

ch5: stacks (Last In First Out)

堆疊是一種按照先進先出 (LIFO) 原則運作的線性資料結構。這意味最後加入堆疊的元素將是第一個被刪除的元素。

堆疊可以用陣列或串連結串列來實現

以下是常見的堆疊操作

push: 將元素加入堆疊頂部

pop: 從堆疊中刪除頂部元素

peek: 返回堆疊頂部元素的值, 但不刪它

isEmpty: 如果堆疊為空, 則返回 true, 否則返回 false

堆疊常用於解答運算式, 也可以搜尋兩點間的路徑

ch6: 佇列

佇列是一種先進先出 (FIFO) 的數據結構。這代表佇列中的第一個元素 (front) 是最早添加到佇列中的元素, 而佇列中的最後一個元素 (rear) 是最近添加到佇列的元素。先插入的元素會先得到處理。

以下為常見的佇列操作:

isEmpty: 如果佇列為空, 則返回 true, 否則返回 false

enqueue: 向佇列中新增一個元素

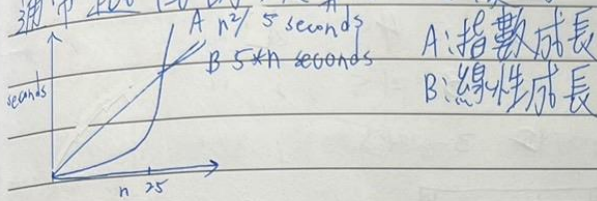
dequeue: 刪除佇列中最早插入的元素

getFront: 返回佇列中最早插入的元素, 但不刪除

ch7: 演算法效率

演算法效率是指演算法在解決問題時所需要的時間和空間。常用的度量演算法效率的方法有時間複雜度和空間複雜度。

時間複雜度是演算法執行時間和輸入數據之間的增長關係。通常是用大O表示法來表示， $O(n)$ 表線性時間複雜度，通常越低的演算法越優秀。



Ex1. $(n+1) * (c * a) + n * w$

$\forall n \geq n_0, (n+1) * (c * a) + n * w \leq k * f(n)$

$\forall n \geq 1, (n+1) \leq n \rightarrow$

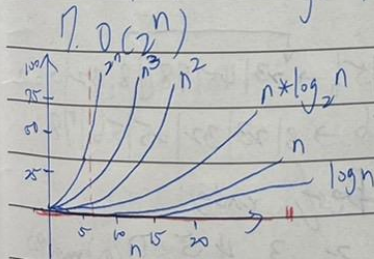
$(n+1) * (c * a) + n * w \leq n * (c * a + w) \leq k * n$ (A) 最好的

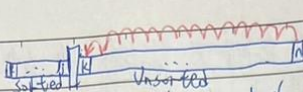
Ex2: $2^n - 1$

$\forall n \geq n_0, (2^n - 1) \leq k * f(n)$

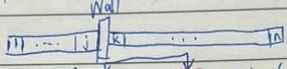
$\forall n \geq 1, (2^n - 1) \leq 1 * 2^n \rightarrow O(2^n)$

1. $O(1)$ 2. $O(\log_2 n)$ 3. $O(n)$ 4. $O(n \log_2 n)$ 5. $O(n^2)$ 6. $O(n^3)$

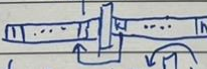


Bubble Sort:  + swap the smallest
 $23|78|45|8|32|56 \rightarrow 8|23|78|45|32|56 \rightarrow 8|23|32|78|45|56 \rightarrow$
 $8|23|32|45|78|56 \rightarrow 8|23|32|45|56|78$

Worst case $\frac{n*(n-1)}{2} = 10$ best case $0(n^2)$
 $5\ 4\ 3\ 2\ 1$ 4 swaps $1\ 2\ 3\ 4\ 5$ 4 compare
 $4\ 3\ 2\ 1\ 5$ 3 " $1\ 2\ 3\ 4\ 5$ 3 "
 $3\ 2\ 1\ 4\ 5$ 2 " $1\ 2\ 3\ 4\ 5$ 2 "
 $2\ 1\ 3\ 4\ 5$ 1 " $1\ 2\ 3\ 4\ 5$ 1 "

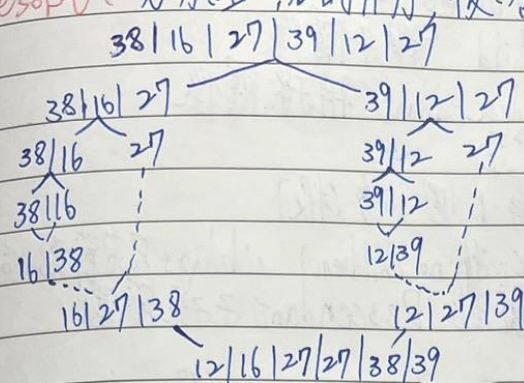
Selection Sort: 
 $23|78|45|8|32|56 \rightarrow 8|78|45|23|32|56 \rightarrow 8|23|45|78|32|56 \rightarrow$
 $8|23|32|45|78|56 \rightarrow 8|23|32|45|56|78$

Worst case $0(n^2)$ best case still $0(n^2)$
 $5\ 4\ 3\ 2\ 1$ 4 compare $1\ 2\ 3\ 4\ 5$ 4 compare $\rightarrow 0(n)$ swaps $\rightarrow 0(n)$
 $1\ 4\ 3\ 2\ 5$ 3 compare $1\ 2\ 3\ 4\ 5$ 3 "
 $1\ 2\ 3\ 4\ 5$ 2 compare $1\ 2\ 3\ 4\ 5$ 2 "
 $1\ 2\ 3\ 4\ 5$ 1 compare $1\ 2\ 3\ 4\ 5$ 1 "

Insertion Sort: 
 $23|78|45|8|32|56 \rightarrow 23|78|45|8|32|56 \rightarrow 23|45|78|8|32|56 \rightarrow$
 $8|23|45|78|32|56 \rightarrow 8|23|32|45|78|56 \rightarrow 8|23|32|45|56|78$

Worst case $\frac{n*(n-1)}{2} + (n-1) = 14$ Best case $0(n)$
 $5\ 4\ 3\ 2\ 1$ 4 compare $1\ 2\ 3\ 4\ 5$ 0 move
 $4\ 5\ 3\ 2\ 1$ 3 compare $1\ 2\ 3\ 4\ 5$ 1 move
 $3\ 4\ 5\ 2\ 1$ 2 compare $1\ 2\ 3\ 4\ 5$ 1 move
 $2\ 3\ 4\ 5\ 1$ 1 compare $1\ 2\ 3\ 4\ 5$ 1 move
 $1\ 2\ 3\ 4\ 5$ 0 compare $1\ 2\ 3\ 4\ 5$ 0 move

Mergesort: 先分組, 各自排序, 後: 合併



$2n$ moves (每個資料會搬2次)
 $\Rightarrow (n-1) + 2n = 3n - 1 \Rightarrow O(n)$

Worst case: $O(n \times \log_2 n)$ Average case: $O(n \times \log_2 n)$

Quick Sort:

先挑 pivot

↳ 分組: 1. items < pivot 2. items >= pivot 先: 分組

↳ sort left section / sort right section 後: 遞迴呼叫

pivot 最後才換到要去的位子上

Average case: $O(n \times \log_2 n)$ worst case: $O(n^2)$

平均效率好, 但最差的情況慢

Radix sort:

基數排序原理是將整數按位數切割成不同數字, 按每個位數比較, 將數據分配到桶之中 \Rightarrow 先分組/後串接

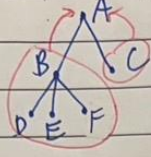
$\Rightarrow O(n)$ 快!

	Worst	Average	Best
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Mergesort	$O(n \times \log n)$	$O(n \times \log n)$	$O(n \times \log n)$
Quicksort	$O(n^2)$	$O(n \times \log n)$	$O(n \times \log n)$
Radix sort	$O(n)$	$O(n)$	$O(n)$

No.

Date

ch8 樹:



具有階層的結構

parent-child 親子關係

Ancestor-descendant 祖孫關係

subtree 子樹

* 樹結構只有 1 個 root (根)

Leaf 葉節點 (with no children) siblings 兄弟節點 (同父)

Ancestor 祖先節點 Descendant 子孫節點

* 二元樹 Binary Tree

最多 2 個, 亦可 1 個或無

樹高影響訊息的傳遞
↳ 最長那條 = 最大階層



一般



二元

走訪 = 元: traverse (in binary tree: Binary Tree)

if (binary tree is not Empty) {

traverse (left subtree of binary tree's root)

traverse (right subtree of binary tree's root)

也能用 stack (堆疊) 實作

→ 前序

→ 中序

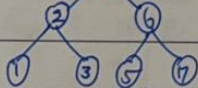
→ 後序

* 二元搜尋樹 Binary Search Tree

存資料能提供搜尋的功能 → 不須指定要放哪個位置

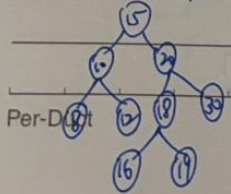
任意節點的左子樹皆 < 父節點, 右子樹皆 > 父節點

2 4 6 7 In Order Traversal: 1 2 3 4 5 6 7

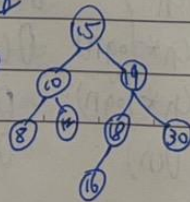


刪除某一個節點

delete



Per-Order



if 沒小孩 → 直接刪非葉結點
則以該刪點為基準以左子最大, 右子最小取代