Titolo unità didattica: Approccio divide et impera [13]

Titolo modulo : Analisi dell'efficienza della ricerca binaria [02-T]

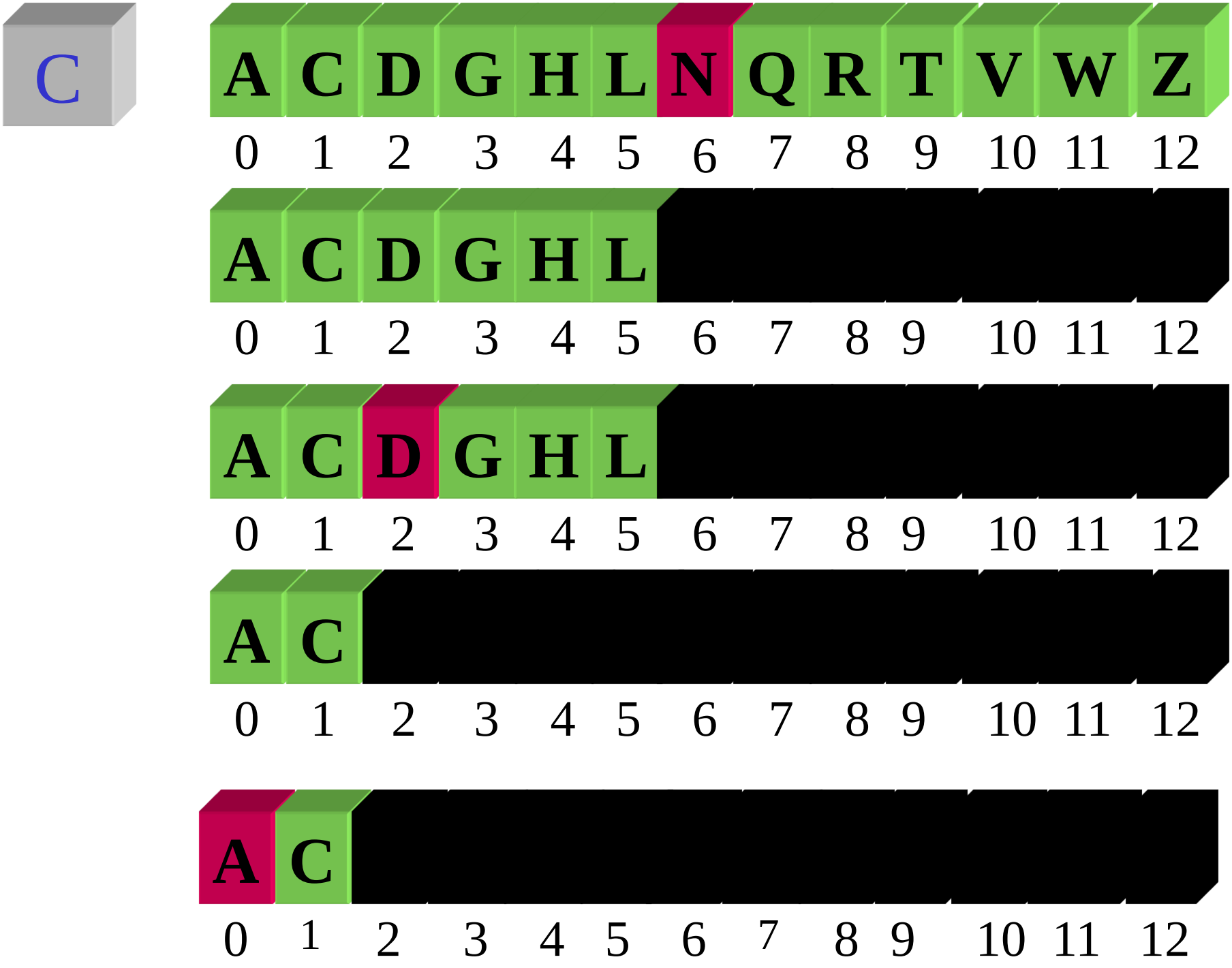
Ricerca a complessità di tempo logaritmica in array ordinati

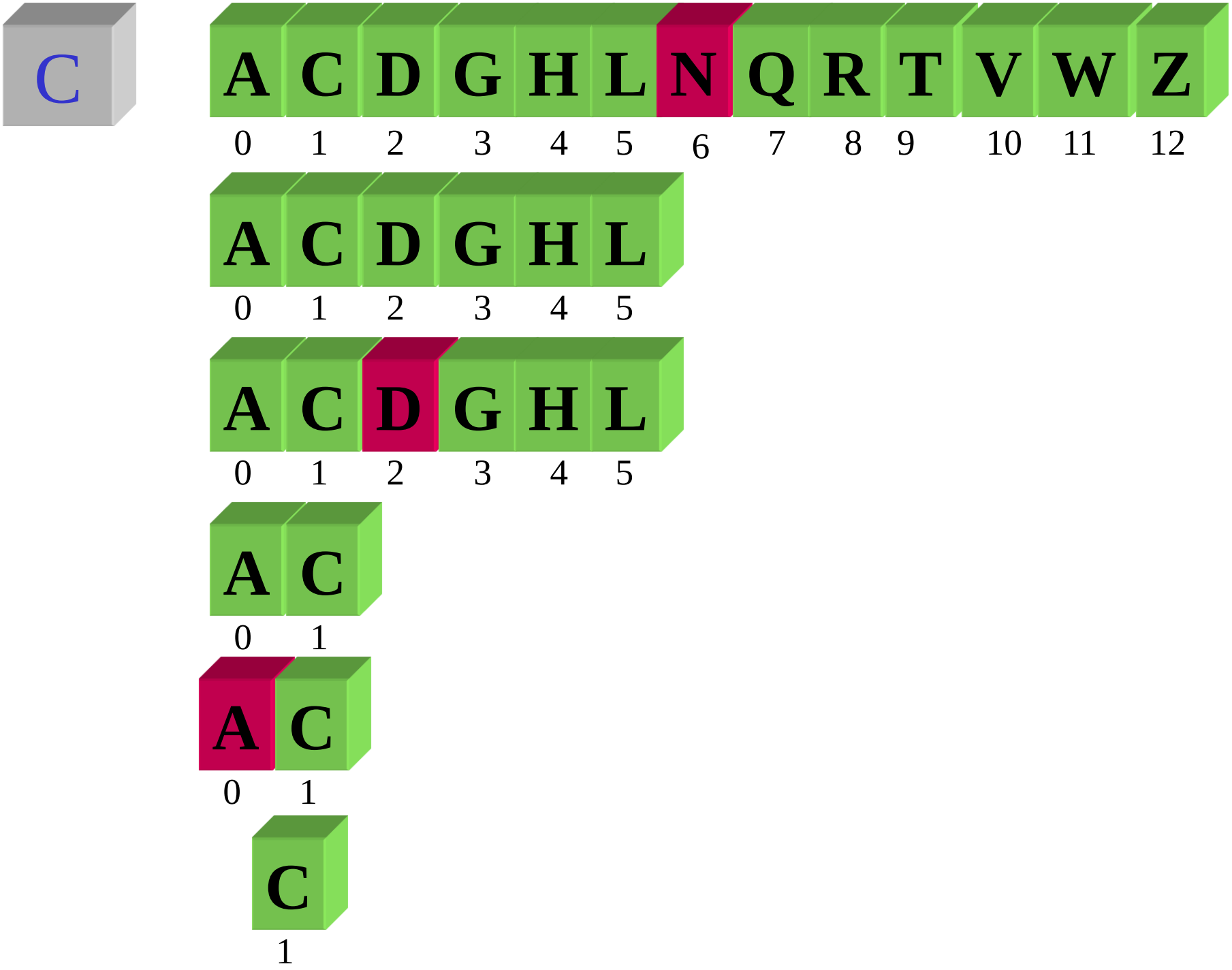
Argomenti trattati:

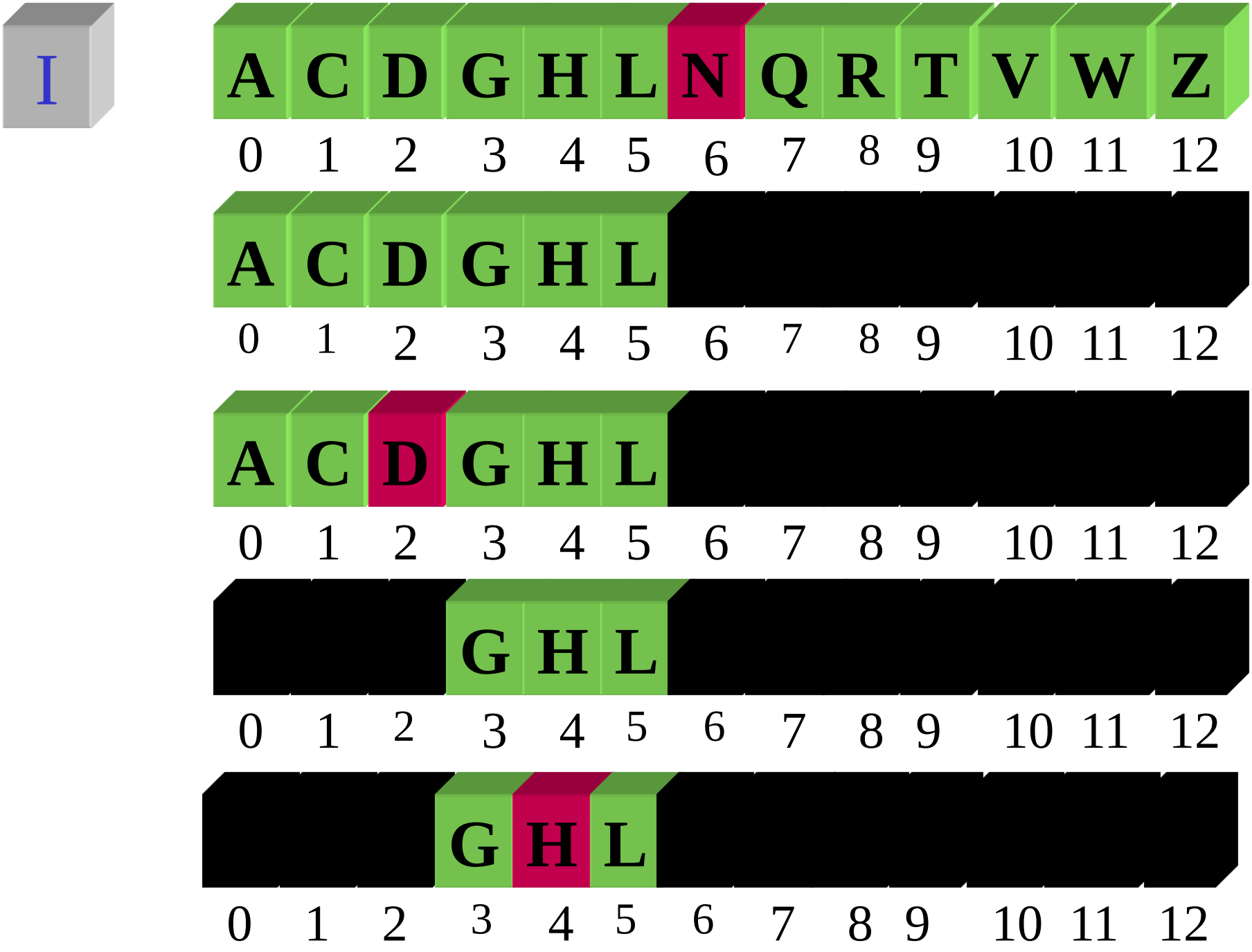
- ✓ alberi di decisione
- ✓ alcune proprietà degli alberi binari
- ✓ complessità di tempo dell'algoritmo di ricerca binaria
- ✓ ottimalità dell'algoritmo di ricerca binaria

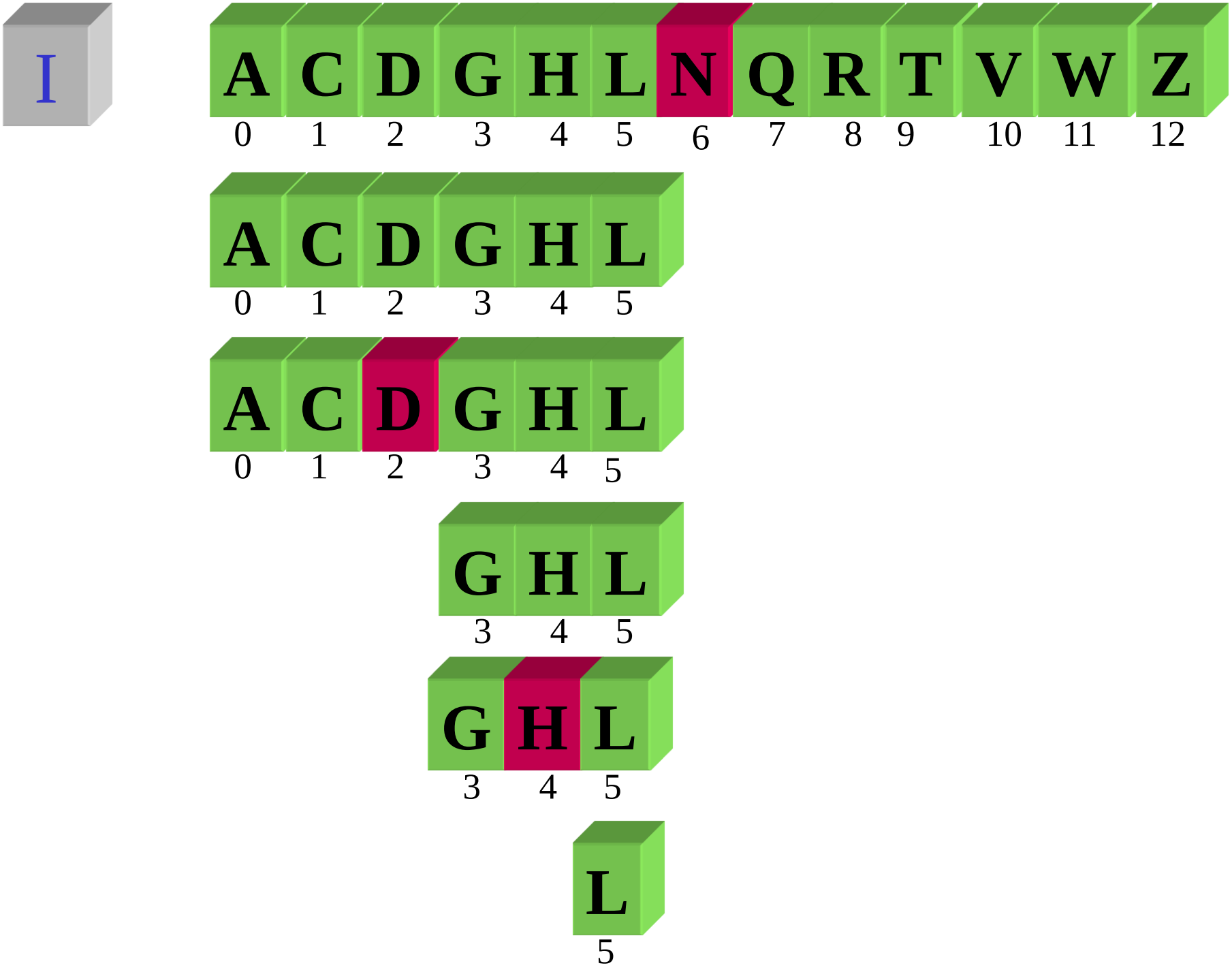
Prerequisiti richiesti: AP-13-01-T

```
int ric_bin (char chiave, char elenco[], int n) {
 int mediano, primo, ultimo;
 primo = 0 ;
 ultimo = n-1 ;
 while (primo <= ultimo) {
   mediano = (primo + ultimo)/2 ;
   if (chiave == elenco[mediano])
        { return mediano ; }
     else if (chiave < elenco[mediano])
             { ultimo = mediano - 1 ; }
             else
             { primo = mediano + 1 ; }
 }
 return -1 ;
}
```

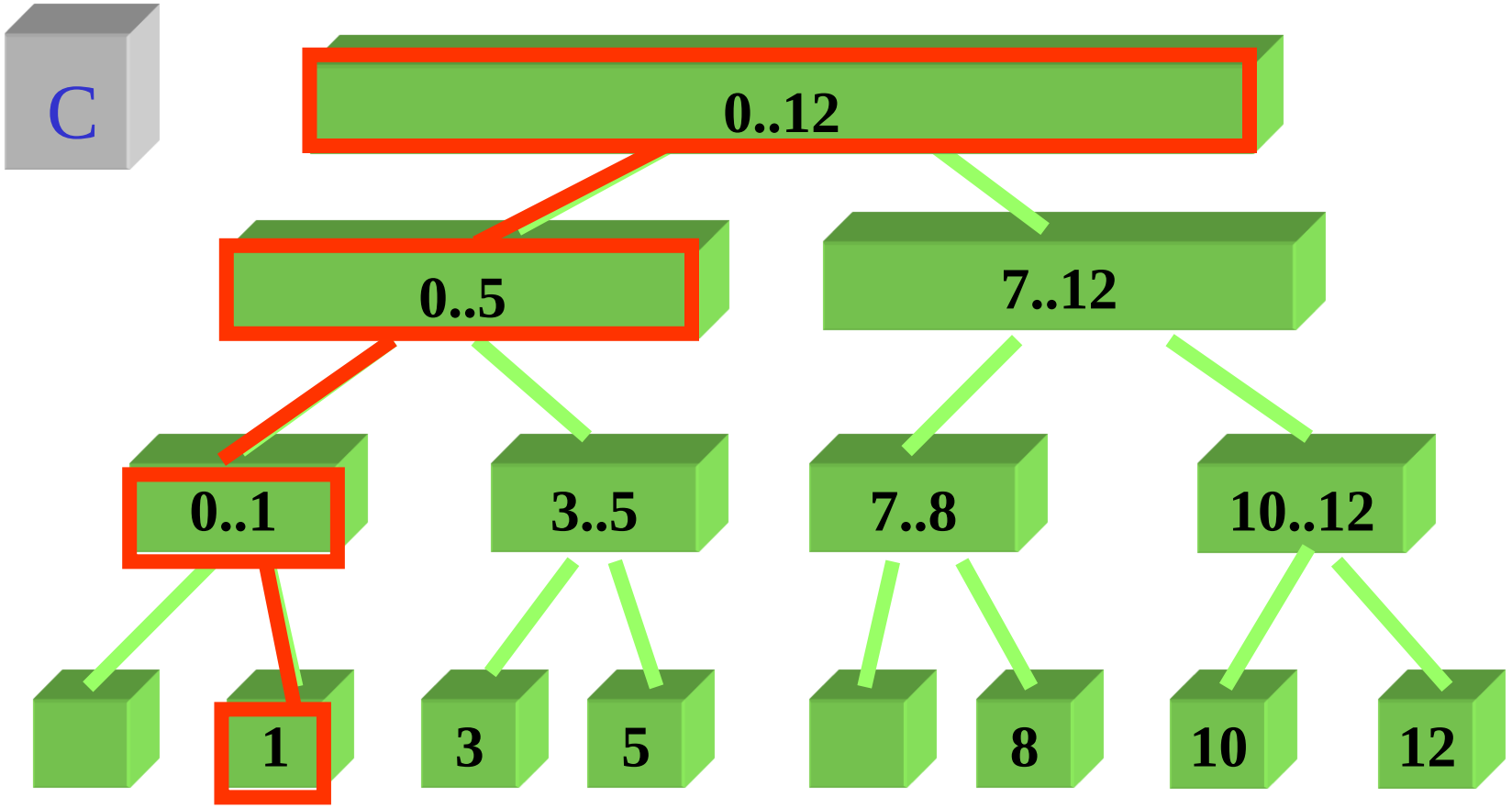

C
A C D G H L N Q R T V W Z
0 1 2 3 4 5 6 7 8 9 10 11 12
A C D G H L
0 1 2 3 4 5 6 7 8 9 10 11 12
A C D G H L
0 1 2 3 4 5 6 7 8 9 10 11 12
A C
0 1 2 3 4 5 6 7 8 9 10 11 12
A C
0 1 2 3 4 5 6 7 8 9 10 11 12


C
A C D G H L N Q R T V W Z
0 1 2 3 4 5 6 7 8 9 10 11 12
A C D G H L
0 1 2 3 4 5
A C D G H L
0 1 2 3 4 5
A C
0 1
A C
0 1
C
1


I
A C D G H L N Q R T V W Z
0 1 2 3 4 5 6 7 8 9 10 11 12
A C D G H L
0 1 2 3 4 5 6 7 8 9 10 11 12
A C D G H L
0 1 2 3 4 5 6 7 8 9 10 11 12
G H L
0 1 2 3 4 5 6 7 8 9 10 11 12
G H L
0 1 2 3 4 5 6 7 8 9 10 11 12

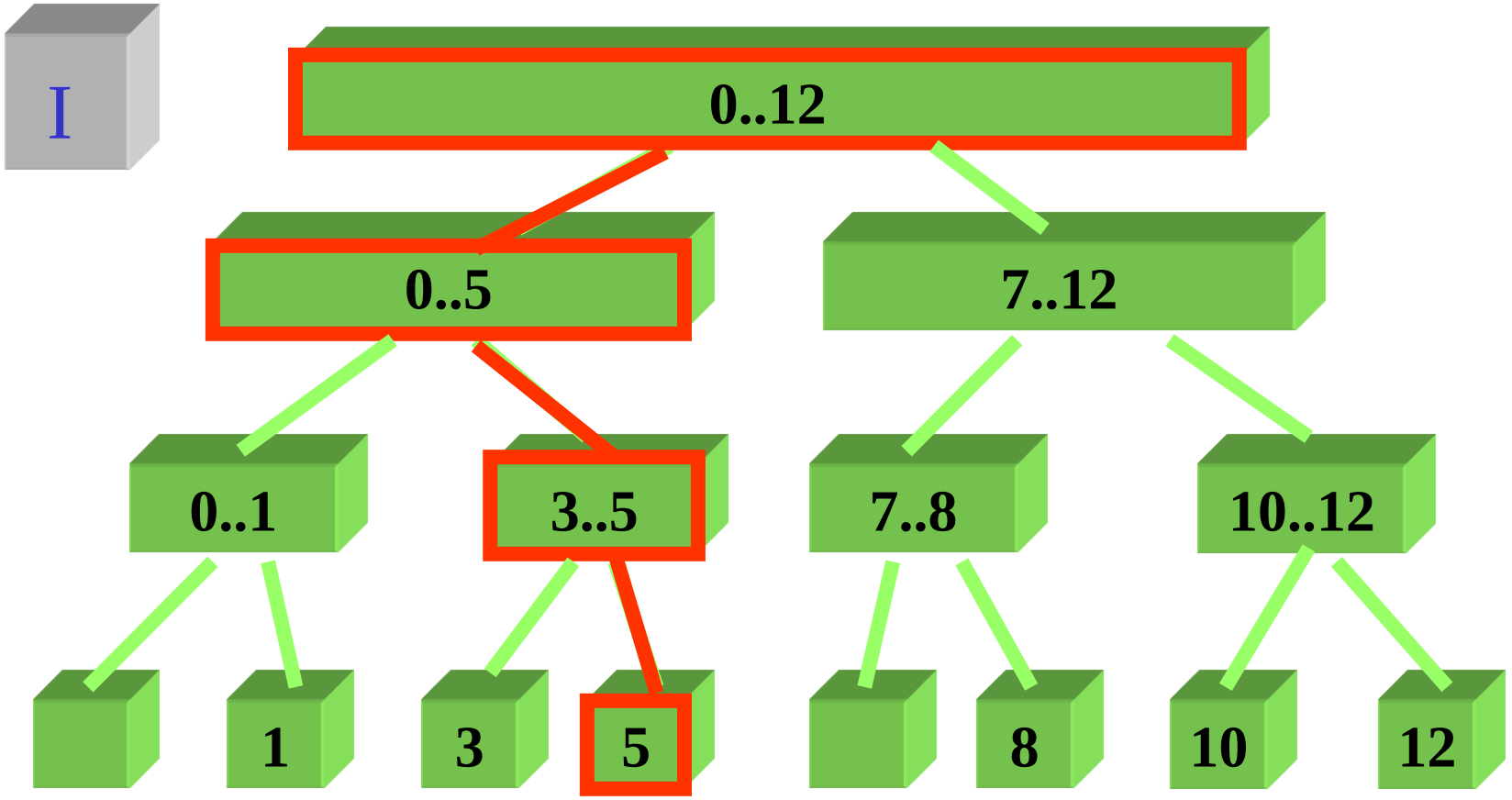

I
A C D G H L N Q R T V W Z
0 1 2 3 4 5 6 7 8 9 10 11 12
A C D G H L
0 1 2 3 4 5
A C D G H L
0 1 2 3 4 5
G H L
3 4 5
G H L
3 4 5
L
5

tutte le possibili porzioni dell'array
(tutte le suddivisioni (a metà) dell'array)



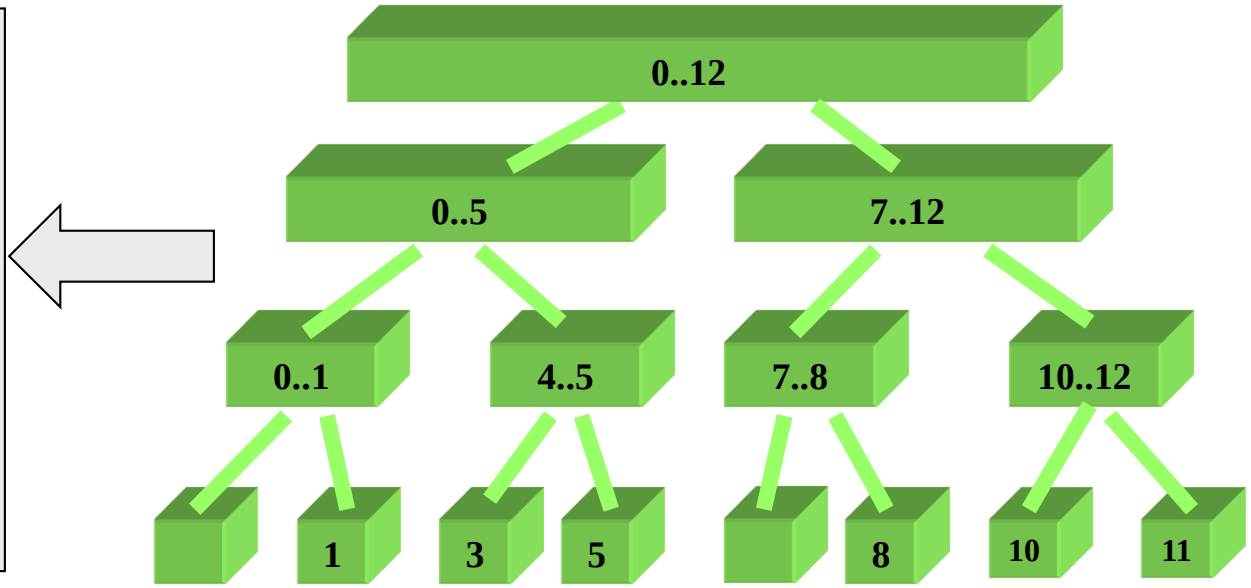
dinamica dei confronti chiave – elemento dell'array

tutte le possibili porzioni dell'array
(tutte le suddivisioni (a metà) dell'array)



dinamica dei confronti chiave – elemento dell'array

albero binario:
un elemento (*nodo*) padre può avere al più due elementi figli



mostra le possibili suddivisioni in successive metà di un array di 13 elementi

l'esecuzione dell'algoritmo di ricerca binaria è un **percorso di discesa** nell'albero

0..12; 0..5; 0..1; 1

0..12; 0..5; 3..5; 5

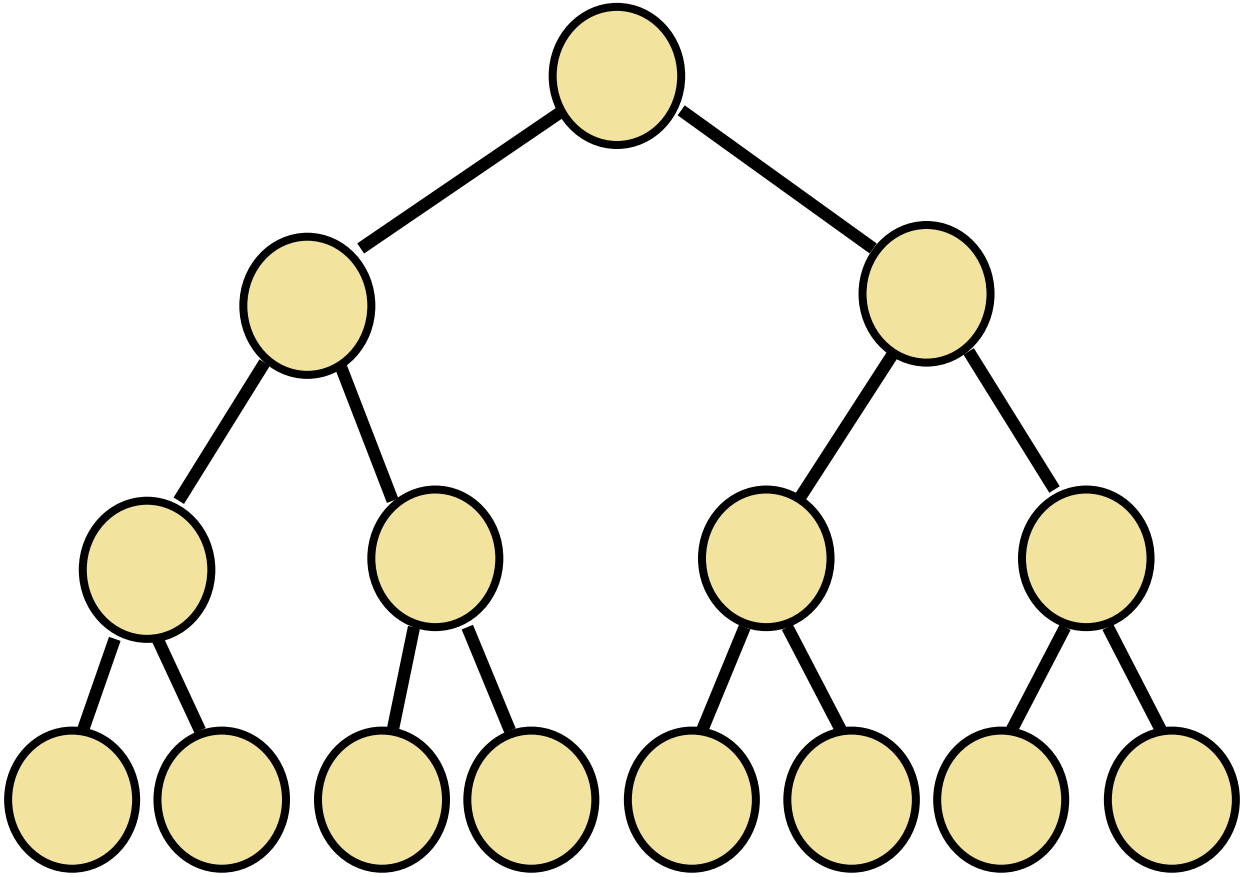
albero binario: terminologia

livello 0

livello 1

livello 2

livello 3



profondità = numero di livelli

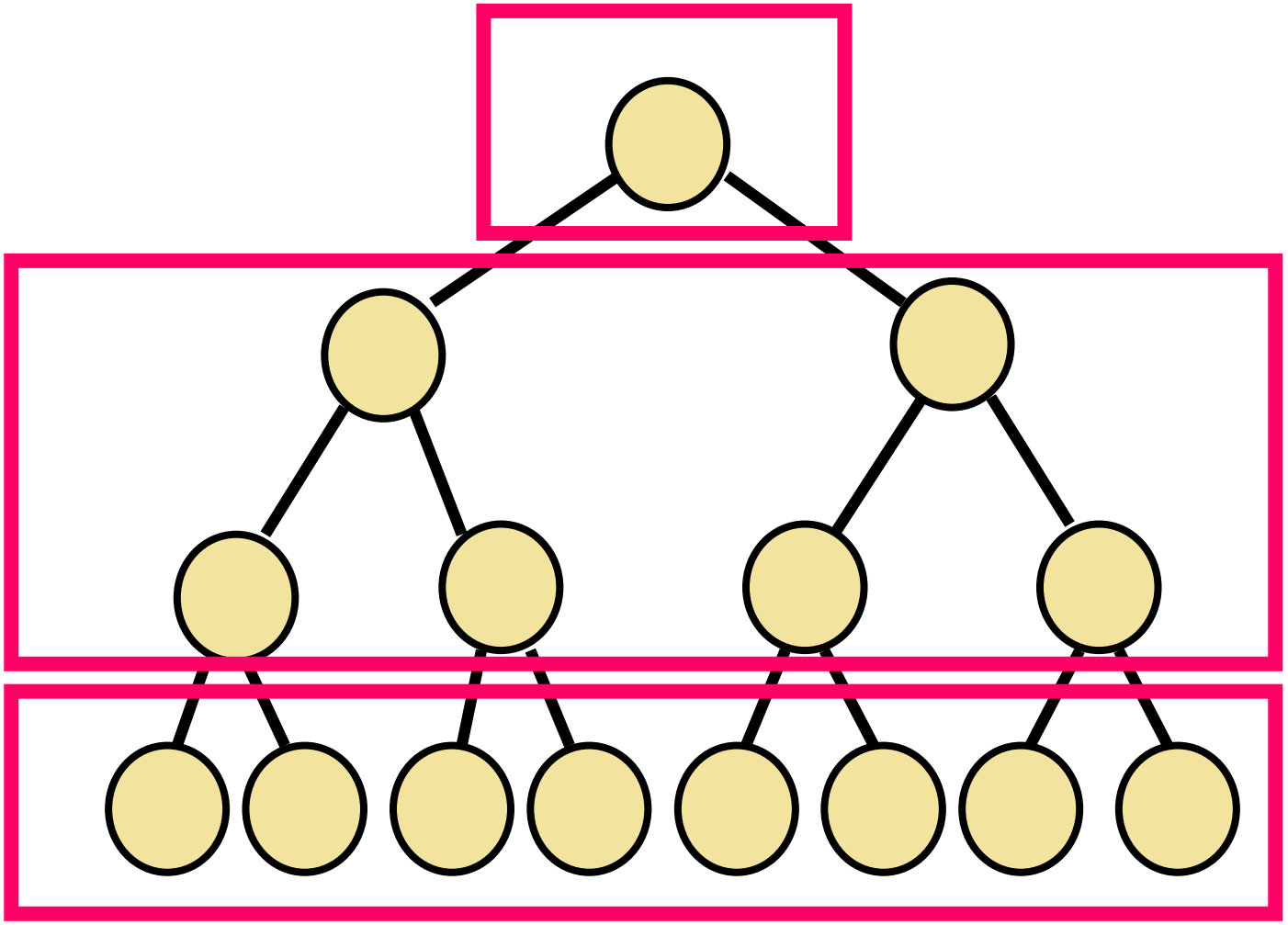
$p = 4$

albero binario: terminologia

radice

nodi interni

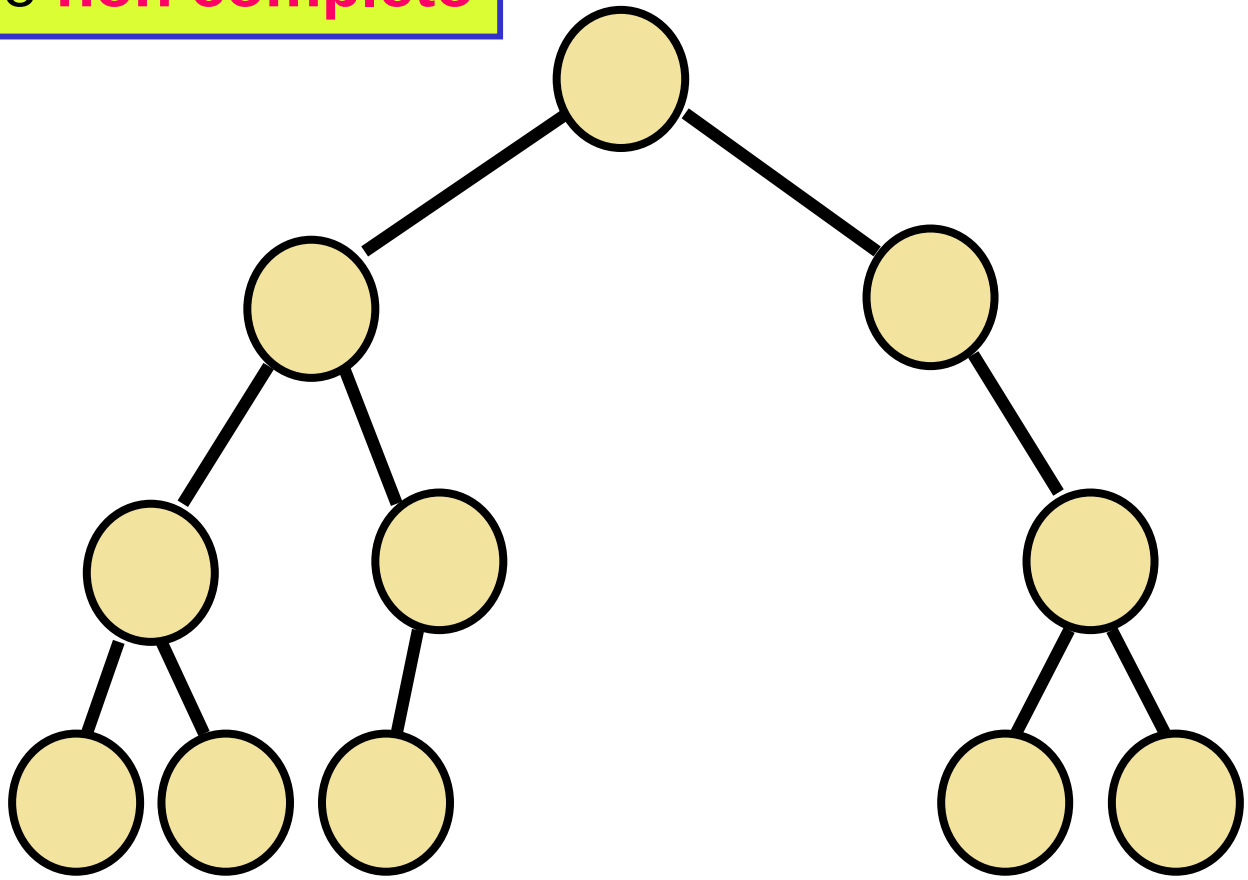
foglie



albero **completo**: ci sono tutti i nodi di tutti i livelli

albero binario: terminologia

esempio di albero **non completo**



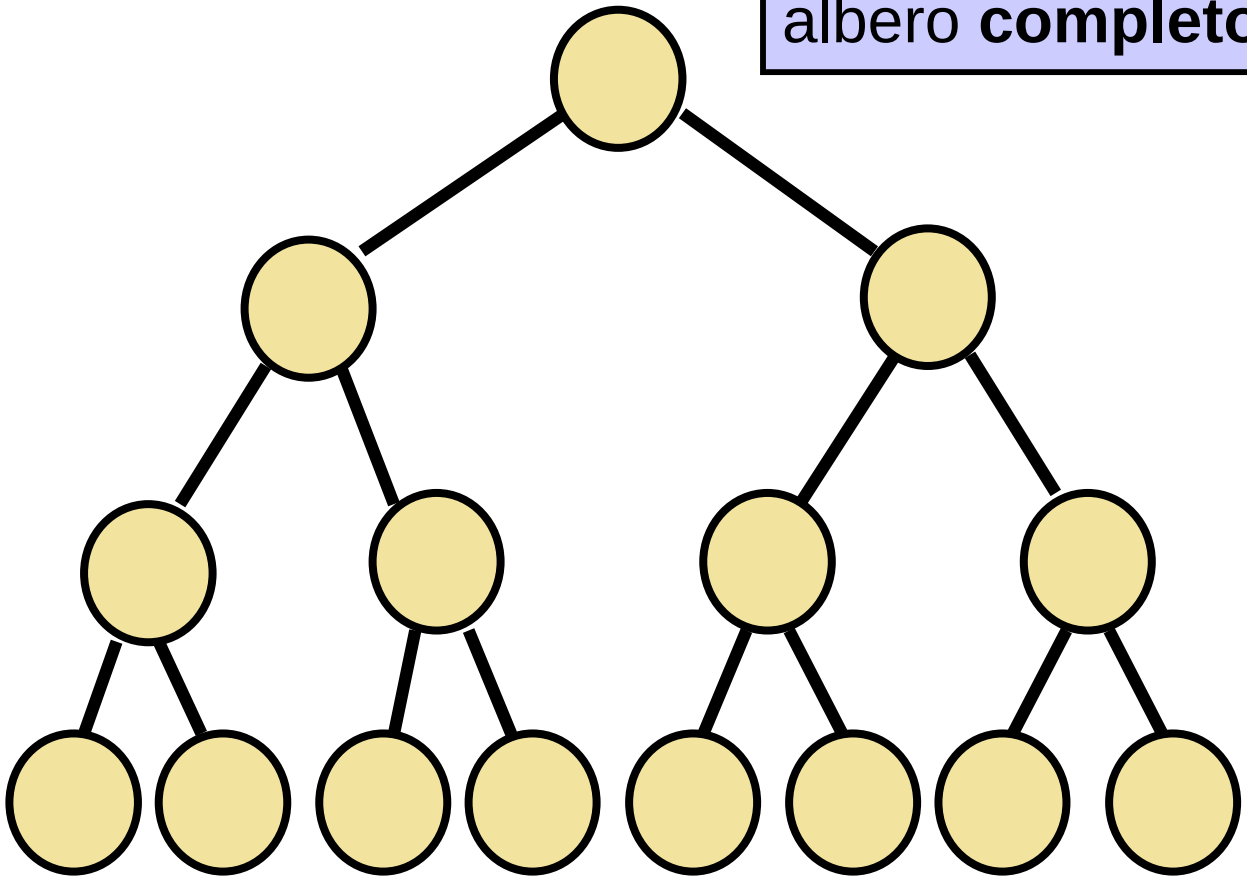
albero **completo**

livello 0

livello 1

livello 2

livello 3



numero di nodi del livello k : 2^k

numero di nodi di un a. b. completo di profondità p : $N = 2^p - 1$

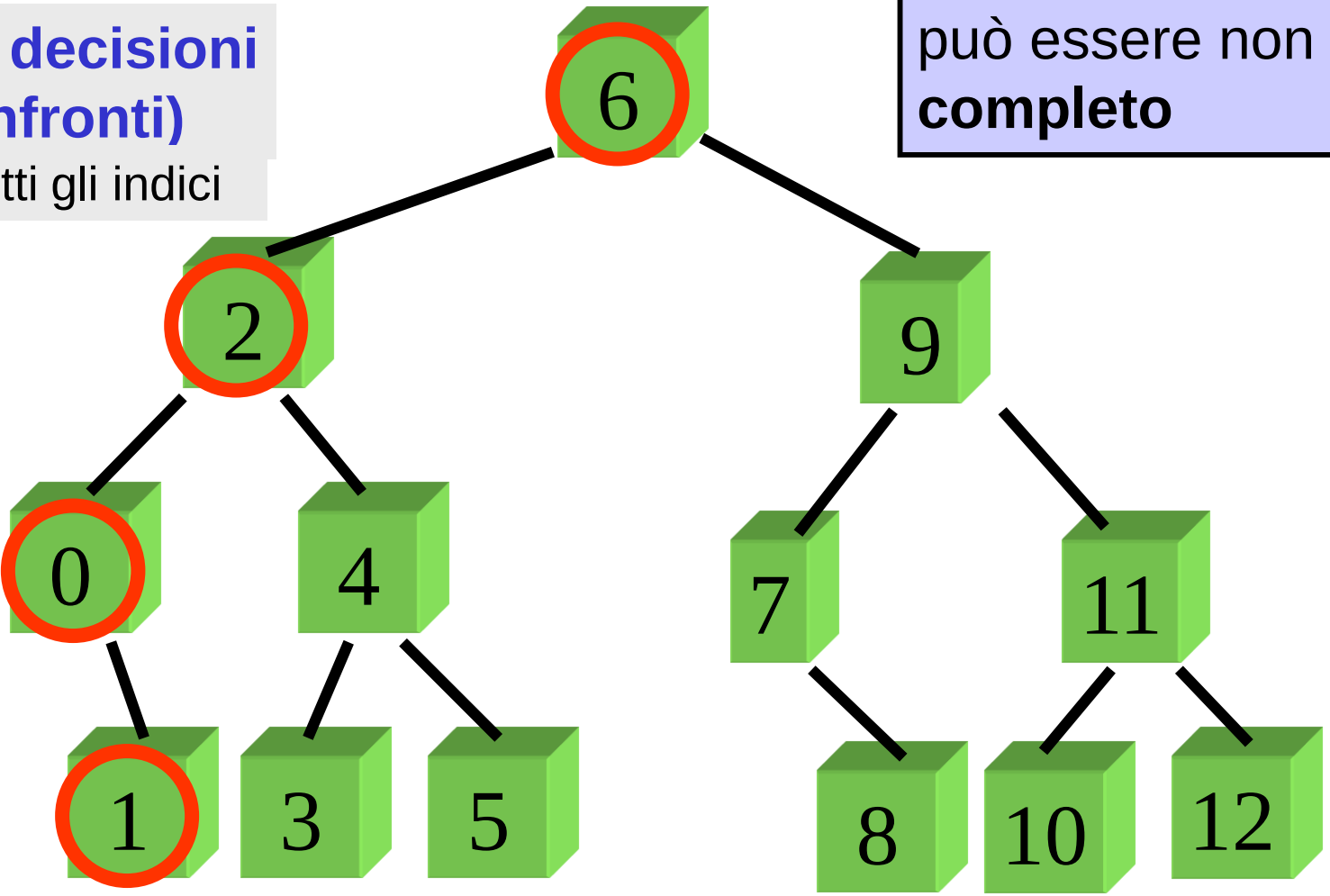
profondità di un a. b. completo di N nodi : $p = \log_2(N+1)$

albero delle decisioni (o dei confronti)

compaiono tutti gli indici

può essere non **completo**

C



un percorso di discesa (nell'esempio) può comprendere al più 4 confronti (profondità dell'albero)

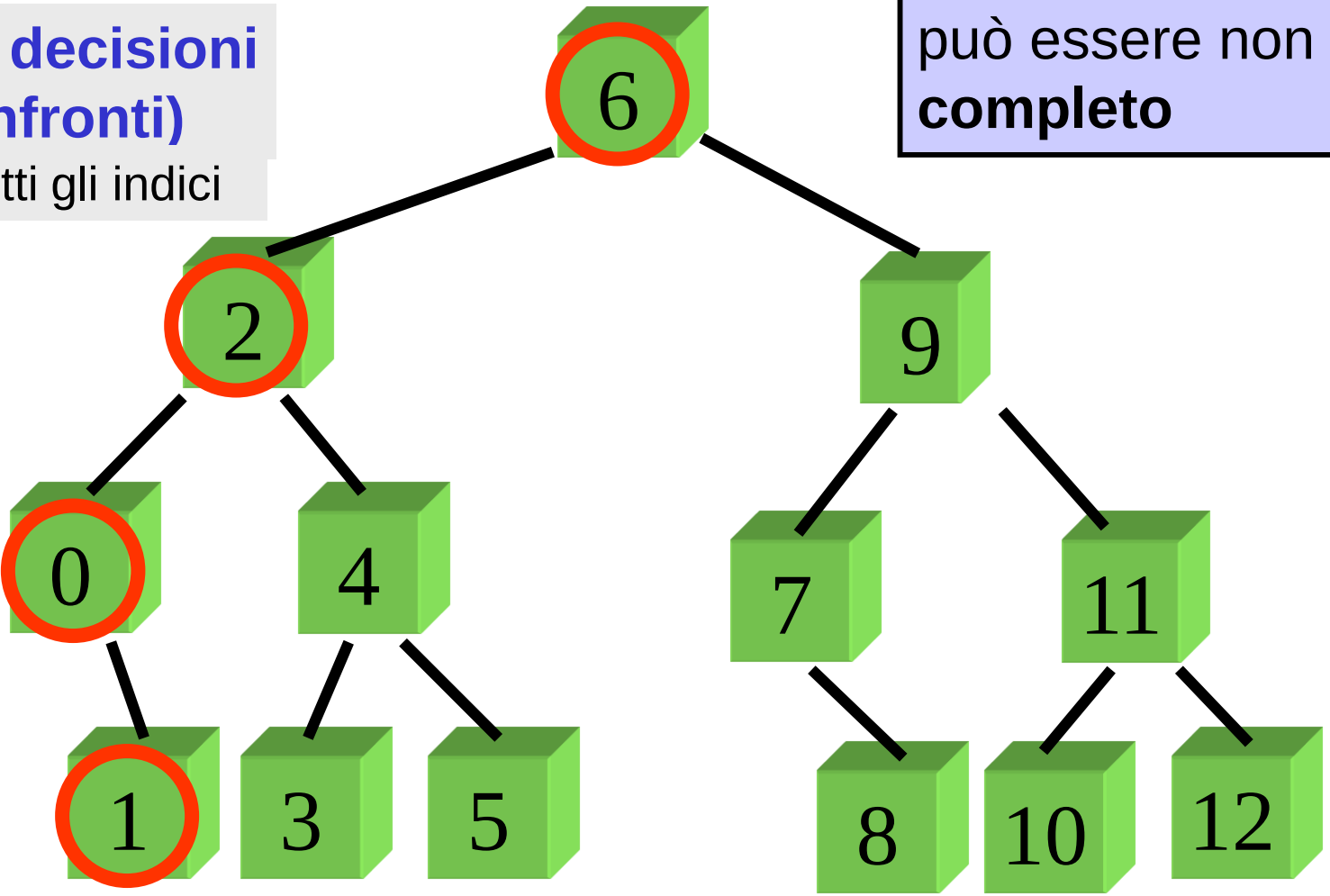
il percorso effettuato è **6,2,0,1**

albero delle decisioni (o dei confronti)

compaiono tutti gli indici

può essere non **completo**

C



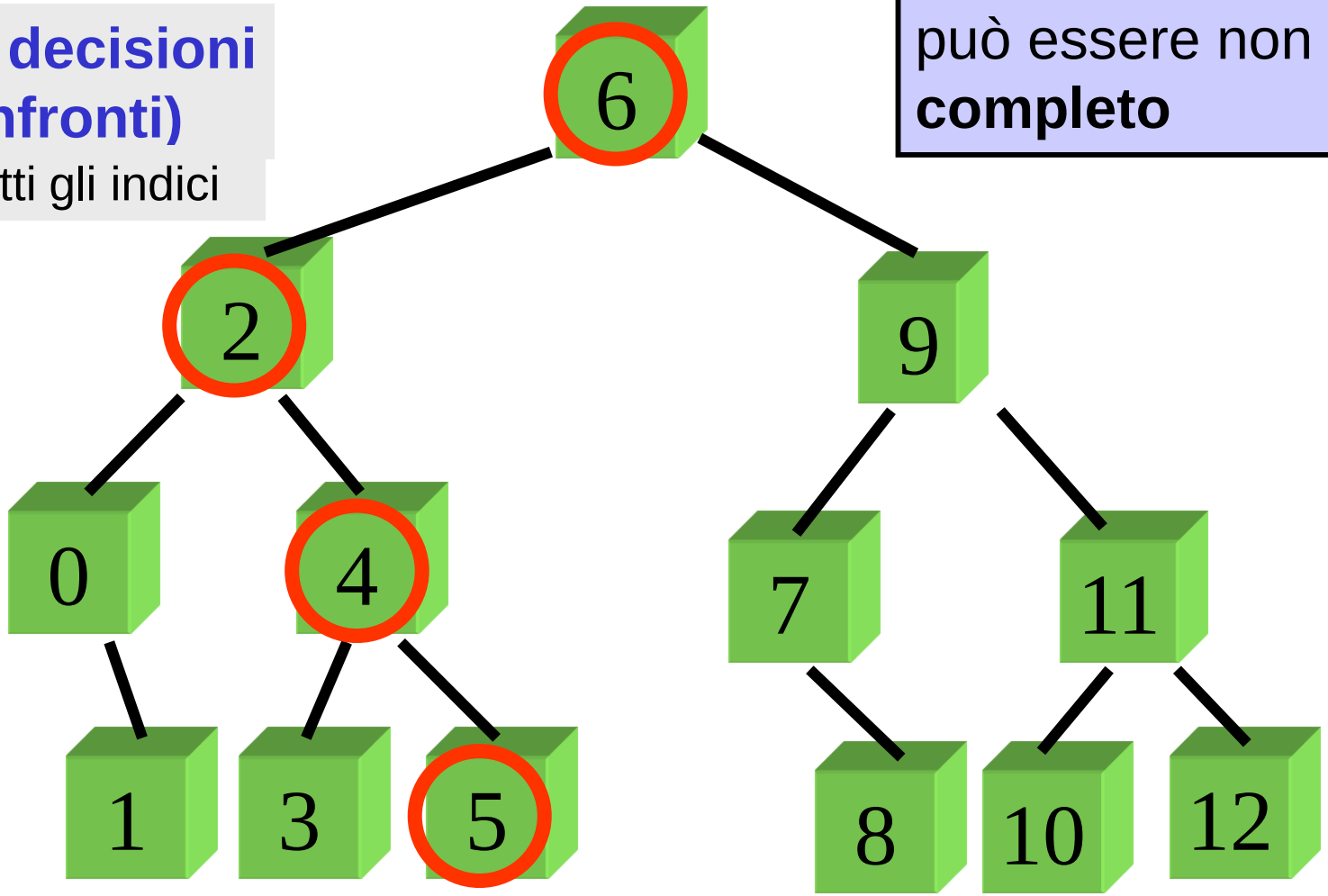
il **numero di confronti** dell'algoritmo di ricerca binaria è (**al più**) uguale alla **profondità** del relativo albero delle decisioni

albero delle decisioni (o dei confronti)

compaiono tutti gli indici

può essere non **completo**

I



un percorso di discesa (nell'esempio) può comprendere al più 4 confronti (profondità dell'albero)

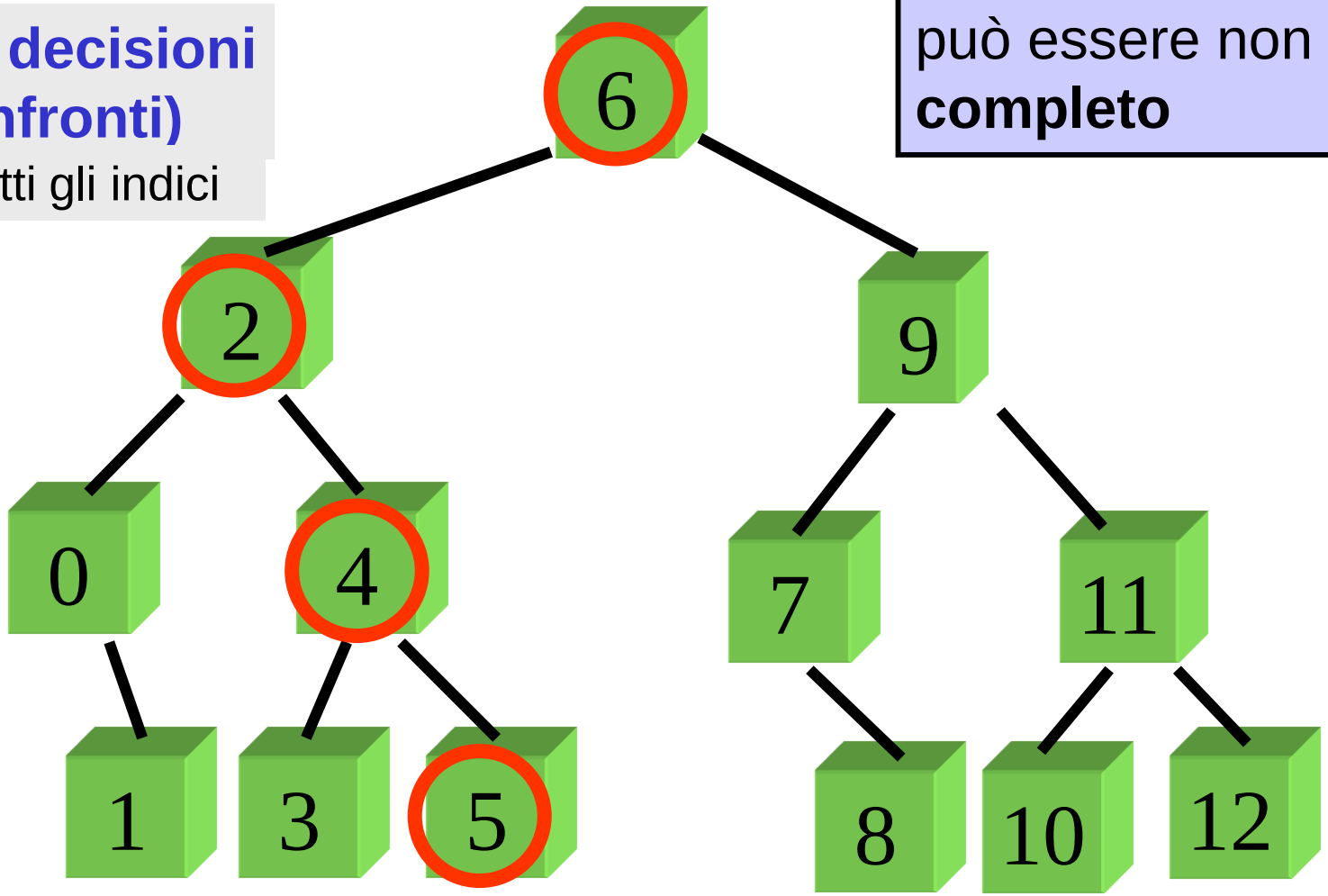
il percorso effettuato è **6,2,4,5**

albero delle decisioni (o dei confronti)

compaiono tutti gli indici

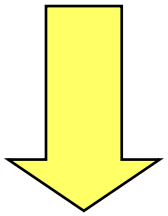
può essere non **completo**

I



il **numero di confronti** dell'algoritmo di ricerca binaria è (**al più**) uguale alla **profondità** del relativo albero delle decisioni

la **profondità** di un
albero **binario delle decisioni** di ***n*** nodi
è uguale alla **profondità**
dell'albero binario completo corrispondente



$$\lfloor \log_2 n \rfloor + 1$$

floor, il più grande intero minore o uguale di

complessità di tempo
(confronti chiave - elemento)
dell'algoritmo di **ricerca binaria**
(caso peggiore)

confronto:
confronto a tre vie (<,=,>)

$$T(n) = \lfloor \log_2 n \rfloor + 1$$

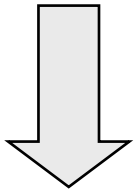
$$T(n) = O(\log_2 n)$$

l'algoritmo di ricerca binaria è ottimale

complessità intrinseca del problema della ricerca in array ordinati

numero di volte in cui è possibile **dividere per 2** un numero intero n, avendo un quoziente maggiore o uguale a 1

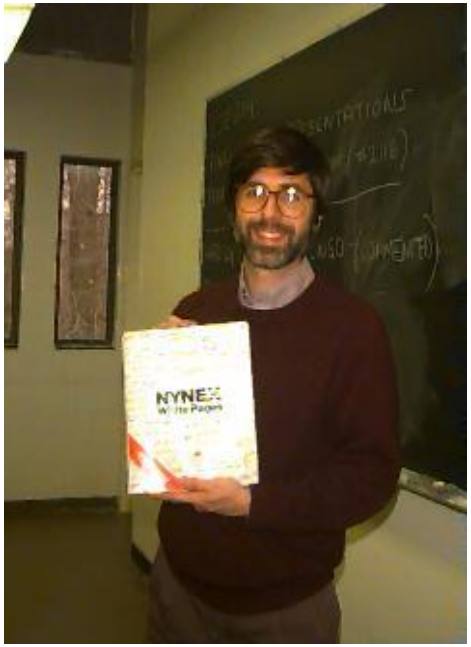
$$\lfloor \log_2 n \rfloor$$

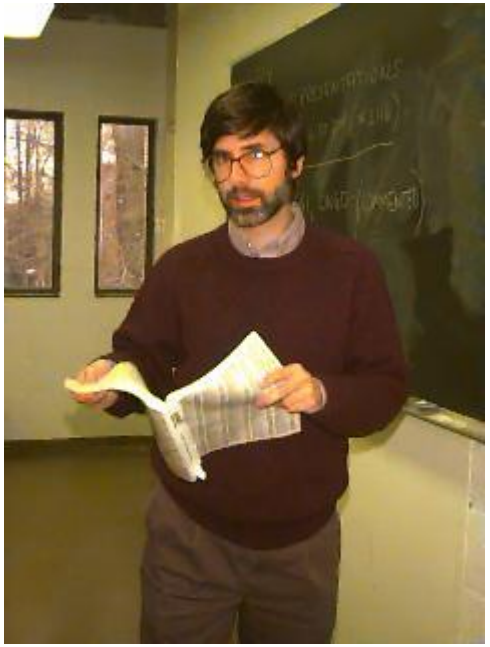


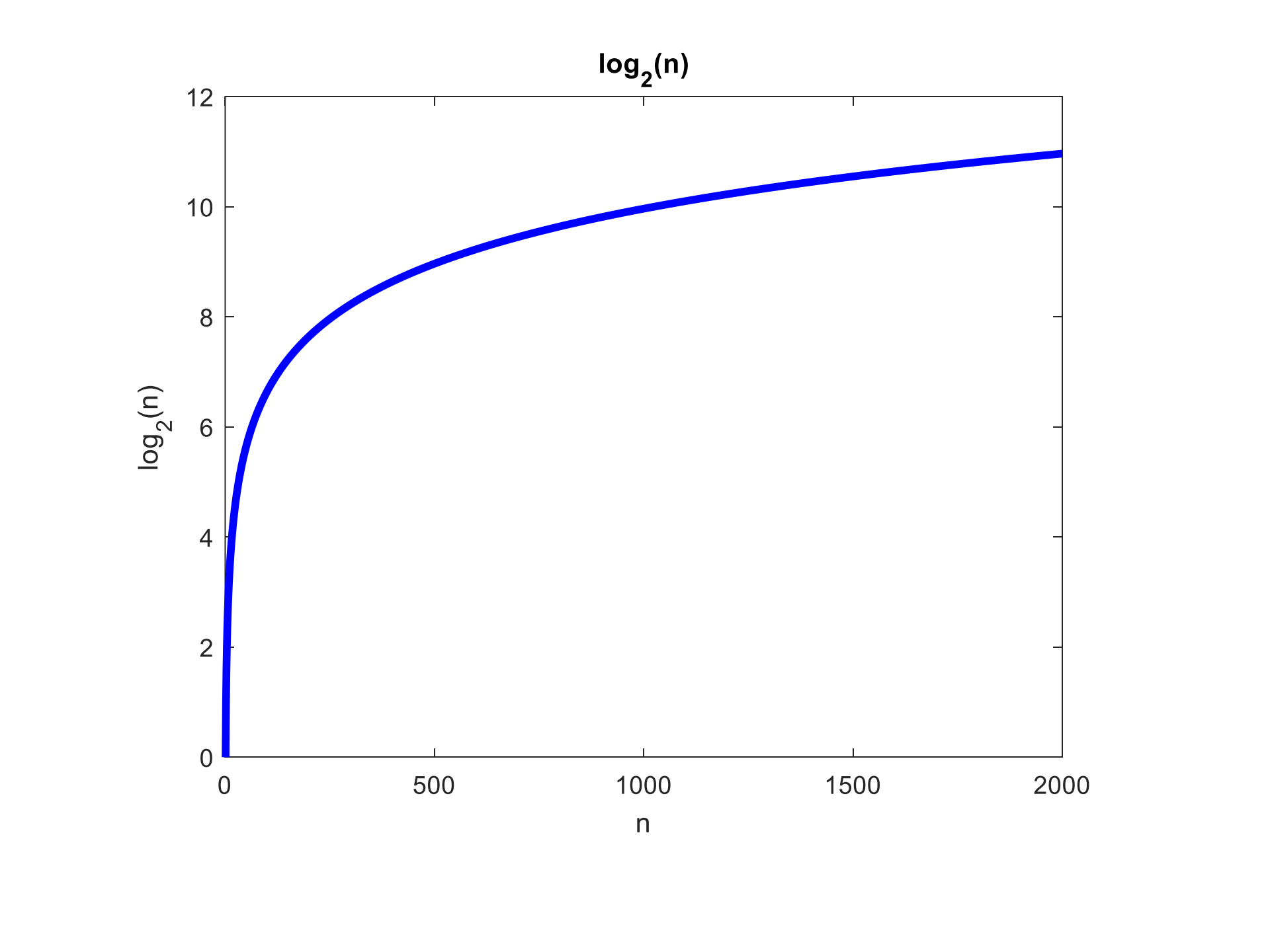
il numero di confronti dell'algoritmo di ricerca binaria è uguale a **uno** più **il numero di volte** in cui è possibile **dividere a metà** l'array 1D nel quale si effettua la ricerca

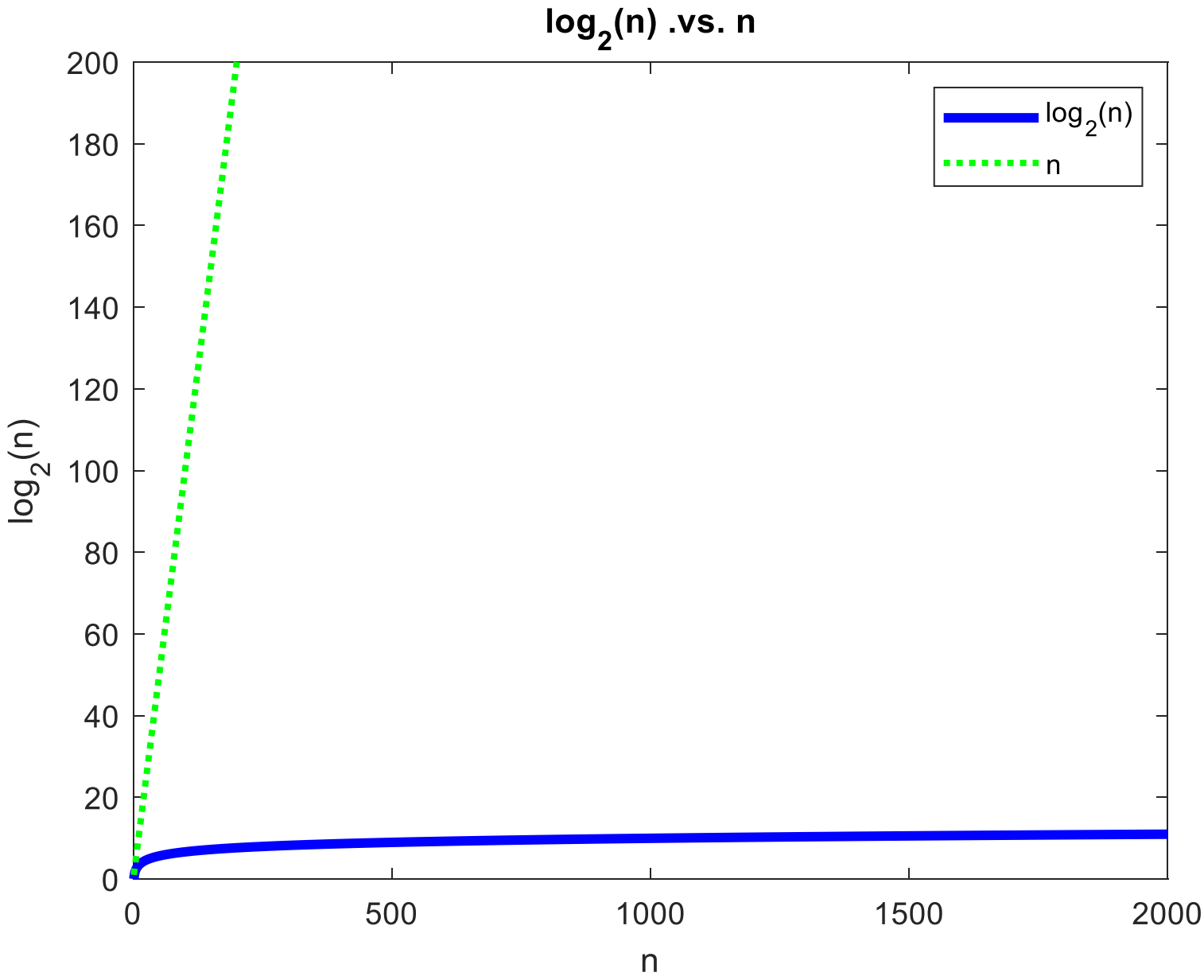
numero massimo di aperture di pagine di un elenco telefonico per ricercare il nome di un abbonato

$$\lfloor \log_2 n \rfloor + 1$$






log2(n)
12
10
8
6
4
2
0
log2(n)
0
500
1000
1500
2000
n


log2(n) .vs. n
log2(n)
n
200
180
160
140
120
100
80
60
40
20
0
0
500
1000
1500
2000
n