# 樹 2

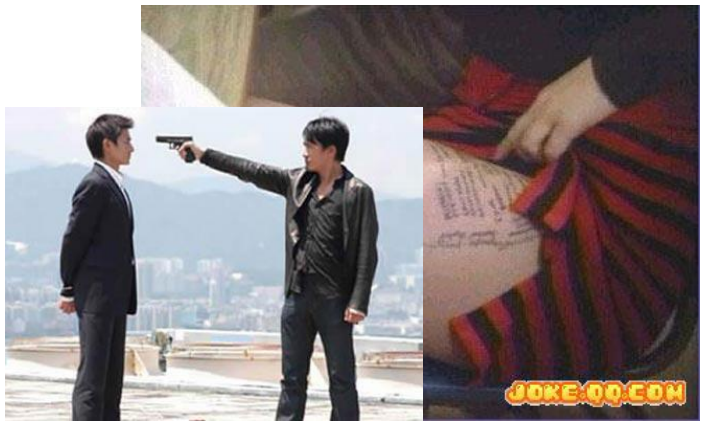
Michael Tsai

2013/3/26

# 期中考 (4/16)!!


- 我的想法:

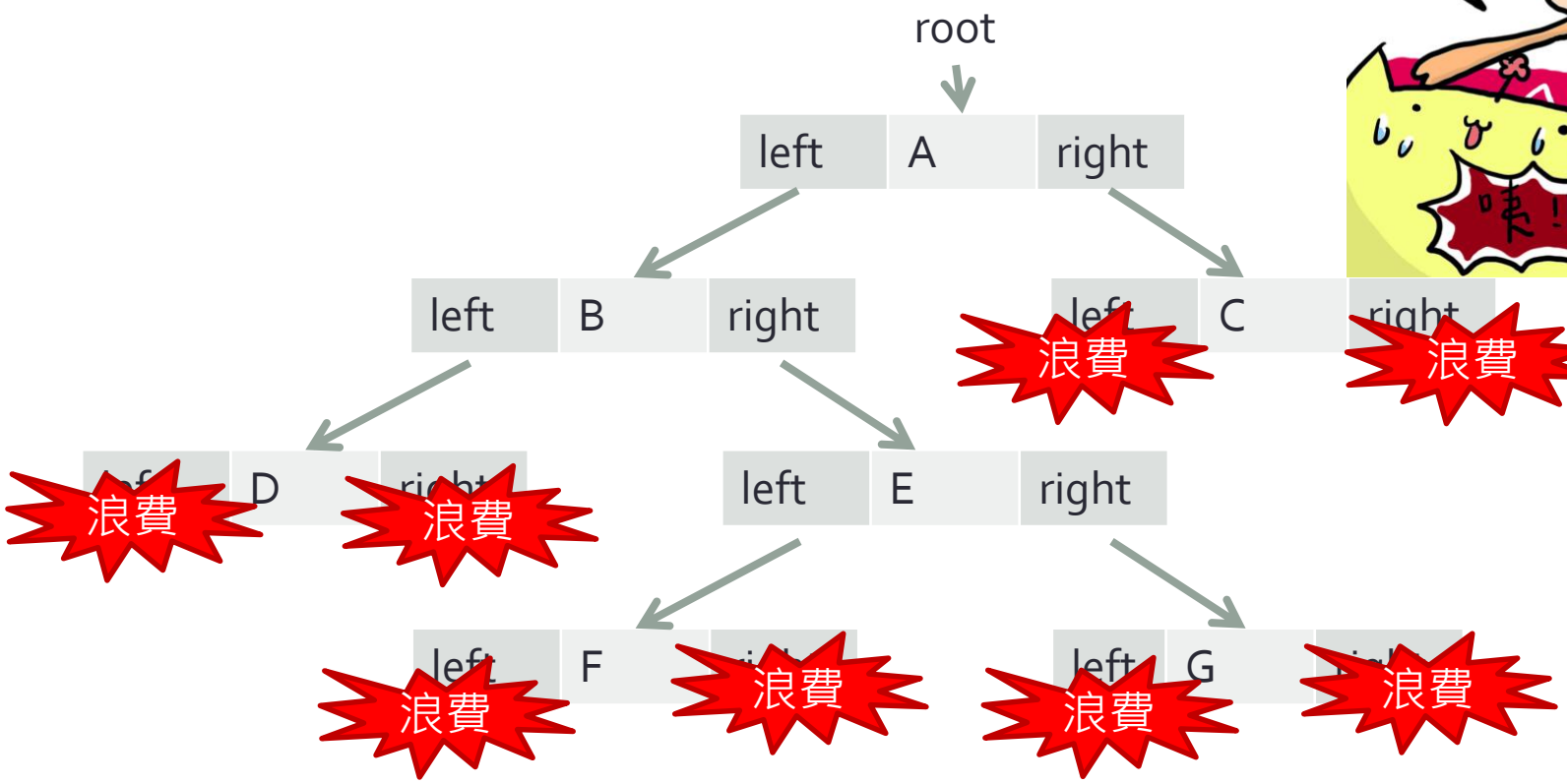
- 關書
- A4 大小一張, 雙面, 抄到你開心為止 (期末考沿用)
- 禁止使用放大鏡、顯微鏡 (供過小字體辨識用) XD
- 題目可能有
  - 是非題 (並解釋原因)
  - 填空題
  - 問答題(寫algorithm, 證明題, 問complexity)
- 請把答案寫清楚, 部分正確就有部分給分




- 作弊的直接砍頭 (當掉+送學校議處)

# 線頭樹??

http://www.wretch.cc/blog/z314159/7248666

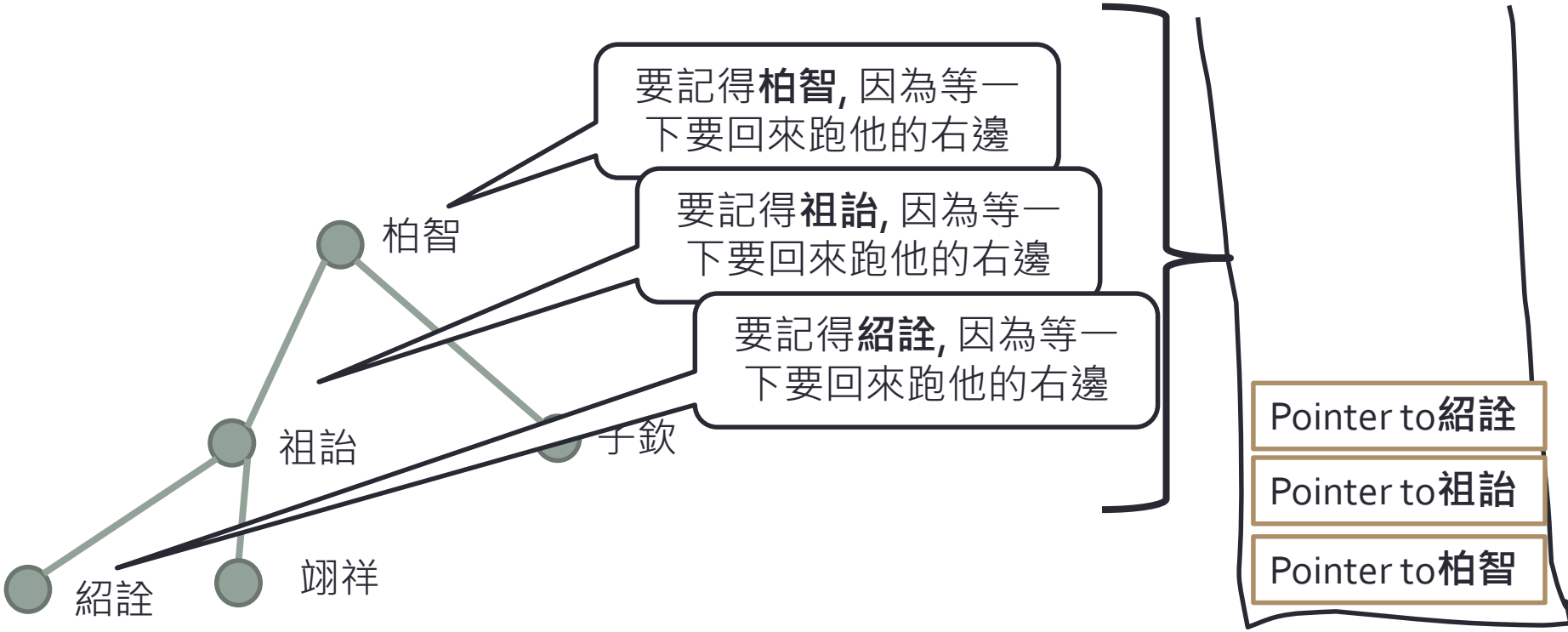




複習: 浪費了幾個pointer? (總共有n個node的話)

A: 總共有2n個pointer, (n-1)個edge/pointer不是NULL, 所以2n-(n-1)=n+1.

所以可以把這些NULL pointer欄位拿來做什麼?

# 怎麼做Inorder Traversal?




需要多少空間?

O(h)=O(n)

我們就拿NULL pointers來幫忙這件事!

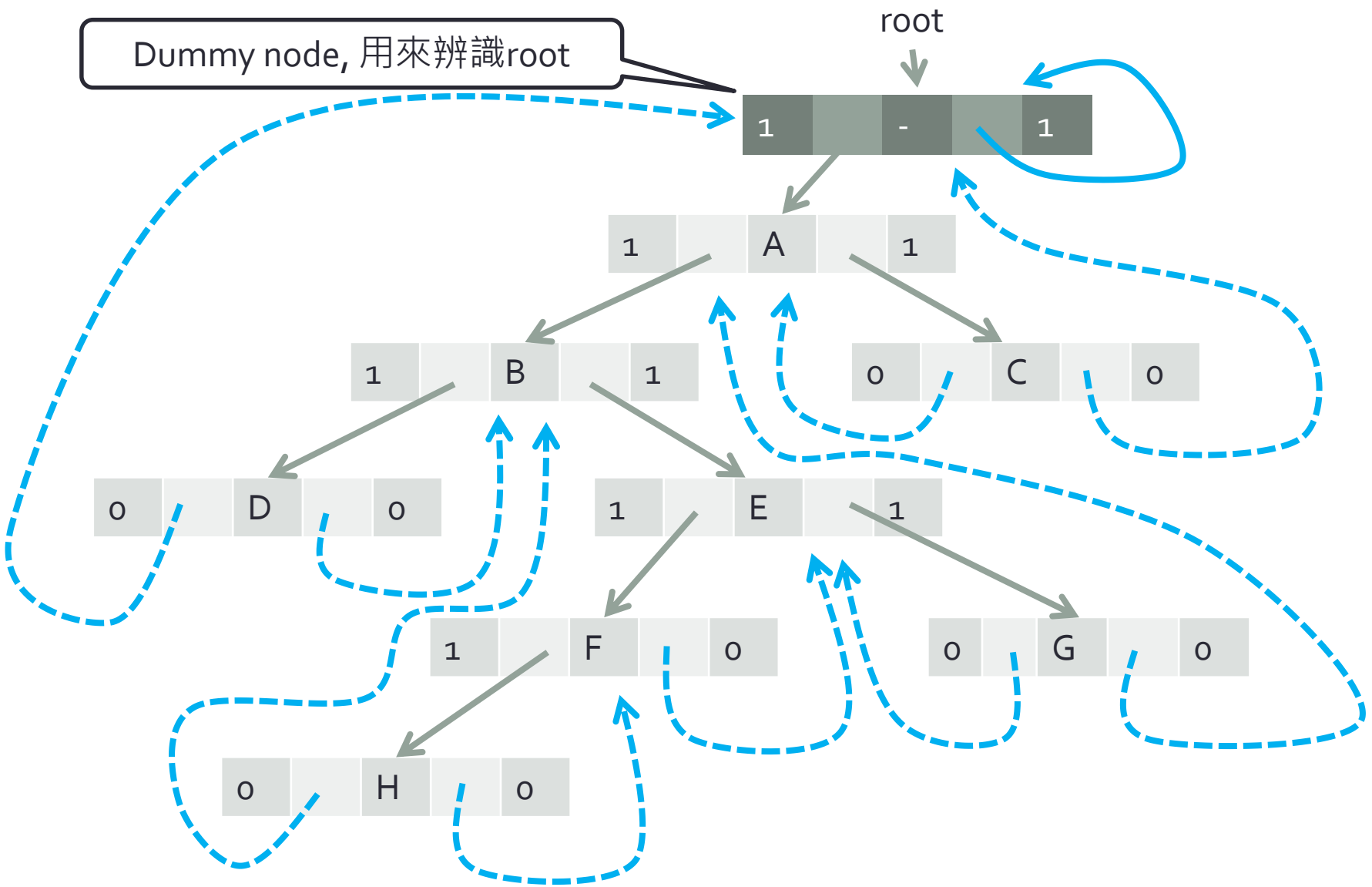
# 浪費掉的pointer們...

- 線頭樹: (Inorder) Threaded Binary Tree

- 1. 如果leftChild是null, 那就改成指到inorder traversal的前一個node (又稱為inorder predecessor) (此為**線頭**)
- 2. 如果rightChild是null,那就改成指到inorder traversal的後一個node (又稱為inorder successor) (此為**線頭**)
- 3. node的structure裡面放兩個額外的boolean欄位, 說明是link還是thread

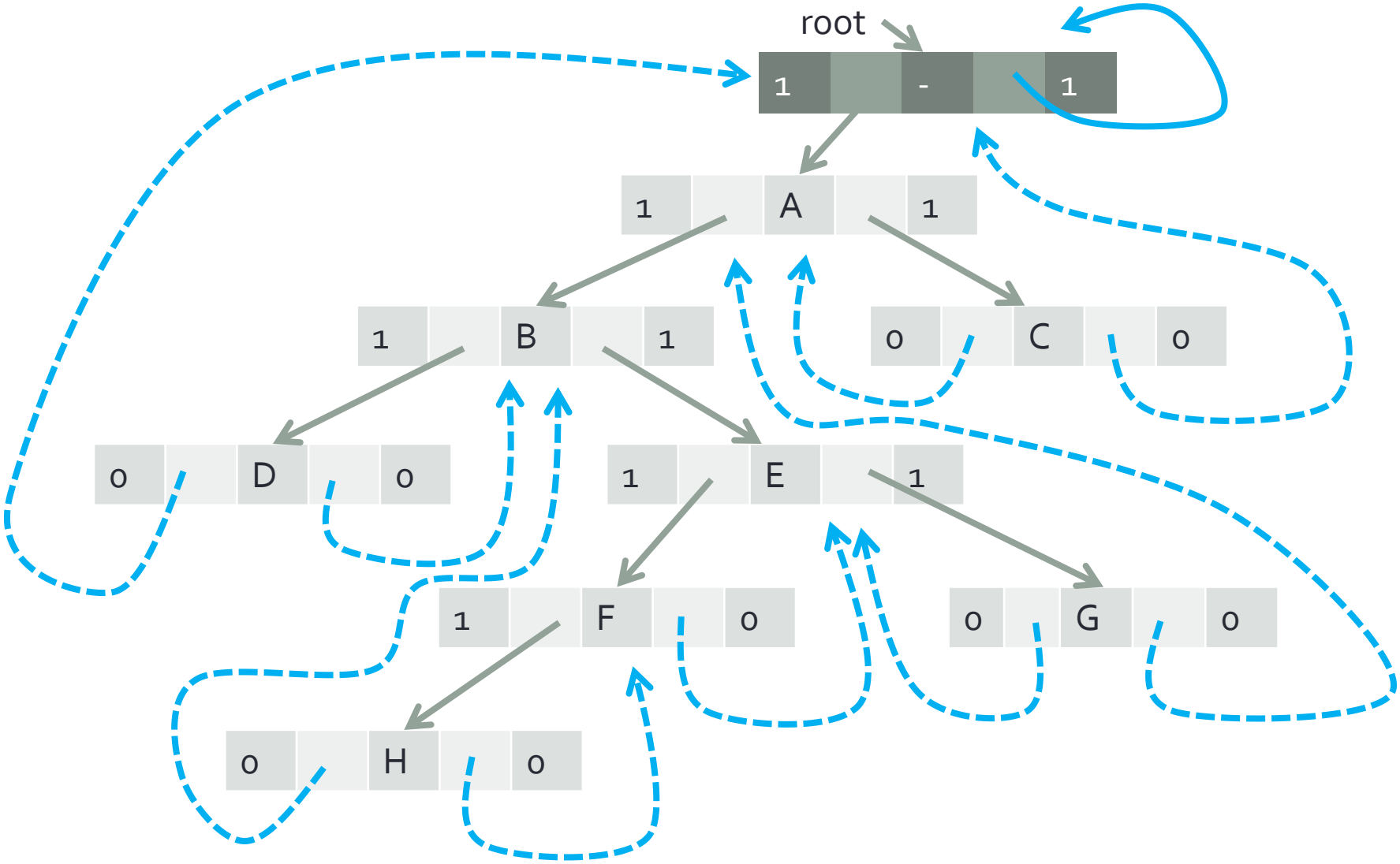
- 效果: 之後做inorder traversal不需要stack了! O(1)!

# Threaded binary tree長這樣



- Inorder Traversal?
- 怎麼找到inorder successor?

1. 如果右邊不是thread, 那就找rightChild的leftChild一直走到底
2. 如果右邊是thread, 那麼就是thread指到的地方



# Threaded Binary Tree

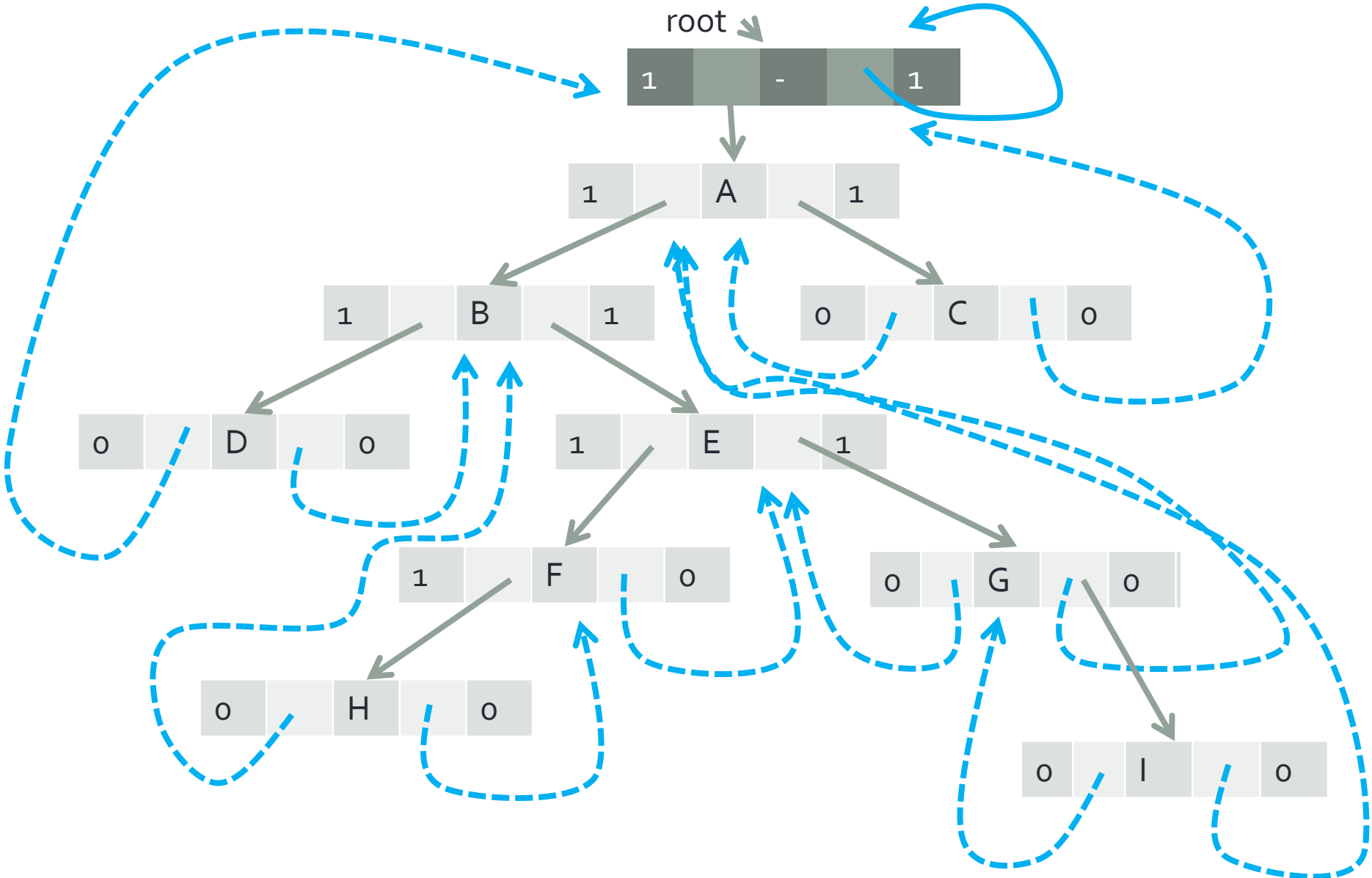
- 接著, 要做inorder traversal就很簡單了

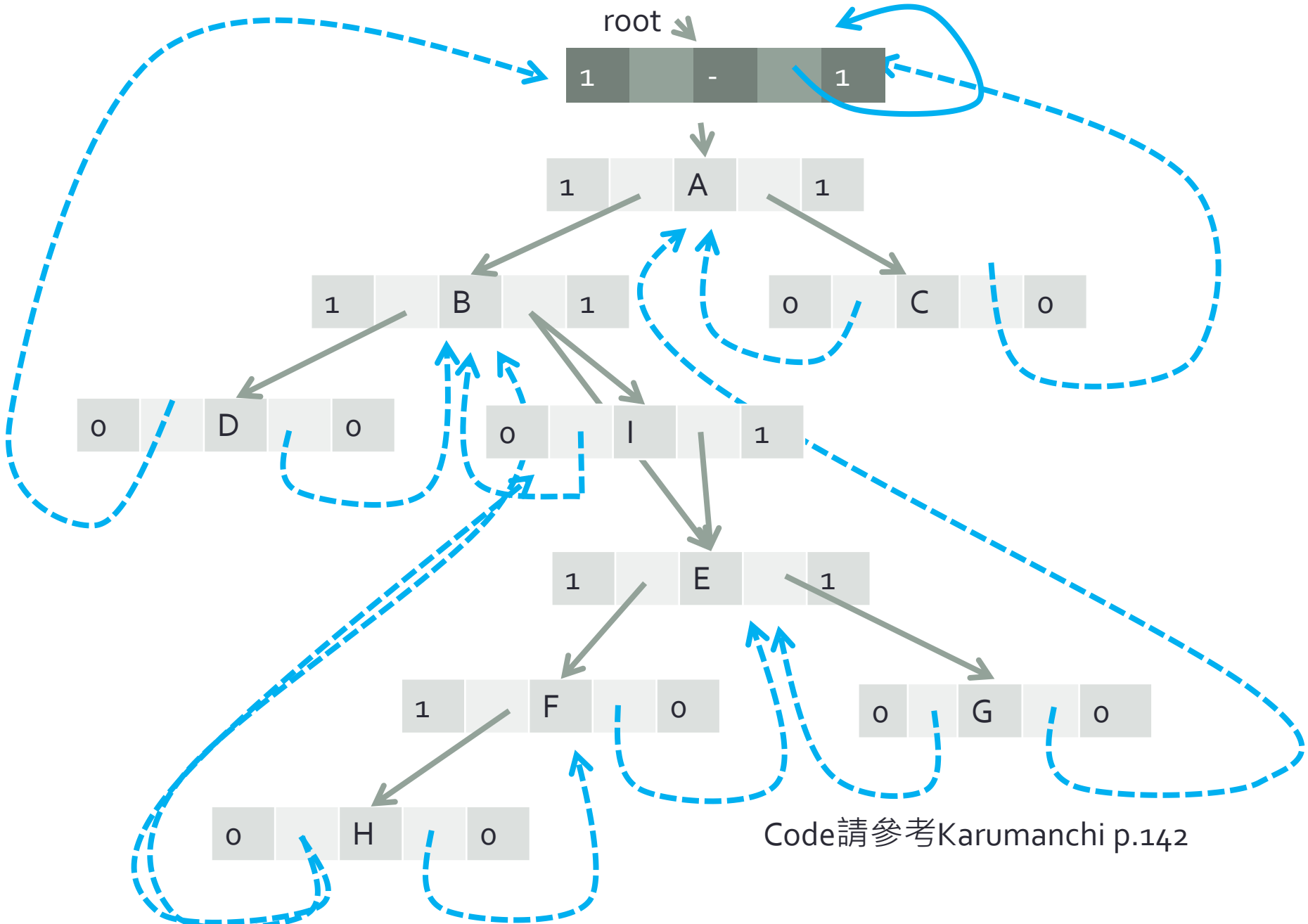
```
void InOrderTraversal(struct TreeNode *root) {
        struct TreeNode *temp=root;
        while(1){
                temp=inorderSuccessor(temp);
                if (temp==root) return;
                printf("%d", temp->data);
        }
}
```

Space Complexity: O(1)

- <動腦時間> 如果要用threaded binary tree做preorder traversal呢?
- 首先要先想想怎麼找到preorder successor
- Postorder仍然無法不使用stack來做traversal!
- Karumanchi p.141

# 在Threaded Binary Tree裡面加一個node






Code請參考Karumanchi p.142

# Priority Queue

- 一種每次都可以拿到priority最高的element的queue
- 直接來定義operations

- Insert(element) 把element放進queue裡面
- DeleteMax() 把element拿出來. 這個element有最高的priority

- (可以想像, 放進去的時候有做一些排序)

- 另外也有FindMax(), isEmpty(), isFull等等的operation
- 請同學想想看, 要怎麼用已經學過的東西來做priority queue?

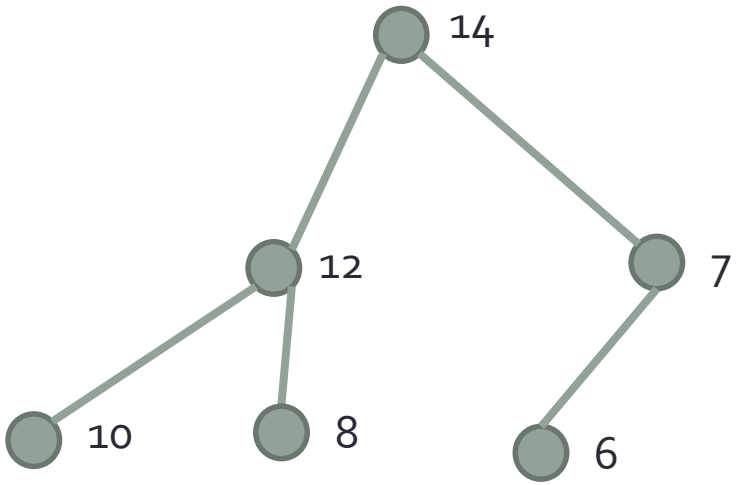
# 用以前學過的方法效果如何?

| | Insert | DeleteMax | FindMax |
|---|---|---|---|
| Unordered Array | 我是遮板 | | |
| Unordered Linked List | 我是遮板 | | |
| Ordered Array | 我是遮板 | | |
| Ordered Linked List | 我是遮板 | | |
| Binary Search Tree | 我是遮板 | | |
| Binary Heap | 我是遮板 | | |

# Heap

- Definition: A **max tree** is a tree in which the key value **in each node is no smaller (larger) than the key values in its children** (if any).
- Definition: A **max heap** is a **complete binary tree** that is also a **max tree**. A **min heap** is a **complete binary tree** that is also a **min tree**.

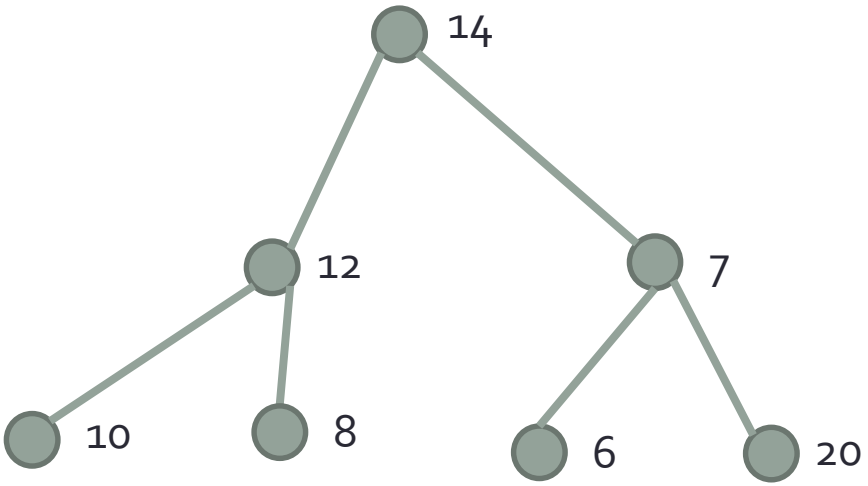
- 有了heap, 我們要怎麼用它來做priority queue?
- Root是不是永遠都是最大?



# Insert一個element到Heap

- 加入的時候每次都能夠繼續維持是一個max heap
- 怎麼加?
- 1. 既然是complete binary tree, 所以一定要加在下一個該出現的地方, 把新的element放在那邊.
- 2. 循序往root的方向移動, 一直到不違反parent > child的規則為止
- Time complexity?

$O(\log n)$



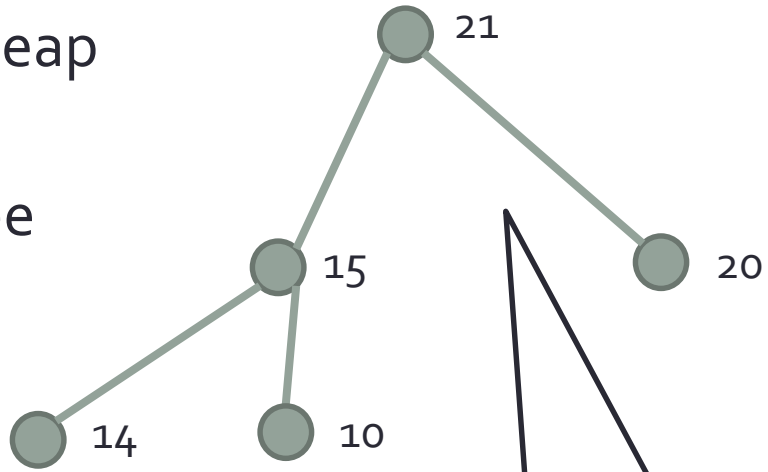

需要和6比嗎?
不用! 因為原本6的parent就會比它大了! 因此20只會更大!

# 從Heap DeleteMax一個element

- 從root拿走一個element (最大的)
- 調整位置, 繼續維持是一個max heap

- 1. 首先既然是complete binary tree
  拿掉的位置就沒有別的選擇.

- 2. 把拿掉的位置的element,
  拿到root的地方. 和child中比較大
  的比較. 如果比其小則與其交換.
  重複以上步驟直到不再違反
  parent > child的規則為止.
- Time complexity?




需要跟15和20都比!
因為如果拿到不是最大的
child,
則可能違反heap原則!

$O(\log n)$

# 用什麼data structure來implement heap?

- Array? 比較簡單? 偷懶?
- 可以. 因為每次都只會加在最後面 (complete binary tree)

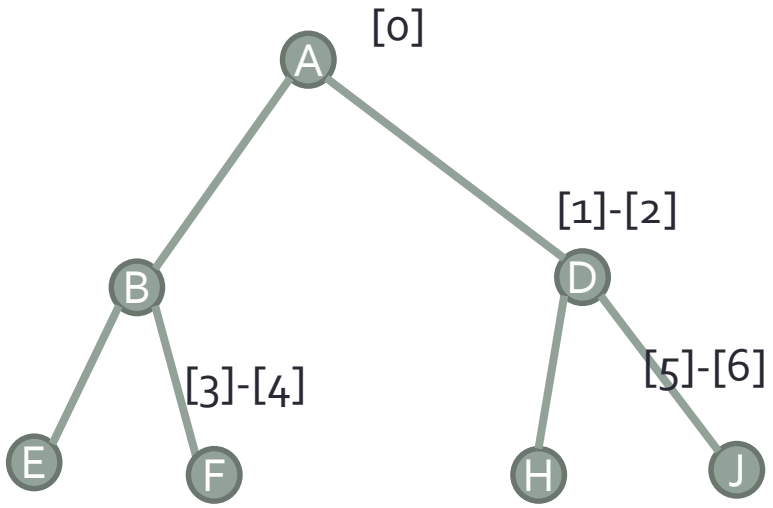
- 複習 "怎麼在記憶體裡面記一棵樹呢? Array法", Binary Tree 版

# 怎麼在記憶體裡面記一棵樹呢? Array法

- Binary Tree
  (每個node最多有2個children)
- 某個node的parent?
- 觀察:
  Index為i的node, 其parent之index為
  $\lfloor (i-1)/2 \rfloor$

- 怎麼找某個node的children
- 觀察:
  Index為i的node, 其children之index為
  $2i+1 \sim 2i+2$

- 這樣要找parent或是child都非常方便!



# 討論時間

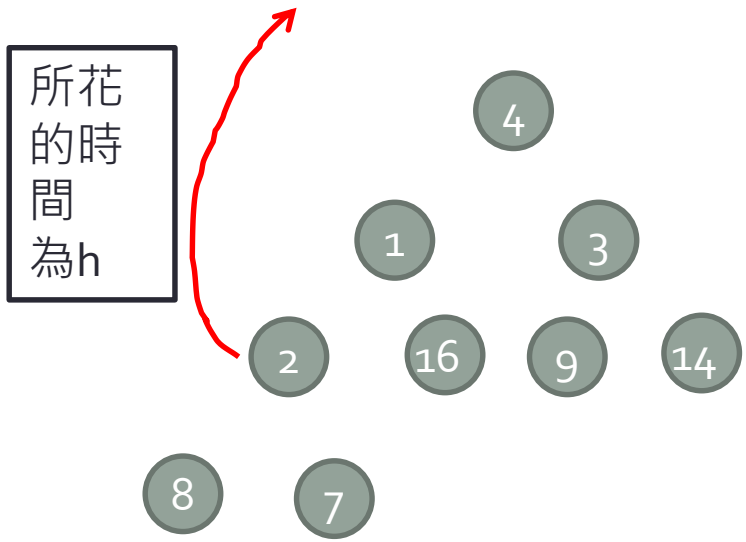
- 給一個沒有處理過的array, 怎麼用最少的時間把它變成heap?

- 概念:
  - 原本的array可以看成一個還沒排好的binary tree(還不是heap)
  - Leaf的部分不用處理 (因為它們沒有children, 不會違反heap原則)
  - 從最後一個(index最大的)非leaf node開始處理 (跟下面的比)
  - Time complexity=O(n) !

# How to “heapify” an array?

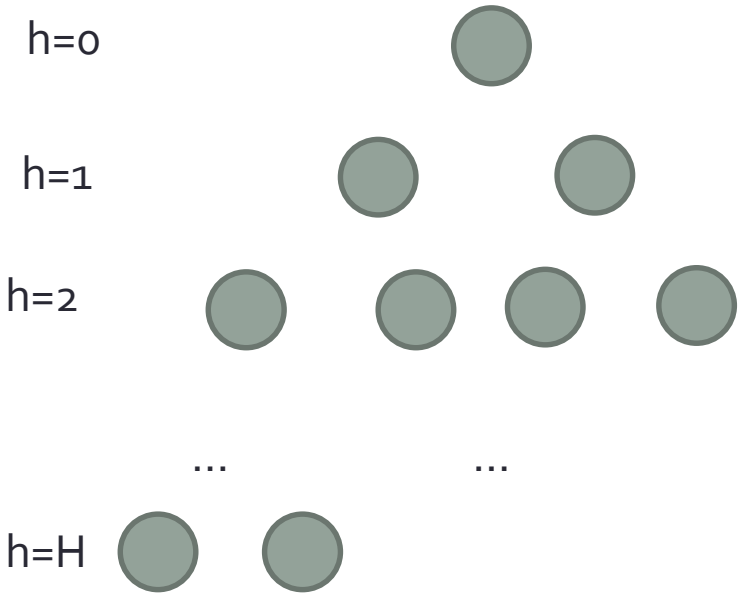
方法一: 每次Insert一個element進去heap.

| 4 | 1 | 3 | 2 | 16 | 9 | 14 | 8 | 7 |
|---|---|---|---|---|---|---|---|---|



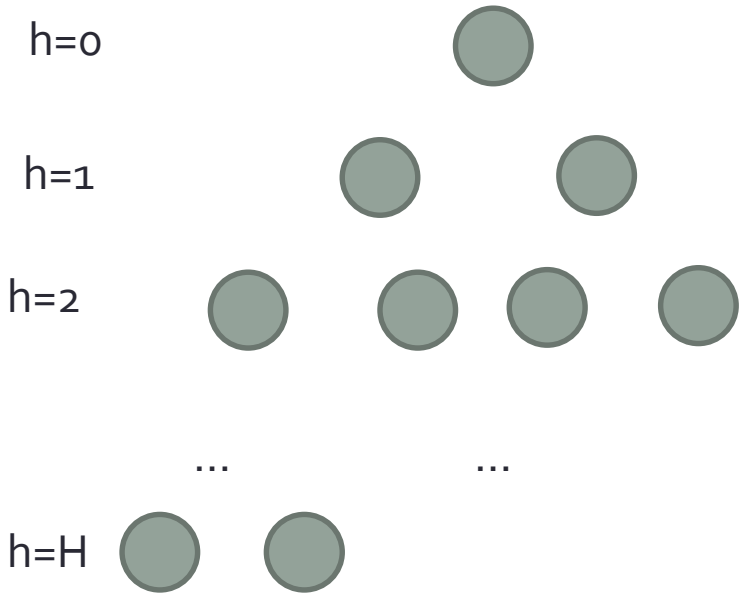

- 這樣所花的時間複雜度是多少?
- 每新insert一個element進去heap, 所花的時間最多為O(h)=O(log n)
- h: 當時的高度
- n: 當時的element數目
- 假設最後總共有N個element,
- 總時間複雜度會是O(N log N)嗎?

# 方法一：時間複雜度分析




- 則所花時間為:
- $2^1 + 2 \times 2^2 + 3 \times 2^3 + \cdots + H \times 2^H$
- $= \sum_{i=1}^{H} 2^i + \sum_{i=2}^{H} 2^i + \cdots + \sum_{i=H}^{H} 2^i$
- $= \sum_{j=1}^{H} \sum_{i=j}^{H} 2^i$
- $= \sum_{j=1}^{H} 2^{j-1} \sum_{i=j}^{H} 2^{i-j+1}$
- $= \sum_{j=1}^{H} 2^{j-1} \left( \sum_{i=1}^{H-j+1} 2^i \right)$
- $= \sum_{j=1}^{H} 2^{j-1} \cdot 2 \left( 2^{H-j+1} - 1 \right)$
- $= \sum_{j=1}^{H} 2^{H+1} - 2^j$
- $= H(2^{H+1}) - 2(2^H - 1)$
- $= (H-1)2^{H+1} + 2$

# 方法一：時間複雜度分析

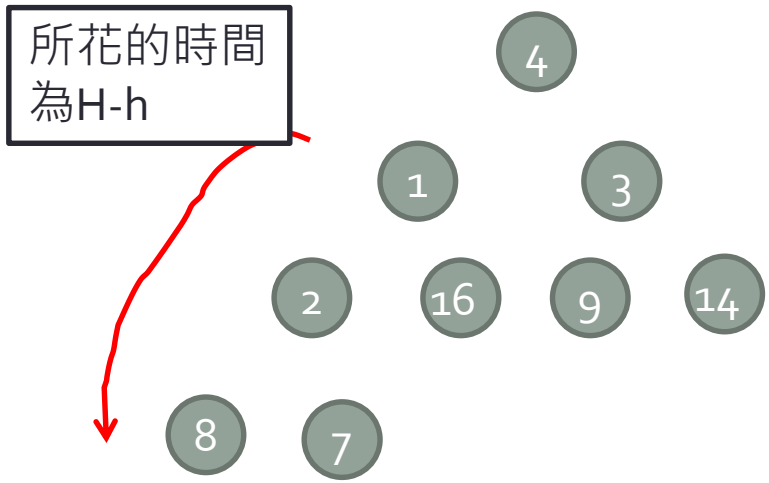



- 時間為: $(H-1)2^{H+1}+2$
- 又H為heap高度, N為element數目, 則兩者可有下列關係:
- $H=\lceil \log_2(N+1) \rceil - 1$ (檢查看看對否?)
- 因此所花時間以N表示為:
- $(\lceil \log_2(N+1) \rceil - 2)2^{\lceil \log_2(N+1) \rceil}+2$
- $\leq (\log_2(N)+1)\,2^{(\log_2(N)+1)}+2$
- $=(\log_2(N)+1)(2N)+2$
- $=O(N\log N)$

# How to “heapify” an array?

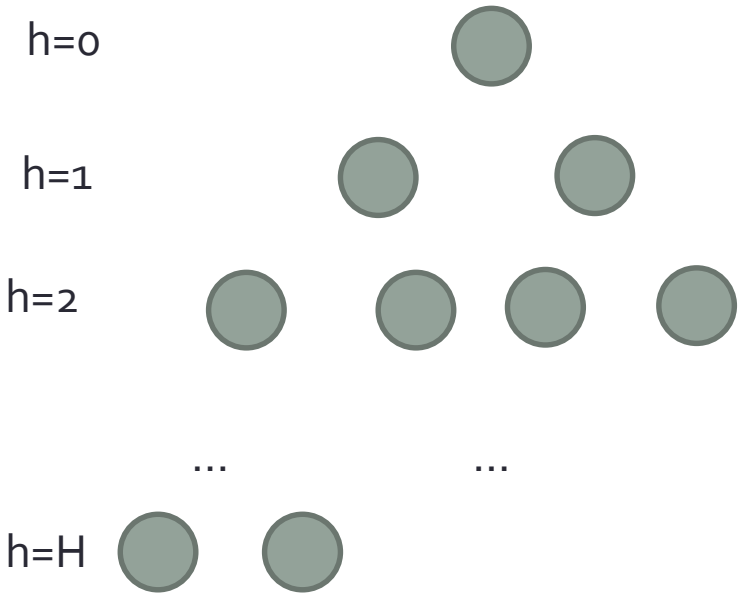
方法二: 從最後一個element開始往前，每次組成一個以該node為root的小heap

| 4 | 1 | 3 | 2 | 16 | 9 | 14 | 8 | 7 |
|---|---|---|---|---|---|---|---|---|




- Leaves不用看 (下面沒有東西, 一定是heap)
- 要怎麼找到第一個非leaf的node?
- 從第一個非leaf的node開始檢查, 往下檢查/移動到符合heap性質為止.
- 每處理完一個node, 那個node為root的sub-tree會變成一個heap
- 等到第一個node(root)也處理好的時候，整個binary tree就變成heap了
- 這樣的話, 時間複雜度會變好嗎?

# 方法二：時間複雜度分析



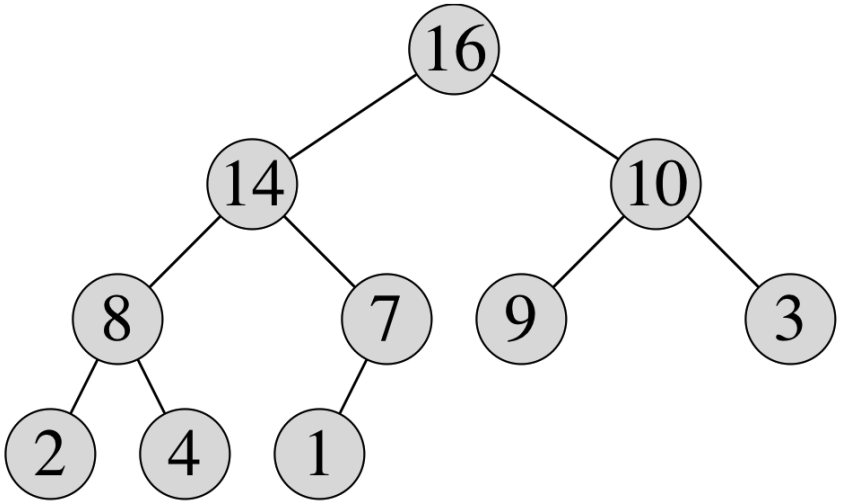

- 所花的時間為:
- $H + 2(H-1) + 2^2(H-2) + \cdots + 2^{H-1} \cdot 1$
- $= \sum_{i=0}^{H} 2^i (H-i)$
- Let $S = \sum_{i=0}^{H} 2^i (H-i)$.
- $2S = 2H + 4(H-1) + \cdots + 2^H$
- $2S - S = -H + 2 + 4 + \cdots + 2^H$
- $S = 2^{H+1} - H - 2$
- 又$H = \lceil \log_2(N+1) \rceil - 1$
- $2^{\lceil \log_2(N+1) \rceil} - \lceil \log_2(N+1) \rceil - 3$
- $\leq 2N - \log_2(N+1) - 3$
- $= O(N)$

# Heapsort：利用heap來排序

- 如何利用heap排序?

1. 先用剛剛的heapify方法把整個array變成heap. $O(N)$
2. 每次從heap拿出一個最大的, (和尾巴交換), 然後把原本尾巴的element依序往下檢查/交換直到符合heap性質.
3. 重複步驟2一直到heap變成空的.

$O(N \log N)$

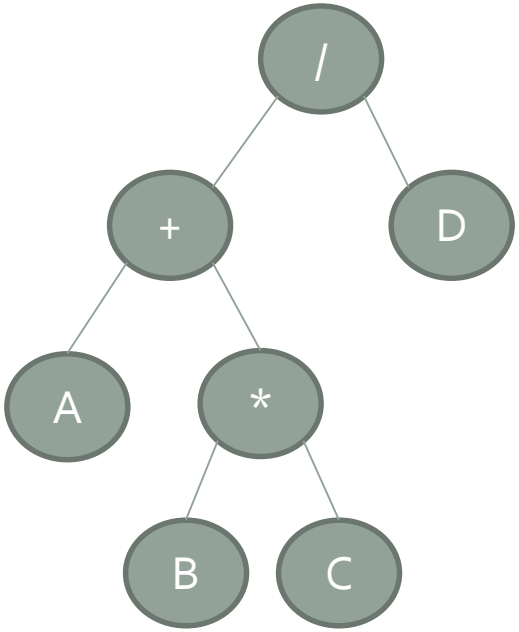



Total: $O(N \log N)$

請同學試試看! (參考Cormen p.161 Figure 6.4)

# Expression Tree

- 用一棵樹來代表expression
- Leaf nodes: operand
- Non-leaf nodes:  operator
- (A+B*C)/D 如何以expression tree表示?




- Traversal的方法可以對應到不同的expression表示法:
  - Preorder → prefix
  - Inorder → infix
  - Postorder → postfix

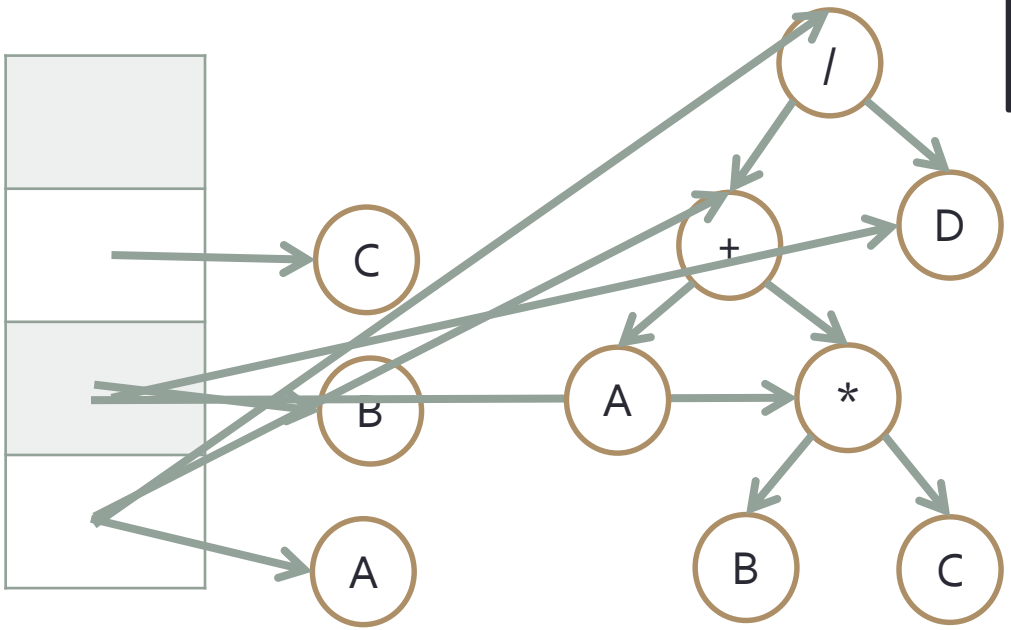
- 因此expression tree建好以後可以:
  - 轉換不同的expression表示法
  - 計算結果(Boolean或一般數學式)
  - Evaluate satisfiability of a boolean expression

- 使用標準traversal方法: code非常簡單!

# 建立Expression Tree

- 假設給的expression為postorder
- 例: ABC*+D/

Stack which can hold **pointers to a node**




最後在stack裡面的一個entry就是我們要的expression tree!

# 計算邏輯運算式

- 變數可為True or False
- 三種operator: $\neg\ (not)$,$\wedge\ (and)$,$\vee\ (or)$
- 可以使用括號
- 例如 $(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_3) \vee \neg x_3$

1. 如何計算當$(x_1, x_2, x_3) = (T, T, F)$時的結果?
2. 進階題: 如何找出所有組合使得結果為true?

# 計算邏輯運算式

$(x_1, x_2, x_3) = (\mathrm{T}, \mathrm{T}, \mathrm{F})$

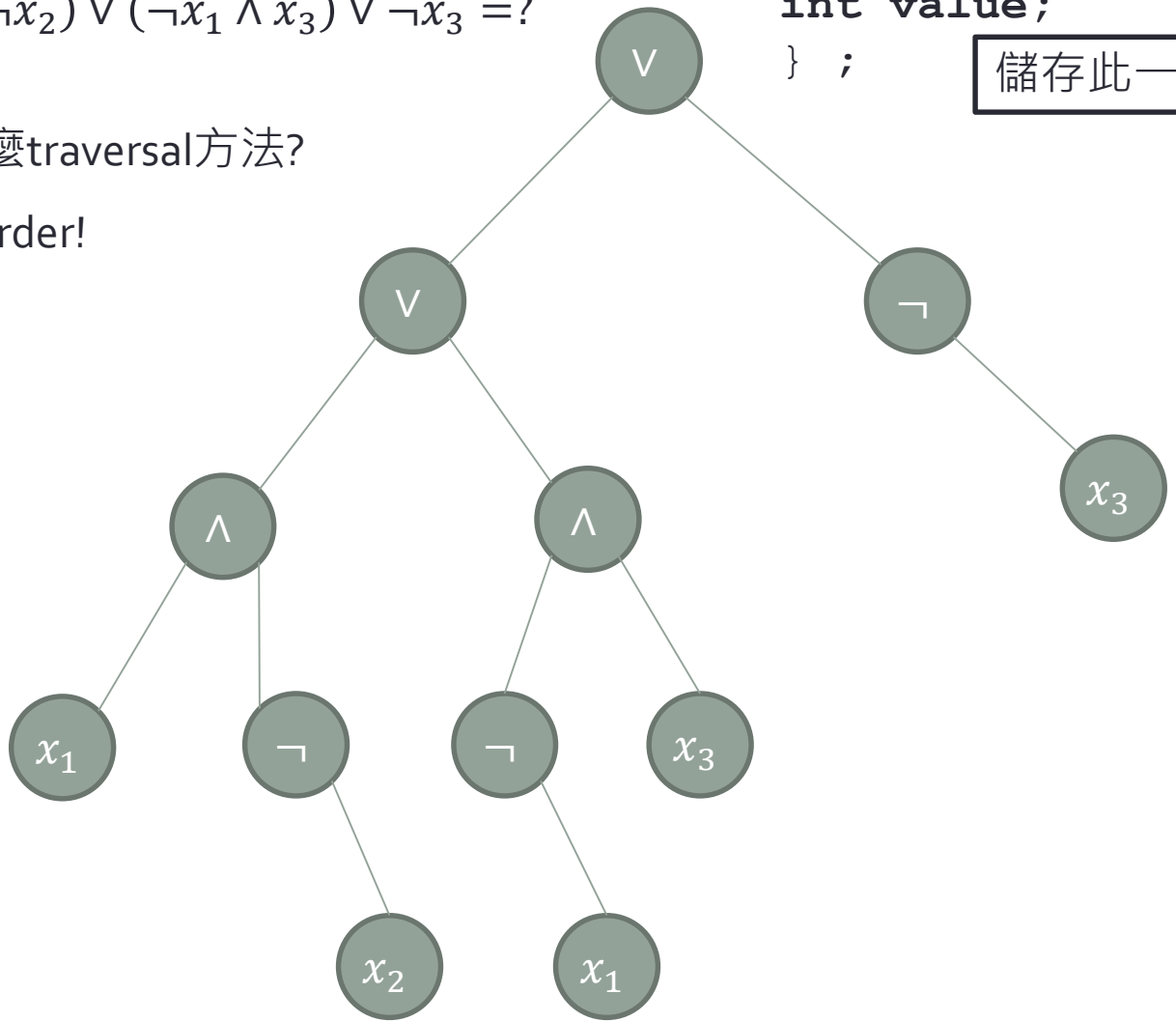
$(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_3) \vee \neg x_3 = ?$

用什麼traversal方法?

Postorder!

```
typedef struct TreeNode {
struct TreeNode *left, *right;
int data;
//either operator or operand
int value;
} ;
```

儲存此一subtree的計算結果



# 邏輯運算式: Satisfiability Problem

- Satisifiability: 是否可以被滿足 → 有沒有一組truth assignment 可以使得最後boolean expression的結果為true
- 如何evaluate satisfiability of a boolean expression?

```
for (all 2^n possible combinations) {
      generate the next combination;
      replace the variables by their values;
      evaluate root by traversing it in postorder;
      if (root->value) {
            printf(<combination>);
            return;
      }
}
printf("No satisfiable combination\n");
```

Pseudo-code: 不是真的程式碼, 但是每一個步驟(可用文字表示)夠詳細, 足以解釋如何執行.

# Today's Reading Assignments

- Karumanchi 6.11, Problem [49-52],59
- Cormen ch 6
- Karumanchi 7.1-7.6, Problem 7,12,13