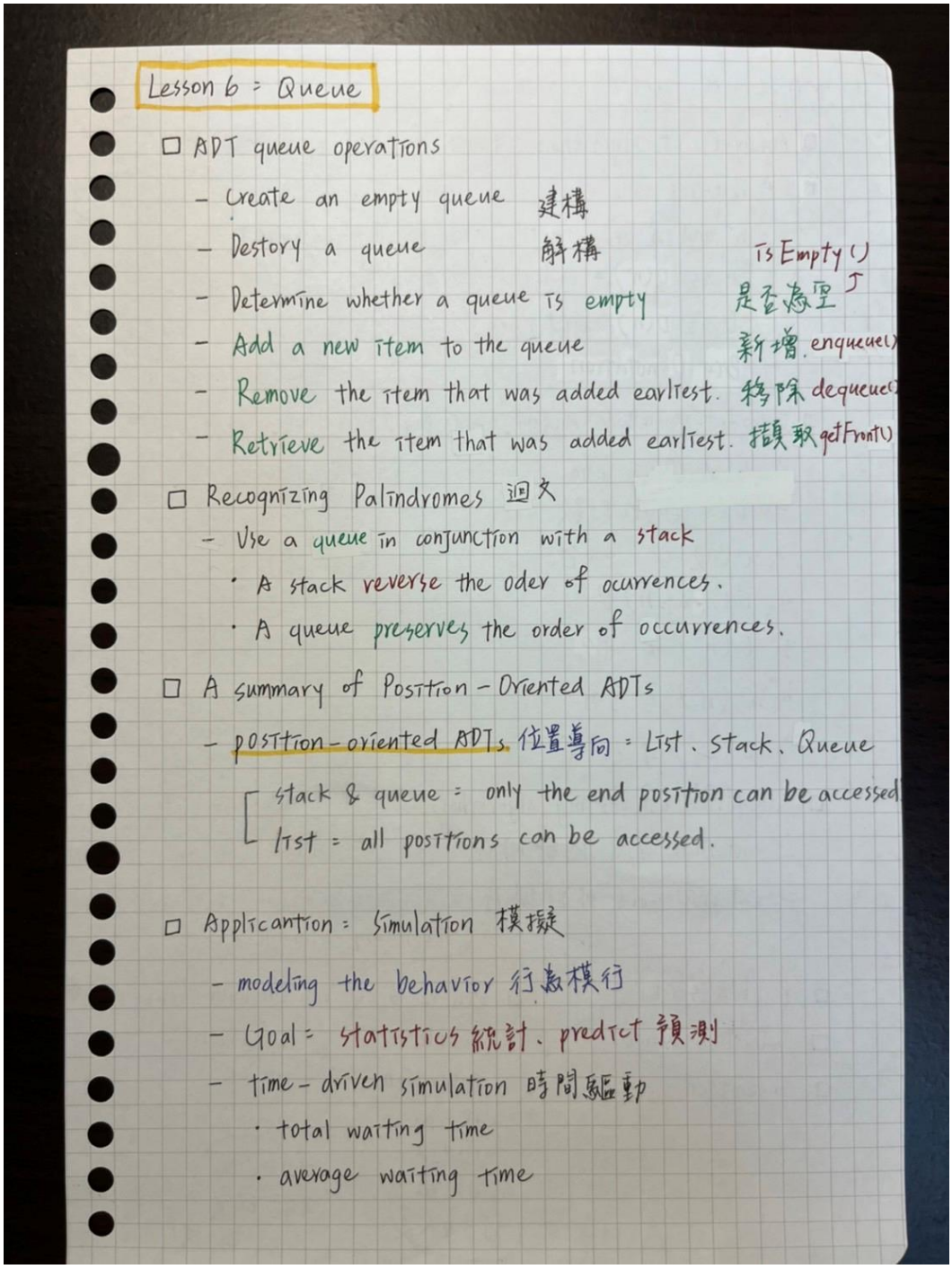
資訊二甲 **10927159** 林玟君



Lesson 6 : Queue

☐ ADT queue operations
- Create an empty queue 建構
- Destory a queue 解構
- Determine whether a queue is empty 是否為空 IsEmpty()
- Add a new item to the queue 新增 enqueue()
- Remove the item that was added earliest. 移除 dequeue()
- Retrieve the item that was added earliest. 擷取 getFront()

☐ Recognizing Palindromes 迴文
- Use a queue in conjunction with a stack
  - A stack reverse the oder of ocurrences.
  - A queue preserves the order of occurrences.

☐ A summary of Position-Oriented ADTs
- position-oriented ADTs 位置導向 : List、Stack、Queue
  - stack & queue : only the end position can be accessed
  - list : all positions can be accessed.

☐ Application : Simulation 模擬
- modeling the behavior 行為模行
- Goal : statistics 統計、predict 預測
- time-driven simulation 時間驅動
  - total waiting time
  - average waiting time

Lesson 1 = Alogorithm Efficiency

- Analysis of algorithms = Time efficiency, space efficient
- A requires time → n^2 ; B requires time → n
  - n^2 and n are growth-rate function 成長函數
  - A is $O(n^2)$ - order n^2 大小位階.
    B is $O(n)$ - order n
  - Big O notation
- Order-of-Magnitude Analysis and Big O Notation.
  1、$O(1)$ A 常數
  2、$O(\log_2 n)$ B 對數
  3、$O(n)$ C 線性
  4、$O(n \log_2 n)$ D
  5、$O(n^2)$ E 平方
  6、$O(n^3)$ F 立方
  7、$O(2^n)$ G 指數
- categories of sorting algorithms
  - internal sort 內部排序 = 電腦內記憶體可容納的data
  - external sort 外部排序 = 內部記憶體無法容納的data
    secondary storage

- Bubble Sort
- Selection Sort
- Insertion Sort

- stable sort v.s. Unstable Sort

stable	Unstable
bubble	selection
insertion	quick
merge	heap
radix	

相同值 維持不變 順序

相同值 可能改變 → 順序

□ Bubble Sort (Bubble down)

```
void BubbleSort (int A[], int n) {
    for (pass = 1 ; pass < n ; ++pass) {
        for ( int index = 0; index < n-pass ; index ++) {
            if (A[index] > A [index+1 ]) // 相鄰兩筆排序
                swap ( A[index], A [index+1]) ;
        } // for
    } // for
} // BubbleSort ()
```

比較次數： $n + n[n(n-1) - \frac{n(n-1)}{2}] + (n-1) = n^2+n-1 \rightarrow O(n^2)$ (worst case)

Best case $\Rightarrow O(n)$

□ Mergesort 合併排序

(遞迴呼叫)

分組 → 各自排序 → 合併

- A recursive sorting algorithm
- strategy : 1、Divide an array into halves
  2、Sort each half
  3、Merge the sorted halves into one sorted array
  4、Divide-and-conquer

3、邊合併邊排序

- stable!
- What is the big-O notation of merge() ? $O(n)$
  What is the number of recursive calls ? $n-1$ → 總數
- Analysis : Worst case : $O(n * \log_2 n)$
  Average case : $O(n * \log_2 n)$
- (優) 快 (缺) : 需要額外2個陣列

□ Quicksort 快速排序

- strategy 1、choose a pivot 樞紐、軸
  2、分組 (依軸的位置)
  - Items < pivot
  - Items >= pivot
  - Pivot is now in correct sorted position.
  3、sort the left section (遞迴呼叫)
  4、sort the right section

- What is the big-O notation of partition()? O(n)
  What is the number of recursive calls? O(logn)
- Worst case = $O(n^2)$
  Average case = $O(n * \log_2 n)$

□ Radix Sort

- 10 is the radix of the decimal system 基數(十進制)
- strategy: Decompose the sort key by the radix 分解取部份值
- LSD (Least Significant Digit) 從右到左
  MSD (Most Significant Digit) 排序時最重要的數
  · 從 LSD 開始做: 最後一次依 MSD 分配、串接.
  · 從 MSD 開始做: 最後一次依 LSD 分配、串接
- Unstable!

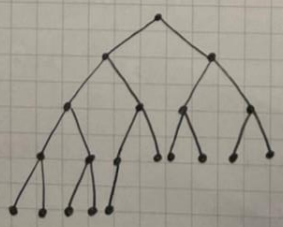
Lesson 8 : Trees

- General : Operations that Insert、delete、ask questions、
  - Position-oriented 位置導向
    - eg. list、stack、queue、binary tree
  - Value-oriented 內容導向
    - eg. sorted list、binary search tree

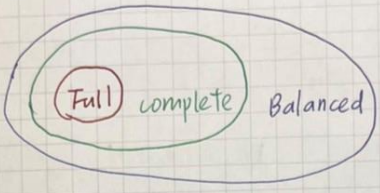
- Terminology
  - Trees are hierarchical : Parent-child、Ancestor-descendant 親子關係 祖孫關係
  - subtree of tree 子樹
  - General tree : one or more nodes 至少一個節點
    - A single node r, the root
  - Leaf : A node with no children (葉結點)
  - siblings : Nodes with a common parent (兄弟結點)

- Binary Tree
  - Height of Tree : the longest path

- Full Binary Tree 完全樹 : 沒有缺項，資料量固定，限制多
- Complete Binary Tree 完整樹 : full to level h-1
  filled from left to right



□ Balanced Binary Tree 平衡樹：任兩點的樹高差值不超過1.



□ Minimum height (complete binary tree):

$n \le 2^h - 1 \Rightarrow \log_2(n+1) \le \log_2(2^h) \Rightarrow h = \lceil \log_2(n+1) \rceil$

□ Maxmum height (complete binary tree):

$h = \lfloor \log_2(n) \rfloor + 1$

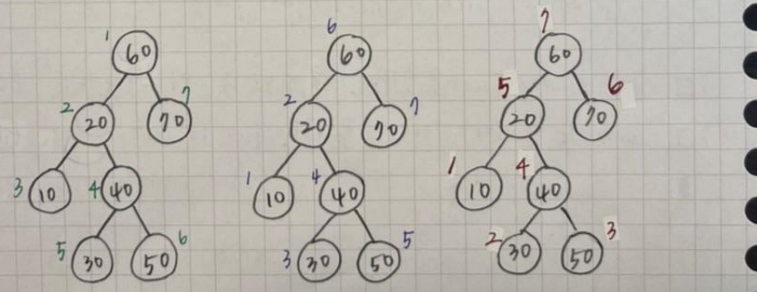
□ Traversals of a Binary Tree:

```
traverse ( Node* walk ) {
    if ( walk != NULL) {
        // printf
        traverse ( walk -> left);
        // printf
        traverse ( walk -> right);
        // printf
    } // if
```

← 前序 preorder

← 中序 Inorder

← 後序 postorder



(a) preorder:
60、20、10、40、30、50、70

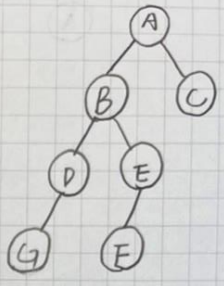
(b) Inorder:
10、20、30、50、40、60、70

(c) postorder:
10、30、50、40、20、70、60

□ Practice 8-1 :

ex: Postorder : GDFEBCA (低 ← 樹根)

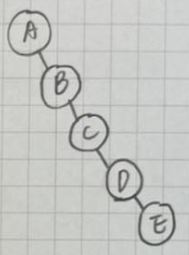
Inorder : GDBFEAC (左 → 右)



ex: Preorder : ABCDE (樹根 → 低)

Inorder : ABCDE

(Inorder = sort 排序)



□ Binary Search Tree

- Average case : O(logn)
- Worst case : O(n)

□ Efficiency = 最高比較次數 = 樹高、
加入移除的次序會影響樹高、
隨機次序加入可趨近最小樹高

□ Tree sort : 建立二元樹 → 中序走訪 (排序)

- Build a binary search tree by n case :
  - Average case : O(n * log n)
  - Worst case : O(n * n)
- Inorder traversal : O(n)