

Algoritmi e Strutture Dati

Lezione 3

www.iet.unipi.it/a.virdis

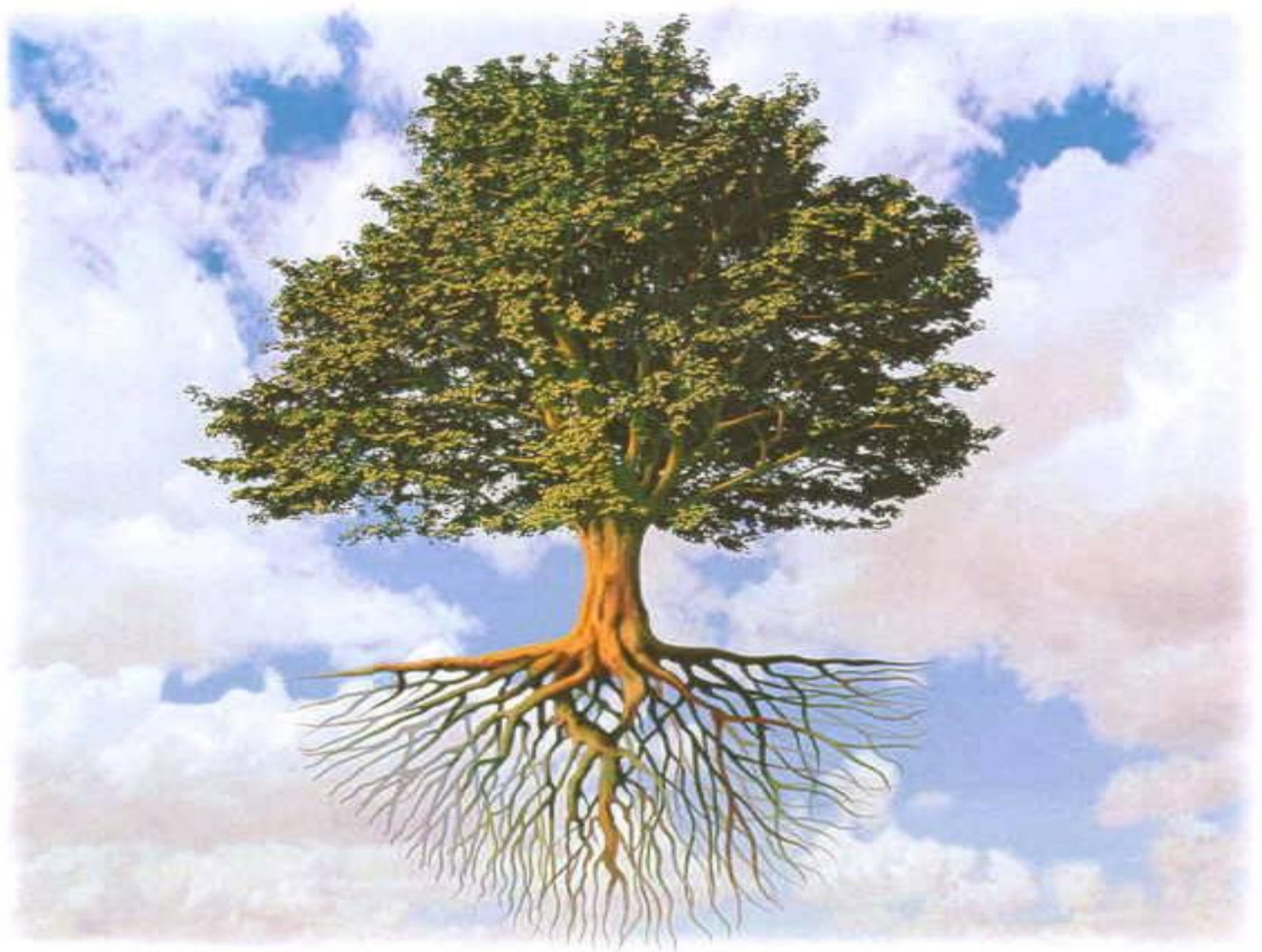
Antonio Virdis

antonio.virdis@unipi.it



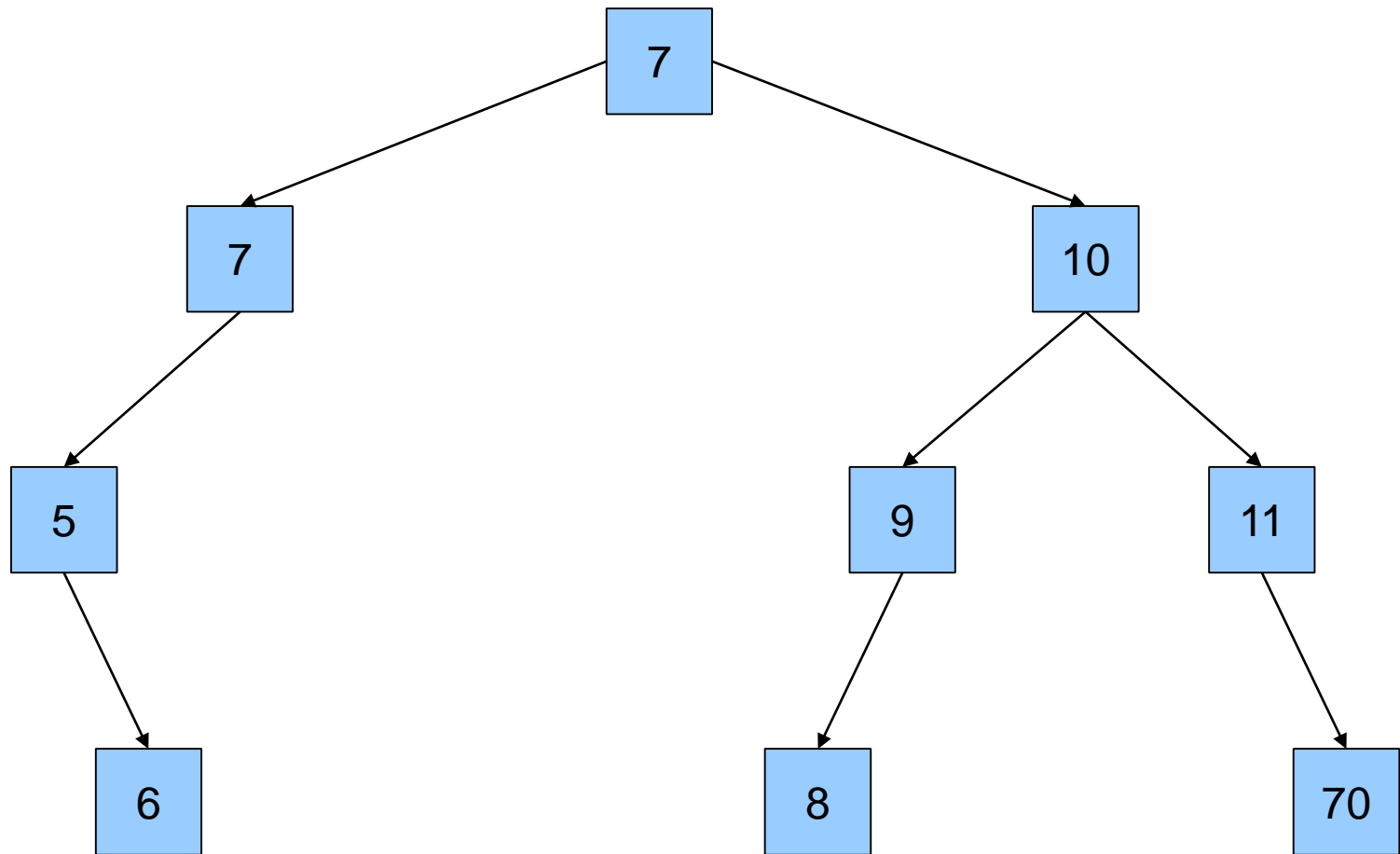
Sommario

- Alberi Binari di Ricerca
- Gestione Stringhe
- Progettazione
- Esercizi

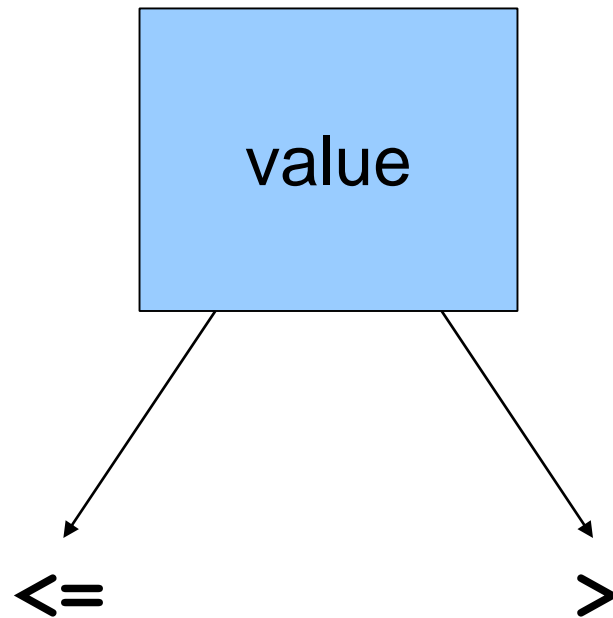




Alberi Binari



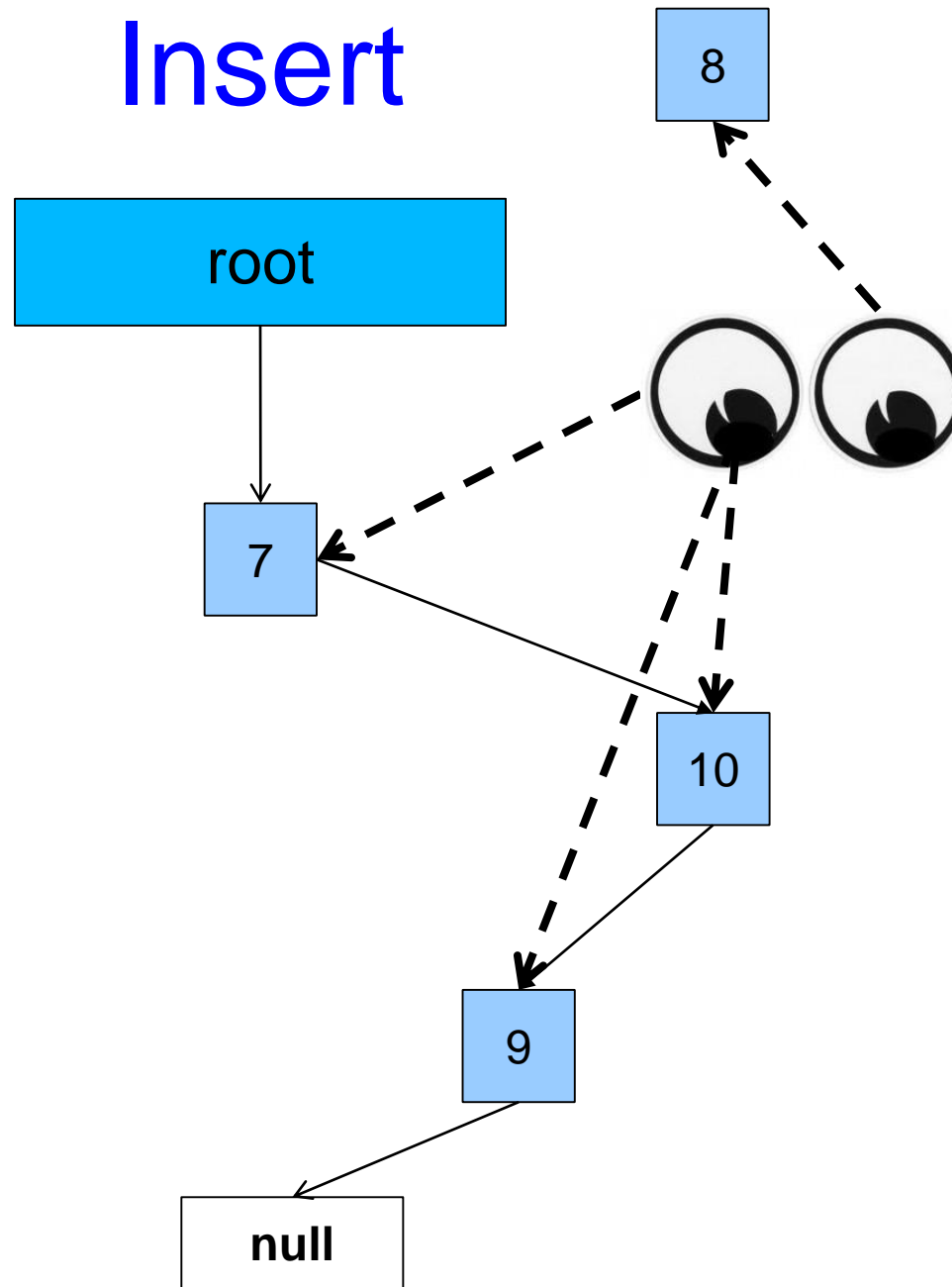
Alberi Binari di Ricerca



binTree

```
1 struct Node
2 {
3     int value;
4     Node * left;
5     Node * right;
6
7     Node(int val) :
8         value(val) , left(NULL) , right(NULL) {}
9 };
10
11 class BinTree
12 {
13     Node * root_;
14
15 public:
16
17     BinTree() { root_ = NULL; }
18
19     Node * getRoot() { return root_; }
20 }
```

Insert



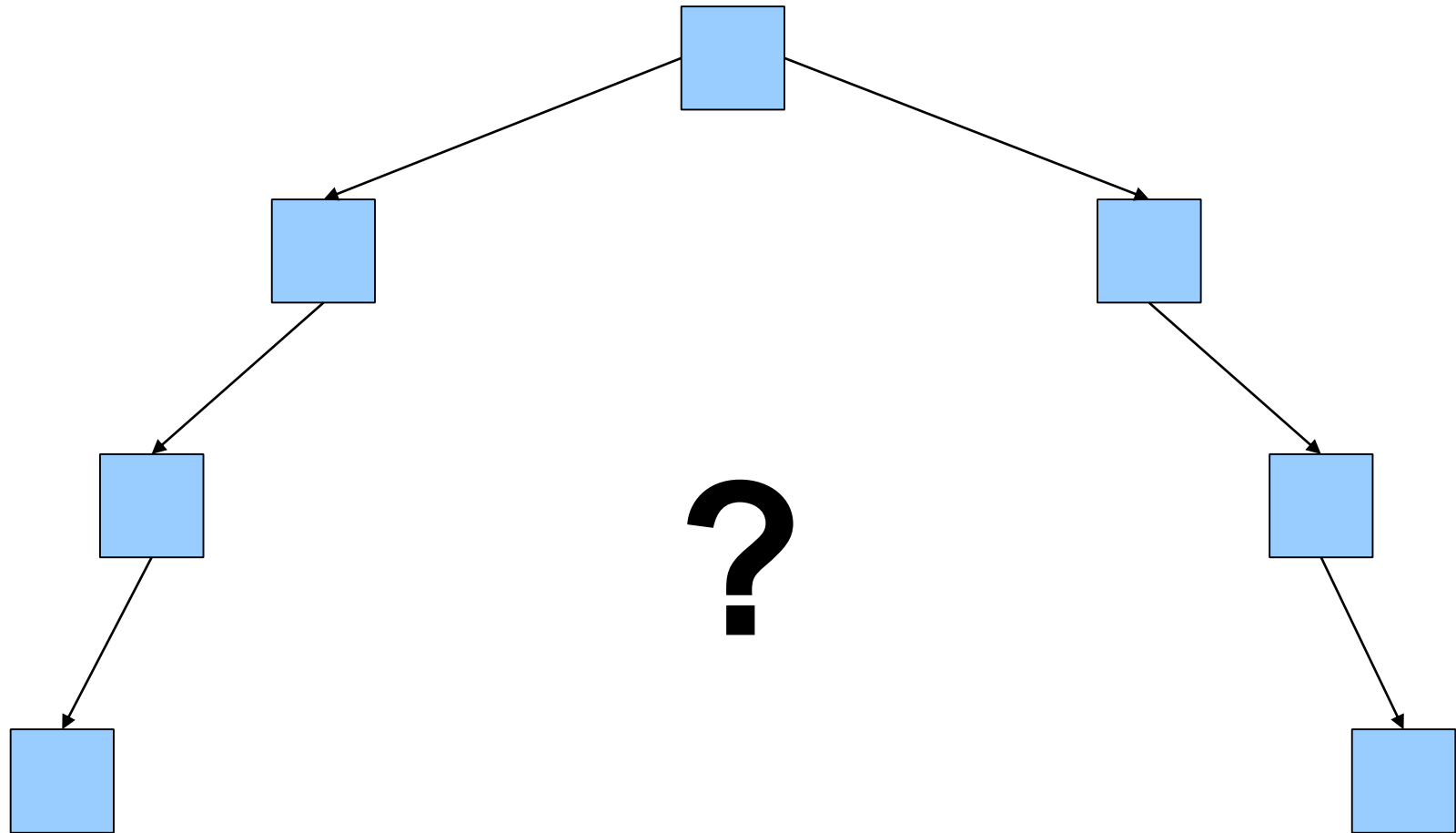
Insert

```
1 void insert( int val )
2 {
3     // inizializzo nuovo elemento    INIZIALIZZAZIONE
4     // inizializzo variabili appoggio
5
6
7     // finchè non arrivo ad una foglia
8     {
9         // aggiorno variabili
10        // se <=
11            // vado a sinistra
12        // altrimenti
13            // vado a destra
14    }
15    // se albero vuoto
16        // aggiorno radice
17
18    // decido se diventare figlio left o right
19
20
21 }
```

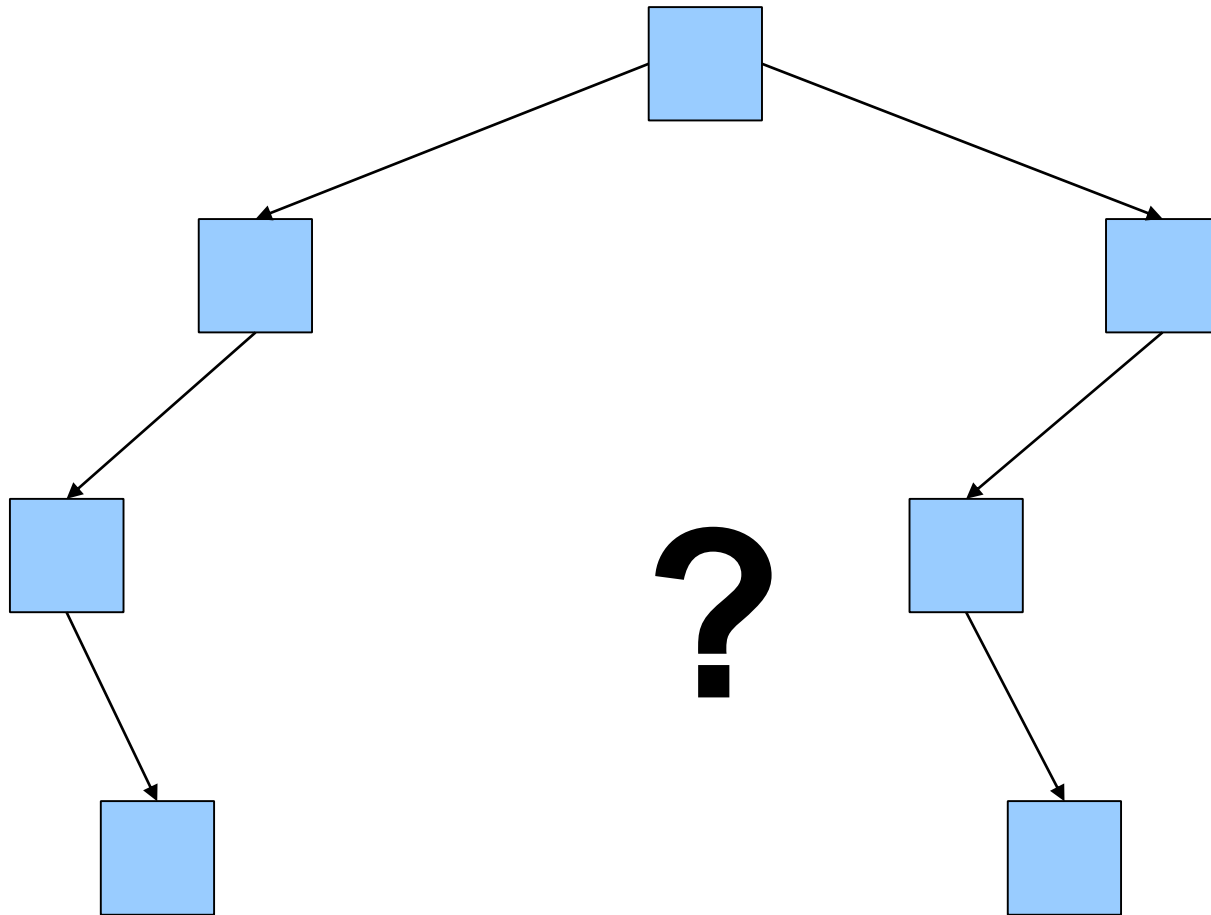
Insert

```
1 void insert( int val )
2 {
3     Node * node = new Node(val);           INIZIALIZZAZIONE
4     Node * pre = NULL;
5     Node * post = root_;
6
7     while( post != NULL )
8     {
9         pre = post;
10        if( val <= post->value )
11            post = post->left;               INDIVIDUO
12        else                                     POSIZIONE
13            post = post->right;
14    }
15    if( pre == NULL )                          INSERIMENTO
16        root_ = node;
17    else if( val <= pre->value )
18        pre->left = node;
19    else
20        pre->right = node;
21 }
```

min & MAX



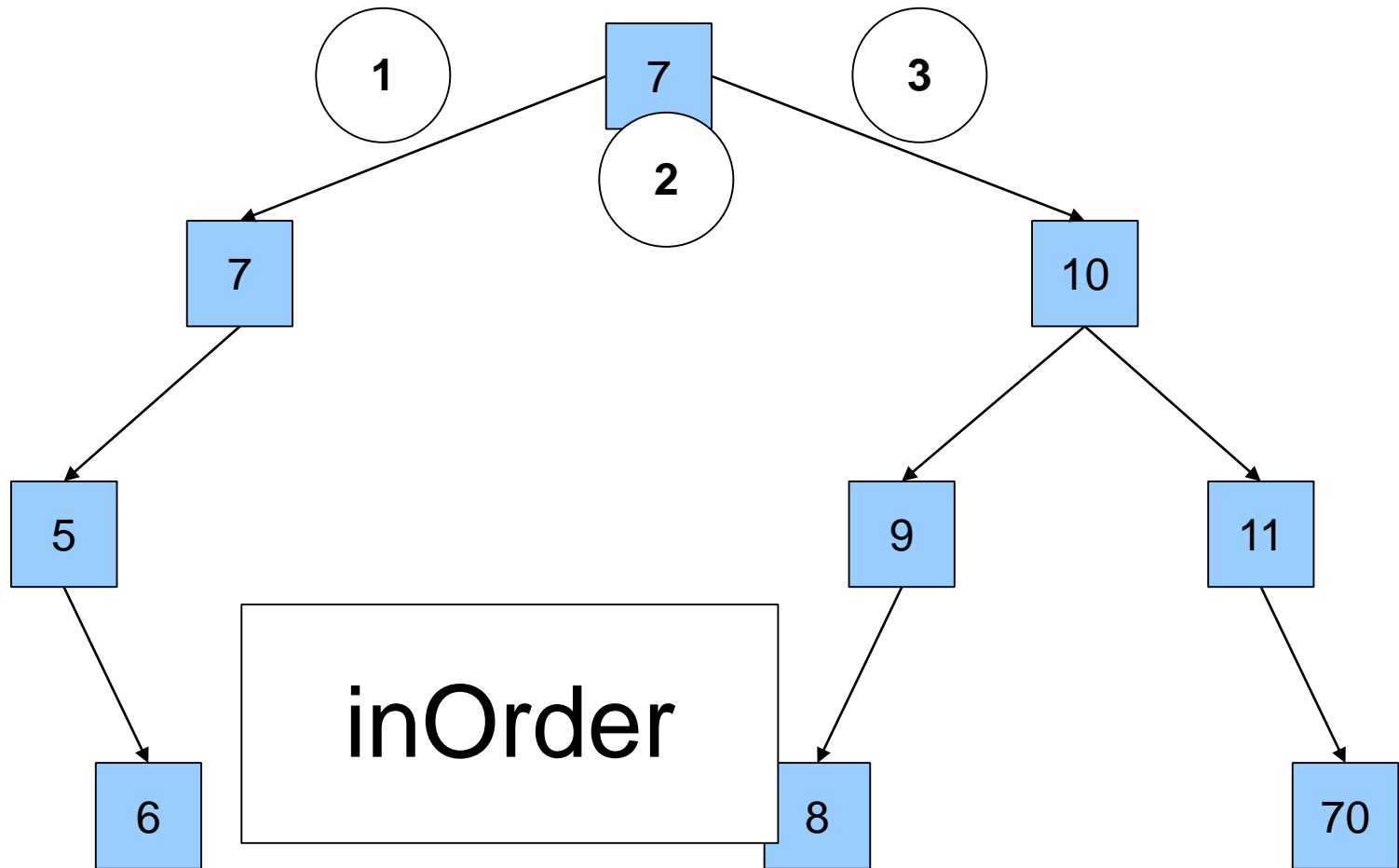
min & MAX



Min/Max

```
1 Node * min()  
2 {  
3     Node * temp = root_;  
4     while( temp->left != NULL )  
5         temp = temp->left;  
6     return temp;  
7 }  
8  
9  
10  
11 Node * max()  
12 {  
13     Node * temp = root_;  
14     while( temp->right != NULL )  
15         temp = temp->right;  
16     return temp;  
17 }  
18  
19  
20
```

visite



In-Order

```
1
2
3
4 void inOrder( Node * tree )
5 {
6     // se non sono in una foglia
7     {
8
9         // visito verso left
10
11        // stampo questo valore
12
13        // visito verso right
14
15    }
16 }
17
18
19
20
```

In-Order

```
1
2
3
4 void inOrder( Node * tree )
5 {
6     if(tree!=NULL)
7     {
8
9         visitaNode(tree->left);
10
11         cout << tree->value << "\t";
12
13         visitaNode (tree->right);
14
15     }
16 }
17
18
19
20
```

In-Order

```
1
2
3
4 void inOrder( Node * tree )
5 {
6     if (tree!=NULL)
7     {
8
9         inOrder (tree->left) ;
10
11         cout << tree->value << "\t";
12
13         inOrder (tree->right) ;
14
15     }
16 }
17
18
19
20
```

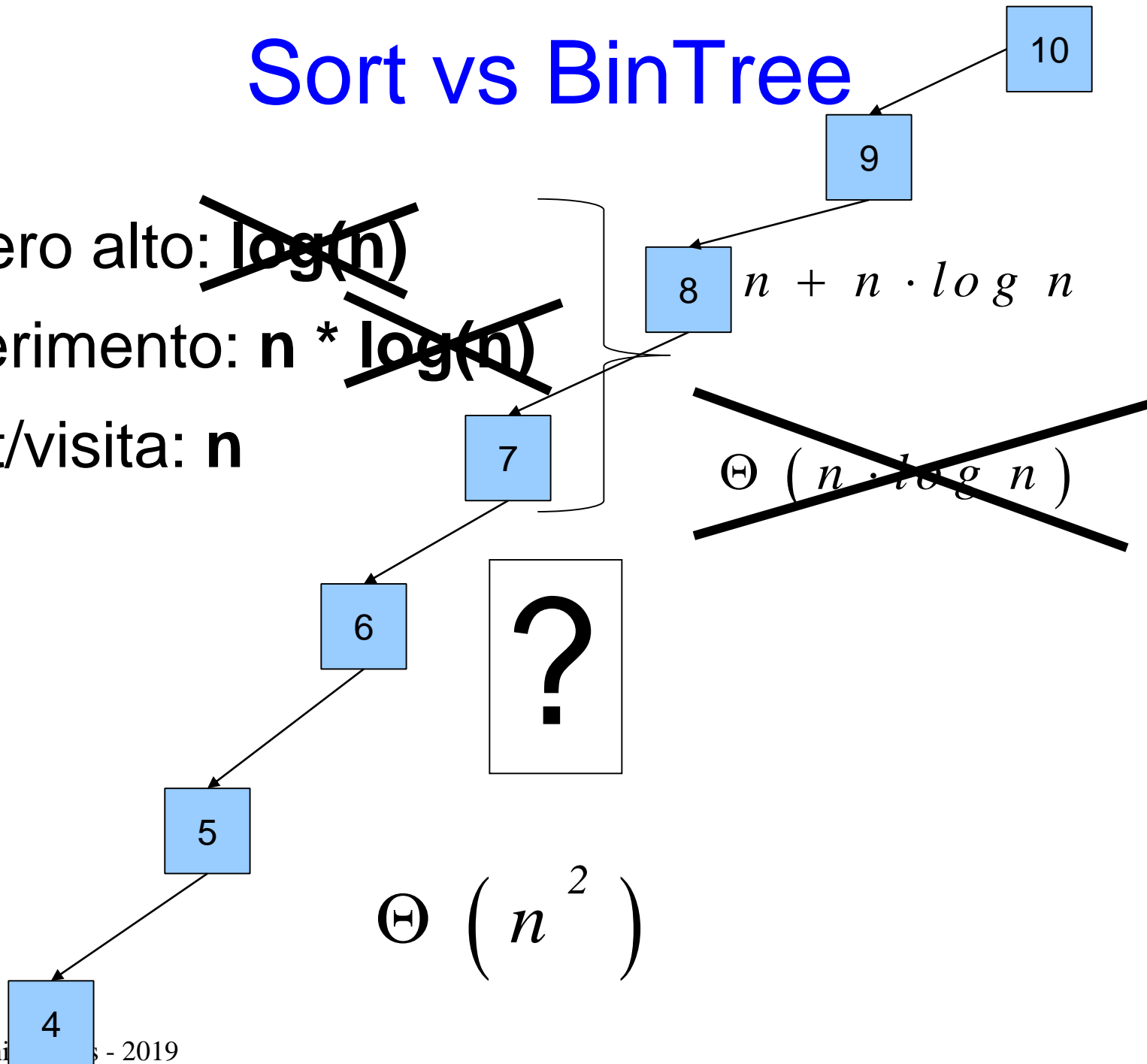
Sort vs BinTree

- Albero alto: $\log(n)$
 - Inserimento: $n * \log(n)$
 - Sort/visita: n
- $n + n \cdot \log n$
- $\Theta (n \cdot \log n)$

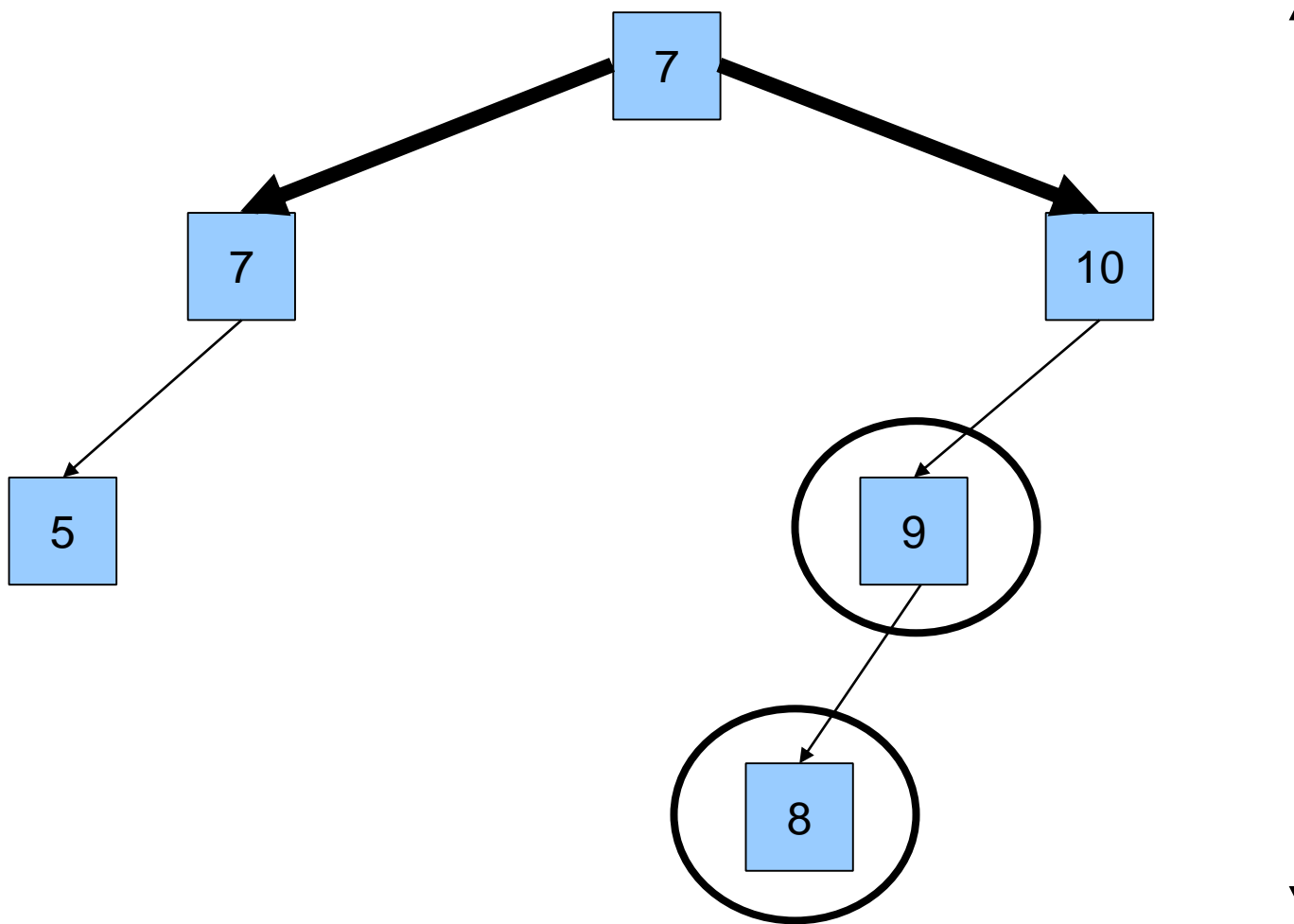
giusto?

Sort vs BinTree

- Albero alto: ~~$\log(n)$~~
- Inserimento: ~~$n * \log(n)$~~
- Sort/visita: n



Height



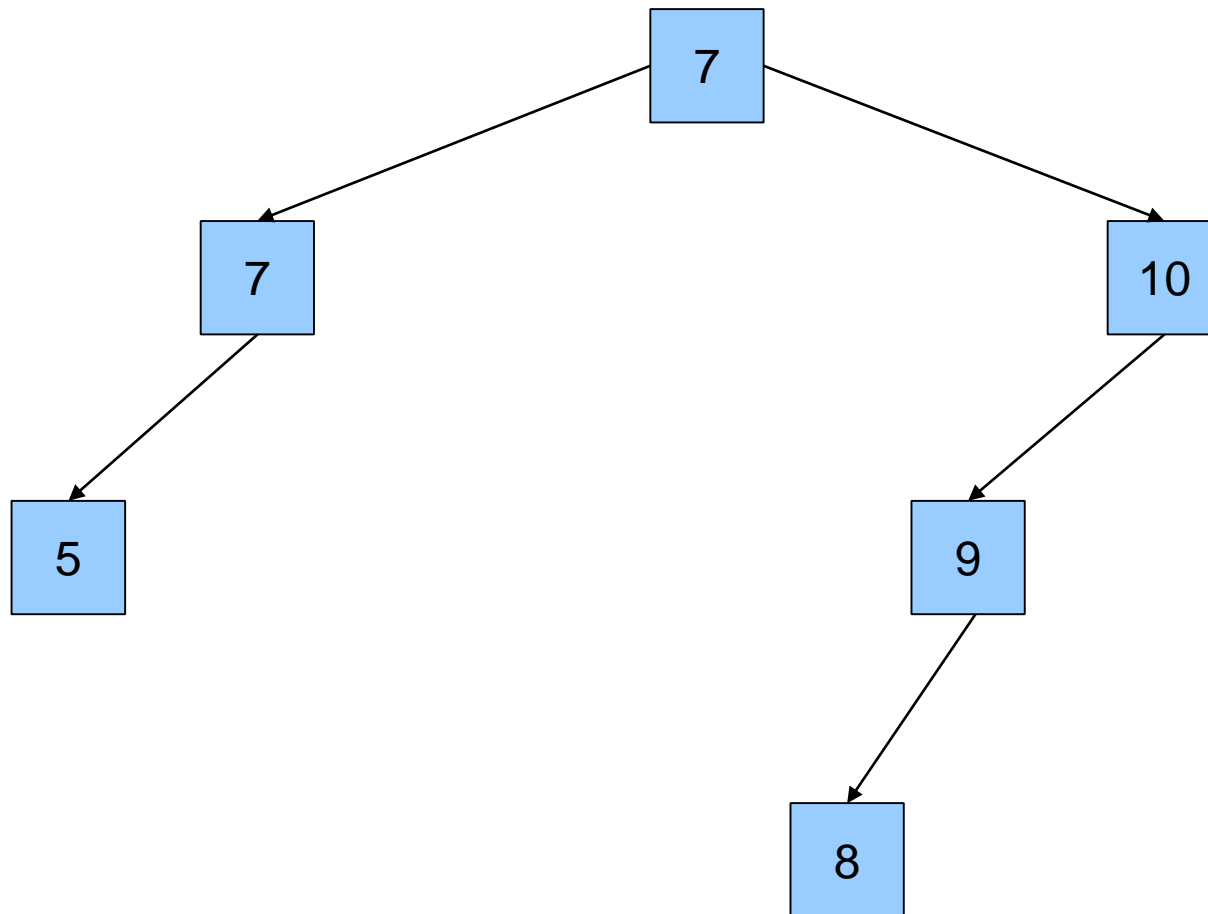
Altezza albero

```
1  int height( Node * tree )
2  {
3
4      int hLeft;
5      int hRight;
6
7
8      if( tree == NULL )
9          return 0;
10
11
12     hLeft = height(tree->left) ;
13
14     hRight = height(tree->right) ;
15
16
17     return 1 + max(hLeft, hRight) ;
18 }
19
20
```

Trova chiave

- Dato
 - Un albero binario con valori distinti
 - Un valore K
- Trovare
 - Se il valore esiste

Search K=9



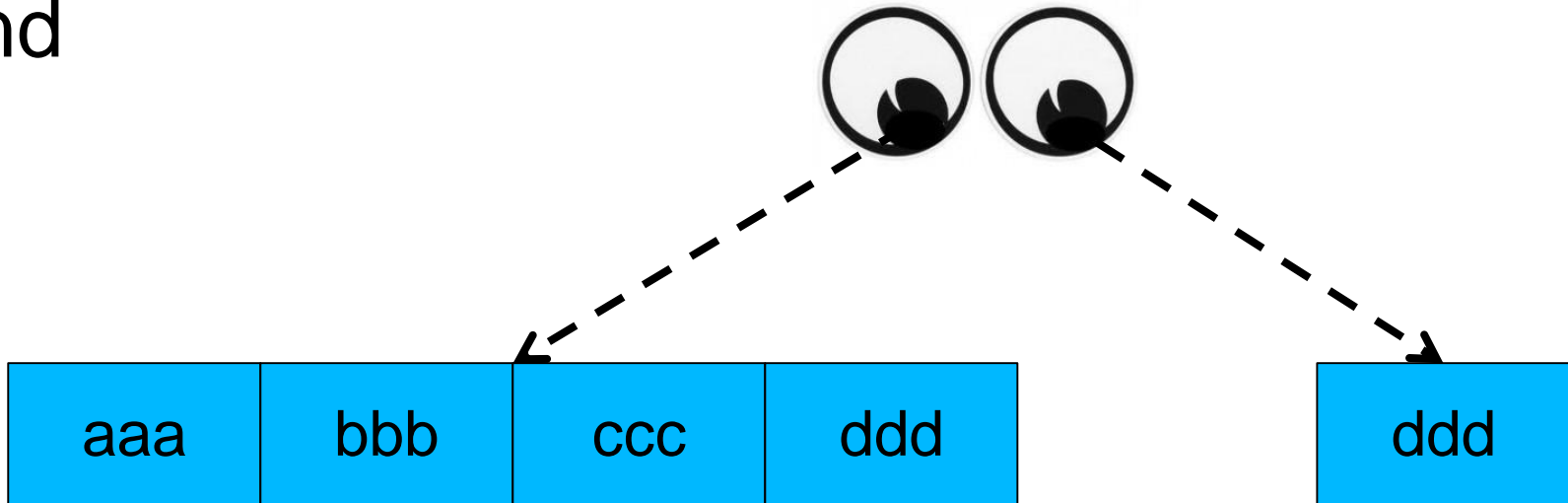
Search

```
1  bool search( Node * tree , int val )
2  {
3      if( tree == NULL )
4          return false;
5
6      bool found;
7
8      if( tree->value == val )
9          return true;
10
11
12      else if( val <= tree->value )
13          found = search( tree->left , val );
14      else
15          found = search( tree->right , val );
16
17
18      return found;
19  }
20
```

**INDIVIDUO
DIREZIONE**

stringhe

- Creazione
- Concatenazione
- Compare
- Find



Stringhe

```
1  #include <string>
2
3  String parola = "liste";
4
5  String frase = "mi piacciono le liste";
6
7  String parola2 = "non ";
8
9  String frase2 = parola2 + frase;
10
11 frase.find(parola);
12 // se fallisce -> string::npos
13
14 parola.compare(parola2);
```

<http://www.cplusplus.com/reference/string/string/>



Esercizio Stringhe

- Input
 - Una testo T formato da più parole
 - Un insieme S di N parole
- Output:
 1. Le parole di S *contenute* in T, ordinate per posizione in T (insieme R1)
 2. Le parole di S *non contenute* in T, in ordine lessicografico (insieme R2)

Analisi

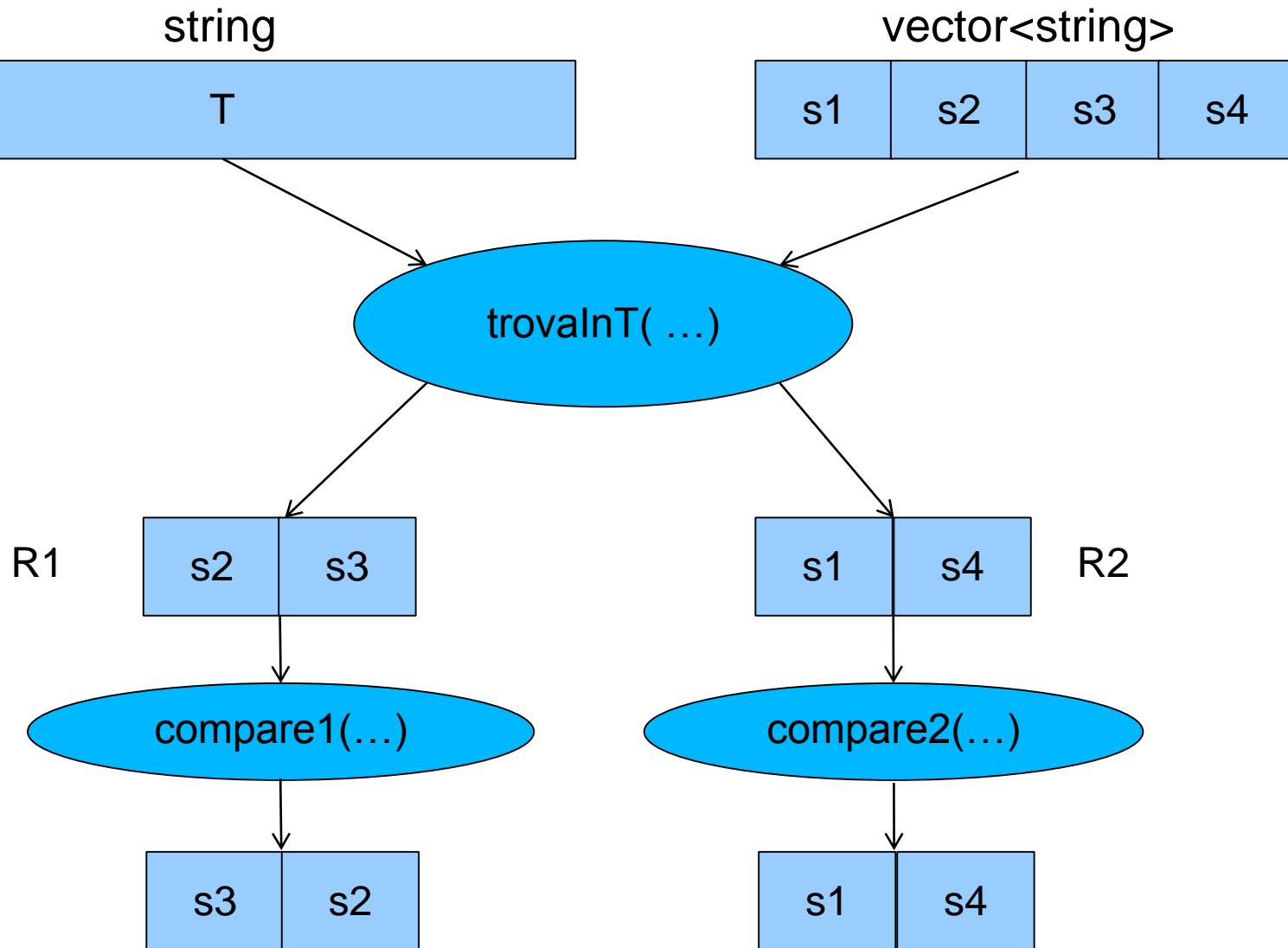
Strutture Dati:

- Dove salvo T?
- Dove salvo S?

Operazioni:

- Come ottengo gli elementi di 1?
- Come ottengo gli elementi di 2?
- Come ordino 1?
- Come ordino 2?

Analisi (2)



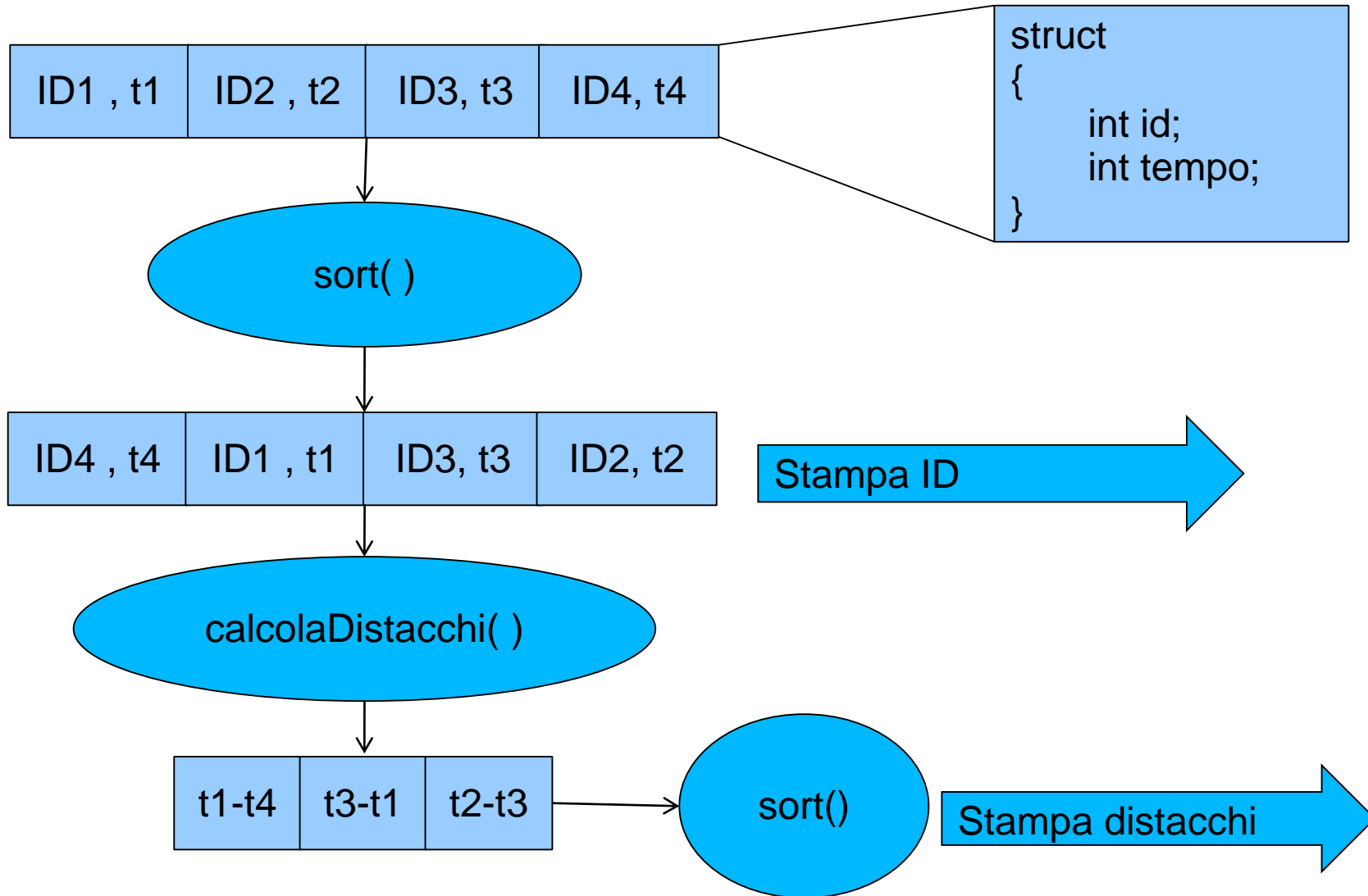
Implementazione

```
1 // cerca stringhe di S dentro T
2 void trovaInT( ... ){    }
3
4 // implementa confronto per posizione
5 bool compare1(string a, string b){    }
6
7 // implementa confronto lessicografico
8 bool compare2(string a, string b){    }
9
10 int main()
11 {
12     string T;
13     vector <string> S, R1, R2;
14
15     // lettura T ed S
16
17     trovaInT( ... );
18     sort( R1.begin(), R1.end(), compare1 )
19     sort( R2.begin(), R2.end(), compare2 )
20     print();
}
```

Gara

- Ad una gara partecipano N concorrenti
- Ogni concorrente e' caratterizzato da:
 - Un ID intero
 - Un tempo di arrivo espresso in secondi
- Calcolare:
 - Classifica
 - K distacchi più ampi di utenti **consecutivi**

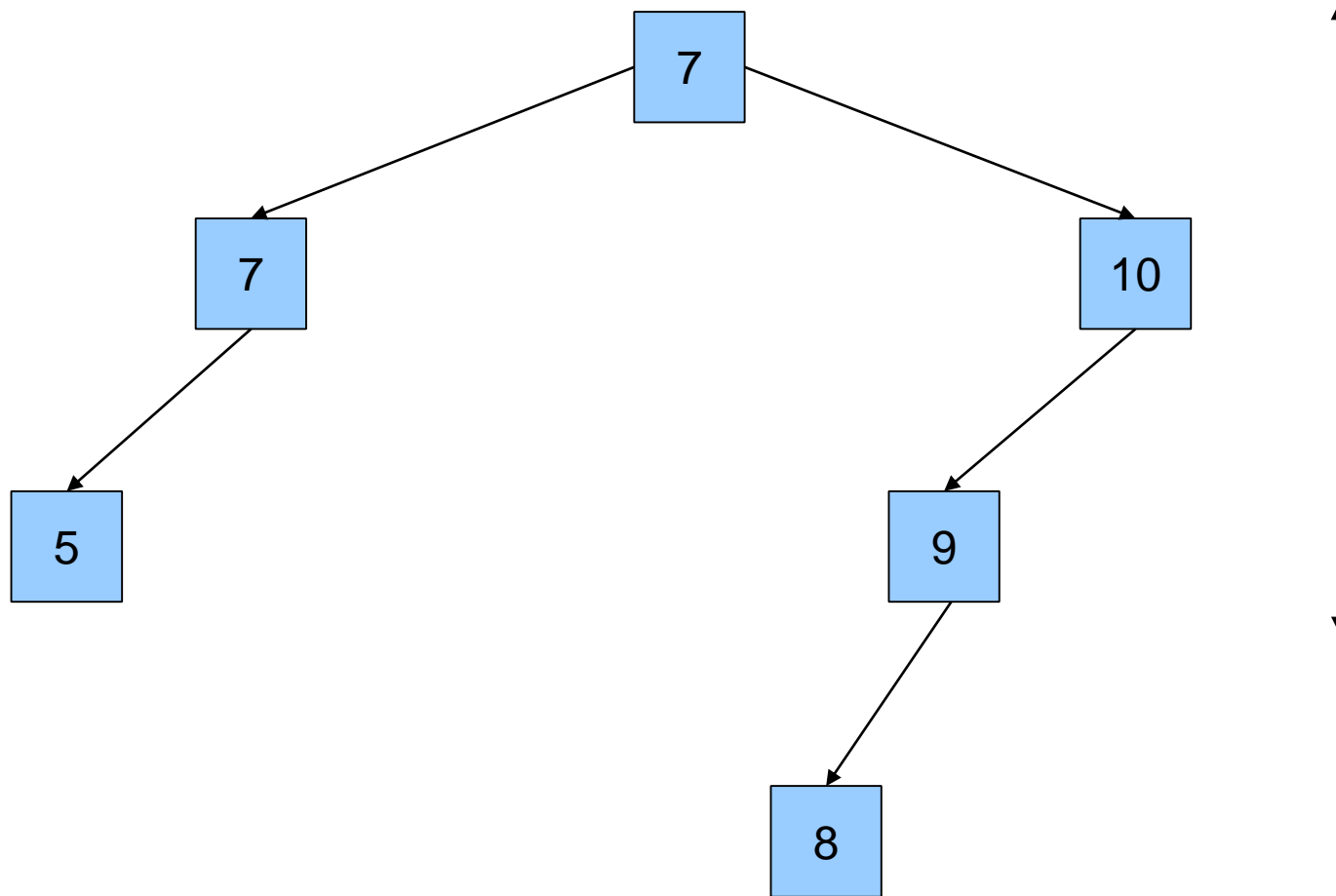
Analisi



Trovare altezza chiave K

- Input
 - Una sequenza di N interi positivi
 - Chiave K
- Output
 - L'altezza della chiave K dentro l'albero (se esiste)

Search K=9



Altezza

```
1  int search( Node * tree , int val )
2  {
3      if( tree == NULL )
4          return 0;
5
6      int cont = 0;
7      if( tree->value == val )
8          return 1;
9
10     else if( val <= tree->value )
11         cont = search( tree->left , val );
12     else
13         cont = search( tree->right , val );
14
15     if( cont != 0 )
16         return cont+1;
17
18     else return 0;
19 }
20
```

SEARCH

HEIGHT

Esercizi

- Esperimenti
 - Sort vs Binary Tree
 - Sort vs Min/Max
- Esercizi
 - Visite pre- e post-order
 - Esercizi di progettazione