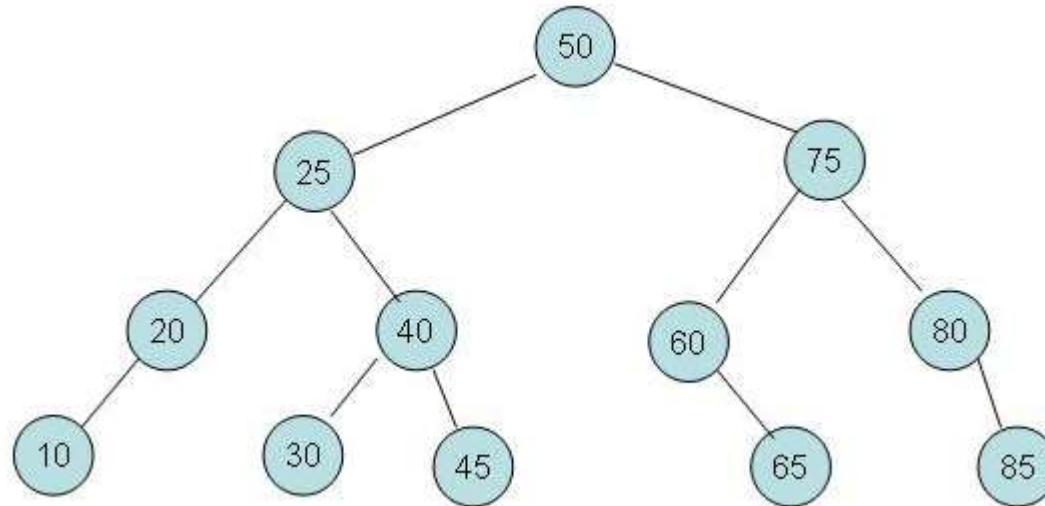


# Binary Search Trees



Alberi binari di ricerca (binary search trees – **BST**):

Strutture dati costituite da **nodi**, collegati tra loro mediante **archi o rami**.

Ogni nodo ha un solo arco entrante (tranne il nodo **radice** che non ne ha).

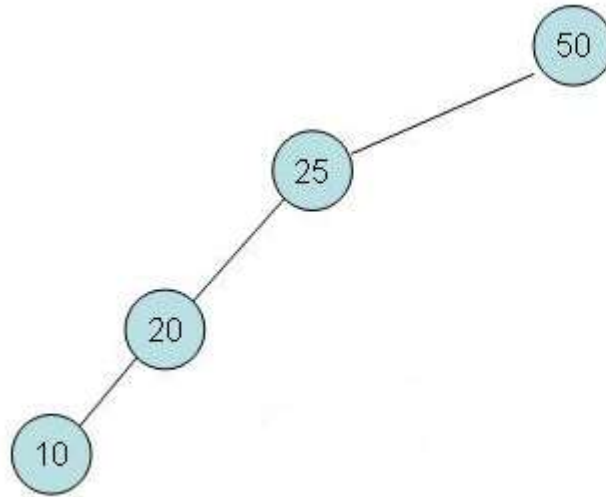
I nodi possono avere al più **due figli** (fattore di ramificazione binario).

Un nodo è detto **foglia** se non ha archi uscenti.

L'**altezza** dell'albero è la profondità massima  $h$  a cui si trovano i nodi foglia (la radice ha altezza 0).

Un albero è **ordinato** se, per ogni nodo, il suo valore non è minore dei valori contenuti nel sottoalbero sinistro e allo stesso tempo non è maggiore dei valori contenuti nel sottoalbero destro.

# Binary Search Trees

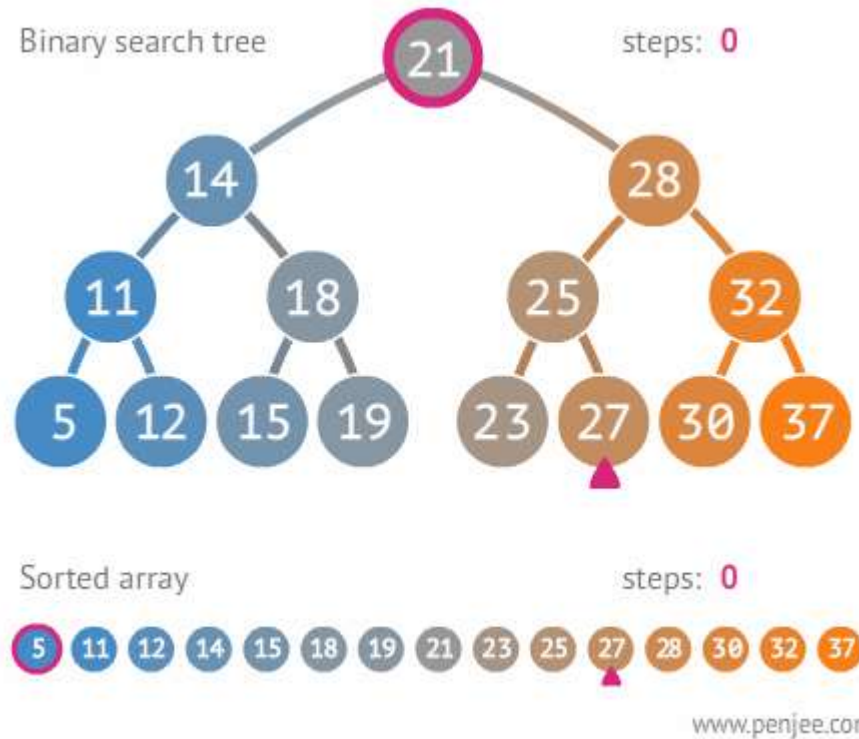


Un albero è:

- **Pieno** o Completo – se tutti i nodi hanno due foglie e comunque altezza  $h$  o  $h-1$  (quasi pieno in quest'ultimo caso)
- **Bilanciato** – se tutte le foglie hanno altezza  $h$  o  $h-1$  (perfettamente bilanciato nel primo caso)

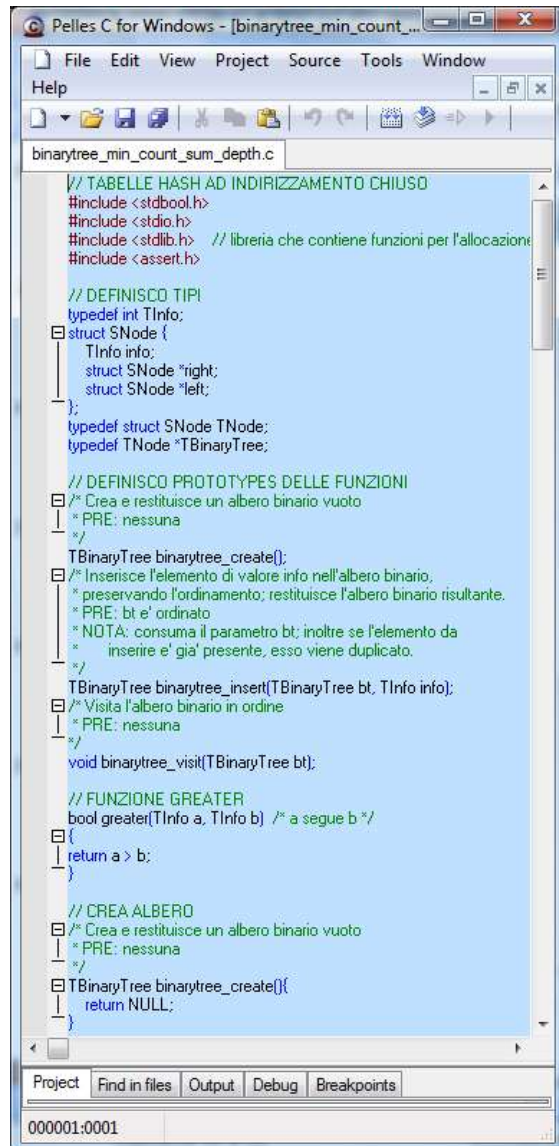
In un albero ordinato la complessità computazione di un'operazione di **ricerca** è logaritmica  $\Theta[\log(n)]$ , e dipende infatti anche dall'altezza  **$h$**  – nei casi di non pienezza la complessità può quindi diventare funzione  $\Theta[(n)]$  esattamente come nelle liste dinamiche.

# Binary Search Trees



Esempio: differenza nel numero di passi – se cerco il numero 27 in un ALBERO BINARIO DI RICERCA ottimizzato [3], o in un semplice ARRAY ORDINATO [10]

# Binary Search Trees



```
// TABELLE HASH AD INDIRIZZAMENTO CHIUSO
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h> // libreria che contiene funzioni per l'allocazione
#include <assert.h>

// DEFINISCO TIPI
typedef int TInfo;
struct SNode {
    TInfo info;
    struct SNode *right;
    struct SNode *left;
};
typedef struct SNode TNode;
typedef TNode *TBinaryTree;

// DEFINISCO PROTOTYPES DELLE FUNZIONI
/* Crea e restituisce un albero binario vuoto
 * PRE: nessuna
 */
TBinaryTree binarytree_create();
/* Inserisce l'elemento di valore info nell'albero binario,
 * preservando l'ordinamento; restituisce l'albero binario risultante.
 * PRE: bt e' ordinato
 * NOTA: consuma il parametro bt; inoltre se l'elemento da
 * inserire e' gia' presente, esso viene duplicato.
 */
TBinaryTree binarytree_insert(TBinaryTree bt, TInfo info);
/* Visita l'albero binario in ordine
 * PRE: nessuna
 */
void binarytree_visit(TBinaryTree bt);

// FUNZIONE GREATER
bool greater(TInfo a, TInfo b) /* a segue b */
{
    return a > b;
}

// CREA ALBERO
/* Crea e restituisce un albero binario vuoto
 * PRE: nessuna
 */
TBinaryTree binarytree_create(){
    return NULL;
}
```

## BINARYTREE\_MIN\_COUNT\_SUM\_DEPTH.c

- # creazione di un BST ordinato
- # sua visita/stampa in ordine, pre-ordine e post-ordine
- # ricerca del nodo con valore minimo
- # conteggio delle foglie
- # somma dei valori dei nodi
- # calcolo dell'altezza

