DS 單元1

Basic of Priority Queue

- Sorting Algorithm

	Worst case	Average case
Selection sort	n^2	n^2
Bubble sort	n^2	n^2
Insertion sort	n^2	n^2
Mergesort	$n\log n$	$n\log n$
Quicksort	n^2	$n\log n$
Radix sort	n	n

(P1, 5, 23:55), (P2, 5, 00:05) (P3, 3, 00:10) (P4, 4, 00:30)

↓ pqInsert(): O(1)

$\boxed{\text{Selection Sort: Unsorted list}}$

↓ pqDelete(): O(n)

(P3, 3, 00:10) (P4, 4, 00:30) (P1, 5, 23:55) (P2, 5, 00:05)

(P1, 5, 23:55) (P2, 5, 00:05), (P3, 3, 00:10) (P4, 4, 00:30)

↓ pqInsert(): O(n)

$\boxed{\text{InsertionSort: Sorted list}}$

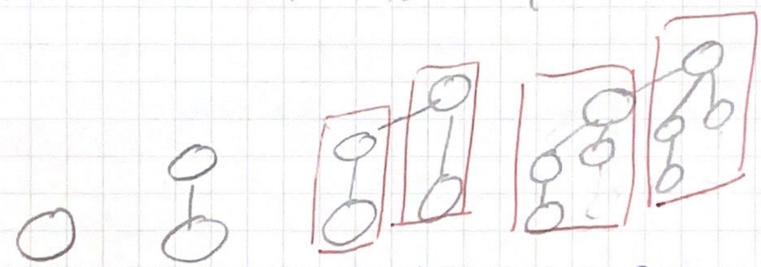
↓ pqDelete(): O(1)

(P3, 3, 00:10) (P4, 4, 00:30) (P1, 5, 23:55) (P2, 5, 00:05)

DS U2

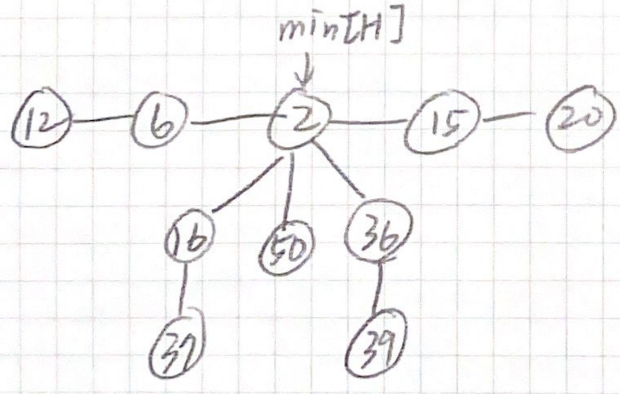
其他堆積變形

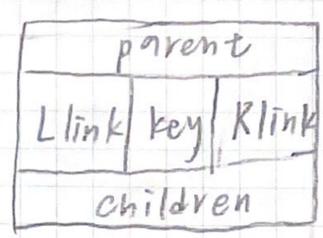
- Binomial tree of order K (B_K) 可合併的堆積結構
  - The root has K children
  - Merged by two binomial tree of order $K-1$
  - number of nodes $= 2^K$
  - tree height $= K+1 \to O(\log n)$
  - C^K_i nodes at level i, for $i = 0 \dots K$



B_0 $2^0=1$ B_1 $2^1=2$ B_2 $2^2=4$ B_3 $2^3=8$

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

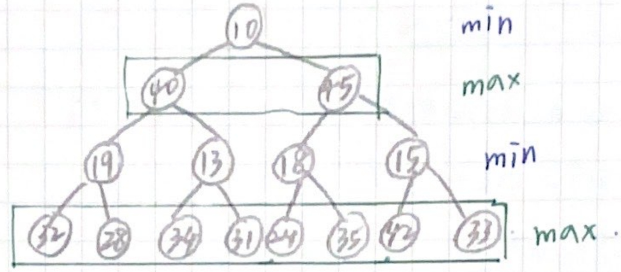




DS2 ex1.

雙向優先佇列 Double-ended Priority Queue

• Min-Max heap



1個 min heap 2個 max heap.

- Min-Max Heap: Insert

① 先判斷在奇數層 or 偶數層

↳ min ↳ max

② 判斷父節點有沒有比較小/大有的話 (min 小, max 大)

就要交換.(繼續往上升與祖父節點換)

③ 往上升與祖父節點比較

沒有的話

- Delete

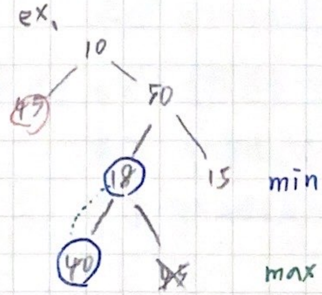
① 先跟小孩比(換/不換)

② 跟孫子比 繼續往下比

(如果停在樹葉(18)有可能會比max大!要比較) (在min層)

(min 小)

在max層



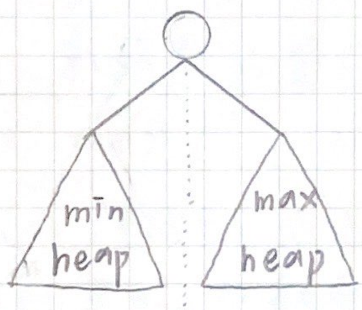
- 基本堆積

  - How to build a Heap?

```
struct HeapType {
  * void ReheapDown (int, int);
  * void ReheapUp (int, int);
    ItemType * elements;
    int numElements;
};
```

- Deap

Left < Right



  - Deap: Insert

① 先判斷加入的資料在左半邊 or 右半邊 / 算出間隔

→ 左半邊：(1) 與右半邊相應位子的父節點比較（要換(2) / 不換(3)）
(2) Reheap Maxheap 右半
(3) Reheap Minheap 左半

→ 右半邊：(1) 與左半邊相應位子比較（換(2) / 不換(3)）
(2) Reheap Minheap 左半
(3) Reheap Maxheap 右半

U3

- 2-3-4 tree
  - have 2 nodes, 3 nodes, 4 nodes.

    2-node : one data item two children.

    3-node : two data item three children

    4-node : tree data item four children.
  - general tree, $\boxed{\text{not}}$ binary trees
  - never taller than a 2-3 tree.
  - Search and traversal algorithms for a 2-3-4 tree are simple extensions of the corresponding algorithms for a 2-3 tree.

U4

- 紅黑樹
  - Represent each 3-node and 4-node in a 2-3-4 tree as an equivalent binary search tree.
  - A binary search tree to represent a 2-3-4 tree.
  - Has the advantages of a 2-3-4 tree, without the storage overhead.

DS2 ex2

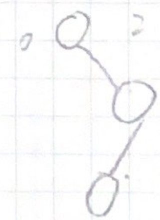
由上而下成長的平衡二元樹

- AVL
  - A balanced binary search tree.
  - can be search almost as efficiently as a minimum-height binary search tree.
  - Maintains the tree height close to the minimum
  - Balance Factor 平衡係數

    BF = h(left subtree) − h(right subtree)

- AVL Tree : Actions (Double Rotations)

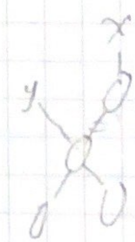
⊕ → 左大 ⊖ → 右大



⊕⊕ : LL ⊕⊖ : LR
10
⊖⊖ : RR ⊖⊕ : RL } Double rotation

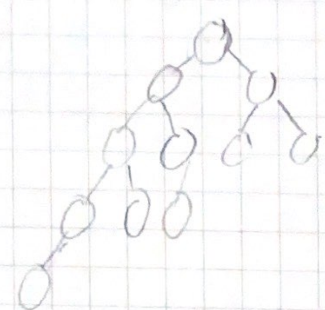
- AVL tree : Insert

① 先找到要加入的位子



② 往回比較左右subtree樹高

if(=+2) → 判斷新的資料在左or右
L
if左: LL()
if右: LR()



(=−2) → 〃
R
if左: RL()
if右: RR()

由下而上成長的平衡二元樹

- 2-3 tree

  - 完整樹

  - 2-3 tree : Insert

    ① 先找到要新增的位子

    若未滿2個 → 直接加進去並排序.

    滿了 → 就要判斷各稱的大小，把中間的往上提 → 新增一個temp存放.

    ② 之後往回判斷有沒有資料要新增到前面的節點.

    temp != NULL

    如果滿2個 → 就要把先前要新增的與這次要新增的接起來，再往回比.

    temp2

    temp

    沒滿 → 就把要加的temp樹根裡的資料加進去 → middle → temp → 左

    把 right → temp → 右

    其餘資料接起來!