主題：二元搜尋常用演算法實作與比較

組員：陳羿錦　　指導老師：鄭瑞南

一 動機

當我在練習程式設計題目的時候，看見了一個有關搜尋的題目，題目敘述是這樣的，給定一個數量為 n 的數列（由小到大），並給予一個數字 N，判斷他是否有出現在數列，這個題目直覺地讓我想到二元搜尋法與之相關的二元搜尋樹的演算法，於是我決定用這兩種來時做看看並比較其差異。

二、樹狀結構定義：

樹狀結構的定義為每個節點(node)之間都可以找到路徑(edge)連通，但不會形成循環，且設定其中一個點為根節點(root)，與根節點相連的子樹(子樹 1、子樹 2、…與子樹 n)，任兩個子樹之間沒有路徑(edge)相連，若可以連通就會形成循環，且子樹 1、子樹 2、…與子樹 n 也都是樹狀資料結構。

如此圖，node 1 ~ 9 每個點之間都可以找到 edge 連通，且沒有形成循環。

node 1 為 root，其下方有三個子樹，子樹之間沒有邊相連，node 2、3 與 4 也是子樹。

三、二分搜尋法

定義： 如果資料已先排序過，則可使用二分法來進行搜尋。 二分法是將資料分成兩部份，再將鍵值(key 值為欲搜尋的值)與中間值比較，如鍵值相等則找到，如小於中間值則比前半段，如大於中間值則比後半段。 如此，分段比較至找到或無資料為止。

四、二分搜尋樹

定義： 二元搜尋樹法是先將資料列建立為一棵二元搜尋樹，樹中每節點皆不小於左子樹(葉)，也不大於右子樹(葉)，也就是 左子樹(lnode)的值≦樹根(root)值≦右子樹(rnode)的值。除去樹的結構概念，其餘概念即為二元搜尋演算法。

五、題目輸出入說明

輸入說明：
第一行：輸入 n (1 <= n <= 10000)
第二行：輸入 n 個數字 P (1 <= P <= 10000，且不重複由小到大輸入)，代表此數列各個元素。
第三行：輸入要尋找的數字 N
輸出說明：
第一行：輸出 Yes 或 No

以二元搜尋樹程式實作：

```
#include <bits/stdc++.h>
using namespace std ;
int n ;
struct node          // 宣告節點結構
{
        double num ;    // 節點名稱
        node *lnode ;   // 左子節點
        node *rnode ;   // 右子節點
} ;
// rnode > num > lnode (恆不等式，二分搜尋樹之定義)

bool search ( double obj , node *ptr , node *root )
// 搜尋 函式
{ptr = root ;
while ( true ){
if ( ptr -> num == obj )   // 判斷是否找到數值在樹中
        return true ;
else if ( ptr -> num < obj && ptr -> rnode != NULL )
        ptr = ptr -> rnode ;
// 假如 ptr 的名稱(num) 比 obj (搜尋值)小 且 rnode 非 空元素 則向右尋找
else if ( ptr -> num < obj && ptr -> rnode == NULL )
        return false ;
// 但 如果 rnode 為 空元素 表示 沒有比 ptr 大的數值在樹中了
else if ( ptr -> num > obj && ptr -> lnode != NULL )
        ptr = ptr -> lnode ;
//則向右尋找
else if ( ptr -> num > obj && ptr -> lnode == NULL )
        return false ;
//表示 沒有比 ptr 小的數值在樹中了
} }

void insert ( double obj , node *ptr , node *root )
// 宣告 插入元素 的函式
{ node *newNode = new node ;
// 配置空間 並 宣告 newNode(新的元素)
newNode -> num = 0 ; // newNode 初始化
newNode -> rnode = NULL ;newNode -> lnode = NULL ;
ptr = root ;
while ( true ){
if ( ptr -> num == obj ) // 判斷該元素是否早在樹中出現了
{cout << "We have it in the tree!!" << endl ;break ;}
else if ( ptr -> num > obj && ptr -> lnode != NULL )
        ptr = ptr -> lnode ;
 // 假如 ptr 的名稱 (num)  比 obj (搜尋值) 大 且 lnode 非 空元素 則向左繼續向下延伸子節點
else if ( ptr -> num > obj && ptr -> lnode == NULL )
{// 但 如果 lnode 為 空元素 表示 沒有比 ptr 小的數值在樹中了
ptr -> lnode = newNode ;
newNode -> num = obj ; break ;}
else if ( ptr -> num < obj && ptr -> rnode != NULL )
        ptr = ptr -> rnode ;
        //向右繼續向下延伸子節點
else if ( ptr -> num < obj && ptr -> rnode == NULL )
{
ptr -> rnode = newNode ; newNode -> num = obj ;break ;}}}
// 所以 ptr 的 rnode 就是此數 儲存起來


int main ()
{
double num ;            // 宣告變數
node *root = new node ; // 宣告 root 並初始化
root -> num = 1000 ;root -> lnode = NULL ;
root -> rnode = NULL ;
node *ptr ;
ptr = root ;

cin >> n ;

for ( int i = 0 ; i < n ; i++ ) // 建立二元搜尋樹
{
   int point ;
   cin >> point ;
   insert( point , ptr , root ) ;
}

int find_point ;
cin >> find_point ;
if ( search( find_point , ptr , root ) ) // 利用二元搜尋樹進行搜尋
        cout << "Yes\n" ;
else
        cout << "No\n" ;
return 0 ;
}
```

以二元搜尋法程式實作：

```
#include <iostream>
using namespace std ;
const int nmax = 10000 ;
int Binary[nmax] ;   // 儲存原陣列
int n ;              // 儲存的數字數量
bool Binary_search( int tmp ) // tmp 是 尋找值
{
        int max = n-1 , min = 0 ;
        while ( max > min )
        {
                int mid = ( max + min ) / 2 ;
                if ( Binary[ mid ] < tmp )
                        min += mid + 1 ;
        //當中心的位置<tmp 代表 tmp 在陣列的右邊
                else if ( Binary[ mid ] > tmp )
                        max = mid ;
        // 反之 則代表在 左邊的陣列
                else
                        return true ;
        // 代表 ( Binary[mid] == tmp ) 找到了 回傳 true
        }
        return false ;
        // 代表 tmp 不在陣列之中 回傳 false
}
int main (){
        cin >> n ;
        for ( int i = 0 ; i < n ; i++ )
        {
                cin >> Binary[i] ;
                // 輸入陣列元素
        }
        int find_point ;
        cin >> find_point ;
         // 輸入要尋找的數字 ( 元素)
        if ( Binary_search( find_point ) )
                cout << "Yes\n" ;
        else
                cout << "No\n" ;
        return 0 ;}
```

六、比較與結論

綜合比較：

在程式執行時間上，建樹版會比一般無建樹來得久一些，在程式執行空間上，由於無建樹需要預先宣告空間，於是建樹版會略勝一籌。

結論：

不管事有建樹的還是無建樹的，兩者皆有各自的好處與壞處，於是並沒有所謂確切的好壞分割，只有哪一種題目適合哪一種類型而已。建樹版較為複雜，比較不建議初學者去練習，相較之下，無建樹會比較親民一點。各有所長，只看如何運用。

適合題目練習與難易度優缺點比較：

	二元搜尋法	二元搜尋樹
適合題目	純粹陣列的搜尋模式，無上下或前後關係的記憶體儲存方式	有上下關係的記憶體儲存模式，有上下根子節點關係的題目形式
難易度優缺點 比較	較簡，只需陣列即可處理，但可能會過度使用到不必要的記憶體空間	較難，需要自行建構 struct 和指標去處理，但可省一些記憶體空間

七、資料來源

搜尋：

http://spaces.isu.edu.tw/upload/18833/3/web/search.htm#_Toc231546392

點部落-優游在技術的海洋：

https://dotblogs.com.tw/j883988/2013/11/15/129675