前言：

Binary Heap 是一種資料結構，其最大的用途在於，當演算過程需要多次抽出資料中的最小或最大值時，可以在 O(logn) 複雜度完成。以下將簡介 Binary Heap 的特性、操作以及使用場合。

Binary Heap 的特性：

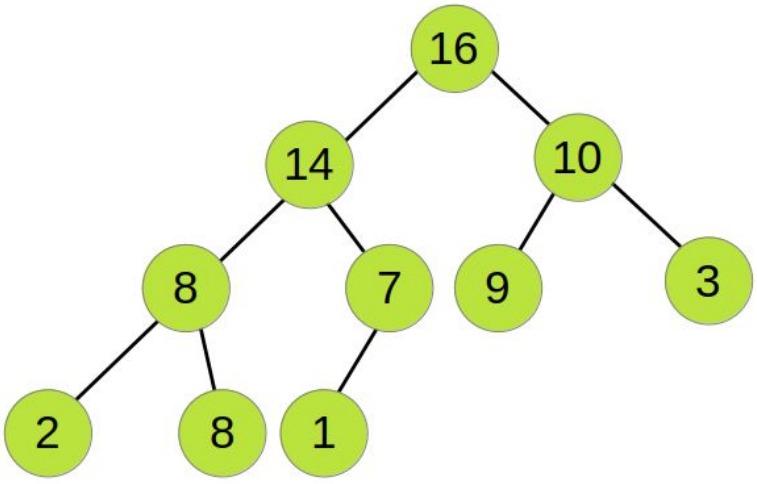
Binary Heap 是一種二元樹的結構，以下為其特性：

1. 一個 node 至多左右兩個子節點。
2. Max/Min Heap 父節點的值一定大於/小於子節點。所以根節點為樹的最大/最小值。
3. 必須為完整的樹結構(也就是說，每一層由左至右依序填滿，才換下一層)

因完整結構的特性， Binary Heap 在實做上通常使用陣列。樹跟陣列間的轉換關係：

1. 根節點的索引為 1。
2. 父節點的索引為 i：
   a. 左子節點的索引為 2i。
   b. 右子節點的索引為 2i+1。

以下為 Max Binary Heap 其樹結構及陣列表達的範例。



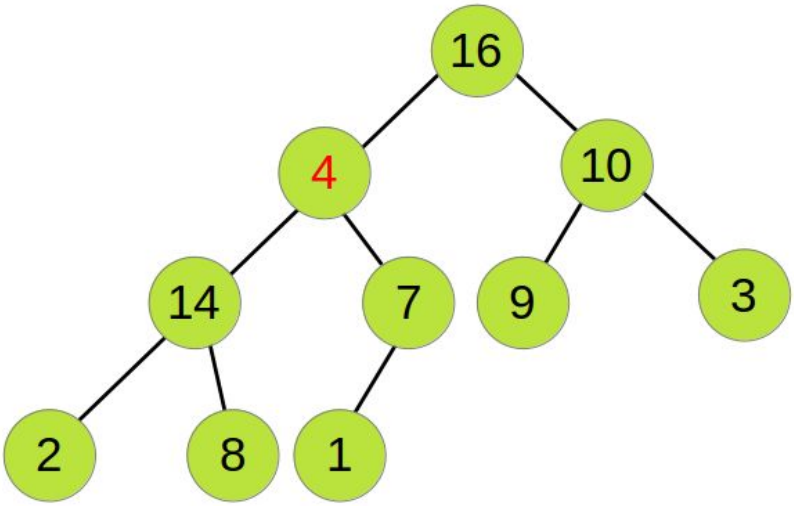
i =	1	2	3	4	5	6	7	8	9	10
	16	14	10	8	7	9	3	2	8	1

Binary Heap 的操作：

1. Build Max/Min Heap → 將無序的輸入資料化為 Max/Min Heap

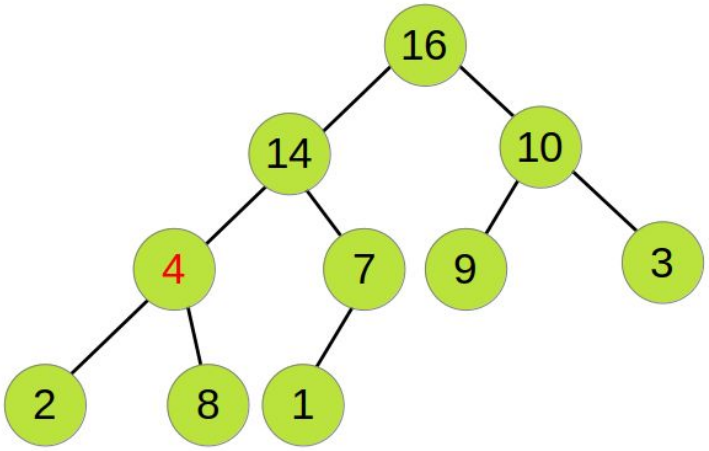
Heap 生成的方式為 Bottom-up，由索引最大的子樹父節點開始，往上使每個子樹，皆符合 Heap 結構，這個動作稱為 Heapify。我們藉由以下的範例說明。

假設在未排序前，某陣列 A 的樹狀結構如下。

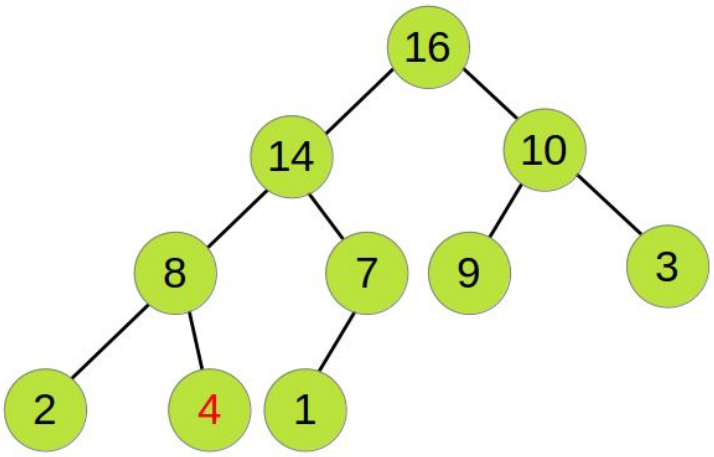


第 2 個元素不符合 Heap 的性質，我們呼叫 Max_Heapify(A, index=2)函式：

Step 1. 因 14 為兩個子節點中較大者，將 A[2]與 A[4]交換。



Step 2. 與 Step 1.相同的邏輯，再將 A[4]與 A[8]交換，即完成 Heapify。



那為何我們要使用 Heapify? 先來看 Build Max/Min Heap 的虛擬碼(以 Max 為例)：

Build_Max_Heap(A)：

```
For i = n/2 to 1：
    call Max_Heapify(A, i)
```

由虛擬碼可以發現，我們不對 leaves 做 Heapify，這是因為 leaves 沒有子節點，所以符合 Heap 特性。我們從最底層的父節點開始，每個父節點形成的子樹，即使不符合 Heap，由於其子節點形成的子樹符合 Heap 特性，所以只有 1 個節點需要移動，如此一步步往上，對所有父節點進行 Heapify，即可完成 Heap 的構建。

每移動一次的時間複雜度為 O(1)，每層需移動的最大節點數與高度有關，約略的計算次數可寫成：

$$n/4*1+n/8*2+...+1*logN$$

利用變數變換及微積分，可算出時間複雜度為 O(n)。

2. Insert 及 Bubble-up → 將新的資料插入 Heap 並保持結構特性

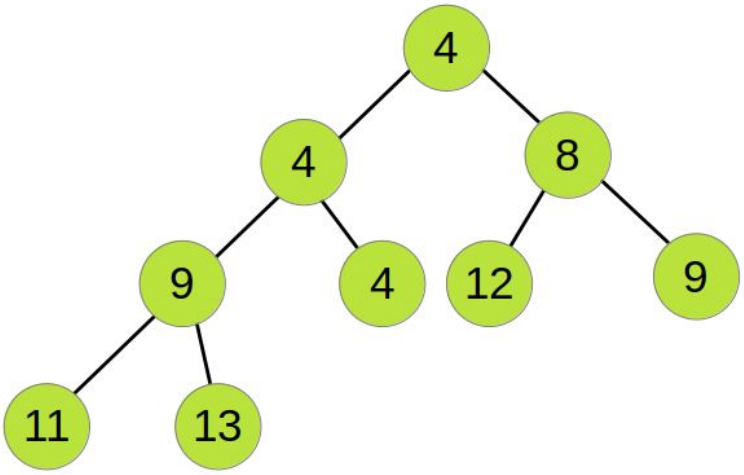
1. 當需要插入新資料時，我們先將資料放到陣列的最後。
2. 然後為了符合 Heap 的特性，若父節點小於(Max Heap)/大於(Min Heap)資料，則與父節點做交換，一路往上確認，直到父節點與子節點的關係滿足 Heap 特性停止。因為類似泡泡上浮，因此稱為 Bubble-up。
3. 時間複雜度為 O(logn)

3. Extract-Max/Min 及 Bubble-down → 取出最大/最小值，保持 Heap 結構特性

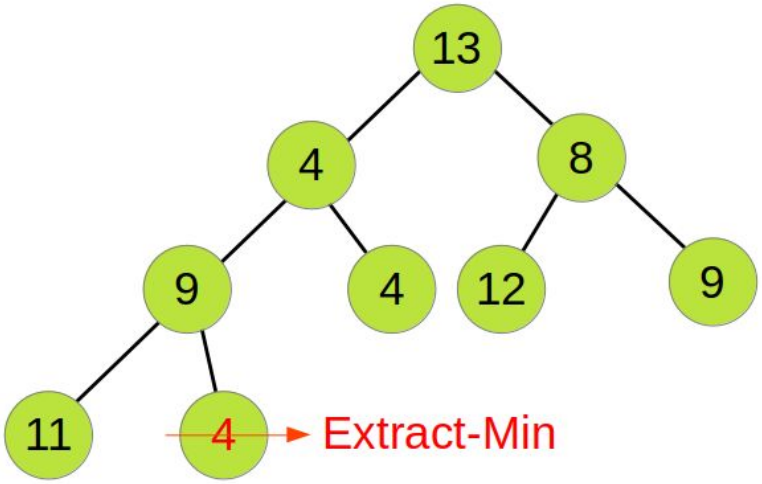
Heap 最重要的特性，就是可以在 O(logn)時間內，取出資料中的極值，其操作原理如下：

1. 將根節點與最後一個葉節點交換，並從 Heap pop。
2. 對新的根節點，確認其與子節點的關係，若不符合 Heap 特性則交換，一路往下確認，直到父節點與子節點的關係滿足 Heap 特性停止。因為類似泡泡下沉，因此稱為 Bubble-down。

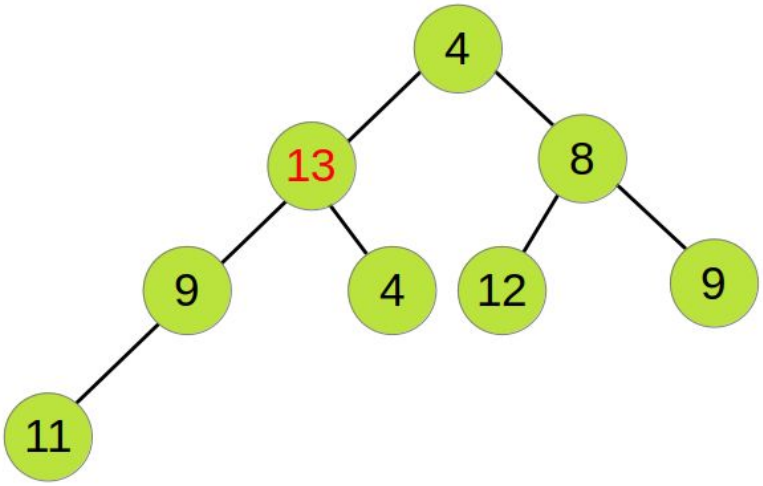
以 Extract-Min 為例：



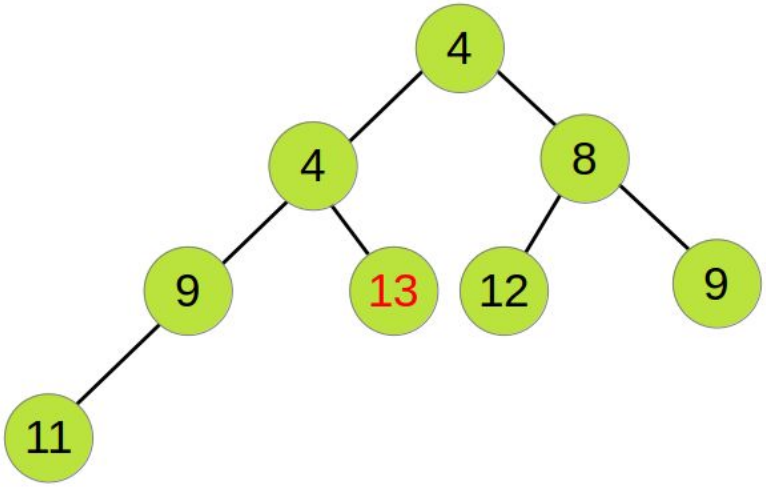
Step 1. 將 A[1] 與 A[9] 交換，再將 A[9] pop 出來取得最小值。



Step 2. 因 A[2] 比 A[3] 小，所以將 A[1] 與 A[2] 交換。



Step 3. 同理，再將 A[2] 與 A[5] 交換，完成 Bubble-Down。



Binary Heap 的使用場合：

1. 排序，時間複雜度 O(nlogn)。
2. 變動數據的中位數計算。
3. 優先佇列 (Priority Queue)。
4. Prim’s、Dijkstra’s 等演算法的加速。