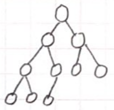
Ch1 優先佇列 Priority Queues

- Queue - FIFO

- Sorting Algorithm

Sorting Algorithm	Worst case	Average case	
Selection	n^2	n^2	選最小或最大資料
Bubble	n^2	n^2	
Insertion	n^2	n^2	
Merge	$n\log n$	$n\log n$	
Quick	n^2	$n\log n$	
Radix	n	n	
Heap	$n\log n$	$n\log n$	
Binary search tree	n^2	$n\log n$	

- Heap 堆積
  - Balanced binary tree
  - Complete
  - min-Heap - 最小的在樹根
  - max-Heap - 最大的在樹根
  - the value stored at a node is greater(smaller) or equal to the value stored at the children



由上到下，由左到右
⇒ complete

- Heap Insert
  → 新增資料於 bottom 後，由下而上追溯資料調整
  → O (log n)

- Heap Delete
  → copy the bottom rightmost element to the root.
  → Delete the bottom rightmost node.
  → Fix the heap property (由上而下)
  → O (log n)

- 以陣列建 heap

$$\text{parent} = \frac{\text{child}-1}{2}$$

$$\text{rightChild} = \text{now} * 2 + 2$$

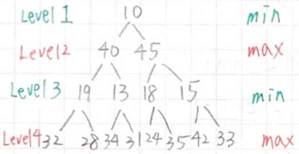
$$\text{leftChild} = \text{now} * 2 + 1$$

- Heap Sort
  → Transform the array into a heap
  → Exchanging the root of the heap with the last element of the heap.
  → Transforms the resulting semi-heap back into a heap.

Ch 2 堆積變形

- Double-ended Priority Queues (DEPQ)
  - Min-max Heap
  - Double-ended Heap (DEAP)
- Forest (union) of Heaps
  - Binomial Heap
  - Fibonacci Heap

- Min-max Heap 最小最大堆積
  - complete binary tree
  - 可知最小或最大的key
  - O(logn)



  - Insert
    1. Decide which Level → min or max
    2. Check whether to swap with its parent
       - 若有交換，從父節點往上檢查 (min-min, max-max)
       - 若沒交換，從現在節點往上檢查 (min-min, max-max)

- Grandparent = $\frac{\text{current node} - 3}{4}$

- Grandchild = $i * 4 + j$, for $j = 3, 4, 5, 6$

- 1 min-Heap + 2 max-Heap

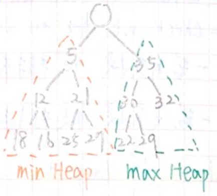
- Delete the smallest
  1. Replace the root with the last element
  2. check whether to swap its smaller child.
     - 若有交換：從樹根往下檢查 (min-min)
     - 若沒交換，從樹根往下檢查 (min-min)

- Delete the Largest
  1. Replace the maximum with the last element
  2. check whether to swap with its larger child.
     - 若有交換：從現在節點往下檢查 (max-max)
     - 若沒交換，從現在節點往下檢查 (max-max)

- 計算在 min or max 層

  $level = \log_2(i+1)$ ，算出的結果除 2 可判斷為 min or max 層

- ~~Double-ended Heap (DEAP) 雙向堆積~~
  - Pseudo root + min-Heap + max-Heap
  - Insert
    1. Examine the corresponding nodes : (對應節點)
       Left < Right



    2. Reheap Up if necessary

       若有交換,(新增資料)交換後的位置往上檢查

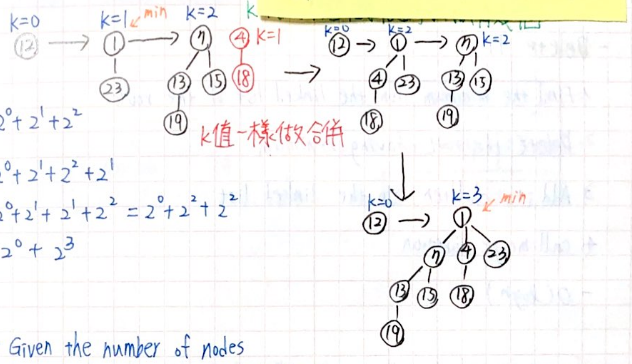
       若找不到對應位置，和對應節點之父節點檢查

  - Delete the smallest
    1. Replace the root of min-heap with the last element
    2. Reheap Down if necessary
    3. Examine the corresponding nodes: left < Right

  - Delete the Largest
    1. Replace the root of max-heap with the last element.
    2. Reheap Down if necessary
    3. Examine the corresponding nodes : left < Right
       - 若對應節點沒交換,且還有下一層,要和對應點小孩比較

- ~~Binomial Heap~~
  - a collection of binomia
    and have distinct order
  - two binomial trees of

- The root has k children
- Merged by two binomial trees of order k-1
- Number of nodes = 2^k
- Tree height = $k+1 \rightarrow O(\log n)$
- C^k_i nodes at level i, for r = 0...k



$7 = 2^0 + 2^1 + 2^2$

$9 = 2^0 + 2^1 + 2^2 + 2^1$

$= 2^0 + 2^1 + 2^1 + 2^2 = 2^0 + 2^2 + 2^2$

$= 2^0 + 2^3$

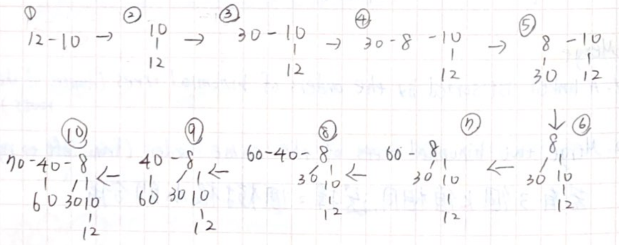
  - Given the number of nodes
    → a unique structure 結構唯一

  - Merge
    1. A linked list sorted by the orders of binomial trees (degrees of the roots)
    2. Merge two binomial trees of the same order (from left to right)
       若有 3 個 k值相同,選擇 2 個樹根小的合併

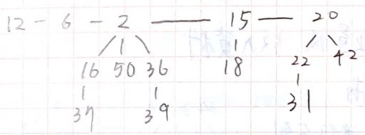
  - O (log n)

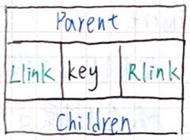
- Insert
  1. Insert into the linked list of the roots
  2. call merge function
  - O(log n)
- Delete ..
  1. Find the minimum from the linked list of the roots.
  2. Delete the root having minimum
  3. Add its children into the linked list
  4. call merge function
  - O(log n)

- Insert 10, 12, 30, 8, 60, 40, 70 (min Heap)



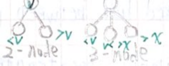
- Fibonacci Heap
  - Doubly linked list on the siblings (tree roots)
  - Doubly linked list between parent and child
  - Merge: simply concatenate two lists of tree roots.





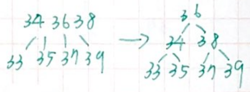
CH3 由下而上成長的平衡二元樹

- 2-3 tree
  - 2-node ⇒ 2個小孩
  - 3-node ⇒ 3個小孩

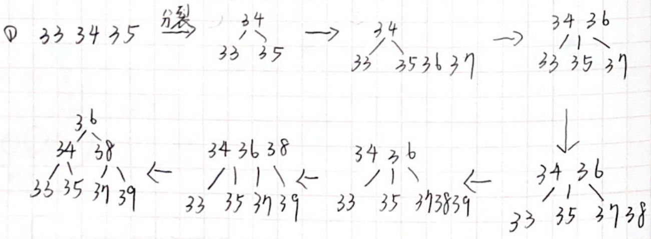


  - Never has height greater than $\log_2(n+1)$

  - Insert
    1. 從樹根往下搜尋位置後，放入資料
    2. 檢查樹葉是否滿
       - 若有3筆資料：進行分裂



- Insert 33, 34, 35, 36, 37, 38, 39



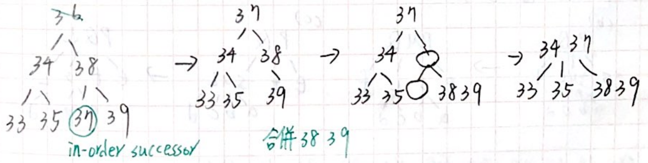
- Delete
  1. 找到所要刪除對象的樹葉，並刪除資料
  2. 檢查樹葉是否空
     - 若為空：1. 看是否需重新分配) 2選1
       2. 合併

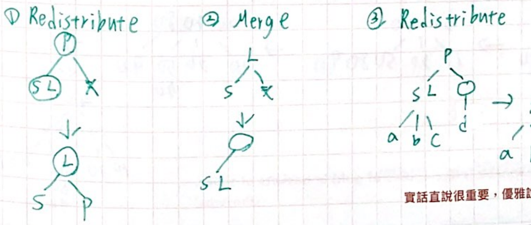
選分配：兄弟節點是 3-node
選合併：兄弟節點是 2-node

當 root 為空，Delete root.

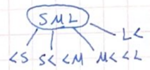
若要刪內部節點，先 1. 交換 in-order successor on a leaf
2. 從 leaf 刪除資料

刪 36





- 2-3-4 tree



· always balanced

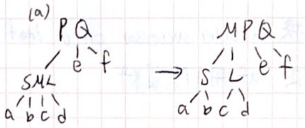
· 樹高比 2-3 tree 矮

- Insert

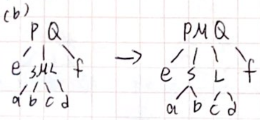
只要遇到滿就分裂

parent is 2-node → same as 2-3 tree
parent is 3-node → move the middle item up

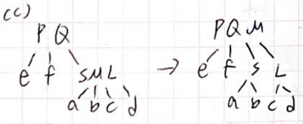
(a)



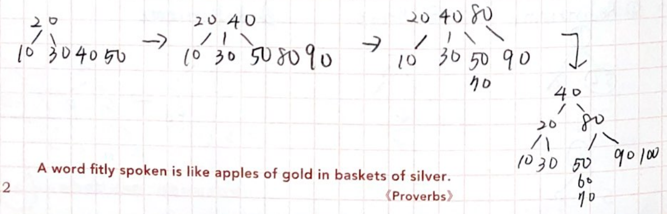
(b)



(c)



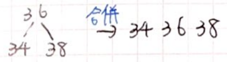
ex: 10, 20, 30, 40, 50, 90, 80, 70, 60, 100



- Delete

1. Locate the node n that contains the item theItem
2. Find theItem's inorder successor and swap it with theItem.
3. If that leaf is a 3-node or a 4-node, remove theItem

→ Parent and sibling are 2node



My Notes

Important Concepts worth keeping

Today :   /   /

Ch4 由上而下成長的平衡二元樹

- AVL

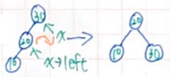
-Balance Factor (BF) 平衡係數

⇒ BF (a node) = h (left subtree) - h (right subtree)

BF > 0 → 左邊重 . BF < 0 → 右邊重

-Single Rotation

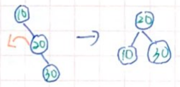
- LL



BF(x) = 2

BF(x→left) = 1

- RR

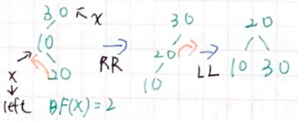


BF(x) = -2

BF(x→right) = -1

- Double Rotation

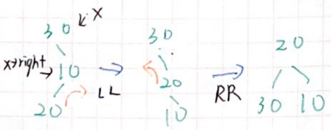
- LR



BF(x) = 2

BF(x→left) = -1

- RL



BF(x) = -2

BF(x→right) = 1

If you don't know where you're going it doesn't matter what path you take.
– Lewis Carroll

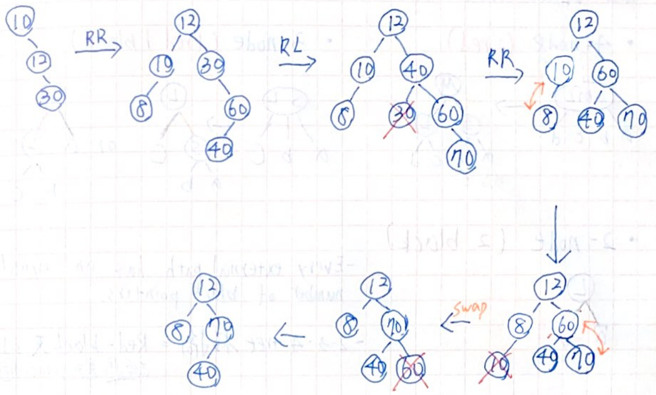
My Questions

Problems & Difficulties needing exploration

My Opinions

Thoughts, inspirations, and suggestions

EX: 10, 12, 30, 8, 60, 40, 70, -30, -10, -60



如果你不知道你要去哪裡，那麼現在你在哪裡一點都不重要。
《愛麗斯夢遊記》

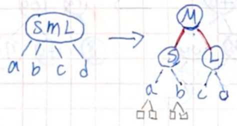
My Notes

Important Concepts worth keeping

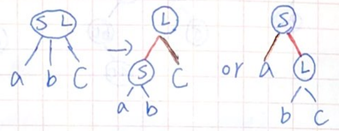
Today :   /   /

- Red - BLack Tree

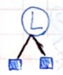
  - 4-node (2red)



  - 3-node (1red 1 black)



  - 2- node (2 black)



-Every external path has an equal number of black pointers.

- 2-3-4 tree 樹高 = Red-black走到leaf 經過黑色 pointer個數

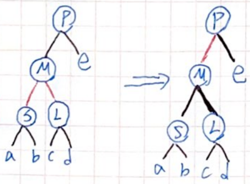
-紅色不能連續出現，連續就旋轉

- red-black 樹高最多為 2倍 2-3-4 tree

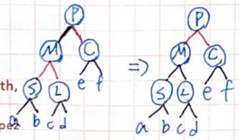
- Split

1. Parent is a 2-node
   only change the color

   → to child : red → black
   → from parent : black → red



2. Parent is a 3 node
   color change + LL
   RR
   LR
   RR



Whenever it feels uncomfortable to tell the truth, that's often the most important time to tell it.

–Jennifer Lopez

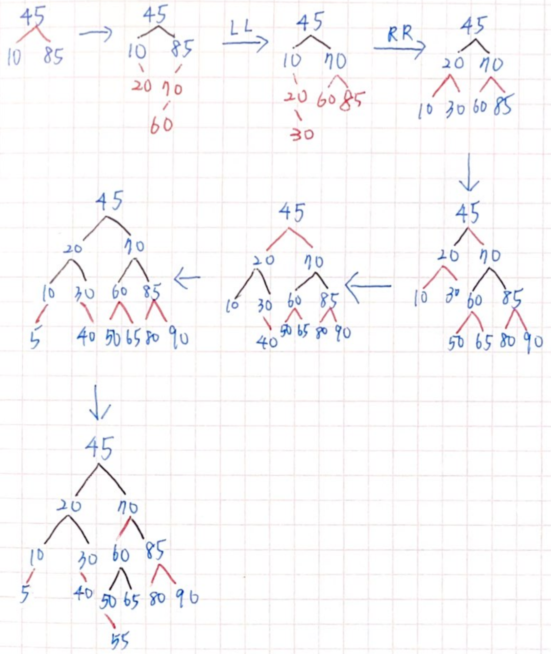
My Questions

Problems & Difficulties needing exploration

My Opinions

Thoughts, inspirations, and suggestions

EX : 45, 85, 10, 70, 20, 60, 30, 50, 65, 80, 90, 40, 5, 55



密碼
cipher key

每當覺得說實話很難，通常正是最應該說實話的重要時刻。