

週四作業討論

HW



Queues and Stacks List(佇列與堆疊串列)

Q:使用單向串列完成以下程式，使輸出結果如右



```
int main() {  
    StackList s;  
    s.Push(99);s.Push(88);s.Push(77); //PushBack? PushFront?  
    cout << s.Top() << endl;  
    s.Pop(); s.Pop();  
    cout << s.GetSize() << endl;  
    cout << s.Top() << endl;  
}
```

77
1
99

週四作業討論

HW



Queues and Stacks List(佇列與堆疊串列)

Q:使用單向串列完成以下程式，使輸出結果如右



```
int main() {  
    QueueList q;  
    q.Push(99); q.Push(88); q.Push(77); //PushBack? PushFront?  
    cout << q.front->data << " " << q.back->data << "\n";  
    q.Pop();q.Pop();  
    cout << q.GetSize() << endl;  
    cout << q.GetFront() << " " << q.GetBack() << endl;  
}
```

```
99 77  
1  
77 77
```

Binary Search Tree



二元樹



每個節點最多有兩個子節點，或兩個子樹，
且左右子樹有區別



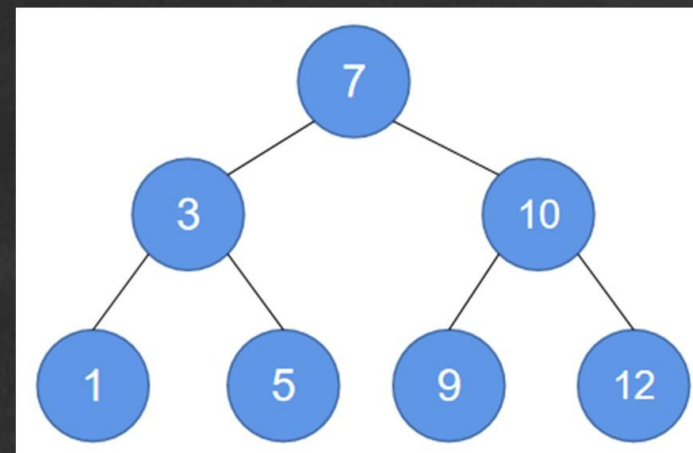
階層 n 的節點數至多為 2^{n-1} 個



深度為 N 的總節點數至多為 $2^N - 1$ 個

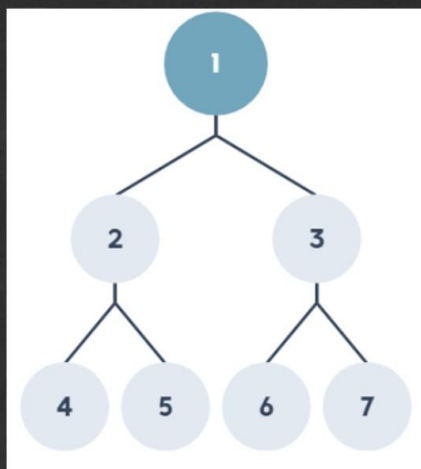


若一非空二元樹leaf個數為 n (e.g. 4)個，
則分支度為2的節點個數為 $n-1$ 個



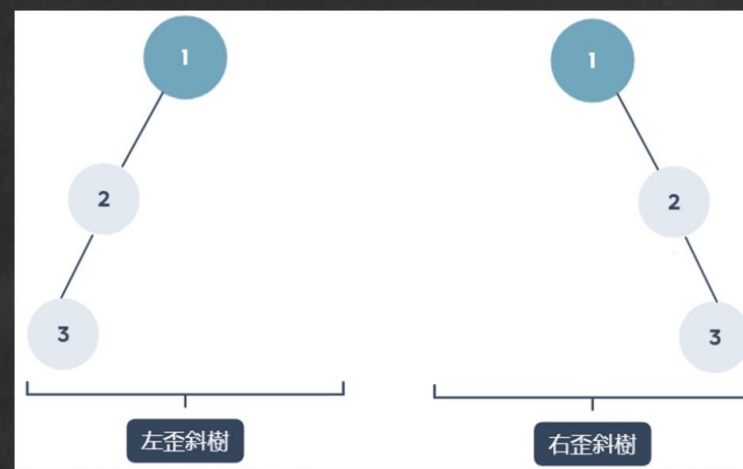
二元樹

➡ 完滿二元樹 (Fully Binary Tree)



深度為N且總節點數為 $2^N - 1$ 個

➡ 歪斜樹 (Skewed Binary Tree)

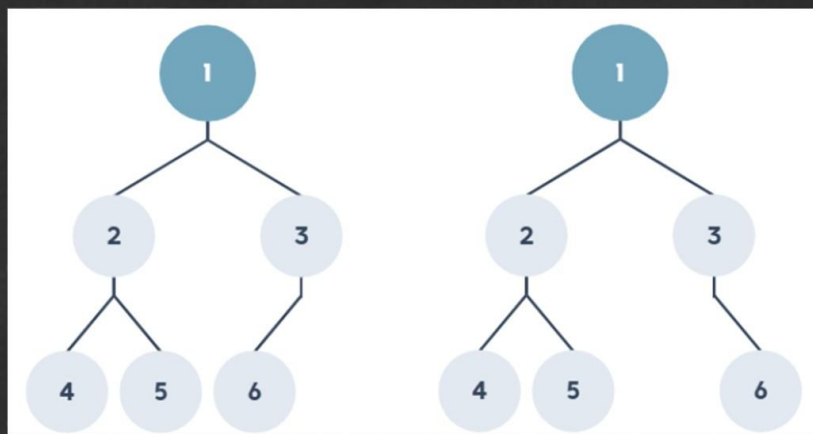


只有左子樹或只有右子樹

二元樹



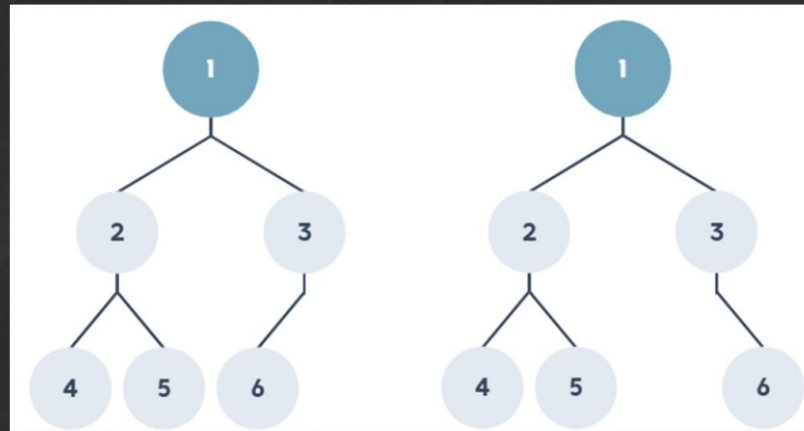
完整二元樹(Complete Binary Tree)



Q:哪一個是完整二元樹?

- 除了最後一層外，所有層的節點都已填滿
- 且最後一層的節點必須從最左邊開始連續地填入，中間不能有空缺
- 深度為N時，總節點數為 $2^{N-1} \sim 2^N - 1$
- 排列順序同完滿二元樹

The hole in the memory of array



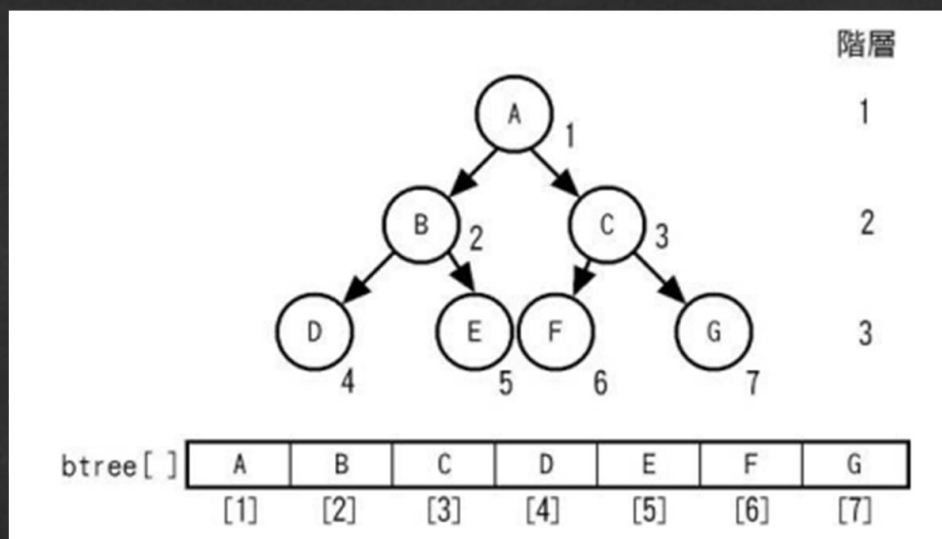
1	2	3	4	5	6
---	---	---	---	---	---

1	2	3	4	5	null	6
---	---	---	---	---	------	---

二元樹



陣列表示法

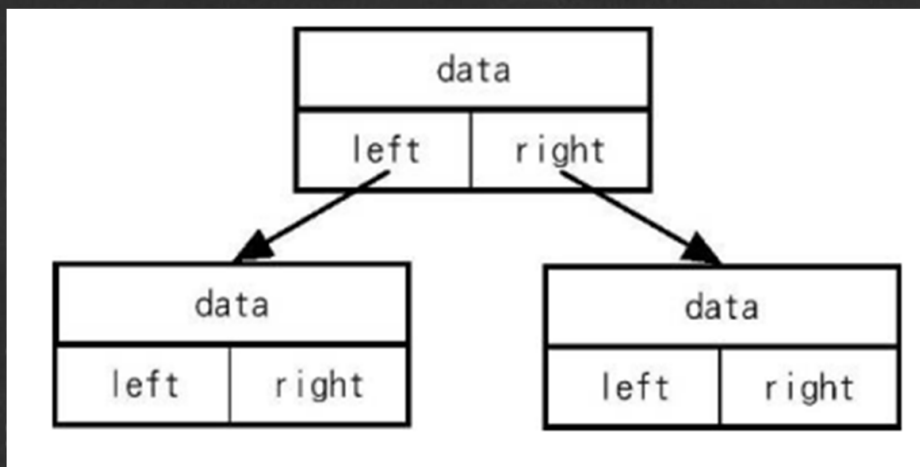


- 左子樹是父節點編號乘以2
- 右子樹是父節點編號乘以2加1
- 節點增刪不易 新增：可能需要找到正確位置、可能產生大量空格。刪除：可能破壞原本的位置關係、若要維持緊密排列，可能需要搬移大量元素
- 歪斜樹浪費空間 節點位置分散、大量索引空缺、浪費空間

二元樹



串列表示法



- 每個節點都有左右指標
- 節點增刪容易 像雙向鏈結一樣，增加：空指標變新節點，刪除：指標變空指標
- (葉片)仍浪費一半空間 葉節點還要記錄左右指標是空指標
- 不易取得父節點 只有記錄子節點，沒有記錄父節點

二元搜尋樹 (BST) 的鐵血規則

如果資料隨便亂擺，樹找起來一樣很慢。Binary Search Tree (BST) 立下了一個嚴格的規定：

左子樹 (Left Child)

數值一定比爸爸 **小**

爸爸 (Parent Node)

判斷大小的基準點

右子樹 (Right Child)

數值一定比爸爸 **大**

* 這種特性讓 BST 每次搜尋都能**砍掉一半**的資料，達成極速 $O(\log N)$!



二元搜索樹

不是往左走就是往右走

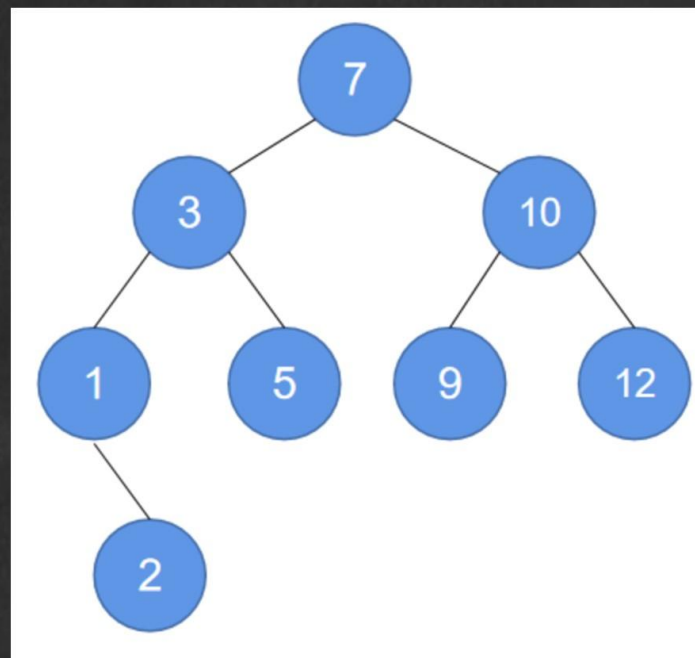


右子節點的值一定 $\geq$ 左子節點
右子樹的值一定 $\geq$ 左子樹



7, 3, 1, 10, 12, 5, 9

Q: 該在哪插入2?



樹的走訪 (Traversals)

要印出樹裡所有的資料，順序很關鍵。我們以「爸爸 (Root)」什麼時候被拜訪來命名：

前序走訪 (Pre-order)

中 → 左 → 右

一走到節點就馬上處理。常用於完整複製一棵樹的結構。

中序走訪 (In-order)

左 → 中 → 右

最常用！對 BST 做中序走訪，印出來的數字**必定是由小到大排序的**。

後序走訪 (Post-order)

左 → 右 → 中

先把底下的小孩處理完，才處理大人。常用於安全地刪除整棵樹。



二元樹走訪



前序走訪(Preorder): 根->左->右

F, B, A, D, C, E, G, I, H



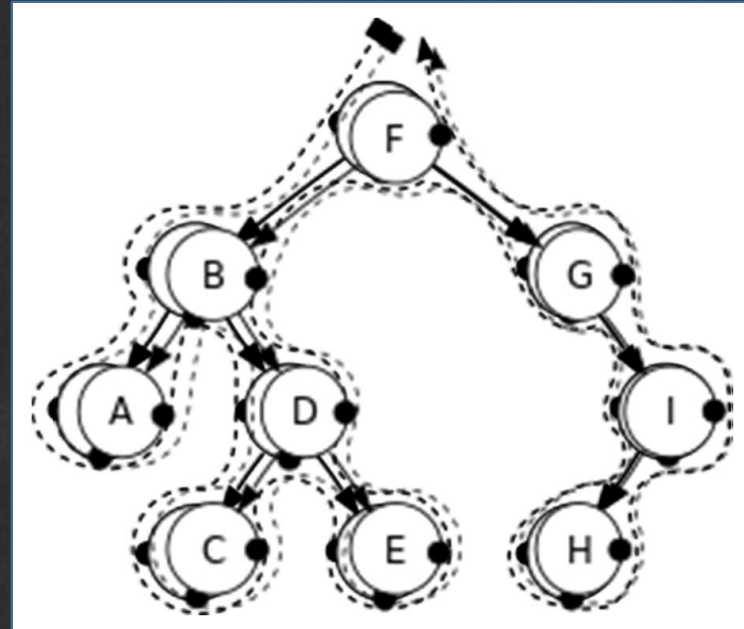
中序走訪(Inorder): 左->根->右

A, B, C, D, E, **F**, G, H, I



後序走訪(Postorder): 左->右->根

A, C, E, D, B, H, I, G, **F**



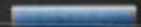


二元搜索樹

Q:畫出以下搜索樹

- 69, -65, -97, 94, -83, 54, -92, -22, -90, 80, -71, 42 <-示範 前序
- -75, 98, 36, -82, -15, -30, -54, 94, 95, -89, -23, 62 <-黃晴 中序
- 26, 96, 34, 58, 21, 59, 68, 76, -15, -23, 74, -51 <-宗群 後序

Sorting Algorithms



為什麼我們需要 Sorting (排序)？

在現實世界中，90% 的運算資源都花在「尋找資料」

。

- 如果資料是亂序的：
我們只能從頭找到尾（線性搜尋），時間複雜度是 $O(N)$ 。
- 如果資料是排序好的：
我們就可以使用二分搜尋法 (Binary Search)，
每次砍掉一半，時間複雜度瞬間降為 $O(\log N)$ ！
- 所以，排序是為了未來的「極速搜尋」做準備。

要選擇速度快、耗能少、花費記憶體少的排序演算法



從 100 萬筆排序好的資料中找人

，

最多只需要找 20 次！

🎵 排序



排序是重新排列元素以使其具有數字或文字順序的動作



穩定與不穩定排序法：
鍵值相同之資料，在排序後相對位置與排序前相同時，稱穩定排序



排序前：3,5,19,1,3*,10
排序後：1,3,3*,5,10,19 (穩定排序)
排序後：1,3*,3,5,10,19 (不穩定排序)





氣泡排序法(Bubble Sort)



重複地走訪整個數列，每次比較相鄰兩個元素，如果順序錯誤就把他們交換，直到不需要再交換為止 $O(N^2)$ $O(N)$

7	2	9	6
7	2	9	6
2	7	9	6
2	7	9	6
2	7	6	9
2	7	6	9
2	7	6	9
2	6	7	9
2	6	7	9
2	6	7	9
2	6	7	9



選擇排序法(Selection Sort)



不斷從未排序部分中找出最小值，接在已排序部分之後，直到全數排序完畢
 $O(N^2)$

7	2	9	6
7	2	9	6
2	7	9	6
2	7	9	6
2	6	9	7
2	6	9	7
2	6	7	9

撲克牌思維：Insertion Sort (插入排序)

- **概念：**想像你正在打撲克牌，整理手牌的過程。
- 把左邊視為「已排序區」。每次從右邊未排序區抽出一張牌，往左邊一路比較，**找到正確的位置「插」進去**。
- **優勢：**如果資料「幾乎已經排好」，速度會逼近 $O(N)$ 。(但最壞情況依然是 $O(N^2)$)。



想像抽出一張新牌，
把前面比它大的牌都往後挪一格，
然後放入騰出的空位。



插入排序法(Insertion Sort)



將尚未排序的資料，逐一與排序好的資料做比對，即可找到合適的插入位置

7	2	9	6
7	2	9	6
7	7	9	6
2	7	9	6
2	7	9	6
2	7	9	6
2	7	9	6
2	7	7	9
2	6	7	9

2

9

6



希爾排序法(Shell Sort)



插入排序法的改良版，初始 $GAP=N/2$ ，依據 GAP 分組並進行插入排序，之後依序除以2直到 $GAP=1$ 最壞情況 $O(N^2)$ ，依 gap 序列而定

6	8	2	3	4	7	5	1
4	7	2	1	6	8	5	3
2	1	4	3	5	7	6	8
1	2	4	3	5	7	6	8
1	2	4	3	5	7	6	8
1	2	3	4	5	7	6	8
1	2	3	4	5	7	6	8
1	2	3	4	5	7	6	8
1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8

$GAP=8/2=4$

$GAP=4/2=2$

$GAP=2/2=1$

插入排序



效能突破：Quick Sort (快速排序)

要打破 $O(N^2)$ 的魔咒，必須使用高階演算法技巧：**Divide and Conquer (分而治之)**。

1. 找基準點 (Pivot)

在陣列中隨便挑一個數字當作基準點（通常選最後一個或中間的）。

2. 兵分兩路 (Partition)

把比基準點「小」的全部丟到左邊，比基準點「大」的全部丟到右邊。

3. 遞迴往下 (Recursion)

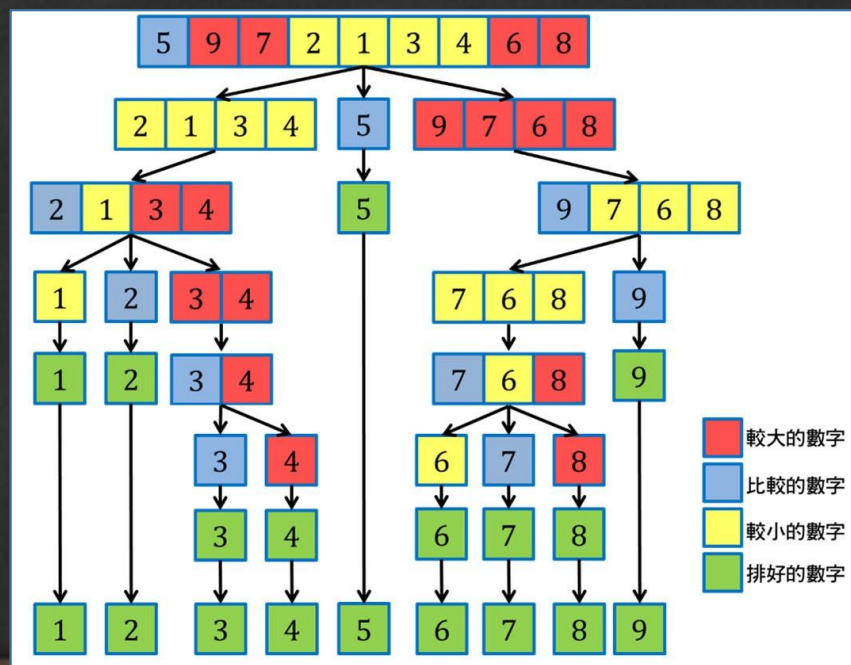
針對左半邊和右半邊，重複 Step 1 和 2。因為每次都拆一半，複雜度降至 **$O(N \log N)$** ！



快速排序法(Quick Sort)



選定一基準值 (Pivot)，將比基準小的數值移到基準值左邊，形成左子串列，比基準值大的數值移到基準值右邊，形成右子串列，再分別對左子串列、右子串列作上述步驟，直到左子串列和右子串列都已排列 $O(N^2)$ (若Pivot挑極端) $O(N \log N)$

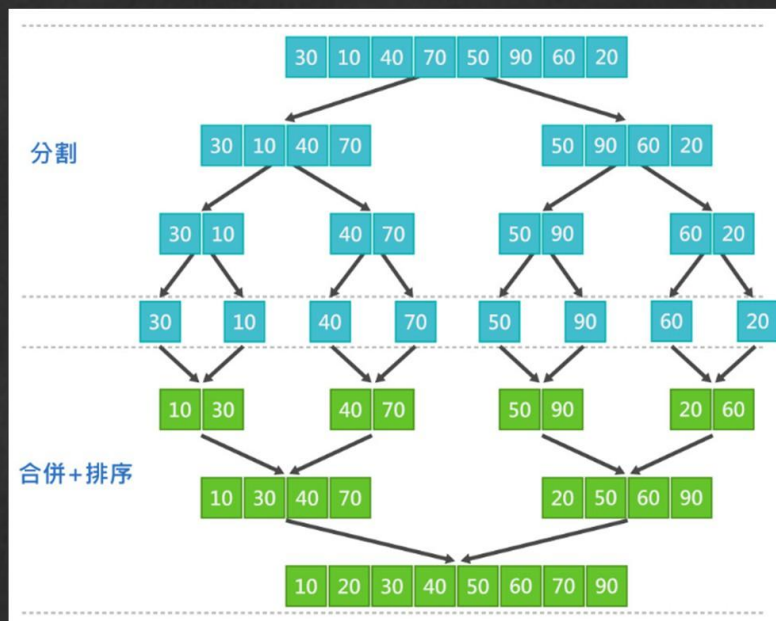




合併排序法(Merge Sort)



把大陣列切一半成為兩個小陣列，再把兩個小陣列再各自切一半，直到每個小陣列都只剩一個元素，將兩個元素排序並合併成小陣列，將小陣列排序並合併成大陣列，直到整個陣列合併完成 $O(N \log N)$





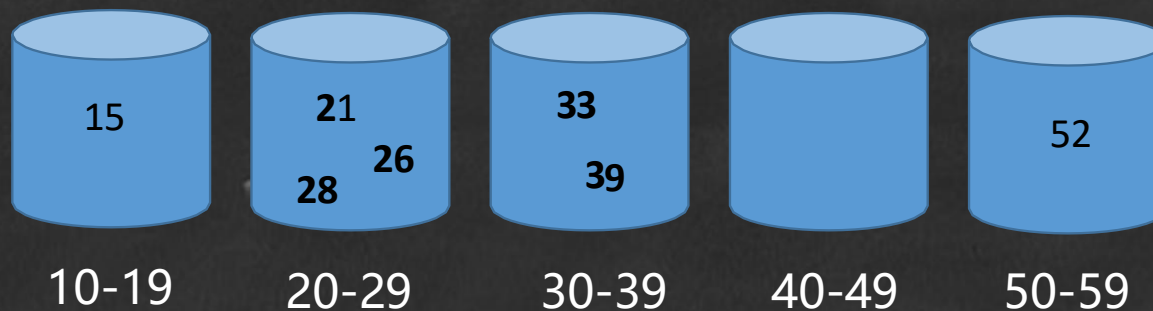
桶排序法(Bucket sort)



建立一定數量的桶子，將資料按照區間放入桶中，將桶中的資料作排序，排序完後，再將所有桶中元素依序放回原始的陣列

最佳: $O(N)$ 最差: $O(N^2)$

EX: [39 ,26 ,21 ,52 ,33 ,15 ,28]



排序後: [15 ,21 ,26 ,28 ,33 ,39 ,52]



基數排序法(Radix Sort)



每回合都根據資料的某位數大小進行分組及排列，直到所有位數都排列完，依序取出桶中的數字放回原陣列 $O(n)$

- LSD: Least Significant Digit First, 從個位數開始，適合位數小的資料
- MSD: Most Significant Digit First, 從最高位數開始，適合位數大的資料

EX: [179, 208, 306, 93, 859, 984, 55, 9, 271, 33]

[271, 93, 33, 984, 55, 306, 208, 179, 859, 9] <- 排完個位數

[306, 208, 09, 33, 55, 859, 271, 179, 984, 93] <- 排完十位數

[009, 033, 055, 093, 179, 208, 271, 306, 859, 984] <- 排完百位數

五天總結

核心武器	關鍵密碼與核心精神
Array & Big-O	記憶體連續性、學會看透程式效率的 $O(n)$ 成長趨勢。
Linked List	尋寶遊戲、指標連線口訣：「先牽新朋友，再放開老朋友」。
Stacks, Queues & Hash	LIFO 疊盤子、FIFO 排隊、以及擁有 $O(1)$ 極速查詢的魔法字典。
Trees (樹狀結構)	階層式分類、二元搜尋樹 (BST) 與前、中、後序的走訪路徑。
Sorting (排序演算法)	亂序重整、體會 Divide & Conquer (分而治之) 的效能突破。

謝謝大家