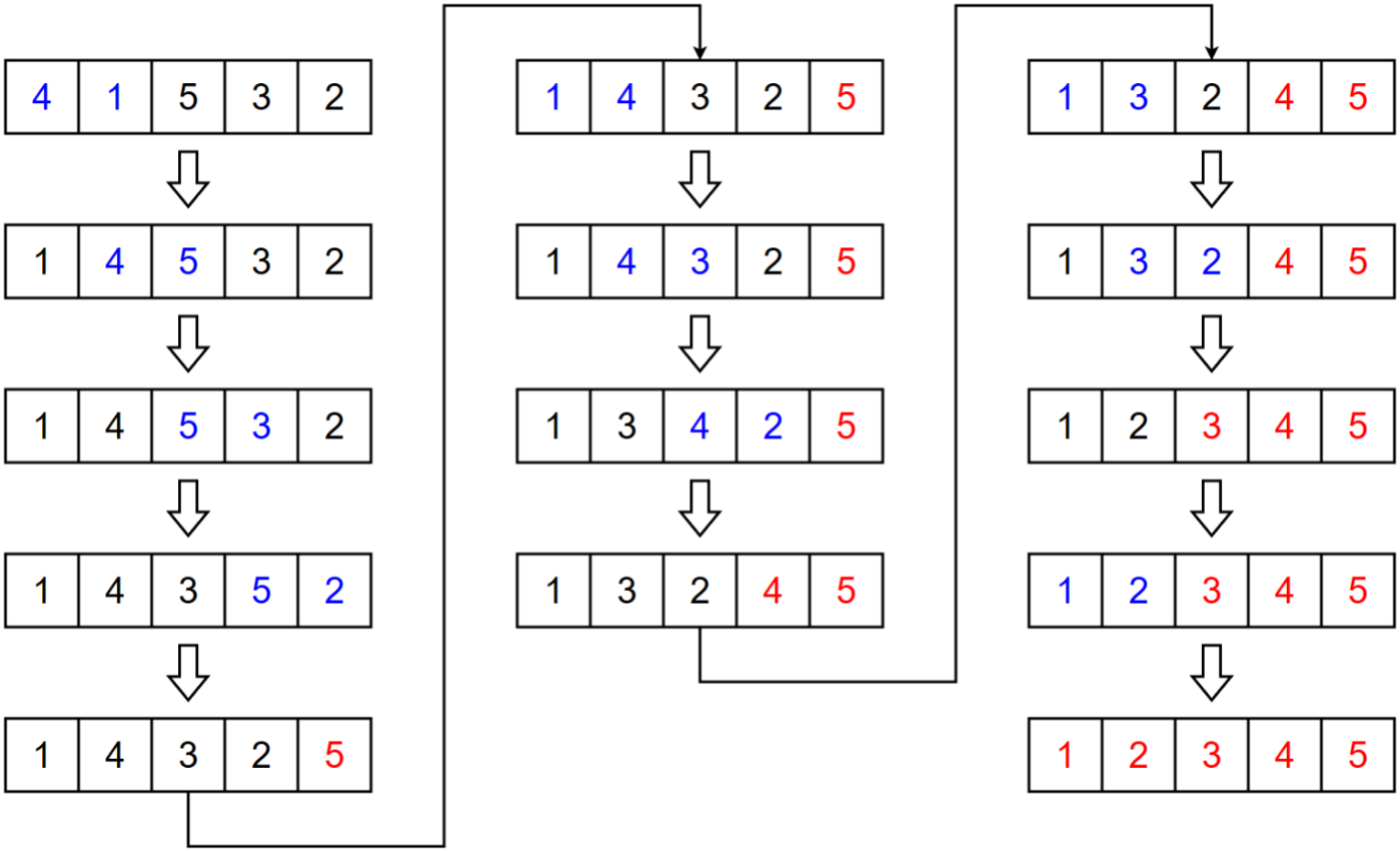
- 氣泡排序是一種較為直觀的排序演算法，它會反覆地遍歷待排序數列，比較每一對相鄰的元素，並且將順序錯誤的元素交換位置。
- 透過不斷地交換相鄰的兩個元素，較大的元素會漸漸地“浮”到數列的右端，較小的元素則逐漸“沉”到數列的左端。
- 這個過程就好像氣泡從水底冒出來一樣，因此被稱為氣泡排序。

氣泡排序 (Bubble Sort)

- 氣泡排序的步驟如下：
  1. 從陣列的第一個元素開始，每次我們比較相鄰的兩個元素，如果前一個元素比後一個元素大，就交換它們的位置。
  2. 繼續向後比較，對每一對相鄰的元素都執行上述比較與交換操作，直到比較到陣列的最後一對元素。
  3. 重複上述步驟 1 和步驟 2，對整個陣列進行多次遍歷，每次遍歷都能確保最後一個元素已經排好序（因為每次交換都會讓最大的元素往上浮，因此浮到最後的元素一定是當前比較的最大值）。

氣泡排序 (Bubble Sort)

- 示意圖如下，藍色代表比較的兩個元素，紅色代表確認排好的元素。



氣泡排序 (Bubble Sort)

- 範例程式碼如下 (由小到大)：

```c
#include<stdio.h>
void bubble_sort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
int main(){
    int arr[] = {4, 1, 5, 3, 2};
    int arr_size = sizeof(arr) / sizeof(arr[0]);
    bubble_sort(arr, arr_size);
    for(int i=0;i<arr_size;i++){
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

演算法分析 (氣泡排序)

- 平均時間複雜度：$O(n^2)$
- 最差時間複雜度：$O(n^2)$
- 額外空間複雜度：$O(1)$
- 是否有穩定性：是

選擇排序
Selection Sort

選擇排序

- 通過不斷地從未排序的元素中找出最小的元素，將其放到已排序的元素的結尾，來實現排序的目的。
- 由於每次找出的最小元素都會直接被放到適當的位置，因此選擇排序除了 swap 的暫存不需要額外的空間，是一種原地演算法 (in-place algorithm)。

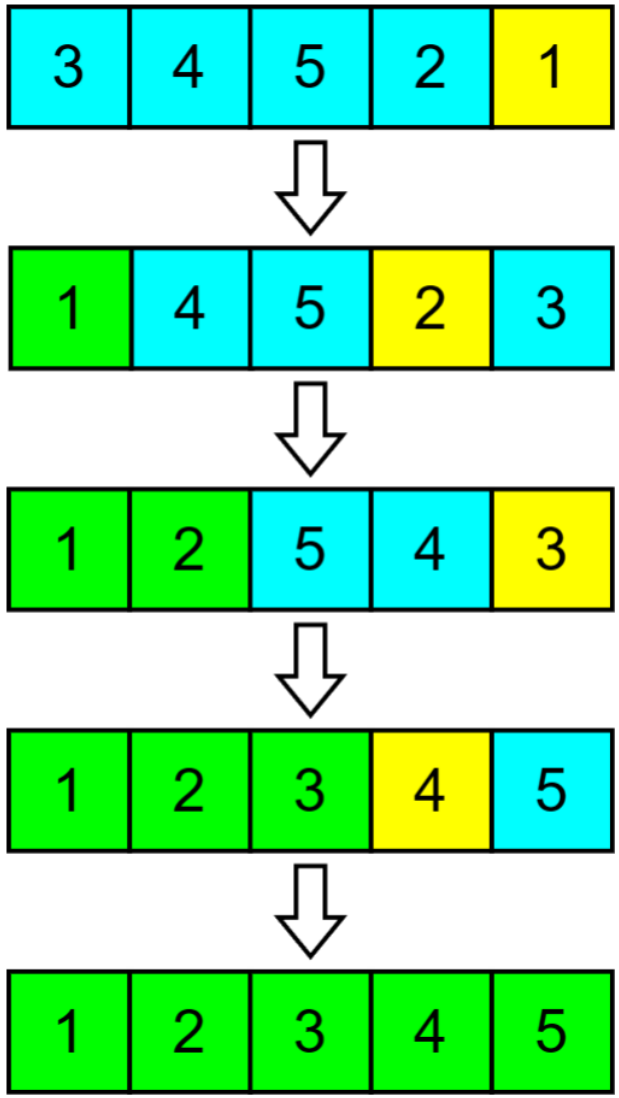
選擇排序 (Selection Sort)

- 選擇排序的步驟如下：
  1. 遍歷整個未排序陣列，從中找出最小的元素。
  2. 把找到的最小元素放到已排序陣列的結尾。
  3. 重複上述步驟，對未排序的陣列繼續遍歷，每次找出最小的元素，並將其放到已排序的序列結尾。

選擇排序 (Selection Sort)

右邊是示意圖：

- 藍色代表未被排序的部分。
- 綠色代表已被排序的部分。
- 黃色未排序部分的最小值。



選擇排序 (Selection Sort)

- 範例程式碼如下 (由小到大)：

```c
#include<stdio.h>
void selection_sort(int arr[], int n) {
    int min_idx;
    for (int i = 0; i < n - 1; i++) {
        min_idx = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[min_idx]) {
                min_idx = j;
            }
        }
        int temp = arr[i];
        arr[i] = arr[min_idx];
        arr[min_idx] = temp;
    }
}
int main() {
    int arr[] = {4, 1, 5, 3, 2};
    int arr_size = sizeof(arr) / sizeof(arr[0]);
    selection_sort(arr, arr_size);
    for (int i = 0;i < arr_size;i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

演算法分析 (選擇排序)

- 平均時間複雜度：$O(n^2)$
- 最差時間複雜度：$O(n^2)$
- 額外空間複雜度：$O(1)$
- 是否有穩定性：是

插入排序
Insertion Sort

插入排序

- 插入排序的基本思想是將一個序列分為已排序部分和未排序部分，然後從未排序的部分中選擇一個元素，插入到已排序部分的合適位置，直到未排序部分為空。

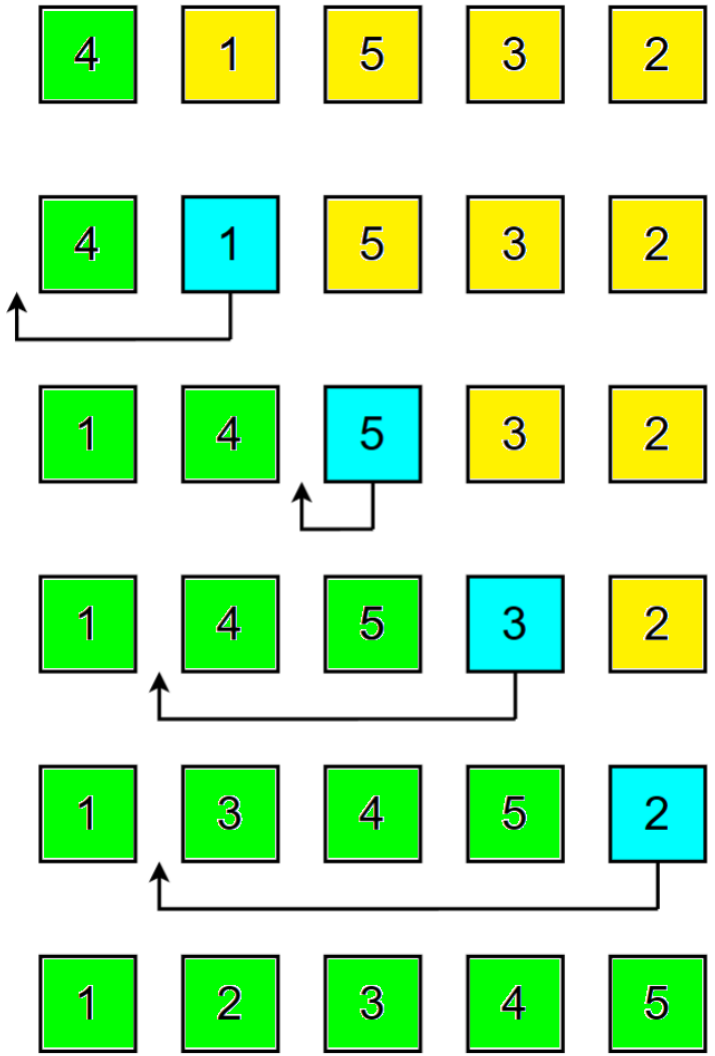
插入排序 (Insertion Sort)

- 插入排序的步驟如下：
  1. 將待排序數列分成已排序區域和未排序區域。一開始已排序區域只包含第一個元素，未排序區域包含剩餘的元素。
  2. 從未排序區域取出第一個元素，記為 key。
  3. 從已排序區域的最右邊開始比較，將所有大於 key 的元素右移一個位置，直到找到第一個小於或等於 key 的元素，記為插入點。
  4. 在插入點的右邊插入 key。
  5. 重複步驟 2 到 4，直到未排序區域為空。

插入排序 (Insertion Sort)

右邊是示意圖：

- 藍色代表正在插入的元素。
- 綠色代表已被排序的部分。
- 黃色未排序部分。



插入排序 (Insertion Sort)

- 範例程式碼如右 (由小到大)：

```c
#include<stdio.h>
void insertion_sort(int arr[], int n) {
    int key, j;
    for (int i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
int main() {
    int arr[] = { 4, 1, 5, 3, 2 };
    int arr_size = sizeof(arr) / sizeof(arr[0]);
    insertion_sort(arr, arr_size);
    for (int i = 0;i < arr_size;i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

演算法分析 (插入排序)

- 平均時間複雜度：$O(n^2)$
- 最差時間複雜度：$O(n^2)$
- 額外空間複雜度：$O(1)$
- 是否有穩定性：是

合併排序
Merge Sort

合併排序

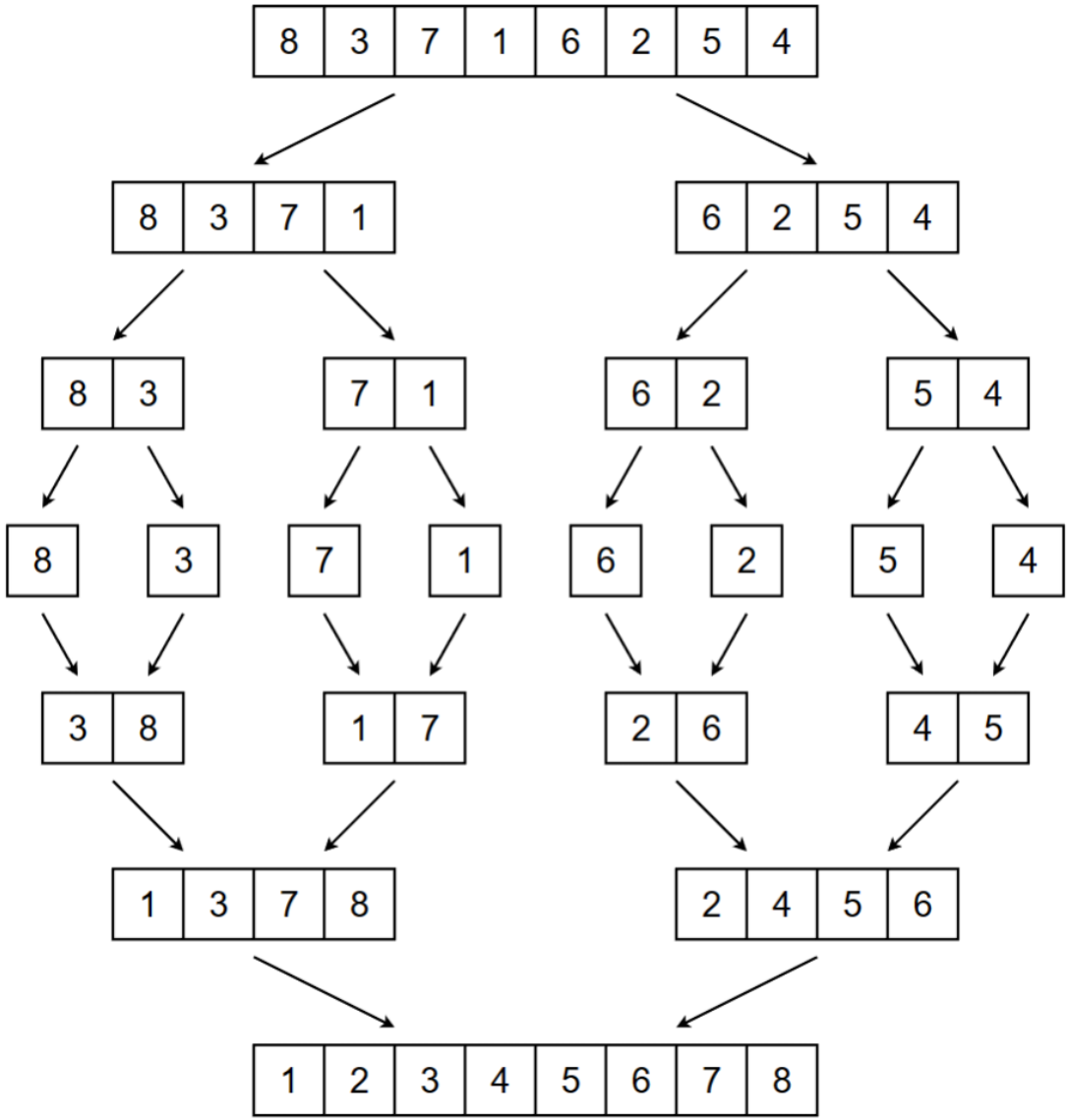
- 合併排序是一種基於分治（Divide and Conquer）的算法。它的基本思想是將一個大問題分解成小問題，然後將小問題的結果合併起來得到整個問題的解。
- 在排序時會將數列不斷切割成兩半，直到每個子數列只剩下一個元素，然後再將相鄰的子數列進行合併，直到最後整個數列都排序完成。

合併排序 (Merge Sort)

- 合併排序的步驟如下：
  1. 如果要排序的數列長度為 1 或 0，那麼它已經是有序的，直接返回該數列。
  2. 將數列拆分為兩個子數列。
  3. 對左半部分遞迴地進行排序，直到左半部分長度為 1 或 0。
  4. 對右半部分遞迴地進行排序，直到右半部分長度為 1 或 0。
  5. 將已排序的左半部分和右半部分進行合併。首先建立一個臨時數列，然後比較左右兩部分的第一個元素，將較小的元素加入到臨時數列中，然後繼續比較兩部分的下一個元素，直到其中一個部分中的元素已全部加入到臨時數列中。
  6. 將臨時數列中的元素複製回原始數列中。
  7. 回傳排序好的數列。

合併排序 (Merge Sort)

右邊是示意圖：



合併排序 (Merge Sort)

- 合併排序範例程式碼 (1/2)：

```
#include<stdio.h>
#define MAX_N 50000
void mergeSort(int* arr, int len) {
    if (len <= 1) return;
    int leftLen = len / 2, rightLen = len - leftLen;
    int *leftArr = arr, *rightArr = arr + leftLen;
    // Recursively sort the left and right halves
    mergeSort(leftArr, leftLen);
    mergeSort(rightArr, rightLen);
    static int tmpArr[MAX_N]; // temporary array to store the merged result
    int tmpLen = 0, l = 0, r = 0;
    while (l < leftLen && r < rightLen) {
        if (leftArr[l] < rightArr[r]) tmpArr[tmpLen++] = leftArr[l++];
        else tmpArr[tmpLen++] = rightArr[r++];
    }
    // Add any remaining elements from the left or right half
    while (l < leftLen) tmpArr[tmpLen++] = leftArr[l++];
    while (r < rightLen) tmpArr[tmpLen++] = rightArr[r++];
    // Copy the merged result back into the original array
    for (int i = 0; i < tmpLen; i++) arr[i] = tmpArr[i];
}
```

合併排序 (Merge Sort)

- 合併排序範例程式碼 (2/2)：

```
int main() {
    int arr[] = { 8, 3, 7, 1, 6, 2, 5, 4 };
    int arr_size = sizeof(arr) / sizeof(arr[0]);
    mergeSort(arr, arr_size);
    for (int i = 0;i < arr_size;i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

演算法分析 (合併排序)

- 平均時間複雜度：$O(n \log n)$
- 最差時間複雜度：$O(n \log n)$
- 額外空間複雜度：$O(n)$
- 是否有穩定性：是

快速排序
Quick Sort

快速排序

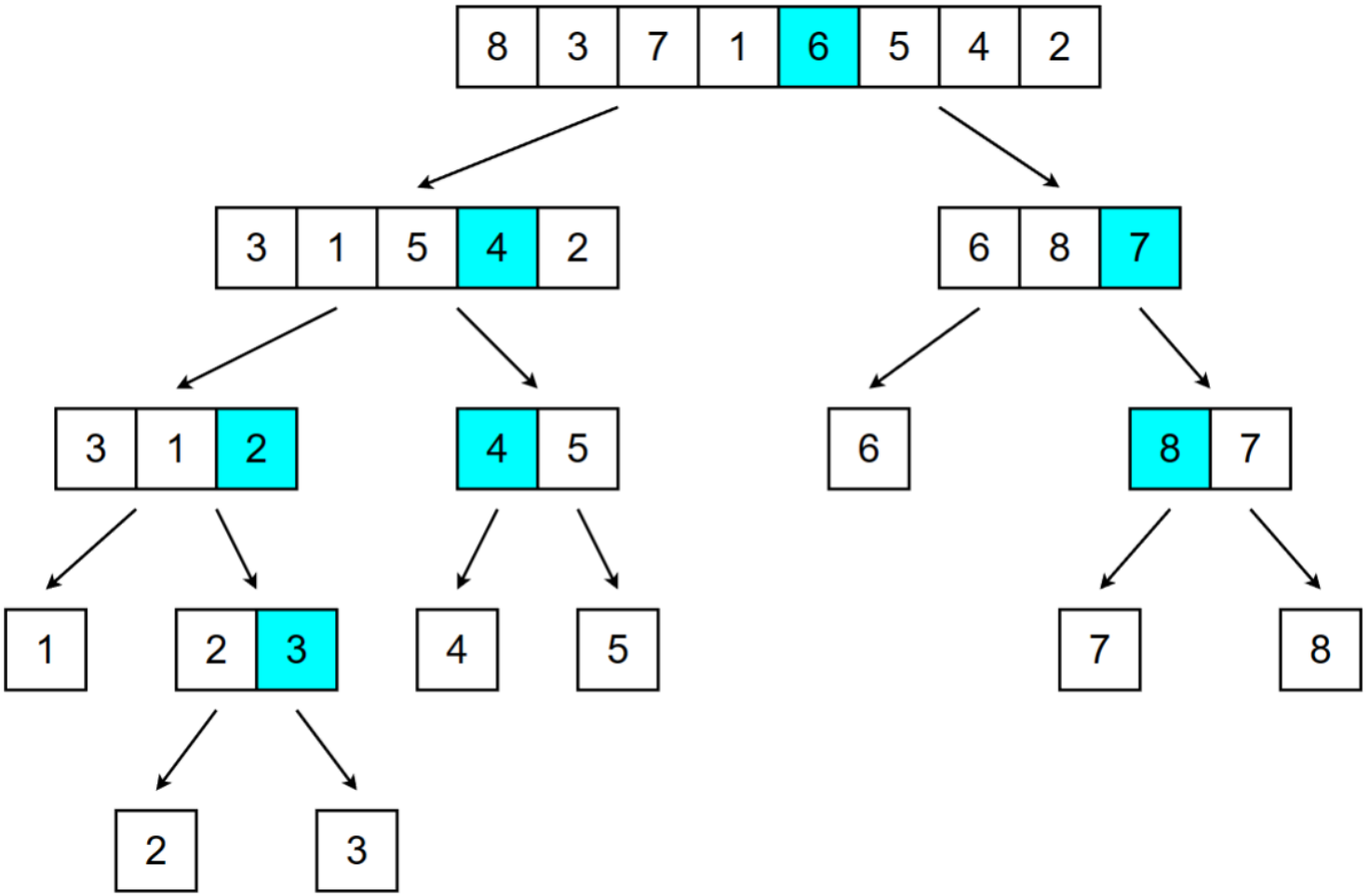
- 快速排序是一種高效的排序算法，其時間複雜度為 $O(n \log n)$。
- 快速排序的基本思想是通過分治法來進行排序。
- 具體來說，它通過選擇一個基準點（pivot），將數列分成左右兩部分，使得左邊部分的所有元素都小於基準點，右邊部分的所有元素都大於等於基準點，然後再分別對左右兩部分進行排序，最終得到一個有序數列。

快速排序 (Quick Sort)

- 快速排序的步驟如下：
  1. 先隨機選擇陣列中的一個數字當作基準點，稱為 pivot。
  2. 把陣列中小於 pivot 的數字移到 pivot 左邊，大於等於 pivot 的數字移到 pivot 右邊。
  3. 再分別對 pivot 左邊的數字區間和右邊的數字區間，重複步驟 1 和 2，直到區間中只剩下一個或沒有數字。
  4. 最終完成排序的陣列會是由 pivot 左邊的數字區間、pivot 和 pivot 右邊的數字區間所組成，而這兩個區間又會重複執行步驟 1 到 4，直到所有的區間都只剩下一個或沒有數字。

快速排序 (Quick Sort)

以下是示意圖，藍色代表被選中的基準點 (pivot)。



快速排序 (Quick Sort)

- 快速排序範例程式碼 (1/2)：

```c
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
void swap(int *x, int *y){
    int temp = *x;
    *x = *y;
    *y = temp;
}
void quickSort(int *arr, int len){
    if(len <= 1) return;
    int pivotIdx = rand()%len, pivot = arr[pivotIdx];
    swap(&arr[len-1], &arr[pivotIdx]);
    int leftLen = 0;
    for(int i=0;i<len;i++) {
        if(arr[i] < pivot) swap(&arr[i], &arr[leftLen++]);
    }
    swap(&arr[leftLen], &arr[len-1]);
    int rightLen = len-leftLen-1;
    int *leftArr = arr, *rightArr = arr+leftLen+1;
    quickSort(leftArr, leftLen);
    quickSort(rightArr, rightLen);
}
```

快速排序 (Quick Sort)

- 快速排序範例程式碼 (2/2)：

```
1  int main() {
2      srand( time(NULL) );
3      int arr[] = { 8, 3, 7, 1, 6, 2, 5, 4 };
4      int arr_size = sizeof(arr) / sizeof(arr[0]);
5      quickSort(arr, arr_size);
6      for (int i = 0;i < arr_size;i++) {
7          printf("%d ", arr[i]);
8      }
9      printf("\n");
10 }
```

演算法分析 (快速排序)

- 平均時間複雜度：$O(n \log n)$
- 最差時間複雜度：$O(n^2)$
- 額外空間複雜度：$O(\log n) \sim O(n)$
- 是否有穩定性：否
- 傳統的快速排序算法通常使用遞迴來實現，在遞迴過程中需要使用額外的堆疊空間來存儲每次遞迴調用的參數和局部變數，因此其額外空間複雜度為 $O(\log n) \sim O(n)$，具體取決於遞迴深度。

計數排序
Counting Sort

計數排序

- 計數排序的基本原理是建立一個計數數列，其中每個元素表示輸入數列中該數字出現的次數。
- 然後通過計算每個元素在計數數列中的累積次數，可以得到每個元素在排序數列中的位置。
- 最後根據這些位置將元素放回到排序數列中。

計數排序 (Counting Sort)

- 計數排序的步驟如下：
  1. 確定待排序數列的最大值 maxValue。
  2. 建立一個計數數列 count，其中 count[i] 表示數列中值為 i 的元素出現的次數。
  3. 再對於待排序數列中的每個元素 arr[i]，將 count[arr[i]] 的值增加 1。
  4. 對於計數數列中的每個元素 count[i]，計算累積次數將其存儲在 count[i] 中。
  5. 建立一個與待排序數列相同大小的輔助數列 output。
  6. 從末尾開始遍歷待排序數列，對於每個元素 arr[i]，將 count[arr[i]] 的值減少 1，並將其存儲在 output 數列中對應的位置。
  7. 將 output 數列中的元素複製回原始數列。

計數排序 (Counting Sort)

- 以下為示意圖，有了累計的數量後，我們就可以得知該數字應該排到的位置，進而去排序整個陣列。

數字	1	2	3	4	5
數量	2	0	1	3	1
累計	2	2	3	6	7

計數排序 (Counting Sort)

- 計數排序範例程式碼：

```
1  void counting_sort(int arr[], int n) {
2      int maxValue = arr[0];  // 確定待排序數組的最大值
3      for (int i = 1; i < n; i++) {
4          if (arr[i] > maxValue)
5              maxValue = arr[i];
6      }
7      int count[maxValue + 1];  // 建立計數數組
8      for (int i = 0; i < maxValue + 1; i++) count[i] = 0;
9      for (int i = 0; i < n; i++) count[arr[i]]++; // 計算每個元素的次數
10     // 計算每個元素在排序數組中的位置
11     for (int i = 1; i < maxValue + 1; i++) count[i] += count[i - 1];
12     int output[n];  // 建立輔助數組
13     for (int i = n - 1; i >= 0; i--) {  // 將元素放回排序數組中
14         output[count[arr[i]] - 1] = arr[i];
15         count[arr[i]]--;
16     }
17     // 將輔助數組中的元素複製回原始數組
18     for (int i = 0; i < n; i++) arr[i] = output[i];
19 }
```

演算法分析 (計數排序)

- 平均時間複雜度：$O(n+k)$
- 最差時間複雜度：$O(n+k)$
- 額外空間複雜度：$O(n+k)$
- 是否有穩定性：是
- 注：k 是數列中元素的範圍。

總結
Summary

總結 (Summary)

總結比較

演算法	平均時間	最差時間	額外空間	穩定性
氣泡排序	$O(n^2)$	$O(n^2)$	$O(1)$	是
選擇排序	$O(n^2)$	$O(n^2)$	$O(1)$	是
插入排序	$O(n^2)$	$O(n^2)$	$O(1)$	是
合併排序	$O(n \log n)$	$O(n \log n)$	$O(n)$	是
快速排序	$O(n \log n)$	$O(n^2)$	$O(\log n) \sim O(n)$	否
計數排序	$O(n+k)$	$O(n+k)$	$O(n+k)$	是

Q & A