

# 非線性資料結構

2021/06/20

By 林品安

iterator  
(迭代器)

# iterator 宣告方式

- ▶ `vector<int>::iterator it;`
- ▶ 資料結構名稱<資料型態>::iterator;

# iterator (迭代器)

- ▶ *iterator* 是作為容器的迭代器使用
- ▶ 除了自行宣告，*v.begin()*、*v.end()*都是*iterator*

# iterator (迭代器)

- ▶ 我們可以在`iterator`前面加上`*`，取代隨機存取進行取值
- ▶ 以排序題目為範例，我們可以這樣進行輸出

```
1  int main(){
2      int n; cin >> n;
3      vector<int>::iterator it;
4      vector<int> v(n);
5      for(int i=0 ; i<n ; ++i){
6          cin >> v[i];
7      }
8      sort(v.begin() , v.end());
9      reverse(v.begin() , v.end());
10     for(it = v.begin() ; it!=v.end() ; ++it){
11         cout << *it << ' ';
12     }
13 }
```

# iterator (迭代器)

- ▶ 再一個例子，如果我們要取得vector第一個元素的值

```
int x = *v.begin();
```

# iterator (迭代器)

- ▶ 最後，`v.end()`沒有值，它是最後一個元素的下一個

```
int x = * (--v.end());
```

# iterator (迭代器)

- ▶ 為甚麼要講iterator?
- ▶ 因為接下來的非線性容器要使用*iterator*才能輸出元素
- ▶ 非線性容器皆為不可隨機存取的，  
所以要使用*iterator*進行存取。



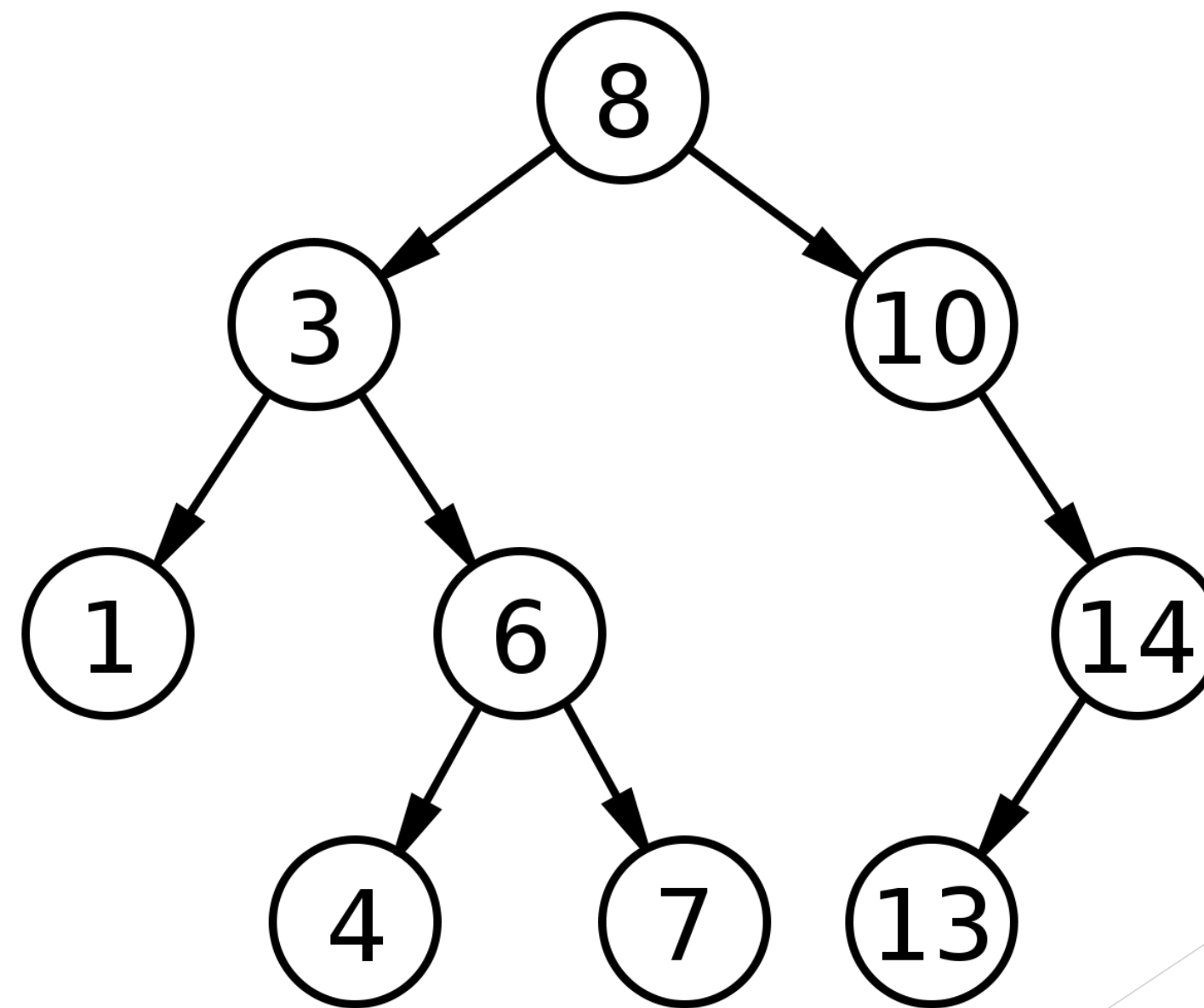
# Binary-Search Tree (二元搜尋樹)

# Binary-Search Tree (二元搜尋樹)

► 常用的資料結構!

► 簡單的帶過，

因為實際上我們不會用C++自己寫這個東西，除非作業要求

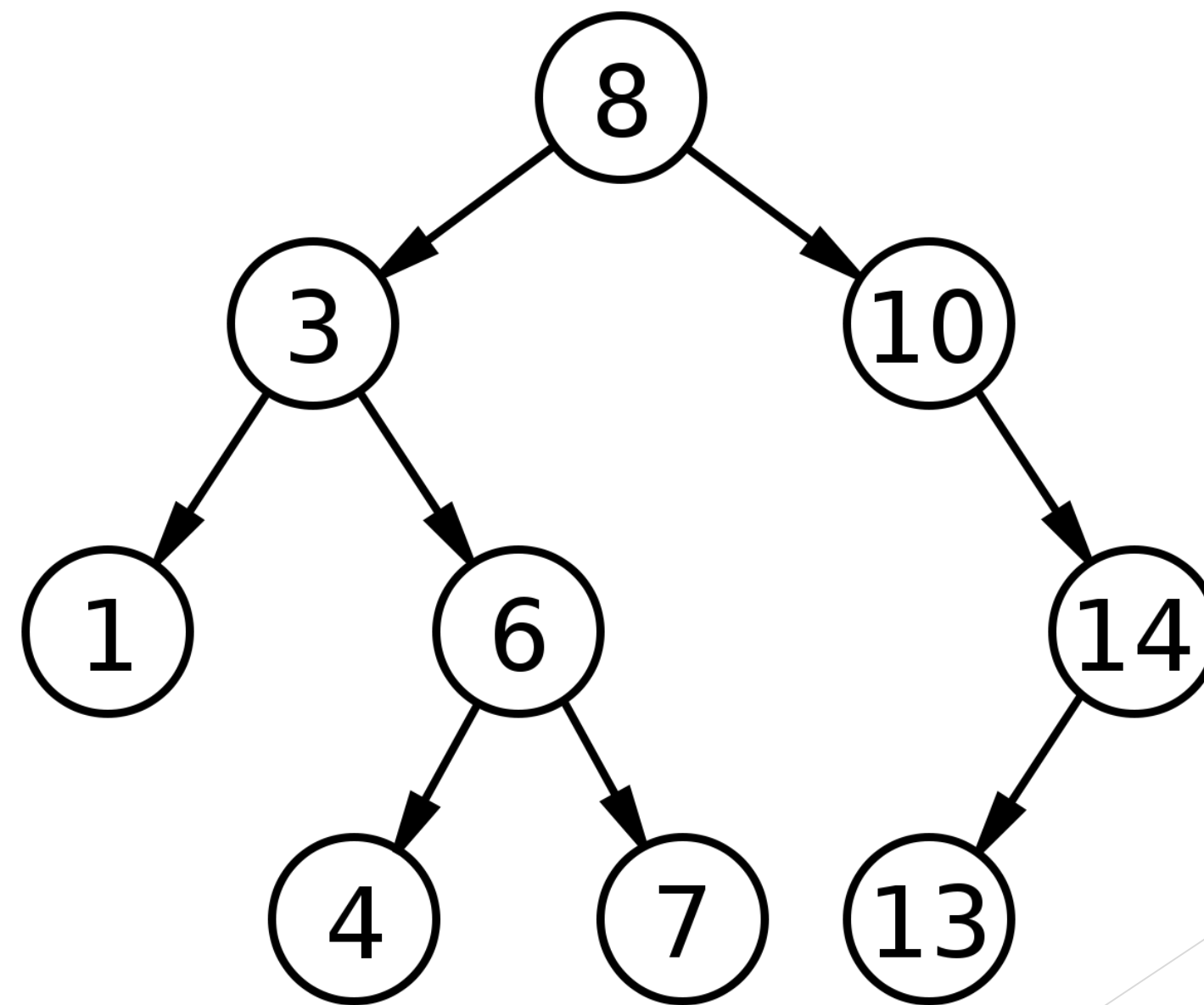


# Binary-Search Tree (二元搜尋樹)

## ► 二元搜尋樹的原理：

### ► 搜尋某個數字的時候分成兩種情況：

1. 小於等於：往左邊走
2. 大於：往右邊走



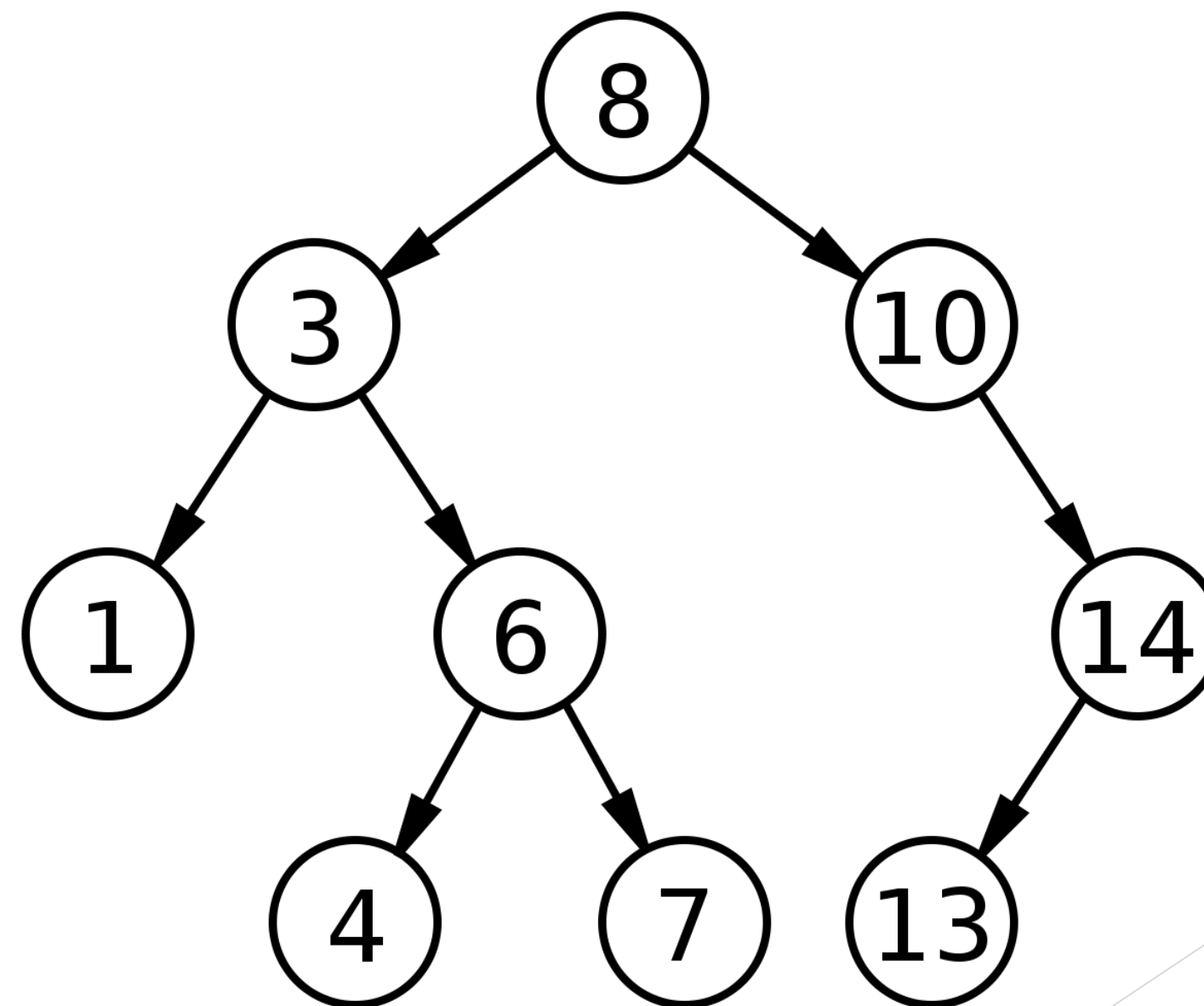
# Binary-Search Tree (二元搜尋樹)

## ► 二元搜尋樹的原理：

### ► 搜尋某個數字的時候分成兩種情況：

1. 小於等於：往左邊走
2. 大於：往右邊走

假設現在要查找6這個數字存不存在

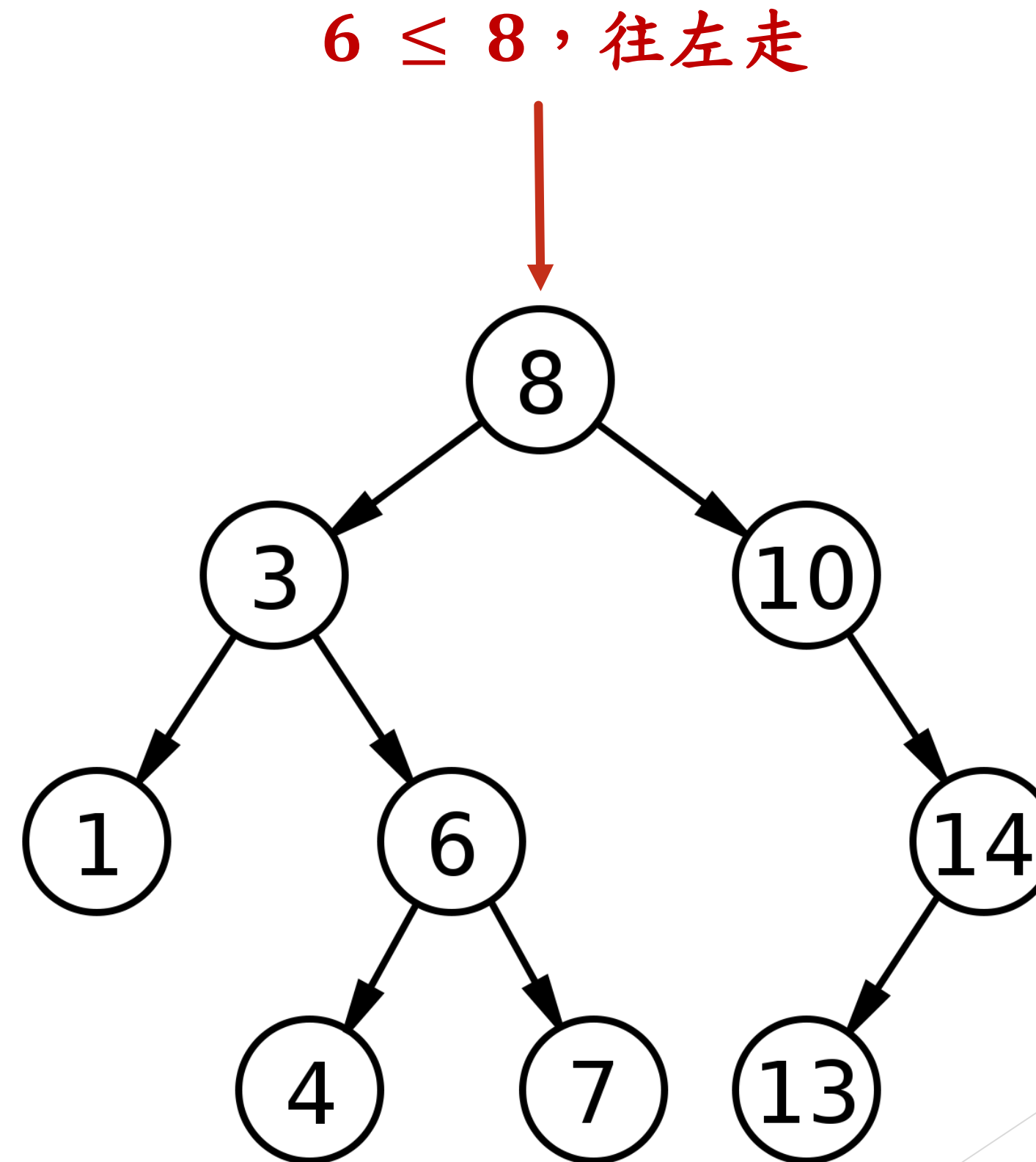


# Binary-Search Tree (二元搜尋樹)

## ► 二元搜尋樹的原理：

### ► 搜尋某個數字的時候分成兩種情況：

1. 小於等於：往左邊走
2. 大於：往右邊走



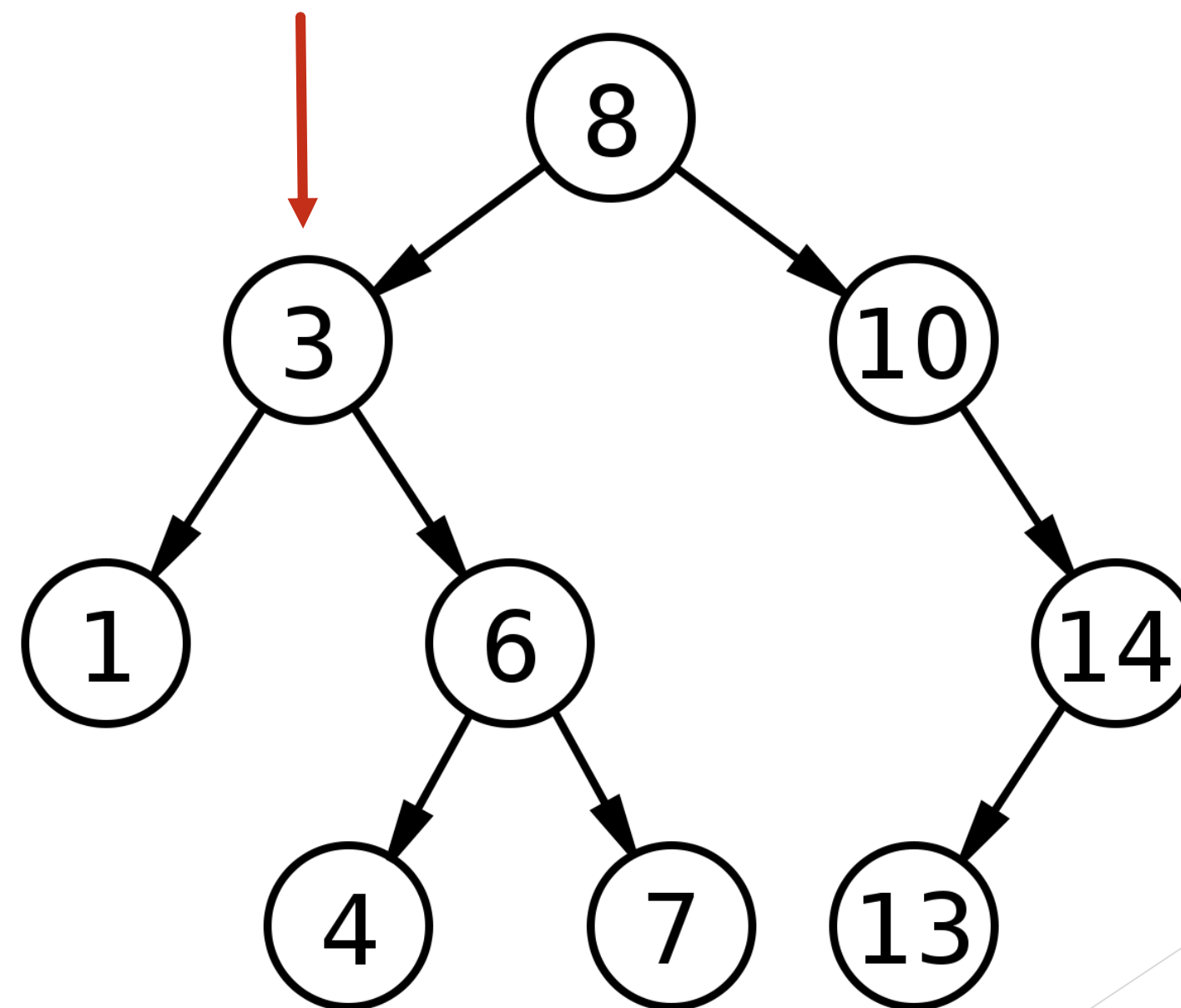
# Binary-Search Tree (二元搜尋樹)

## ► 二元搜尋樹的原理：

### ► 搜尋某個數字的時候分成兩種情況：

1. 小於等於：往左邊走
2. 大於：往右邊走

6 > 3，往右走

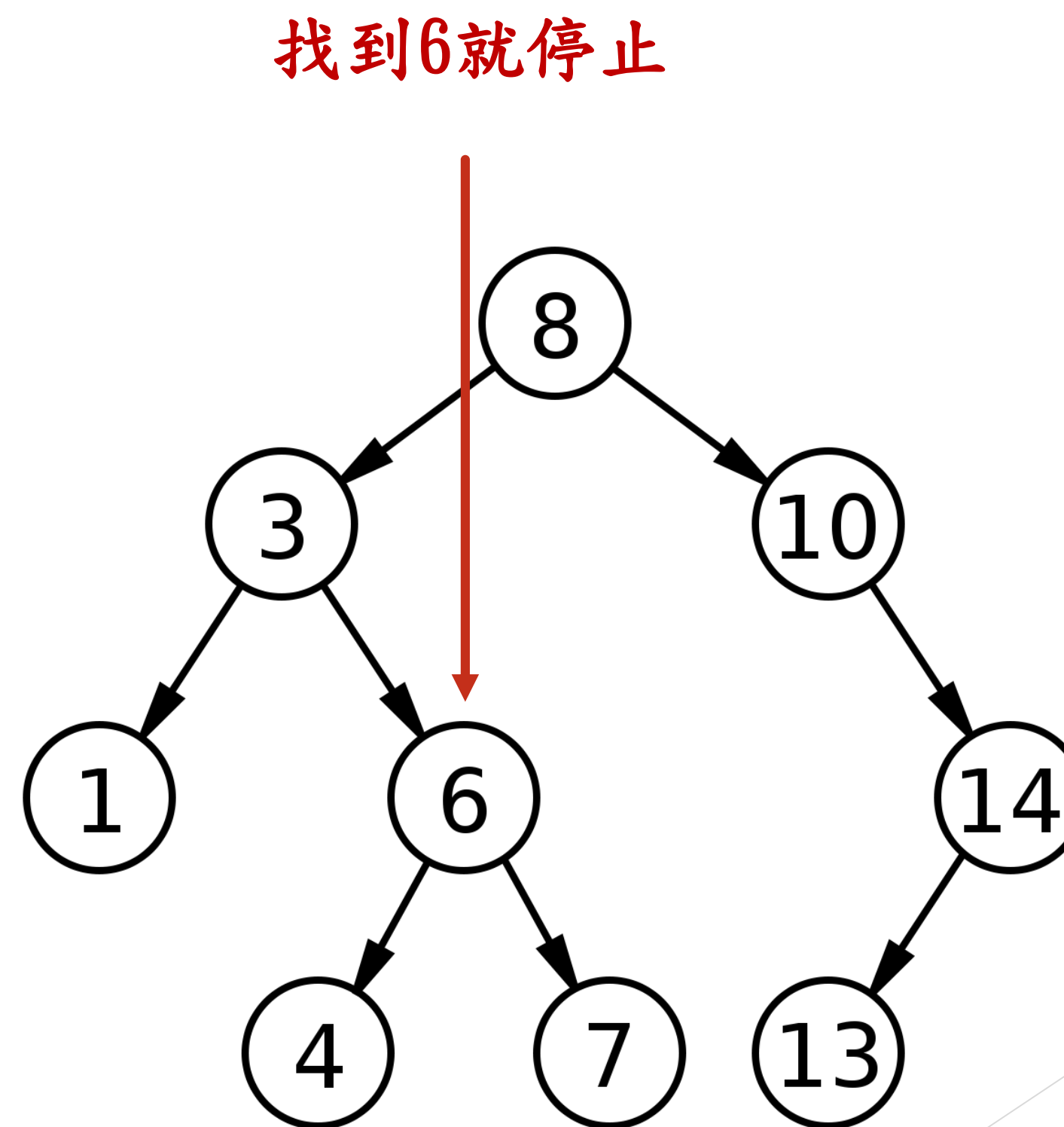


# Binary-Search Tree (二元搜尋樹)

## ► 二元搜尋樹的原理：

### ► 搜尋某個數字的時候分成兩種情況：

1. 小於等於：往左邊走
2. 大於：往右邊走





# Binary-Search Tree (二元搜尋樹)

## ▶ 二元搜尋樹的時間複雜度：

- ▶ 因為每一層都比前一層多兩倍的節點，所以最多  $\log_2 N$  層
- ▶ 我們會期望最多走  $\log_2 N$  層就找到我們要的數字。

如此一來，每次查找一個數字的時間複雜度會降到  $O(\log N)$ 。



# Binary-Search Tree (二元搜尋樹)

## ▶ 二元搜尋樹的時間複雜度：

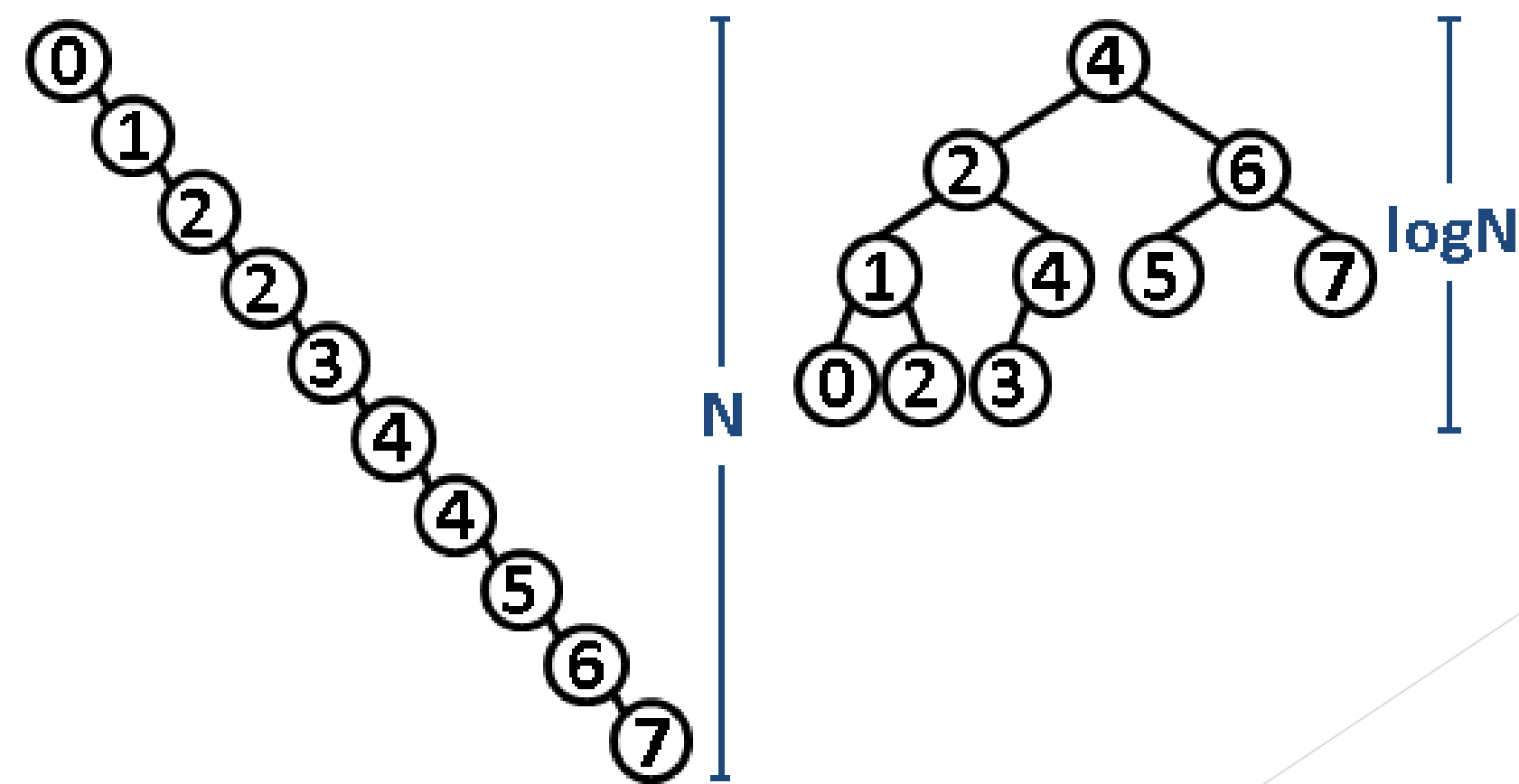
- ▶ 因為每一層都比前一層多兩倍的節點，所以最多  $\log_2 N$  層
- ▶ 我們會期望最多走  $\log_2 N$  層就找到我們要的數字。

如此一來，每次查找一個數字的時間複雜度會降到  $O(\log N)$ 。

## ▶ 但如果…二元搜尋樹是一條鏈呢？

- ▶ 不要讓它變成鏈就好啦！

(說的簡單，做起來很難)



set  
(集合)

# set (集合)

set是沒有重複元素的資料結構，  
也就是說如果你放入兩次 3，  
set裡面還是只有一個 3。

# set (集合)

## ► 宣告方式

```
set<int> st;
```

# set (集合)

- 函數應用- 新增、刪除、查找、取得size
- 時間複雜度： $O(\log N)$

`st.insert(x);` 在 set 裡面插入  $x$ ， $O(\log N)$ 。

`st.erase(x);` 在 set 裡面移除  $x$ 。

`st.find(x);` 在 set 裡面尋找是否有  $x$ ，如果有的話回傳該數字的iterator；否則回傳end()。

# set (集合)

- 當然也可以取得集合大小，時間複雜度 $O(1)$

`st.size(x);` 取得set當前大小

# set (集合)

- 遍歷set的方法

```
1  set<int> st;  
2  for(auto it=st.begin() ; it != st.end() ; ++it){  
3      cout << *it << ' '  
4  }
```

# set (集合)

- 我們隨便insert幾個數字會發現數字由小到大輸出
- 沒錯，在set當中begin()永遠是最小值

```
1  set<int> st;  
2  s.insert(3);  
3  s.insert(177);  
4  s.insert(9);  
5  for(set<int>::iterator it=st.begin() ; it != st.end() ; ++it){  
6      cout << *it << ' ';  
7  }  
8  //輸出 : 3 9 177  
9
```



# set (集合)

- 那最大值呢?
- end()的前一格，\*(--end())

```
1  set<int> st;  
2  s.insert(3);  
3  s.insert(177);  
4  s.insert(9);  
5  cout << *s.begin() << '\n'; //最小值 : 3  
6  cout << *(--s.end()) << '\n'; //最大值 : 177  
7
```

# 去除重複

## #QUESTION

給你一個整數  $N (\leq 100000)$ ，輸入  $N$  個整數，請你告訴我有幾個不一樣的數字，並由小到大輸出有哪些數字。

## #INPUT

輸入的第一行有一個數字  $N$  ( $N \leq 100000$ )，代表有幾個數字。  
第二行有  $N$  個數字用空格隔開。

## #OUTPUT

輸出共兩行。

第一行輸出一個整數  $K$ ，代表有幾種不一樣的數字。  
第二行由小到大輸出  $K$  個數字代表分別是那些數字。

# 去除重複

## #SOLUTION

code : <https://ideone.com/xBhDEW>

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  ▼ int main(){
4      ios::sync_with_stdio(false),cin.tie(0),cout.tie(0);
5      int n ; cin >> n ;
6      set<int> st;
7  ▼  for(int i=0 ; i<n ; ++i){
8          int x; cin >> x;
9          st.insert(x);
10     }
11     cout << st.size() << '\n';
12     for(auto it = st.begin() ; it != st.end() ; ++it){
13         cout << *it << ' ' ;
14     }
15 }
```

# 去除重複

#時間複雜度分析

輸入  $N$  個數字，每個數字都要 *insert*  $\rightarrow O(\log N)$ ，  
故時間複雜度為  $O(N * \log N) = O(N \log N)$

multiset(多重集合)

# multiset(多重集合)

set是沒有重複元素的資料結構，  
而multiset則是有重複的元素。

# multiset(多重集合)

- 宣告、函數都跟set一樣
- 但有些題目我們會需要重複的元素，所以需要multiset

# multiset(多重集合)

- 其實set、multiset的erase的參數可以傳入兩種

1. 原本的資料型態

2. iterator

```
int x;
```

```
mst.erase(x);
```

```
//移除所有數字 x
```

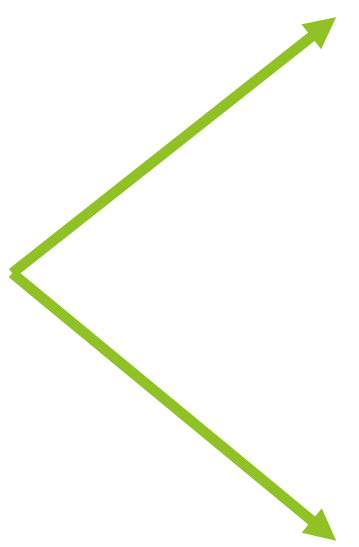
```
multiset<int>::iterator it
```

```
mst.erase(it)
```

```
//移除這個iterator，如果只移除單一元素就要先find(x)再用find出來的erase
```



# multiset(多重集合)

`mst.find(x)` 

- `return` 其中一個`x`的`iterator` //有找到
- `return mst.end()` //沒找到

- `find()`有兩種回傳結果，分別為找到跟沒找到

# 實戰演練

## 輸入說明

輸入只有一行,共有不定數量的整數,

整數可為{-2, -1, 0, 1, 2, ..., 10000},兩整數之間以一個空白隔開。

-2 表示印表機 P1, P2, ..., Pk其中一台有空, 可以列印最高優先權的工作;

-1 表示印表機 Pk+1, Pk+2, ..., Pn其中一台有空, 可以列印最低優先權的工作;1, 2, ..., 10000

代表新增一個優先權為該數字之工作;

0 則代表輸入結束。

若輸入為 -1 或 -2但無等待列印的工作,則不列印,需等待下一個 -1 或 -2 才再列印新的工作。

## 範例輸入 #1

```
// Example 1
20 15 10 -2 -1 -1 0

// Example 2
1 2 3 -2 4 5 6 -1 7 0
```

<https://zerojudge.tw/ShowProblem?problemid=c421>

# 實戰演練

## 239. 滑动窗口最大值

难度 困难  1036  收藏  分享  切换为英文  接收动态  反馈

给你一个整数数组 `nums`，有一个大小为 `k` 的滑动窗口从数组的最左侧移动到数组的最右侧。你只可以看到在滑动窗口内的 `k` 个数字。滑动窗口每次只向右移动一位。

返回滑动窗口中的最大值。

示例 1：

输入：nums = [1,3,-1,-3,5,3,6,7], k = 3

输出：[3,3,5,5,6,7]

解释：

滑动窗口的位置	最大值
-----	-----
[1 3 -1] -3 5 3 6 7	3
1 [3 -1 -3] 5 3 6 7	3
1 3 [-1 -3 5] 3 6 7	5
1 3 -1 [-3 5 3] 6 7	5
1 3 -1 -3 [5 3 6] 7	6
1 3 -1 -3 5 [3 6 7]	7

[leetcode.com/problems/sliding-window-maximum/](https://leetcode.com/problems/sliding-window-maximum/)

map

# map

map簡單來說是一種  
： ” 自己決定索引值 ” 的資料結構。

# map

- 宣告方式。
- 我們稱索引值叫做Key(關鍵字)
- 又稱資料形態裡的叫做Value(值)

```
map<char , int> cnt;
```

```
map<索引值型態 , 資料型態> 變數名稱;
```

```
map<Key , Value> name;
```

# map

- 所以使用map的時候，是用關鍵字(Key)找到對應的值(Value)
- 就很想我們用陣列的索引值找到對應的值，只是現在索引值不是int了~

# map

- 用map我們就可以把任何資料形態當作索引值囉!
- 我們拿計算字母數量當作例子!
- 此時我們的關鍵字就是字母~ 然後用關鍵字來找數量

```
1  map<char , int> cnt;  
2  cnt['a']++; // O(logN)  
3  cnt['A']++; // O(logN)  
4  cnt['a']++; // O(logN)  
5  //在map中，我們可以假設所有數字都初始化為0了~  
6  //此時 cnt['a'] == 2 噲!
```



# 實戰演練

## 451. 根据字符出现频率排序

难度 中等 256 收藏 分享 切换为英文 接收动态 反馈

给定一个字符串，请将字符串里的字符按照出现的频率降序排列。

示例 1:

输入:  
"tree"

输出:  
"eert"

解释:  
'e' 出现两次，'r' 和 't' 都只出现一次。  
因此 'e' 必须出现在 'r' 和 't' 之前。此外，"eetr" 也是一个有效的答案。

[leetcode-cn.com/problems/sort-characters-by-frequency/](https://leetcode-cn.com/problems/sort-characters-by-frequency/)

# 實戰演練

## 692. 前K个高频单词

难度 中等 350 收藏 分享 切换为英文 接收动态 反馈

给一非空的单词列表，返回前  $k$  个出现次数最多的单词。

返回的答案应该按单词出现频率由高到低排序。如果不同的单词有相同出现频率，按字母顺序排序。

示例 1：

输入: ["i", "love", "leetcode", "i", "love", "coding"],  $k = 2$

输出: ["i", "love"]

解析: "i" 和 "love" 为出现次数最多的两个单词，均为2次。

注意，按字母顺序 "i" 在 "love" 之前。

<https://leetcode-cn.com/problems/top-k-frequent-words/>

priority\_queue

# priority\_queue

*priority\_queue* 是 *queue* 的翻版，

- *queue* : 最早進入的最早出來
- *priority\_queue* : 最大的最早出來

# priority\_queue

- 宣告方式。

```
priority_queue<int> pq; // 宣告
```

# priority\_queue

- 移除元素、放入元素： $O(\log N)$
- 取出最大值： $O(1)$
- 由此可知，
- *priority\_queue* 取出最大值的速度很快喔！

```
1  priority_queue<int> pq; // 宣告
2  pq.push(3); //放入元素 O(logN)
3  pq.push(7); //放入元素
4  pq.push(1); //放入元素
5  cout << pq.top() << '\n'; //輸出 7 ， pq.top()是O(1)喔
6  pq.pop(); // O(logN)
7  cout << pq.top() << '\n'; //輸出3
```

# priority\_queue

- 如果你想要top()是最小值...
- 因為C++所有容器都默認less所以改成greater就好

```
priority_queue<int , vector<int> , greater<int> > pq;
```

# 實戰演練

## 輸入說明

輸入只有一行,共有不定數量的整數,

整數可為{-2, -1, 0, 1, 2, ..., 10000},兩整數之間以一個空白隔開。

-2 表示印表機 P1, P2, ..., Pk其中一台有空, 可以列印最高優先權的工作;

-1 表示印表機 Pk+1, Pk+2, ..., Pn其中一台有空, 可以列印最低優先權的工作;1, 2, ..., 10000

代表新增一個優先權為該數字之工作;

0 則代表輸入結束。

若輸入為 -1 或 -2但無等待列印的工作,則不列印,需等待下一個 -1 或 -2 才再列印新的工作。

## 範例輸入 #1

```
// Example 1
20 15 10 -2 -1 -1 0

// Example 2
1 2 3 -2 4 5 6 -1 7 0
```

- 配合使用 `pair<type1,type2>`

<https://zerojudge.tw/ShowProblem?problemid=c421>