# 樹 3

Michael Tsai

2012/3/27

# 前情提要：“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?

- Reading Assignment: p. 182 “Heapifying the Array”

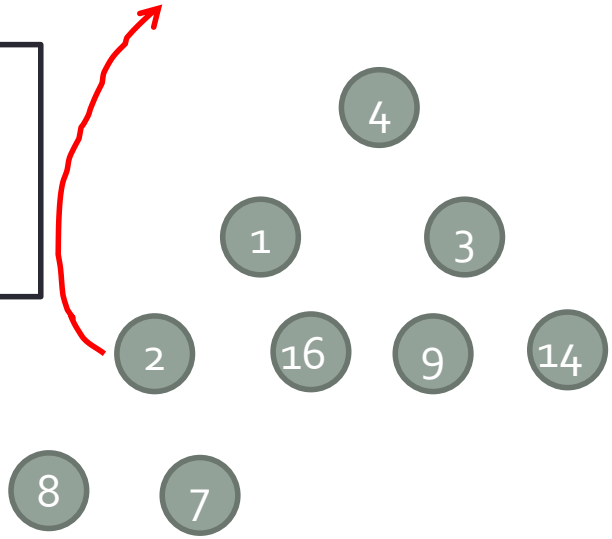
- 概念:
  - 原本的array可以看成一個還沒排好的binary tree(還不是heap)
  - Leaf的部分不用處理 (因為它們沒有children, 不會違反heap原則)
  - 從最後一個(index最大的)非leaf node開始處理 (跟下面的比)
  - Time complexity=O(N log N)

# How to “heapify” an array?

方法一: 每次Insert一個element進去heap.

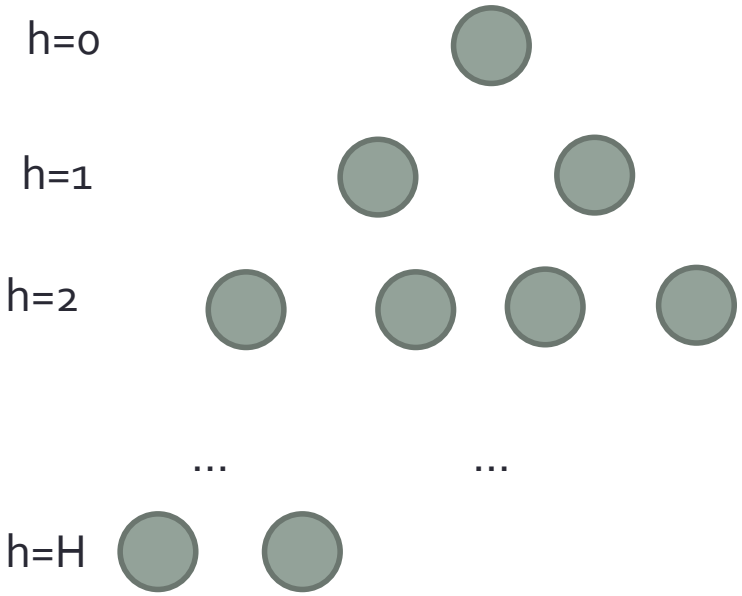
| 4 | 1 | 3 | 2 | 16 | 9 | 10 | 14 | 8 | 7 |
|---|---|---|---|---|---|---|---|---|---|

所花的時間為h



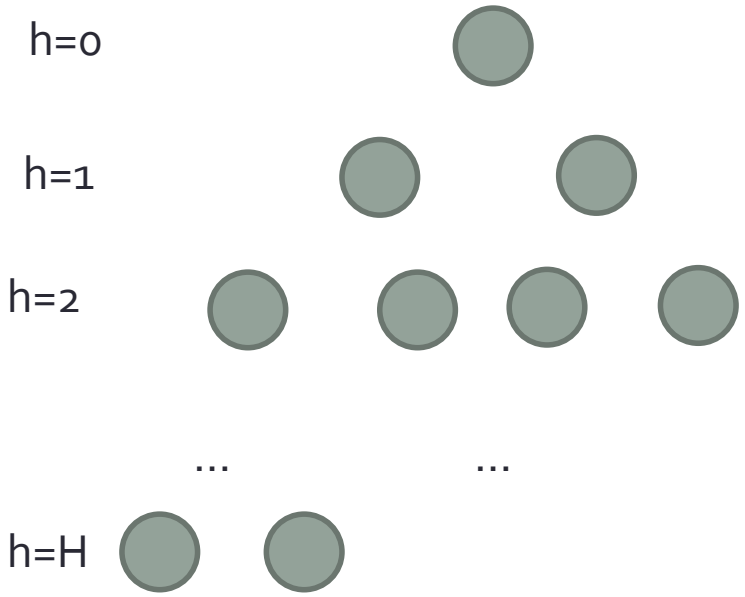

- 這樣所花的時間複雜度是多少?
- 每新insert一個element進去heap, 所花的時間最多為O(h)=O(log n)
- h: 當時的高度
- n: 當時的element數目
- 假設最後總共有N個element,
- 總時間複雜度會是O(N log N)嗎?

# 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

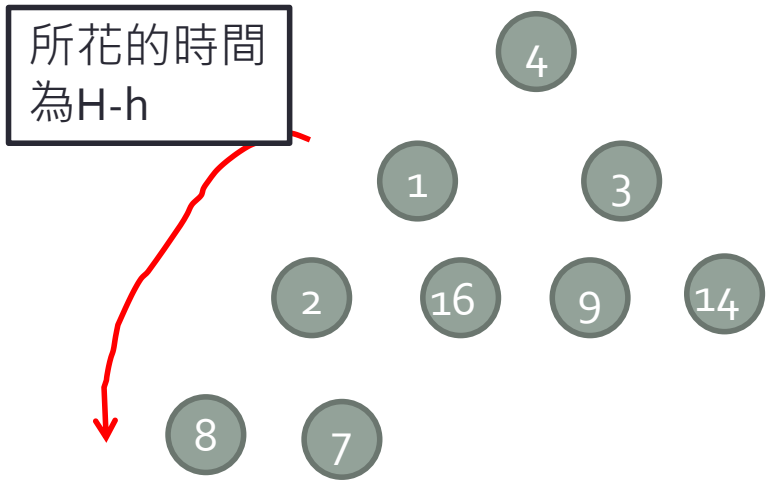



- 時間為: $(H-1)2^{H+1}+2$
- 又H為heap高度, N為element數目, 則兩者可有下列關係:
- $H=\lfloor \log_2(N+1) \rfloor - 1$ (檢查看看對否?)
- 因此所花時間以N表示為:
- $(\lfloor \log_2(N+1) \rfloor - 2)2^{\lfloor \log_2(N+1) \rfloor}+2$
- $\leq \log_2(N+1)\, 2^{\log_2(N+1)}+2$
- $=(N+1)\log_2(N+1)+2$
- $=O(N\log N)$

# How to “heapify” an array?

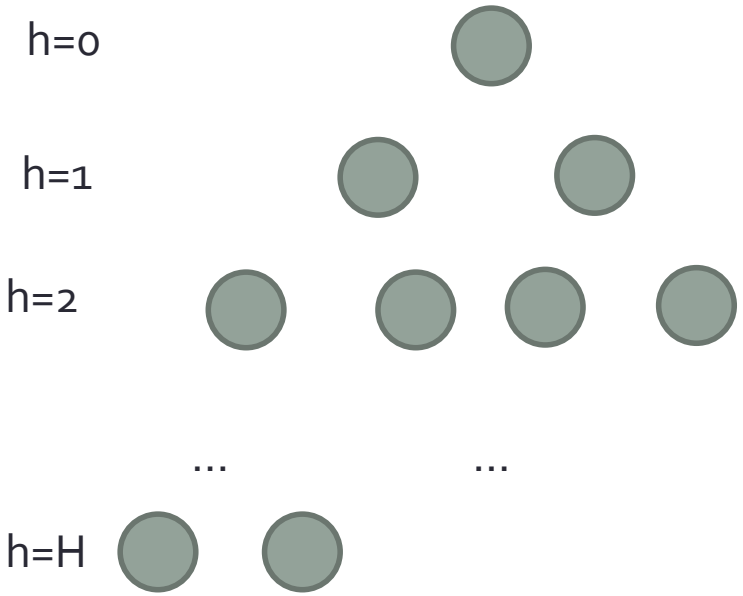
方法二: 從最後一個element開始往前，每次組成一個以該node為root的小heap

| 4 | 1 | 3 | 2 | 16 | 9 | 10 | 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

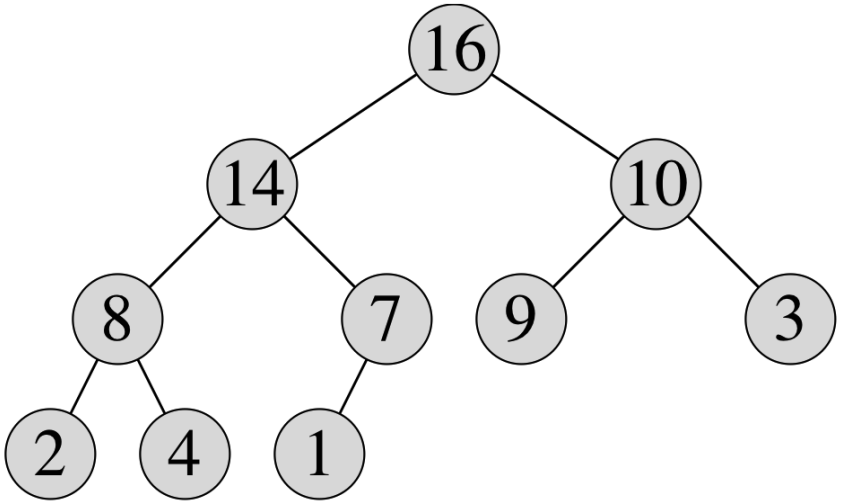



- 所花的時間為:
- $H + 2(H-1) + 2^2(H-2) + \cdots + 2^{H-1} \cdot 1$
- $= \sum_{i=0}^{H} 2^i (H-i)$
- Let $S = \sum_{i=0}^{H} 2^i (H-i)$.
- $2S = 2H + 4(H-1) + \cdots + 2^H$
- $2S - S = -H + 2 + 4 + \cdots + 2^H$
- $S = 2^{H+1} - H - 2$
- 又$H = \lfloor \log_2(N+1) \rfloor - 1$
- $2^{\lfloor \log_2(N+1) \rfloor}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

# 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. $O(N \log N)$

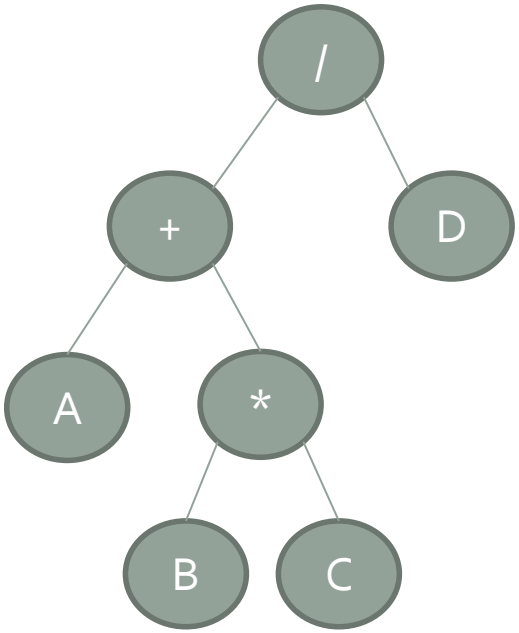



Total: $O(N \log N)$

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

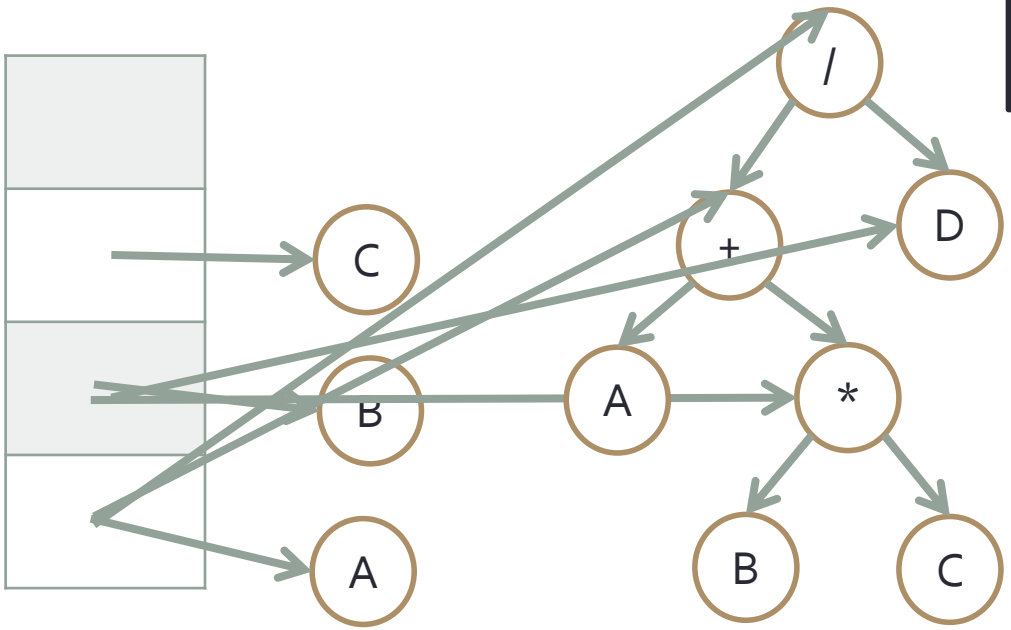
- 因此expression tree建好以後可以:
  - 轉換不同的expression表示法
  - 計算結果(Boolean或一般數學式)
  - Evaluate satisfiability of a boolean expression

- 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: $\neg\ (and)$,$\wedge\ (or)$,$\vee\ (and)$
- 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

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

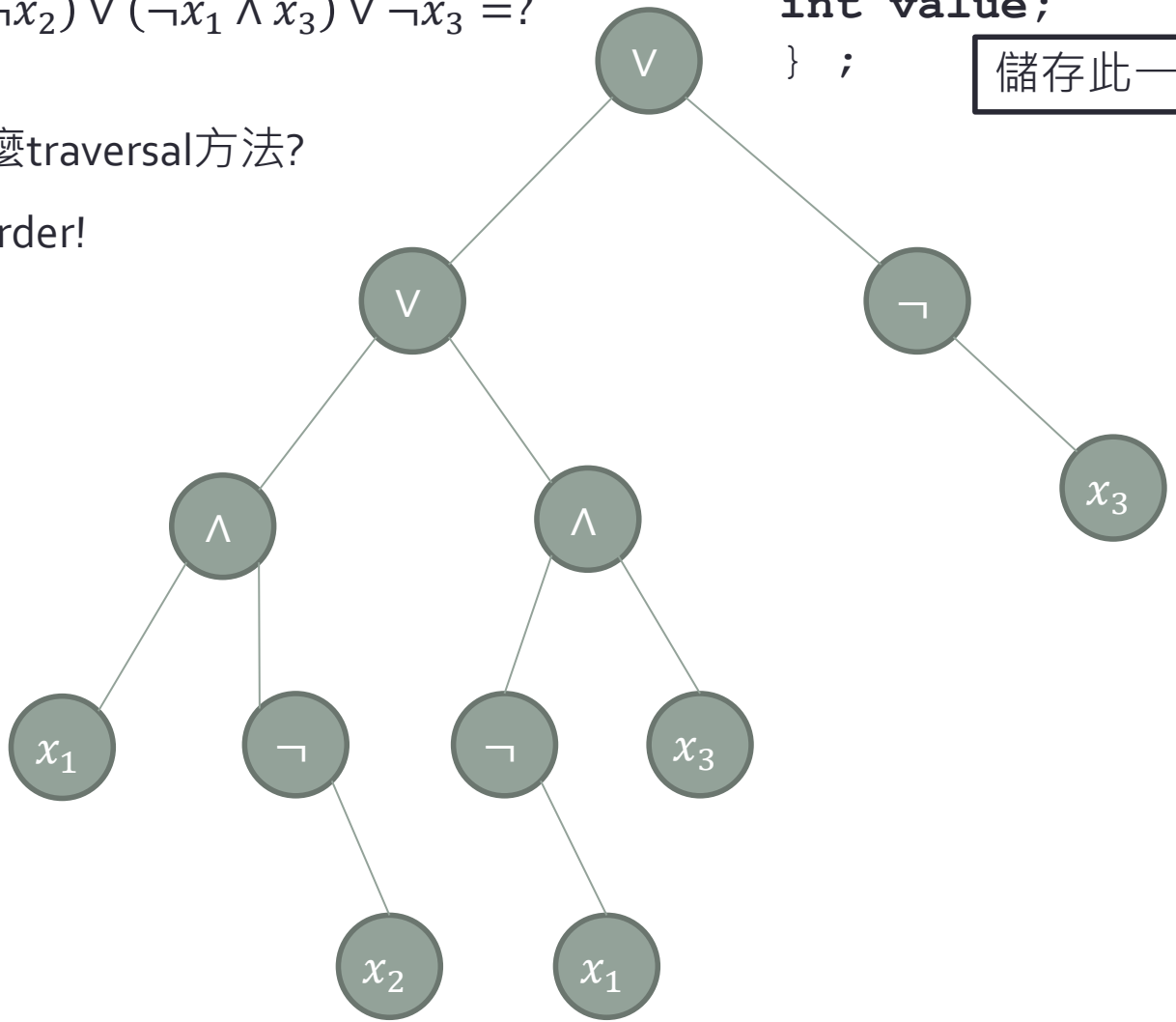
$(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_3) \vee \neg x_3 = ?$

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.