No
Date

Priority Queue 優先佇列

*有"特例"會被優先處理. 比.看急診

應用：找最近距離的城市...

因 selection sort, tree sort 有名詞的時間複雜度

Heap 堆積

*平衡的二元樹 (complete) 完整二元樹 上到下、左到右【連續】沒缺少任何一塊

=> min-Heap、max-Heap Insert(): O(logn) (n)

Build heap?

ReheapDown() pq.Delete(): O(logn)

ReheapUp() pq.Insert(): O(logn)

heapInsert() efficiency: O(logn)

No
Date

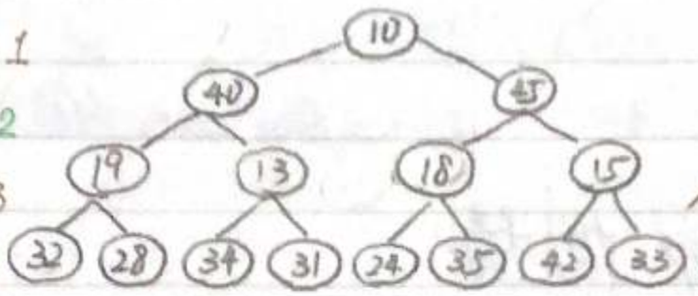
單元二

* Double-ended Priority Queue (DEPQ)

Min-max Heap

- Insert any key
- Delete the smallest key
- Delete the largest key

Level 1 — min
Level 2 — max
Level 3 — min
Level 4 — max



1. Decide which level => min or max
2. Check whether to swap with its parent
   - No => ReheapUp from the current node
   - Yes => ReheapUp from its parents

Min-max heap

Insert

---

1. Replace the root with the last element
2. Check whether to swap with its smaller child
   - No: ReheapDown from the root
   - Yes: ReheapDown from the root

Min-max heap

Delete the smallest

---

1. Replace the maximum with the last element
2. Check whether to swap with its larger child
   - No: ReheapDown from the current node
   - Yes: ReheapDown from the current node

Min-max heap

Delete the largest

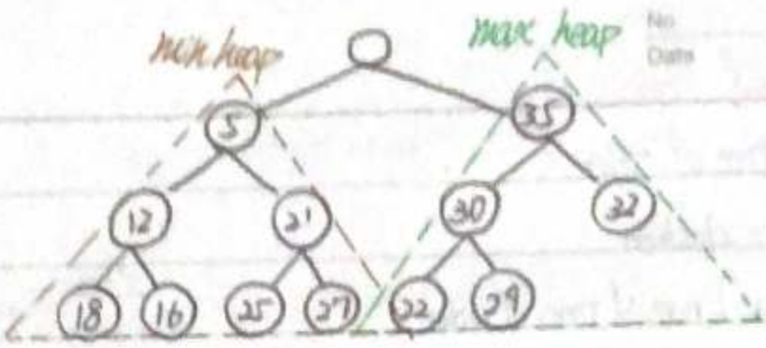
---

* Three 4-way trees : → max heap + min heap + max heap.

Main idea → each node in max-heap has it parent in min heap

Heap 堆積樹

- Insert any key
- Delete the smallest key
- Delete the largest key

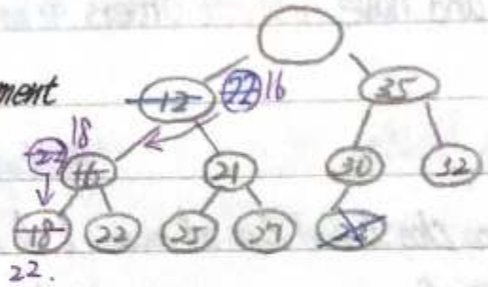


Heap : Insert

1. Examine the corresponding nodes : Left < Right
2. ReheapUp if necessary (recursion)

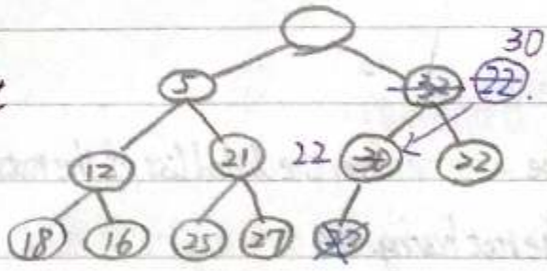
Heap : Delete the Smallest

1. Replace the root of (min)-heap with the last element
2. ReheapDown if necessary
3. Examine the corresponding nodes: Left < Right



Heap : Delete the Largest

1. Replace the root of (max)-heap with the last element
2. ReheapDown if necessary
3. Examine the corresponding nodes : Left < Right



Main idea

- Two heap:

  Pseudo root + min heap + max heap

  Each node in max-heap corresponds to one in min-heap

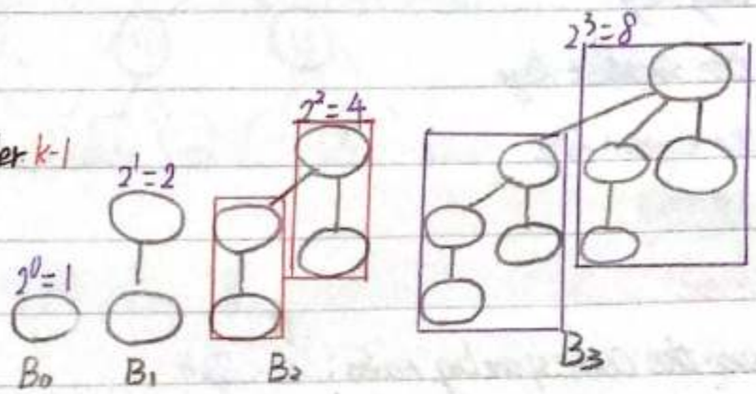
* Mergeable Priority Queues

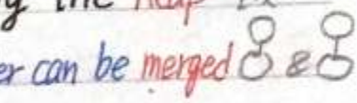
Two cooks → one cook // multiple servers : job queues

No.
Date

Binomial Tree

* Binomial tree of order k
- The root has k children
- Merged by two binomial trees of order k-1
- Number of nodes = 2^k
- Tree height = k+1 => O(logn)
- C_i^k nodes at level i, for i=0...k



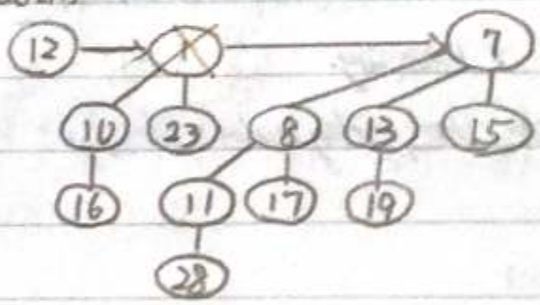
* A binomial heap is a collection of binomial trees that satisfy the heap property and have distinct orders => Two binomial trees of the same order can be merged Ex 

* Insert:
1. Insert into the linked list of roots
2. Call merge function

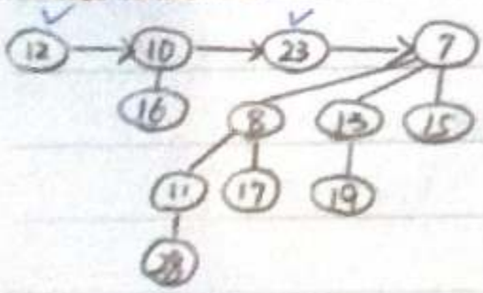
* Merge
1. A linked list sorted by the orders of binomial trees
2. Merge two binomial trees of the same orders [left to right]

* Delete 找到最小的
1. Find the minimum from the linked list of the roots
2. Delete the root having the minimum 刪除它
3. Add its children into the linked list
4. Call merge function

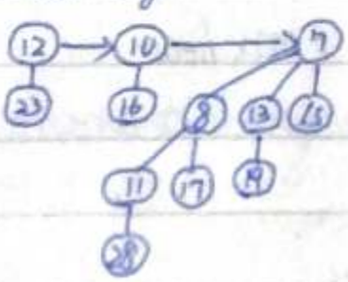
(a)
head[H]



(b) circular linked list



(c) Call merge



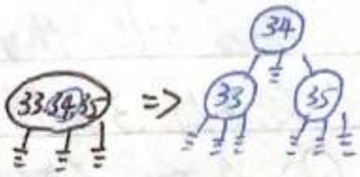
(d) O(logn)

單元3

*23樹 — 2-node & 3-node [幾個小孩]

=>Search: O(log2n)

Insert: a leaf may contain either one or two



*若node有兩個

=>先做排序 => 中間值往上提 => 左右各自處理

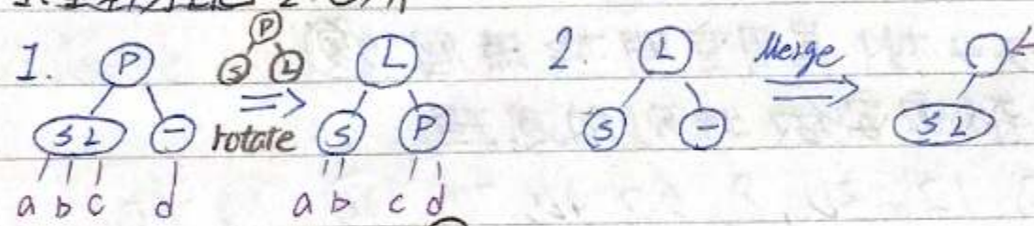
Step:

1. 找到樹葉

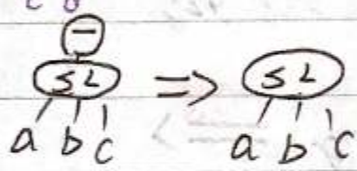
2. 放入 [若空放入] [滿 => 分裂] 樹根->新增, else 左方.勿.刻.

Delete():

1. 重新分配 2. 合併



*特殊刪到root



Step:

1 找到樹葉

2. 刪除[樹葉], 若非樹葉 => in-order successor

2 item 兄弟可借 (a) Redistribute

1 item 借不到 (b) Merge

3 如果還有item => done, else choose

Summary:

搜尋2-3 tree永會比binary search tree有效率, 就它有最小的樹高

∵ 2-3 tree的node have two values.

優點: 它保證平衡

2-3-4 樹

=> have 2-nodes, 3-nodes, 4-nodes.

* Are general tree, not binary trees

Never taller than 2-3 tree

效率比 2-3 樹 好

Q: 4 item 不知道怎麼分 => 看到 3 個就先分

*其它條件和 2-3 樹 差不多

Delete: => 看到有 1 item 先幫它合併

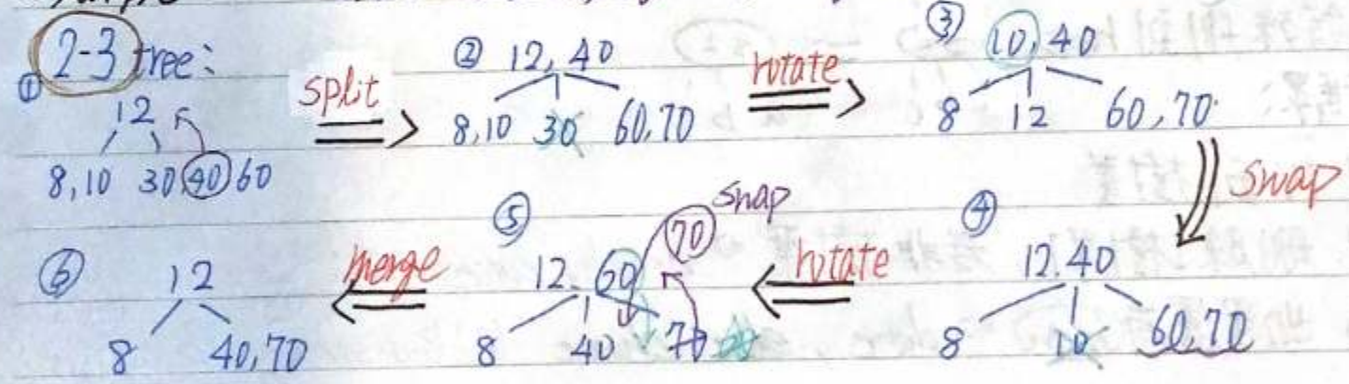
Summary:

2-3 樹 和 2-3-4 樹 是用空間換時間 (優)

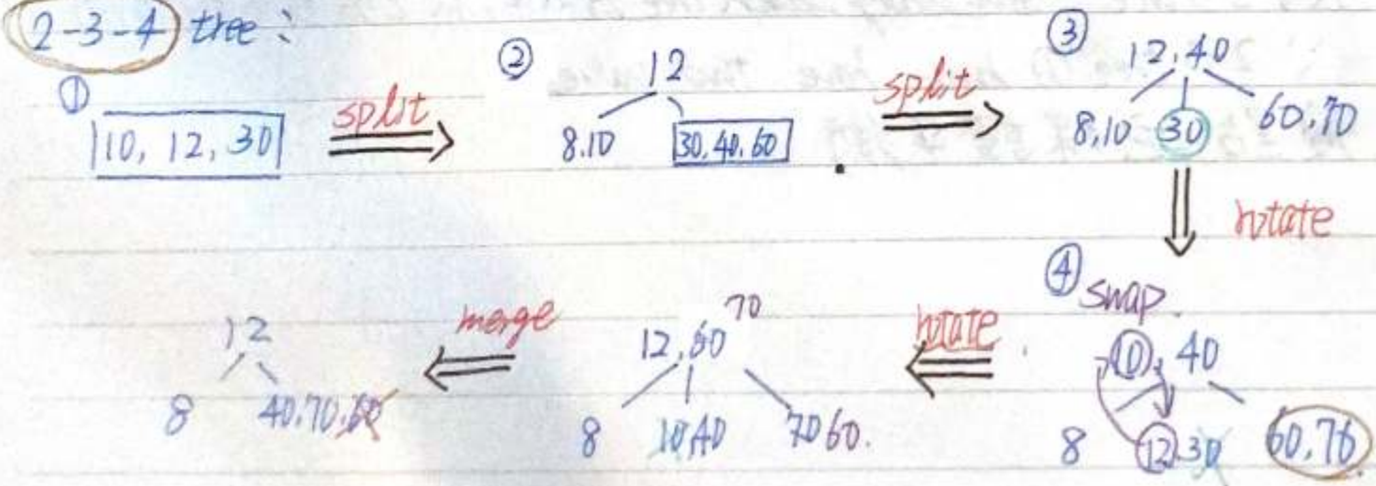
但 (空) or (滿) 都需要做特別的處理.

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

2-3 tree:



2-3-4 tree:



單元4

AVL Tree:

- A balanced binary search tree
- minimum-height

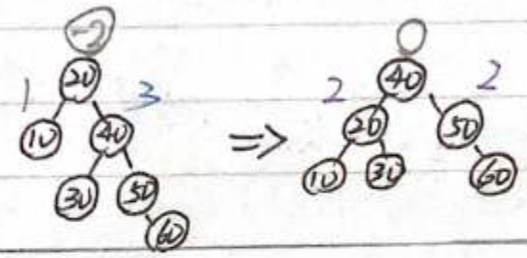
main:

1 check balanced

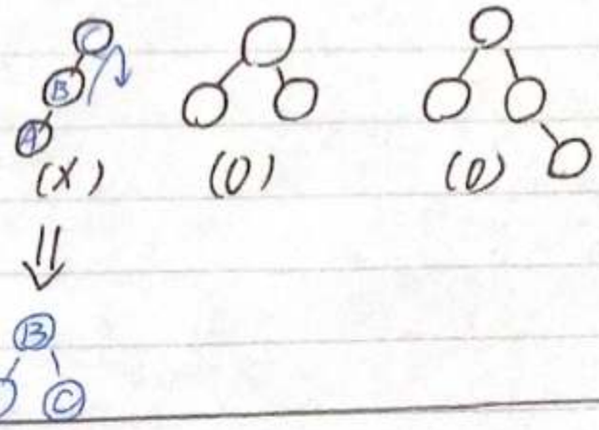
2 if unbalanced => rotate

Rotate:

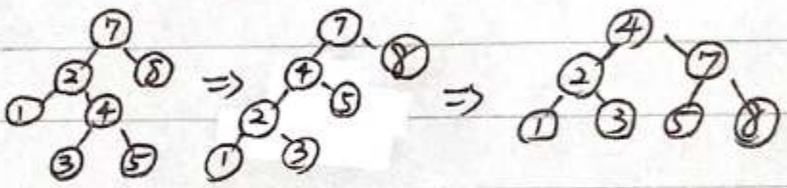
1 single rotation



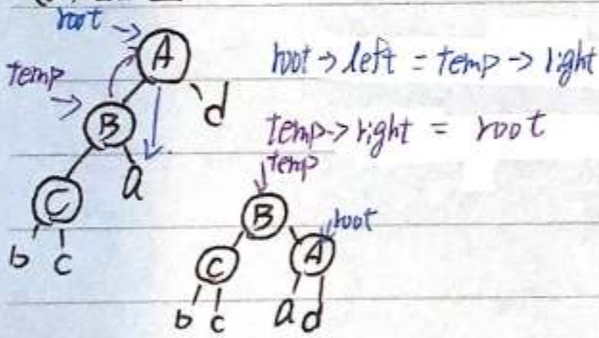
Balance?



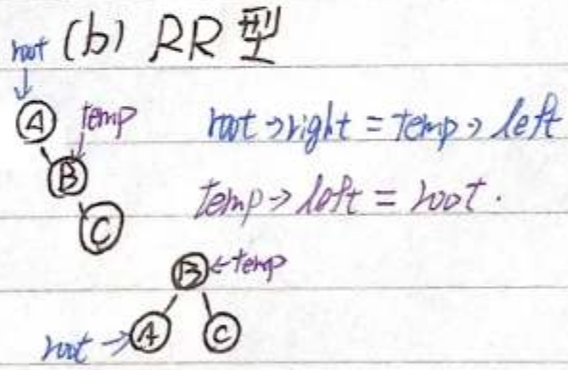
2. Double rotation



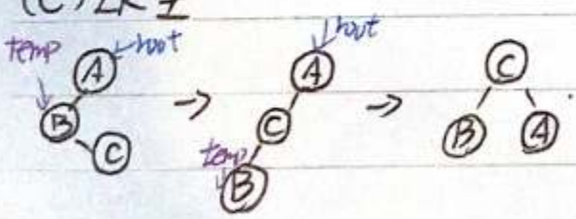
(a) LL型



(b) RR型



(C) LR型



(d) RL型