Heap

by music960633

Sprout

課程內容

- 1. priority queue
- 2. 直接做：O(n)的做法
- 3. 分組：O($\sqrt{n}$)的做法
- 4. heap：O(log(n))的做法

priority queue

- 複習一下之前講過的資料結構
  - stack: last in first out
  - queue: first in first out

- 如果現在不想要pop出來的是最先進去的，也不是最後進去的，而是「權重」最大(小)的呢？
  - priority queue!

- note: 以下為了方便講解，元素皆為整數，「權重」即是數字的大小，如3的權重比2大。

priority queue

- 基本操作
- 1. push：將一個元素放入priority queue中
- 2. top ：詢問現在priority queue中權重最大的元素
- 3. pop ：將priority queue中權重最大的元素拿掉

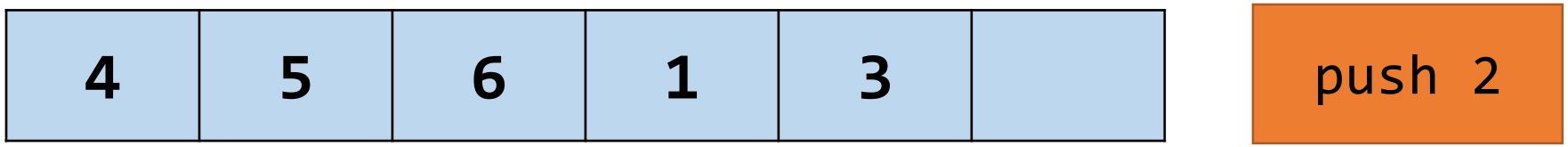
最直接的做法

- 把全部的元素丟進陣列

4	5	6	1	3	

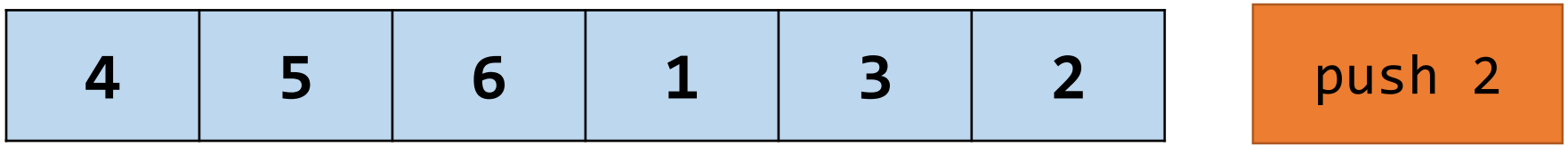
最直接的做法

- 把全部的元素丟進陣列



最直接的做法

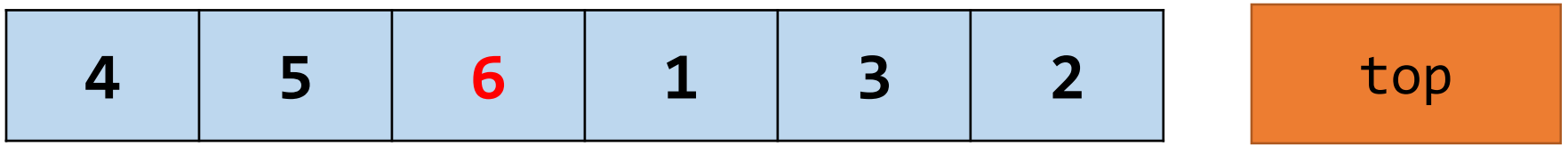
- 把全部的元素丟進陣列



  - 直接放進去就好了，O(1)

最直接的做法

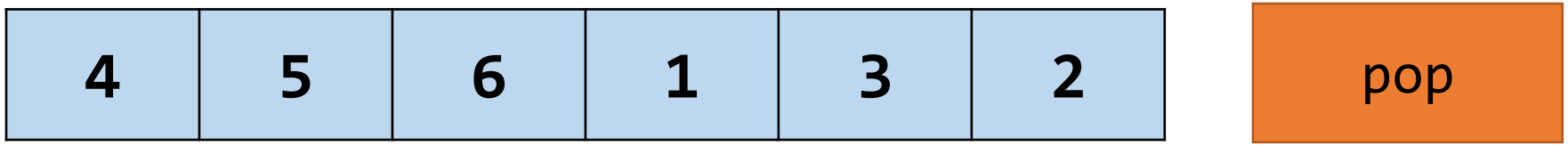
- 把全部的元素丟進陣列



  - 找到最大值：O(n)

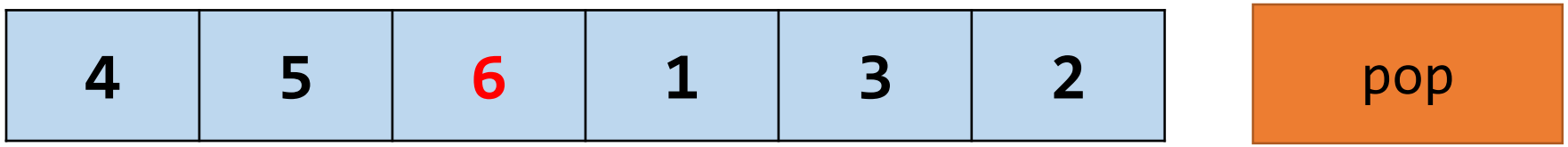
最直接的做法

- 把全部的元素丟進陣列



最直接的做法

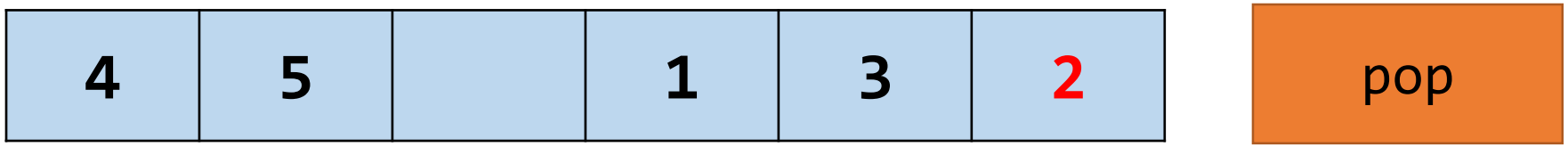
- 把全部的元素丟進陣列



  - 找到最大值，pop掉：O(n)

最直接的做法

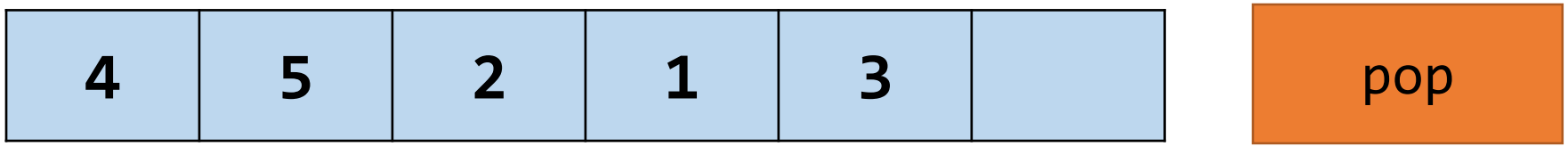
- 把全部的元素丟進陣列



  - 找到最大值，pop掉：O(n)
  - 將最後一個元素補回來：O(1)

最直接的做法

- 把全部的元素丟進陣列



  - 找到最大值，pop掉：O(n)
  - 將最後一個元素補回來：O(1)
  - 整體來看：O(n)

最直接的做法

- 把全部的元素丟進陣列
  - push：O(1)
  - pop ：O(n)
  - top ：O(n)

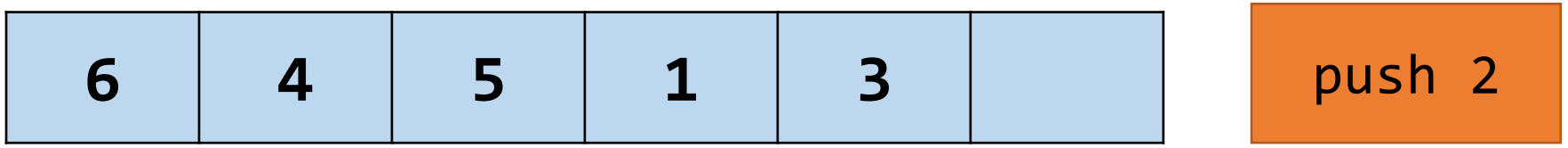
- 改良一下，保持陣列的第一個元素是權重最大的

6	4	5	1	3	

最直接的做法

- 把全部的元素丟進陣列
  - push：O(1)
  - top ：O(n)
  - pop ：O(n)

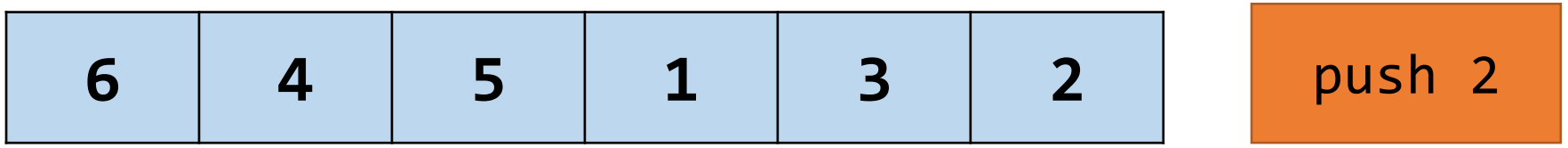
- 改良一下，保持陣列的第一個元素是權重最大的



最直接的做法

- 把全部的元素丟進陣列
  - push：O(1)
  - top ：O(n)
  - pop ：O(n)

- 改良一下，保持陣列的第一個元素是權重最大的

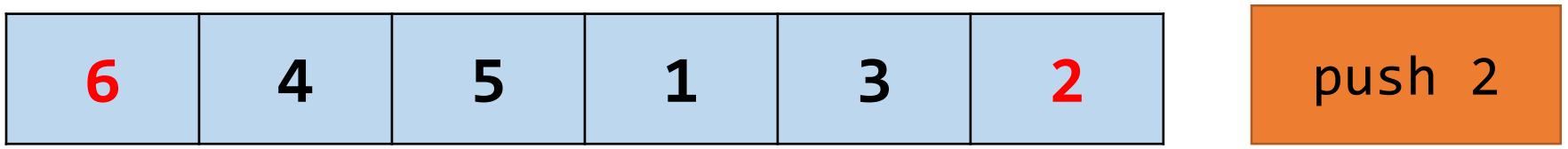


  - 放進去：O(1)

最直接的做法

- 把全部的元素丟進陣列
  - push：O(1)
  - top ：O(n)
  - pop ：O(n)

- 改良一下，保持陣列的第一個元素是權重最大的

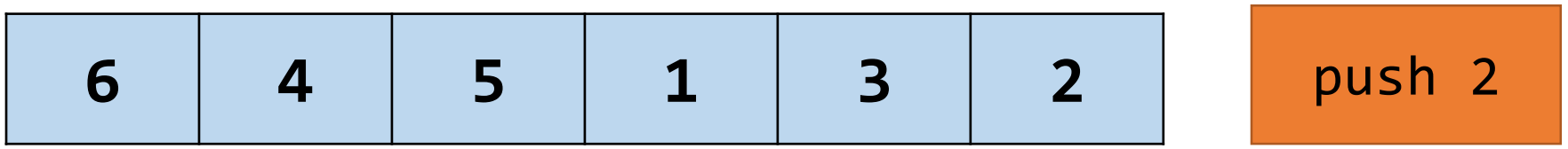


  - 放進去：O(1)
  - 和最大值比較：O(1)

最直接的做法

- 把全部的元素丟進陣列
  - push：O(1)
  - top ：O(n)
  - pop ：O(n)

- 改良一下，保持陣列的第一個元素是權重最大的

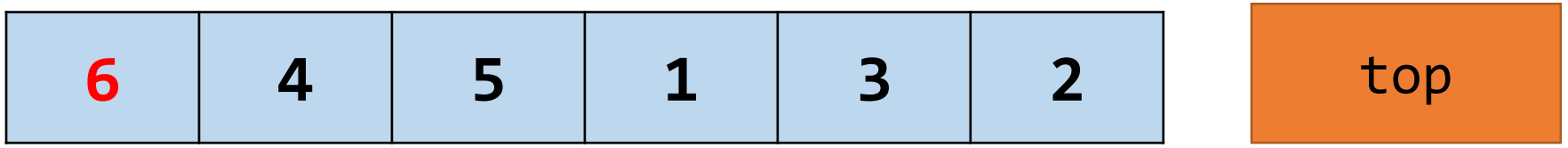


  - 放進去：O(1)
  - 和最大值比較：O(1)
  - 整體來看：O(1)

最直接的做法

- 把全部的元素丟進陣列
  - push：O(1)
  - top ：O(n)
  - pop ：O(n)

- 改良一下，保持陣列的第一個元素是權重最大的

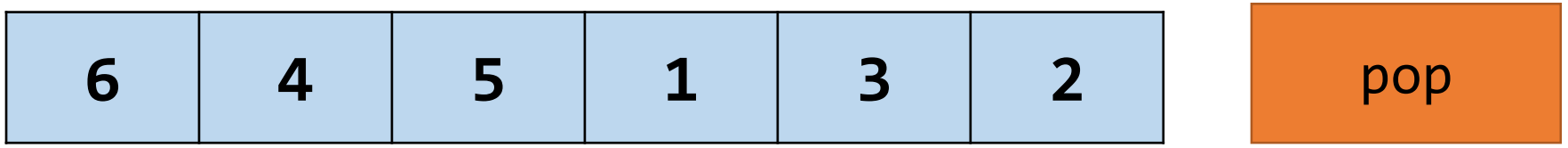


  - 陣列第一個元素就是最大值，O(1)

最直接的做法

- 把全部的元素丟進陣列
  - push：O(1)
  - top ：O(n)
  - pop ：O(n)

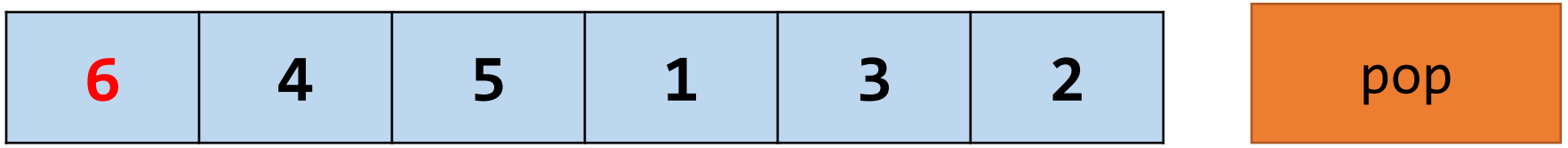
- 改良一下，保持陣列的第一個元素是權重最大的



最直接的做法

- 把全部的元素丟進陣列
  - push：O(1)
  - top ：O(n)
  - pop ：O(n)

- 改良一下，保持陣列的第一個元素是權重最大的

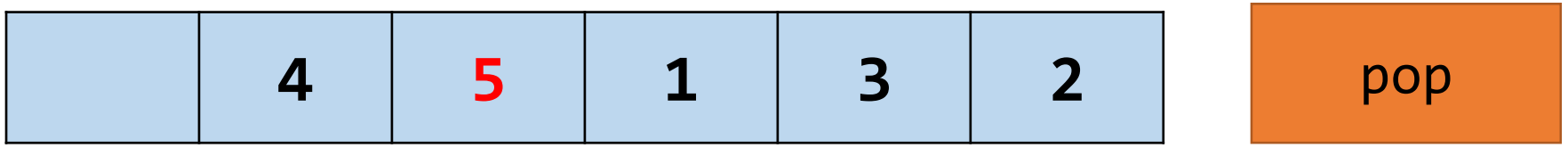


  - 第一個元素權重最大，pop掉

最直接的做法

- 把全部的元素丟進陣列
  - push：O(1)
  - top ：O(n)
  - pop ：O(n)

- 改良一下，保持陣列的第一個元素是權重最大的

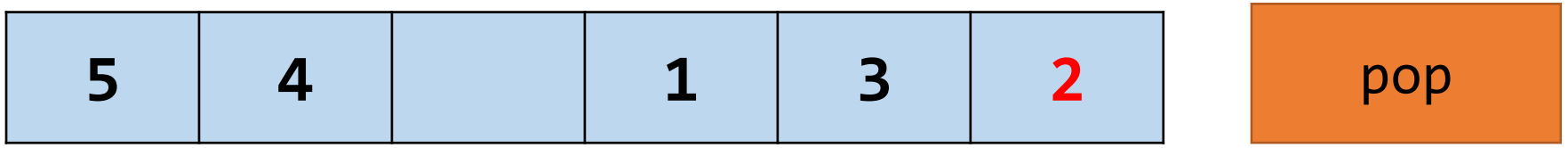


  - 第一個元素權重最大，pop掉
  - 找到剩下元素中的最大值：O(n)

最直接的做法

- 把全部的元素丟進陣列
  - push：O(1)
  - top ：O(n)
  - pop ：O(n)

- 改良一下，保持陣列的第一個元素是權重最大的

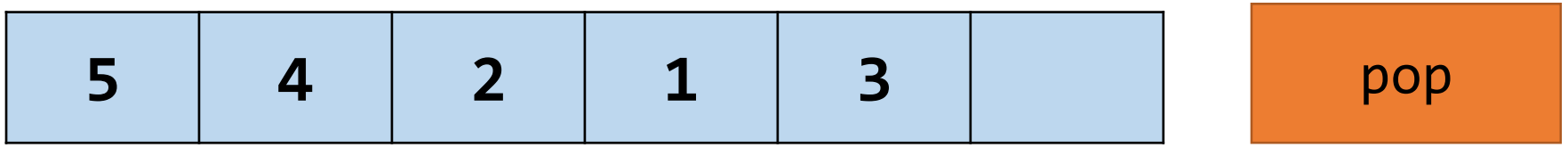


  - 第一個元素權重最大，pop掉
  - 找到剩下元素中的最大值：O(n)
  - 最後一個元素補回來：O(1)

最直接的做法

- 把全部的元素丟進陣列
  - push：O(1)
  - top ：O(n)
  - pop ：O(n)

- 改良一下，保持陣列的第一個元素是權重最大的



  - 第一個元素權重最大，pop掉
  - 找到剩下元素中的最大值：O(n)
  - 最後一個元素補回來：O(1)
  - 整體來看：O(n)

最直接的做法

- 把全部的元素丟進陣列
  - push：O(1)
  - top ：O(n)
  - pop ：O(n)
- 改良一下，保持陣列的第一個元素是權重最大的
  - push：O(1)
  - top ：O(1)
  - pop ：O(n)
- 不管怎樣，pop都還是太慢

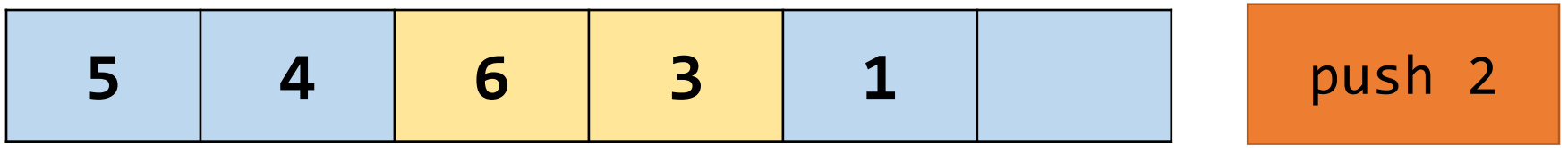
分組做法

- 精神：將k個數字分成一組，並記錄這些數中權重最大的
- ex： k=2，每一組權重最大的放在前面

5	4	6	3	1	

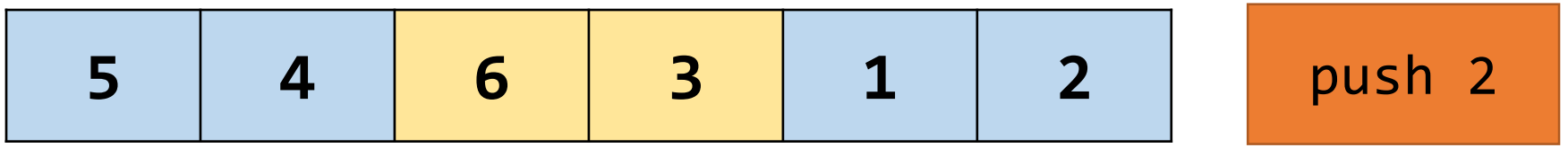
分組做法

- 精神：將k個數字分成一組，並記錄這些數中權重最大的
- ex： k=2，每一組權重最大的放在前面



分組做法

- 精神：將k個數字分成一組，並記錄這些數中權重最大的
- ex： k=2，每一組權重最大的放在前面

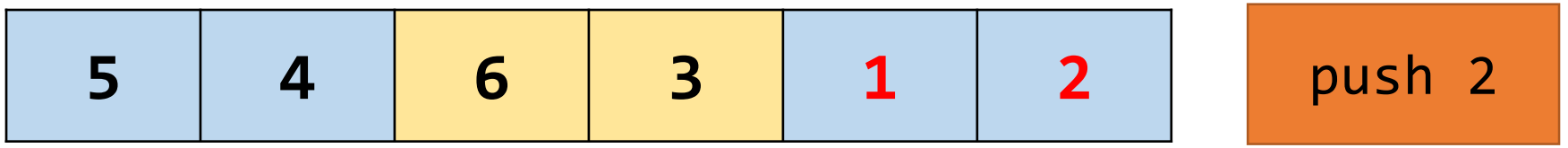


push 2

  - 放進陣列：O(1)

分組做法

- 精神：將k個數字分成一組，並記錄這些數中權重最大的
- ex： k=2，每一組權重最大的放在前面

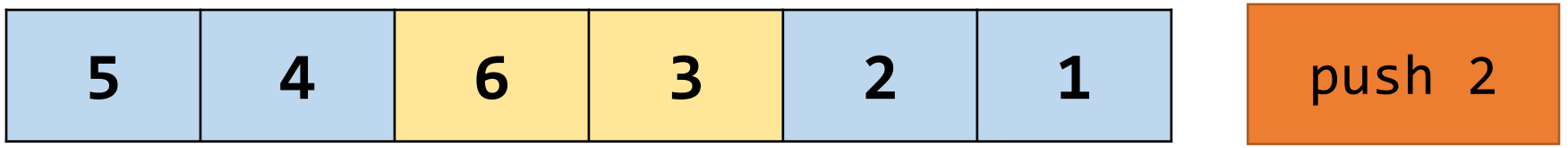


push 2

  - 放進陣列：O(1)
  - 和同組的最大值比較：O(1)

分組做法

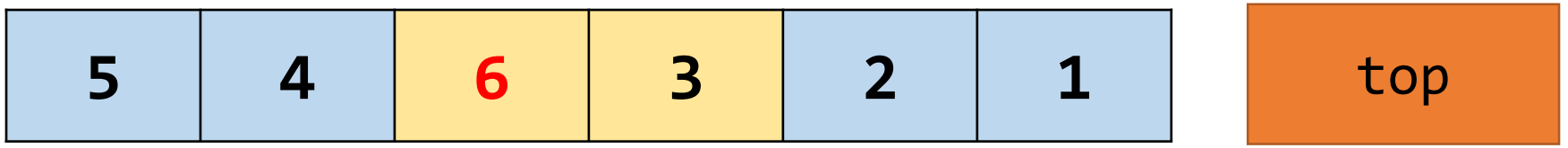
- 精神：將k個數字分成一組，並記錄這些數中權重最大的
- ex： k=2，每一組權重最大的放在前面



  - 放進陣列：O(1)
  - 和同組的最大值比較：O(1)
  - 整體來看：O(1)

分組做法

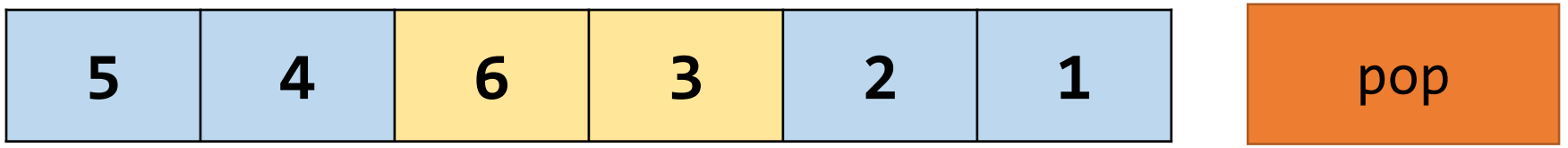
- 精神：將k個數字分成一組，並記錄這些數中權重最大的
- ex: k=2，每一組權重最大的放在前面



  - 比較每組的最大值：O(n/k) // n:元素個數

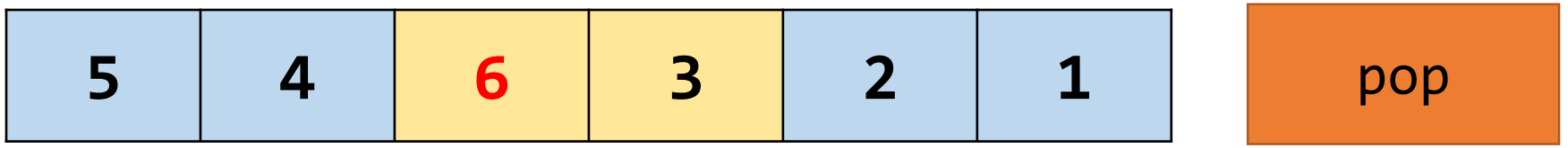
分組做法

- 精神：將k個數字分成一組，並記錄這些數中權重最大的
- ex： k=2，每一組權重最大的放在前面



分組做法

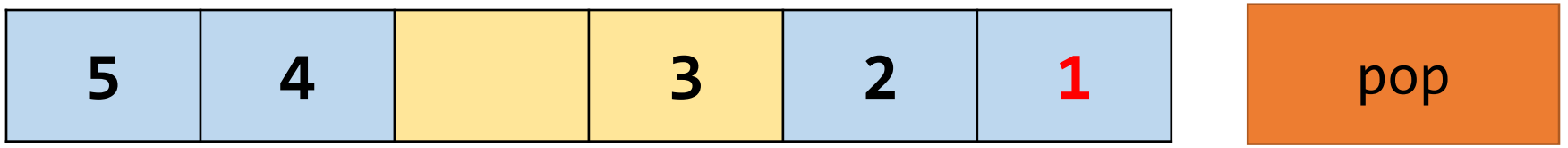
- 精神：將k個數字分成一組，並記錄這些數中權重最大的
- ex： k=2，每一組權重最大的放在前面



  - 找到最大值，pop掉：O(n/k)

分組做法

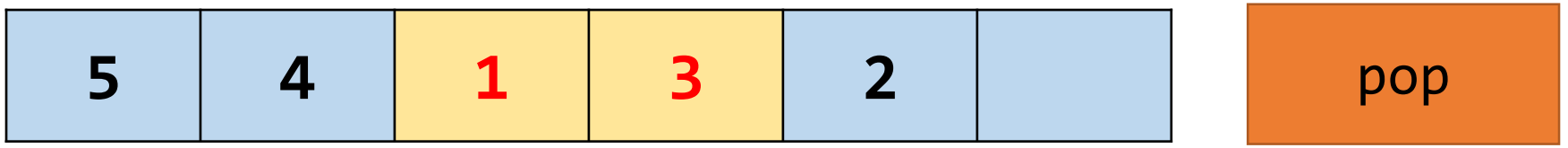
- 精神：將k個數字分成一組，並記錄這些數中權重最大的
- ex： k=2，每一組權重最大的放在前面



  - 找到最大值，pop掉：O(n/k)
  - 最後一個元素補上：O(1)

分組做法

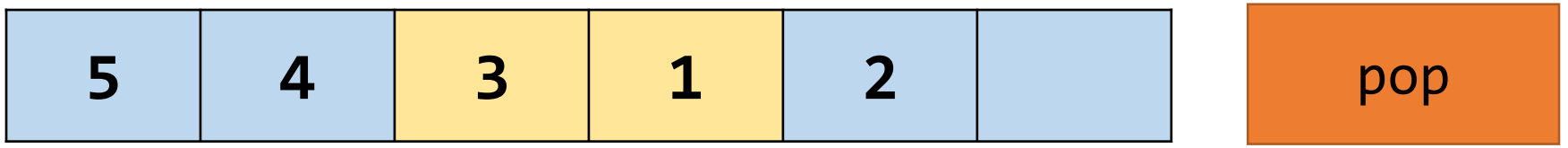
- 精神：將k個數字分成一組，並記錄這些數中權重最大的
- ex： k=2，每一組權重最大的放在前面



  - 找到最大值，pop掉：O(n/k)
  - 最後一個元素補上：O(1)
  - 找出組中最大的元素：O(k)

分組做法

- 精神：將k個數字分成一組，並記錄這些數中權重最大的
- ex： k=2，每一組權重最大的放在前面



  - 找到最大值，pop掉：O(n/k)
  - 最後一個元素補上：O(1)
  - 找出組中最大的元素：O(k)
  - 整體來看：O(n/k)+O(k)，視k的大小而定

分組做法

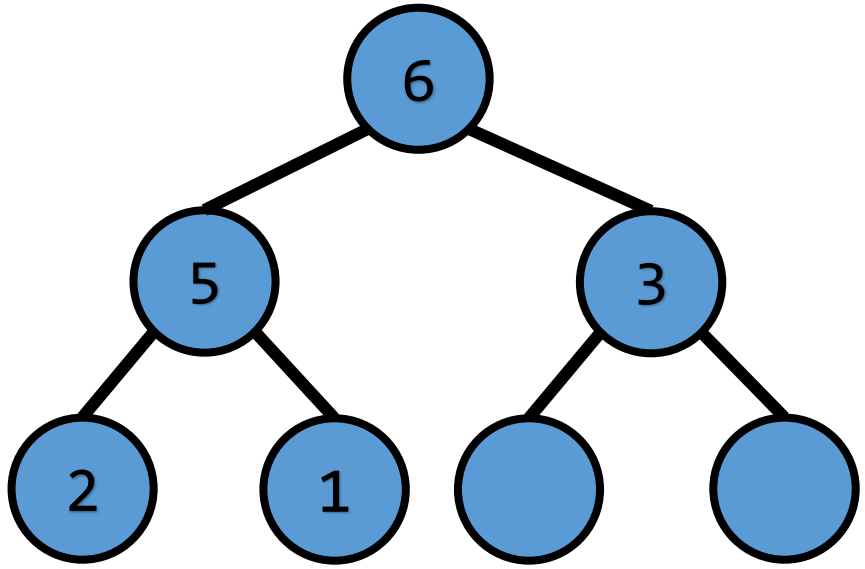
- 操作複雜度
  - push：O(1)
  - top ：O(n/k)
  - pop ：O(n/k)+O(k)

- 選擇k=$\sqrt{n}$
  - push：O(1)
  - top ：O($\sqrt{n}$)
  - pop ：O($\sqrt{n}$)

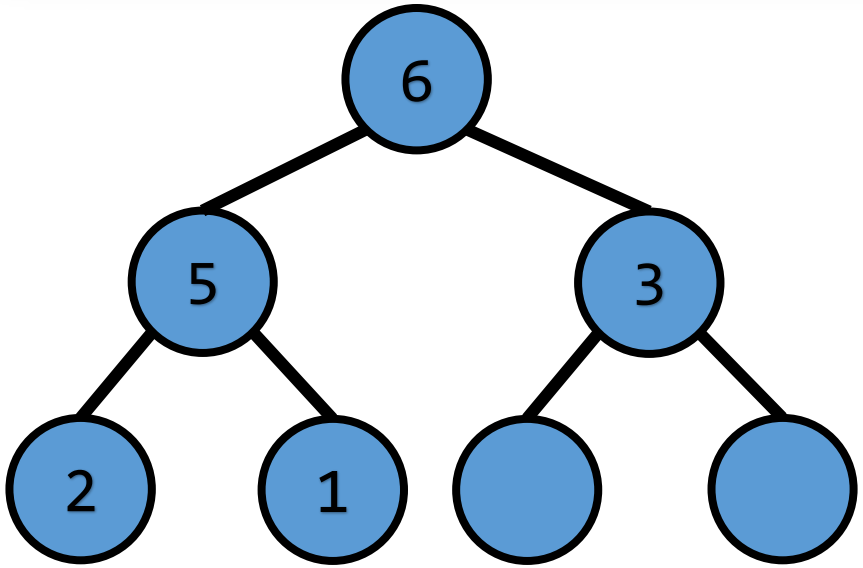
- 當然還可以維護全部的最大值在第一組，在此就不詳細說明了。

heap

- 再來就是這次的主角，heap！
- heap其實就是一棵complete binary tree
- 性質：父節點的權重不小於子節點的權重

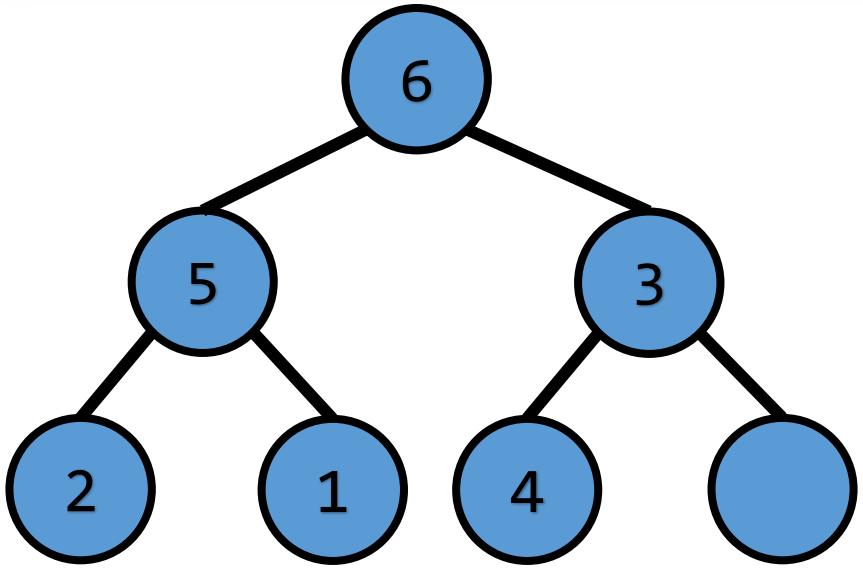


heap



push 4

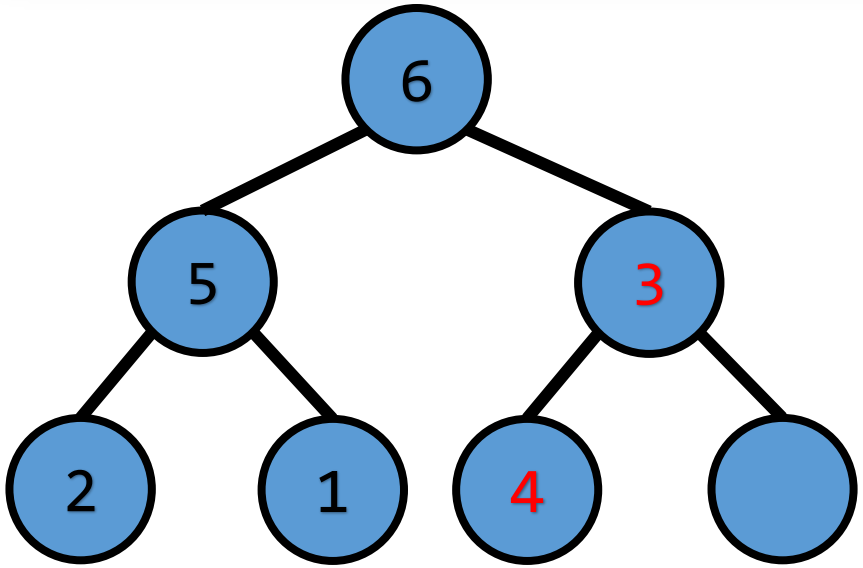
heap



push 4

- 將元素放進tree：O(1)

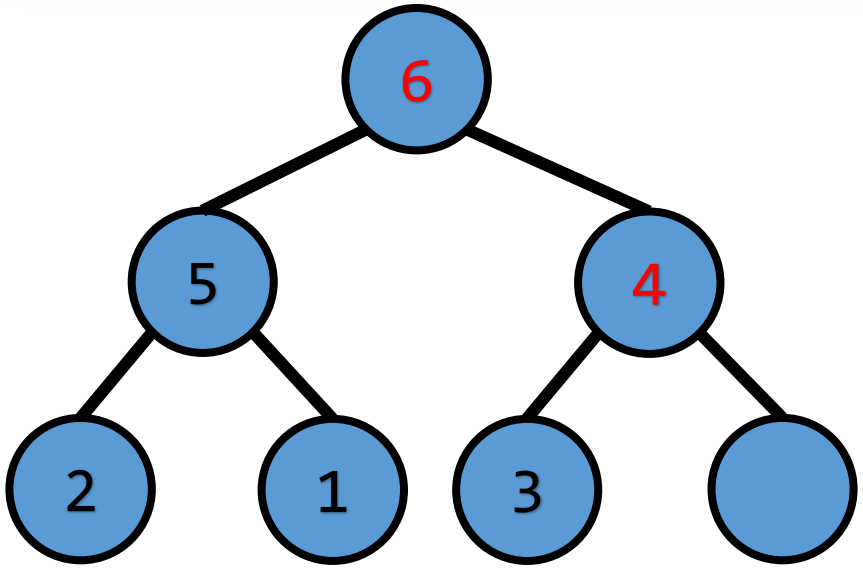
heap



push 4

- 將元素放進tree：O(1)
- 和父節點比較：O(1)

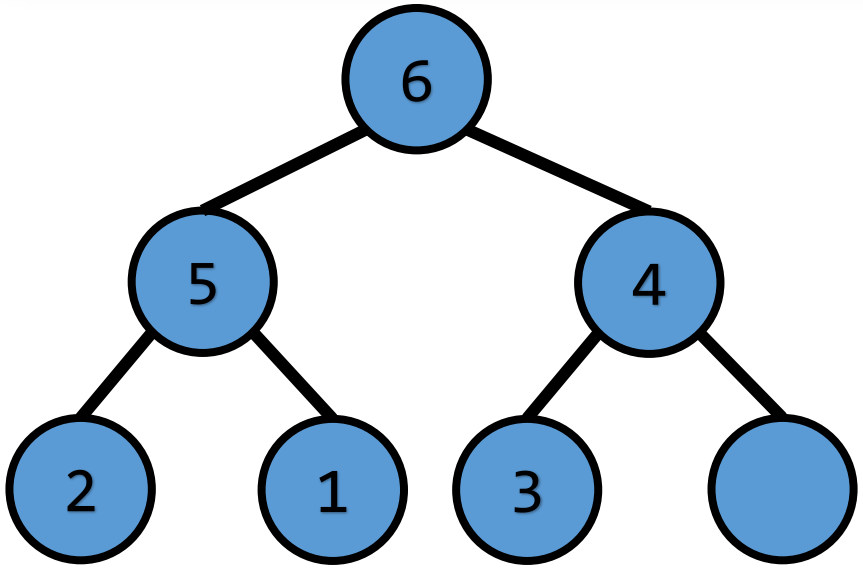
heap



push 4

- 將元素放進tree：O(1)
- 和父節點比較：O(1)*log(n)
- 一直往上浮，直到權重不大於父節點，最多比較log(n)次

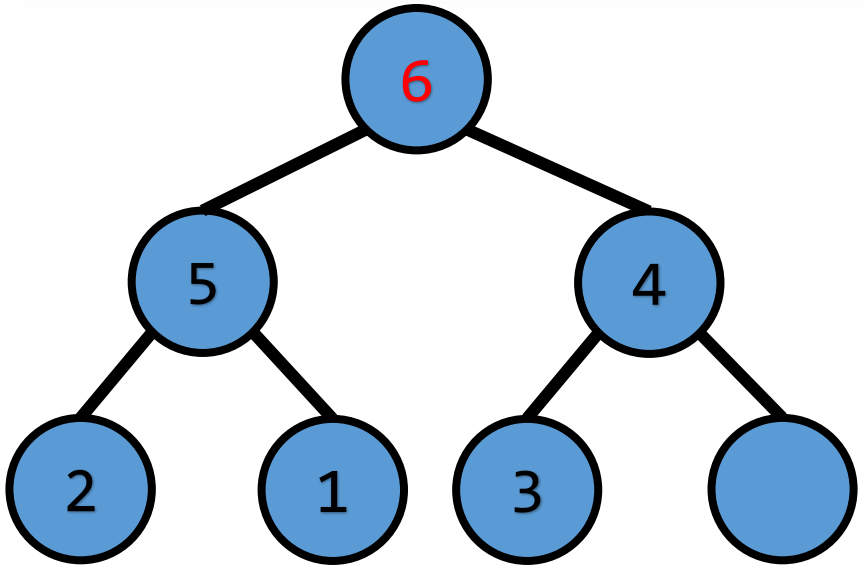
heap



push 4

- 將元素放進tree：O(1)
- 和父節點比較：O(1)*log(n)
- 一直往上浮，直到權重不大於父節點，最多比較log(n)次
- 整體來看：O(log(n))

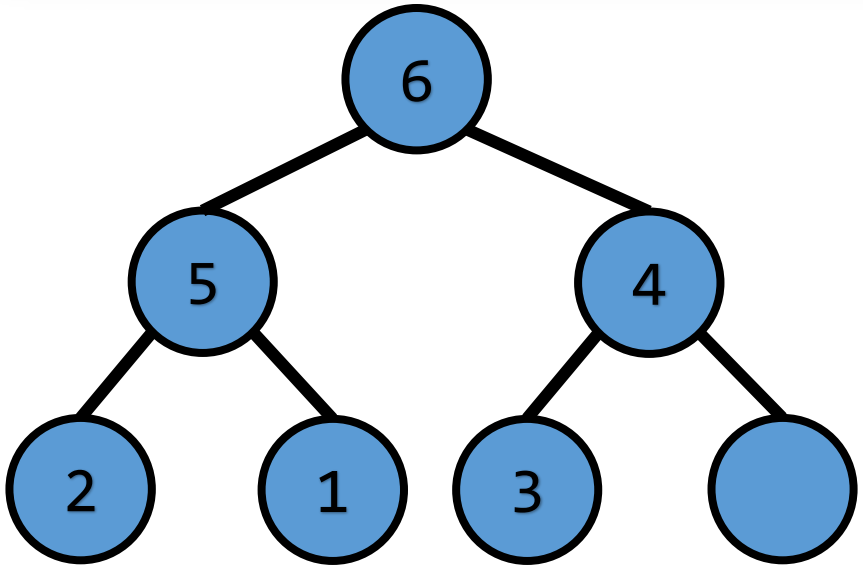
heap



top

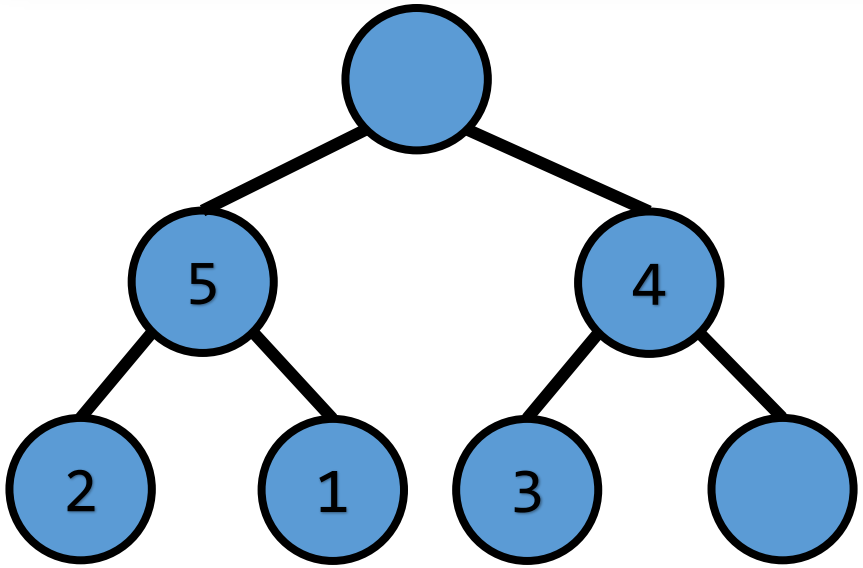
- root的權重一定最大，O(1)

heap



pop

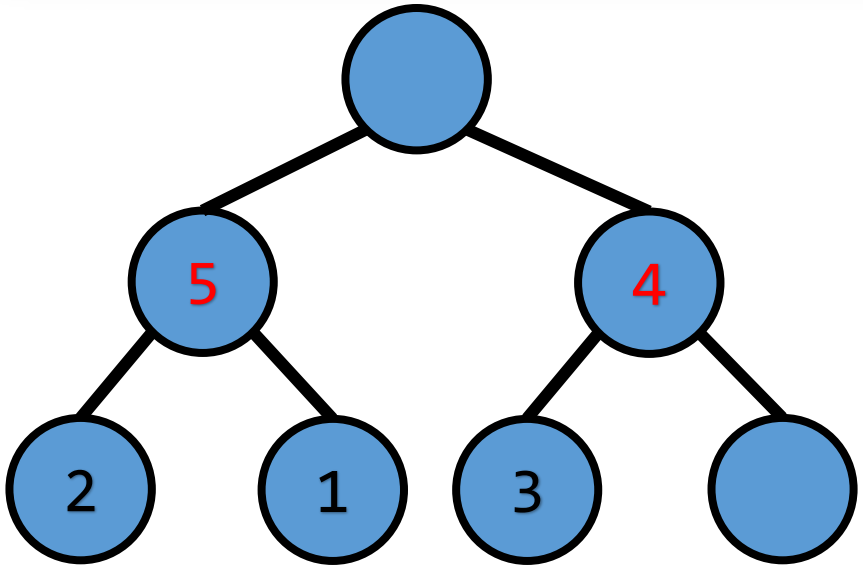
heap



pop

- root的權重最大，pop掉

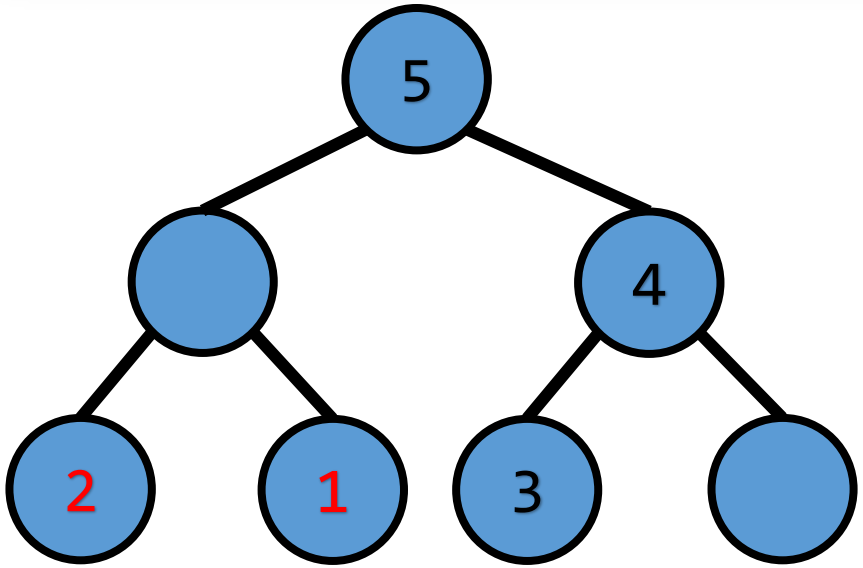
heap



pop

- root的權重最大，pop掉
- 比較兩個子節點，權重大的往上浮：O(1)

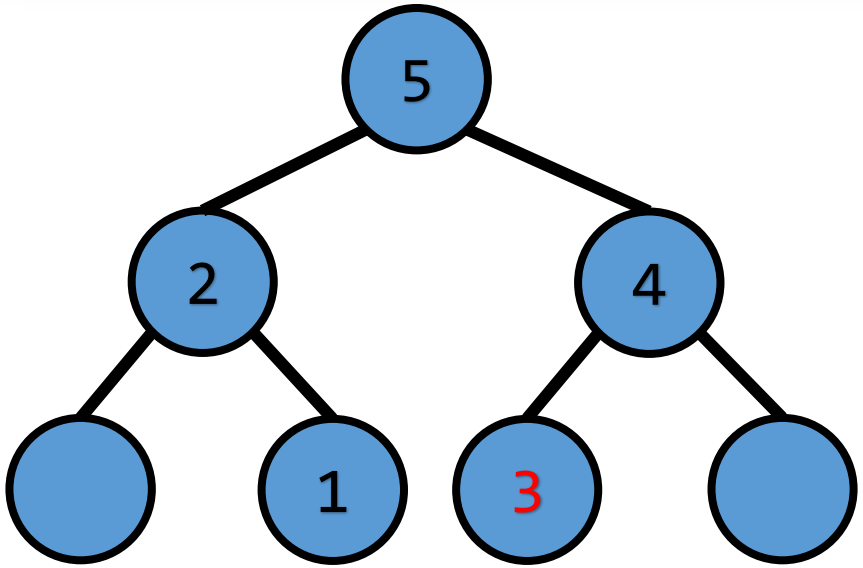
heap



pop

- root的權重最大，pop掉
- 比較兩個子節點，權重大的往上浮：O(1)*log(n)
- 繼續比，最多比log(n)次

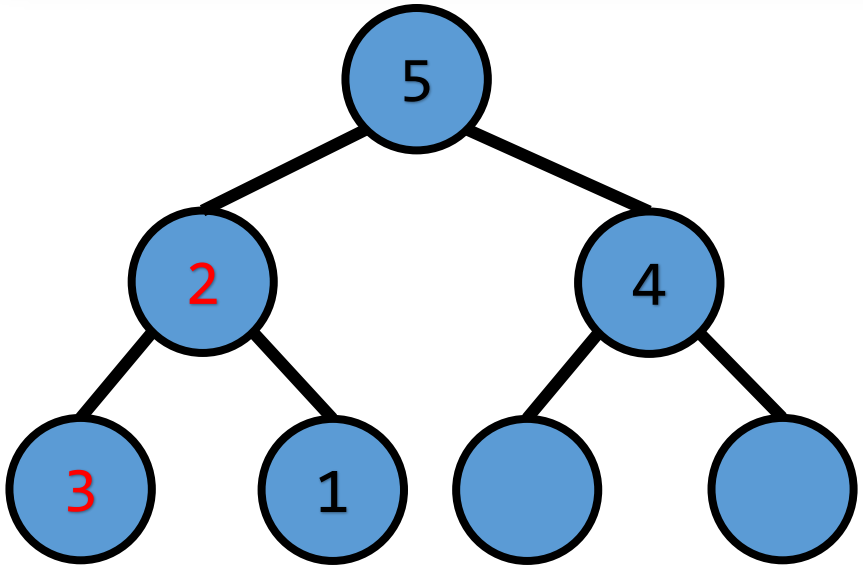
heap



pop

- root的權重最大，pop掉
- 比較兩個子節點，權重大的往上浮：O(1)*log(n)
- 繼續比，最多比log(n)次
- 最後空位用最後一個元素補

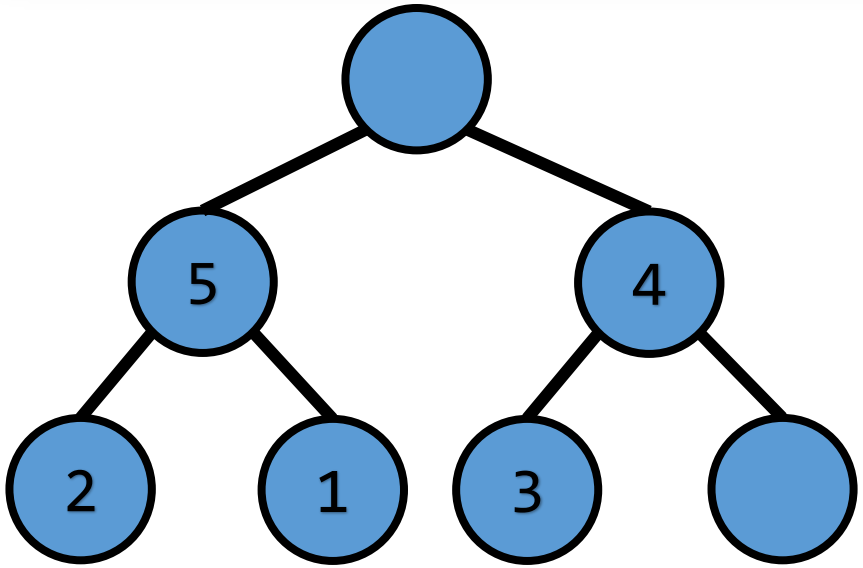
heap



pop

- root的權重最大，pop掉
- 比較兩個子節點，權重大的往上浮：O(1)*log(n)
- 繼續比，最多log(n)次
- 最後空位用最後一個元素補
- 補上後可能權重又比父親大，還要往上浮一次

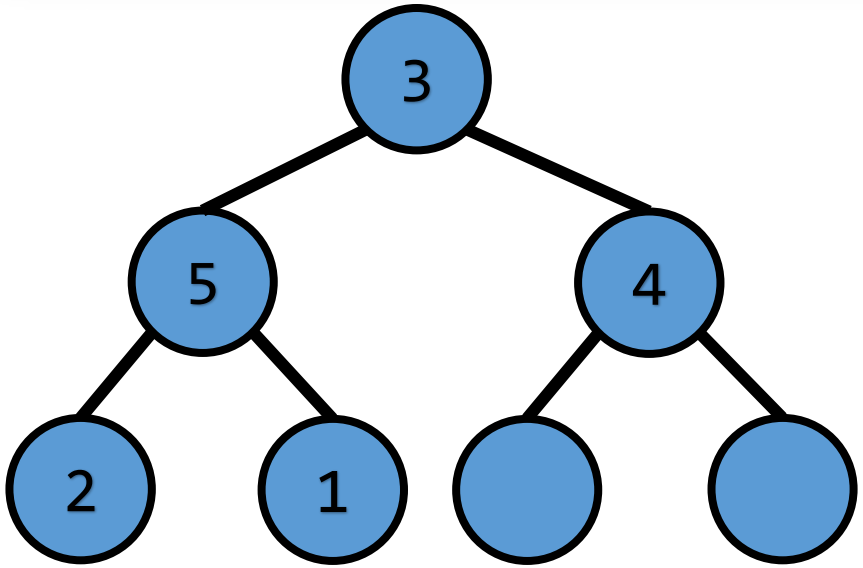
heap



pop

- root的權重最大，pop掉

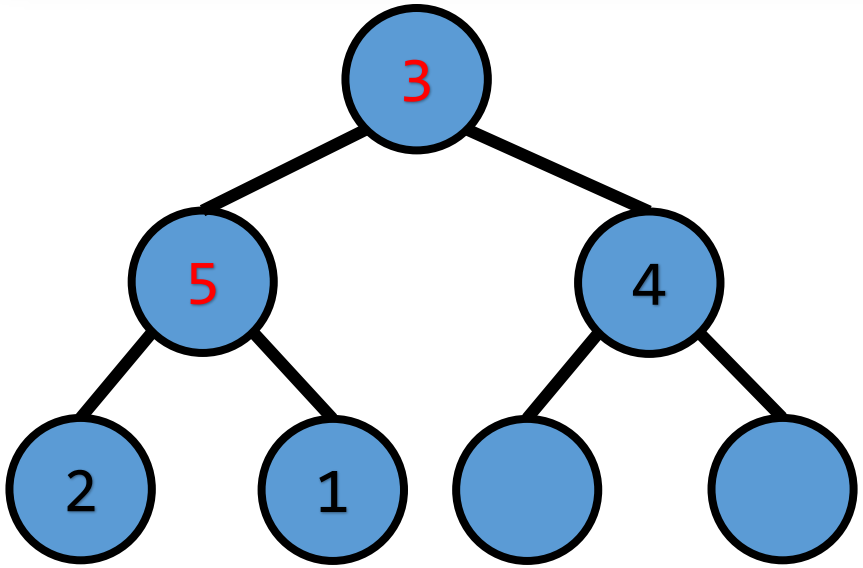
heap



pop

- root的權重最大，pop掉
- 將最後一個元素放到root

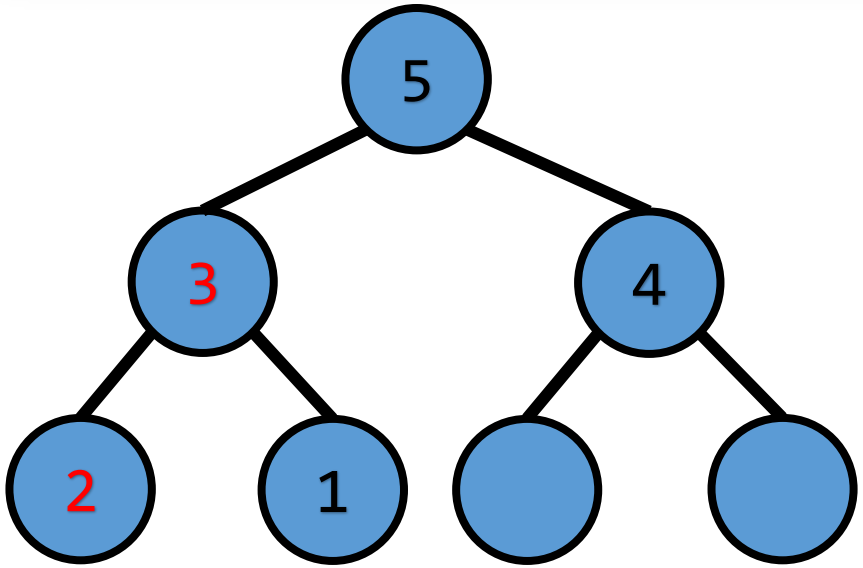
heap



pop

- root的權重最大，pop掉
- 將最後一個元素放到root
- 和兩個子節點中權重較大的節點比較：O(1)

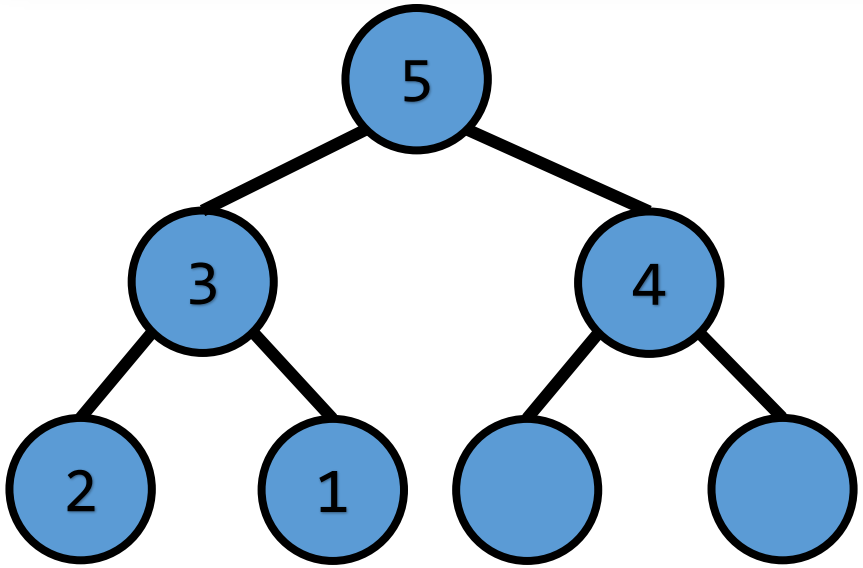
heap



pop

- root的權重最大，pop掉
- 將最後一個元素放到root
- 和兩個子節點中權重較大的節點比較：O(1)*log(n)
- 一直下沉直到權重不小於兩個子節點，最多比較log(n)次

heap



pop

- root的權重最大，pop掉
- 將最後一個元素放到root
- 和兩個子節點中權重較大的節點比較：O(1)*log(n)
- 一直下沉直到權重不小於兩個子節點，最多比較log(n)次
- 整體來看：O(log(n))

heap

- 操作複雜度
  - push：O(log(n))
  - top ：O(1)
  - pop ：O(log(n))

- TOJ: #59 heap練習