Informatica per le Biotecnologie
Algoritmica Lezione 3

RICERCA DEL MASSIMO IN UN INSIEME

Consideriamo un insieme $R = \{r_0, r_1, . . . , r_{n-1}\}$ di n elementi diversi tra loro per cui è definita una relazione di ordinamento indicata con $<$, $>$ e cerchiamo l'elemento massimo m

Valutiamo la complessità degli algoritmi di soluzione come numero di confronti *C(n)* eseguiti, in funzione di *n.*

A causa dell'estrema semplicità del problema proviamo a calcolare il limite in modo preciso anziché in ordine di grandezza.

Cerchiamo un algoritmo di soluzione. Il numero di confronti eseguiti ci fornirà un limite superiore alla complessità del problema.

L'algoritmo più immediato che chiamiamo MASSIMO-ITER ha forma *iterativa*, cioè specifica la sequenza di operazioni in modo esplicito.

L'insieme è memorizzato in un vettore $A[0...n-1]$, **con** $n > 1$. L'algoritmo scandisce il vettore A per valori crescenti dell'indice i, memorizzando in m, al passo i-esimo, il valore massimo trovato nella porzione del vettore tra $A[0]$ e $A[i]$.

Prima di codificarlo in uno pseudocodice vicino al linguaggio C, vediamo come funziona su un esempio.

24 15 7 9 20 49 33 35 22 40 52 1 62 30 8 43

24 15 7 9 20 49 33 35 22 40 52 1 62 30 8 43

m

24 15 7 9 20 49 33 35 22 40 52 1 62 30 8 43

m •

24 15 7 9 20 49 33 35 22 40 52 1 62 30 8 43

m • •

24 15 7 9 20 49 33 35 22 40 52 1 62 30 8 43

m • • •

24 15 7 9 20 49 33 35 22 40 52 1 62 30 8 43

m

24 15 7 9 20 49 33 35 22 40 52 1 62 30 8 43

m

24 15 7 9 20 49 33 35 22 40 52 1 62 30 8 43

m • • • •

24 15 7 9 20 49 33 35 22 40 52 1 62 30 8 43

m

24 15 7 9 20 49 33 35 22 40 52 1 62 30 8 43

m •

24 15 7 9 20 49 33 35 22 40 52 1 62 30 8 43

m

24 15 7 9 20 49 33 35 22 40 52 1 62 30 8 43

m • • •

Dunque il massimo è m = 62

MASSIMO-ITER(A, n, m)

// Ricerca del massimo elemento

in un vettore $A[0...n-1]$ ***//***

input A, n**; output** m**;**

$m = A[0]$**;**

for $(i = 1; i \leq n-1; i++)$

if $(m < A[i])\ m = A[i]$**;**

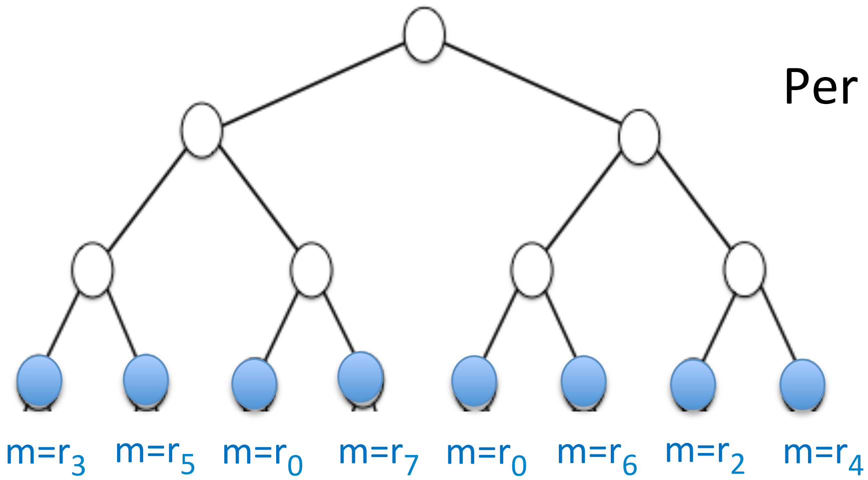
Un'analisi elementare dell'algoritmo mostra che questo è corretto e esegue esattamente n-1 confronti nel caso pessimo, stabilendo un limite superiore $C(n) = n-1$ alla complessità del problema

Valutiamo ora un limite inferiore per il problema.

Limite inferiore per $C(n)$

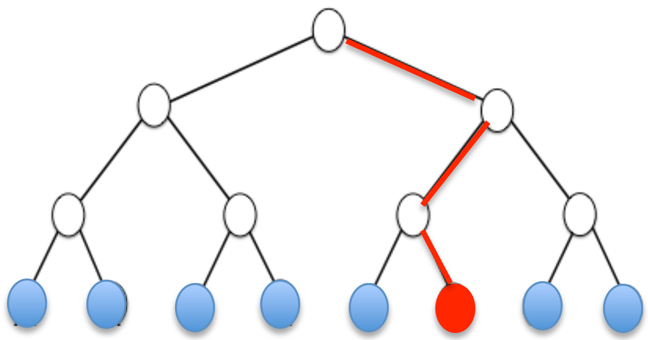
Eseguiremo ragionamenti diversi per determinare questo limite e sceglieremo il limite più alto (quindi più significativo) tra quelli trovati.

1. ***Albero di decisione.*** **Il problema ha** $s = n$ **soluzioni:** $m = r_0, m = r_1, \ldots, m = r_{n-1}$**. I confronti successivi generano un** ***albero binario*** **che contiene le soluzioni nelle foglie.**



Per esempio n = 8

Il percorso più lungo dalla radice a una foglia determina il limite inferiore per il problema



$n = 8 = 2^c$

L'altezza dell'albero è

$c = \log_2 8 = 3$

Se *n* è una potenza di 2 ogni percorso può contenere al minimo $log_2 n$ nodi di diramazione (albero bilanciato), quindi $C(n) \geq log_2 n$

Se *n* non è una potenza di 2 alcuni percorsi dalla radice a una foglia contengono un elemento in meno e il logaritmo si approssima all'intero superiore.

Il limite infriore al problema è sempre $C(n) \geq \log_2 n$ relativo al percorso più lungo

2. *Tutti gli elementi devono essere confrontati almeno una volta.*

Poiché ogni confronto avviene tra due elementi occorrono almeno $n/2$ confronti per considerarli tutti. Dunque $C(n) \geq n/2$

3. La determinazione del massimo implica che gli altri $n-1$ elementi non sono il massimo.

Per certificare che un elemento non è il massimo occorre che risulti minore di un'altro in almeno un confronto.

Poiché da ogni confronto esce esattamente un perdente, sono necessari $n-1$ confronti per dichiarare che ci sono $n-1$ perdenti. Dunque $C(n) \geq n-1$

I tre limiti inferiori trovati

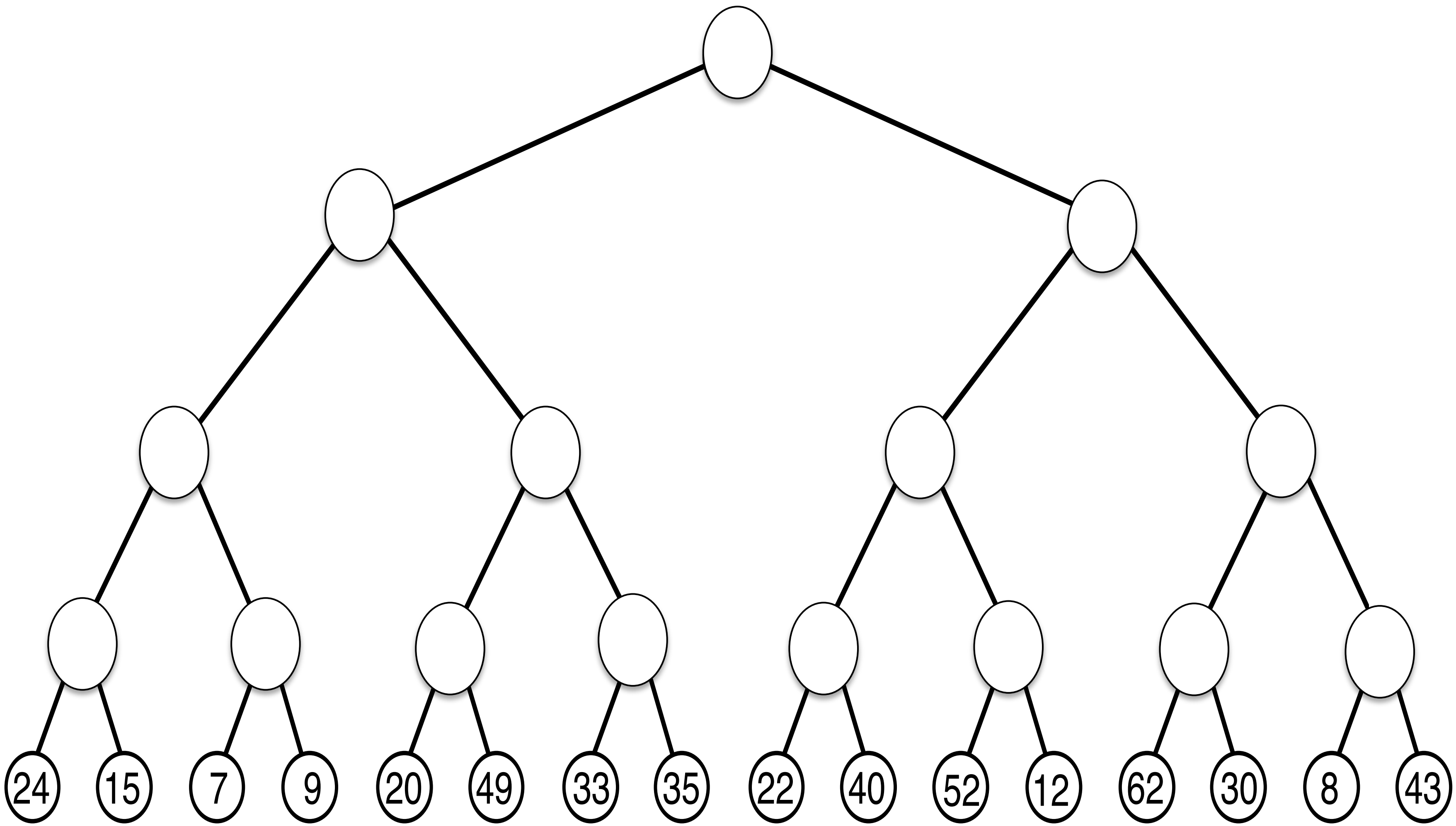
$C(n) \geq \log_2 n$, $C(n) \geq n/2$, $C(n) \geq n\text{-}1$

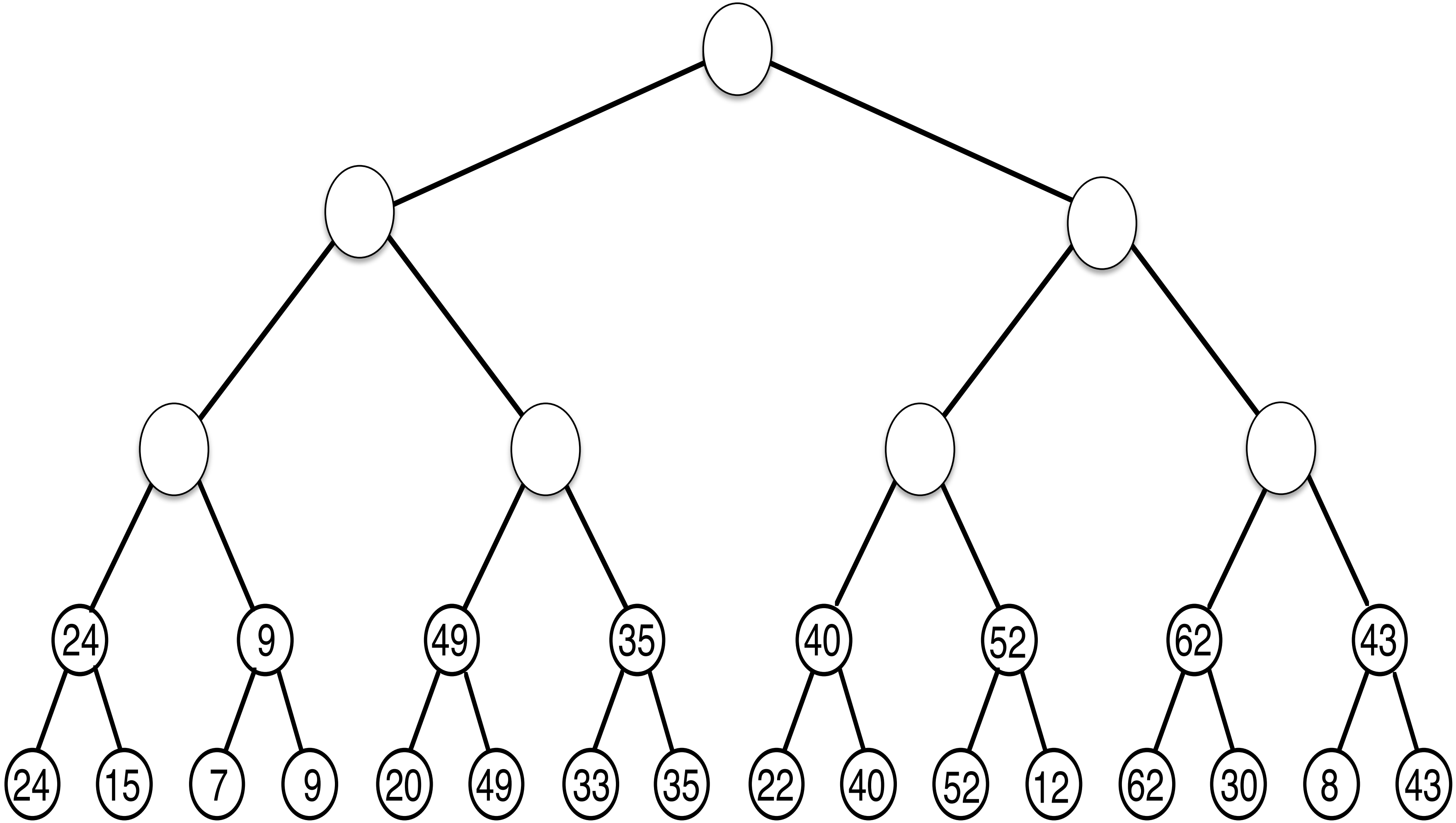
sono tutti legittimi, ma il terzo è il più grande
e viene scelto come il più significativo.

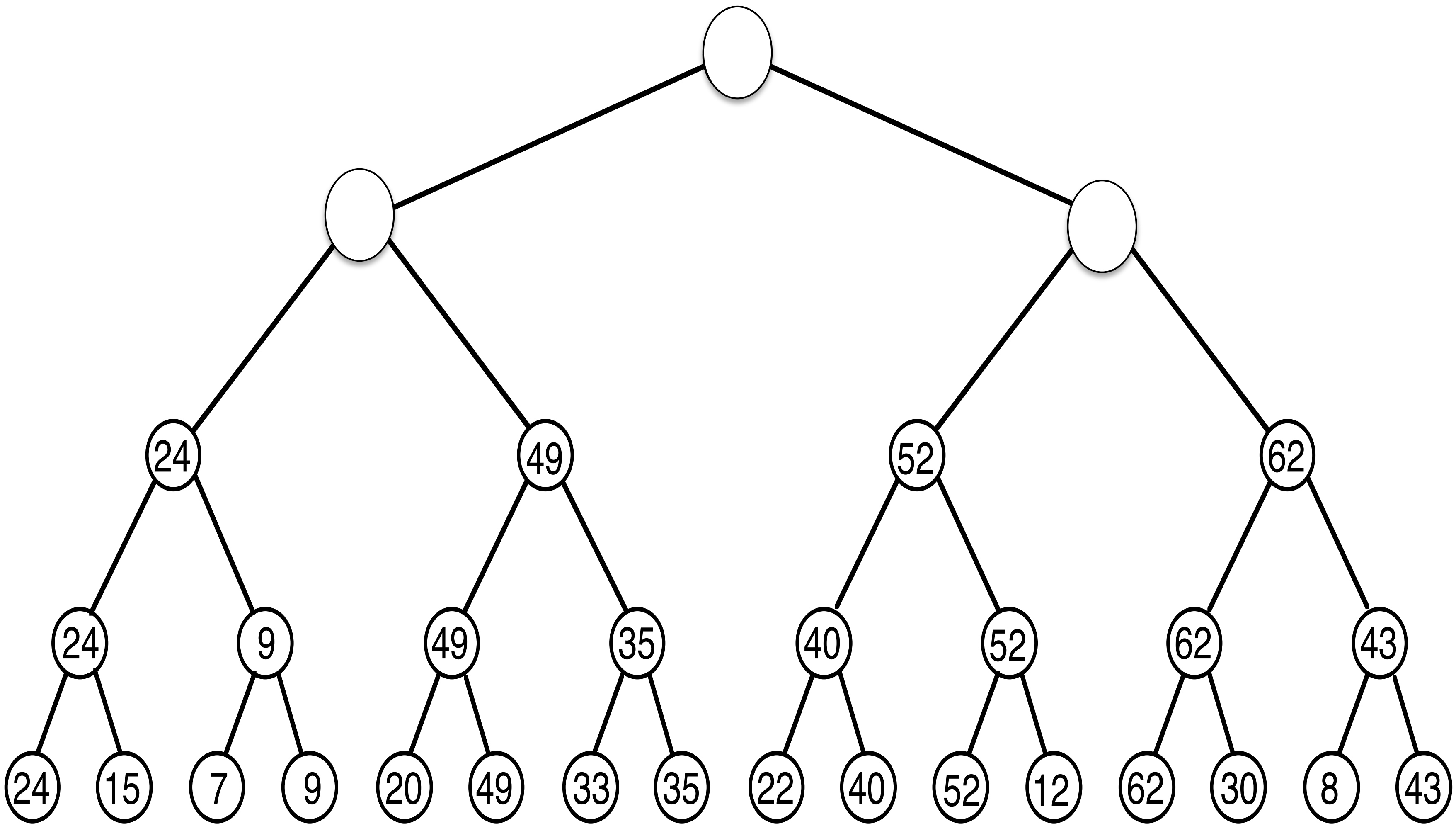
La semplicità del problema ci permette di stabilire limiti esatti. In genere potremo calcolarli solo in ordine di grandezza. Per questo problema il limite 1 è di ordine $\Omega(\log n)$, i limiti 2 e 3 sono di ordiei $\Omega(n)$ quindi equivalenti.

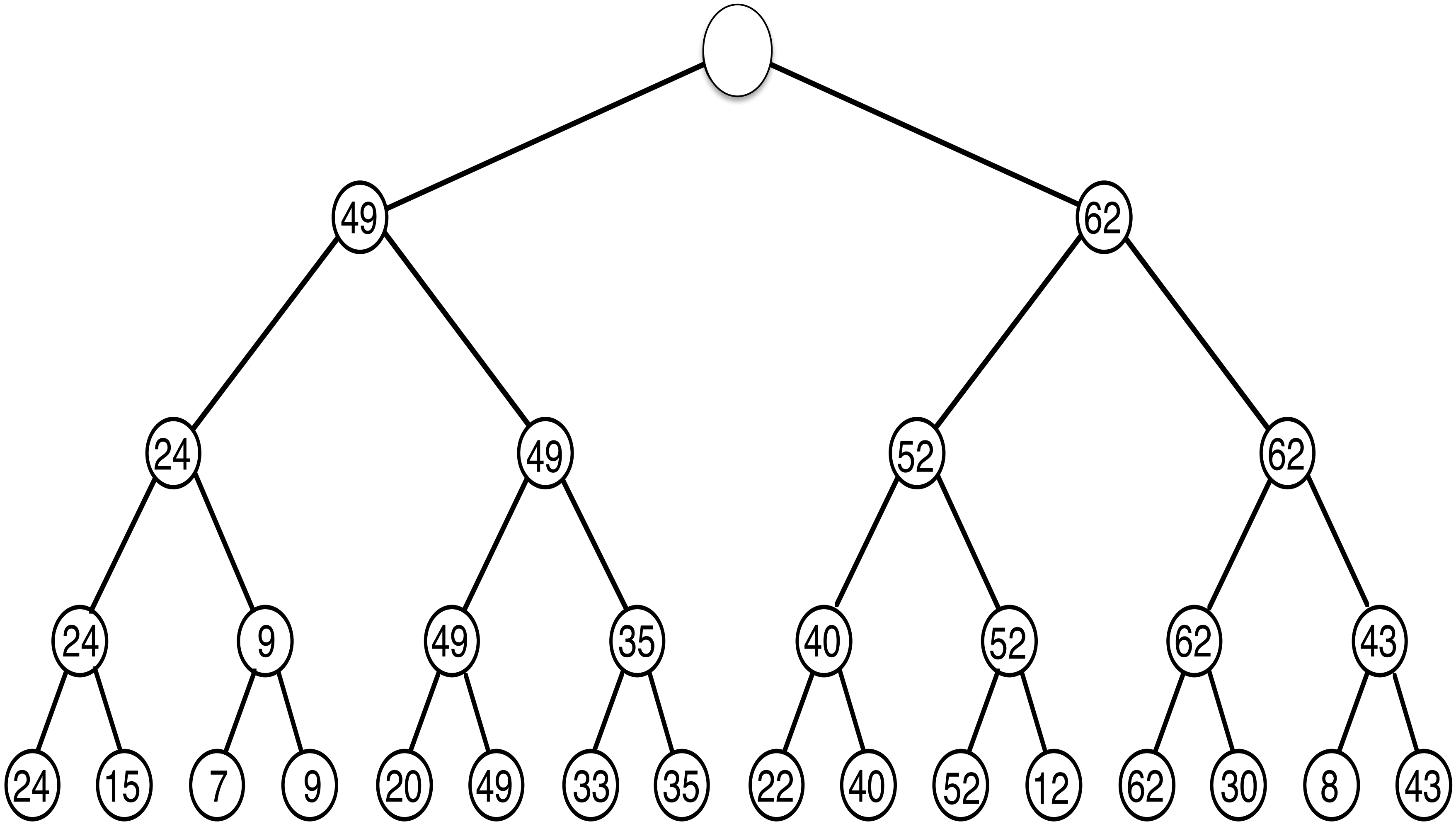
Poiché il valore *n-1* del limite inferiore è uguale al limite superiore dell’agoritmo proposto, questo è ottimo e il problema è computazionalmente chiuso.

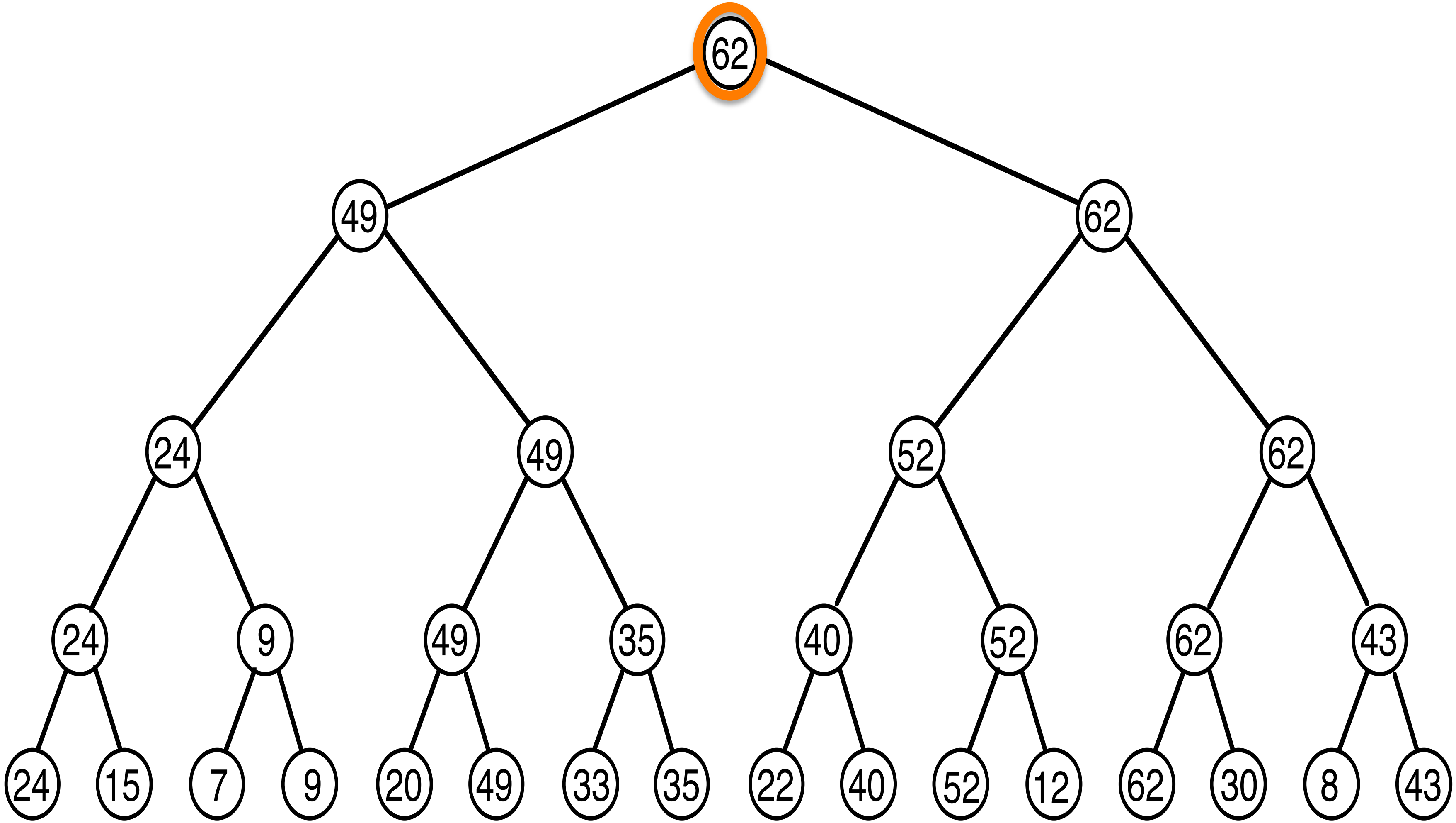
Un altro metodo per trovare il massimo di un insieme impiega la struttura di un torneo a eliminazione diretta


24
15
7
9
20
49
33
35
22
40
52
12
62
30
8
43


24
9
49
35
40
52
62
43
24
15
7
9
20
49
33
35
22
40
52
12
62
30
8
43


24
49
52
62
24
9
49
35
40
52
62
43
24
15
7
9
20
49
33
35
22
40
52
12
62
30
8
43


49
62
24
49
52
62
24
9
49
35
40
52
62
43
24
15
7
9
20
49
33
35
22
40
52
12
62
30
8
43

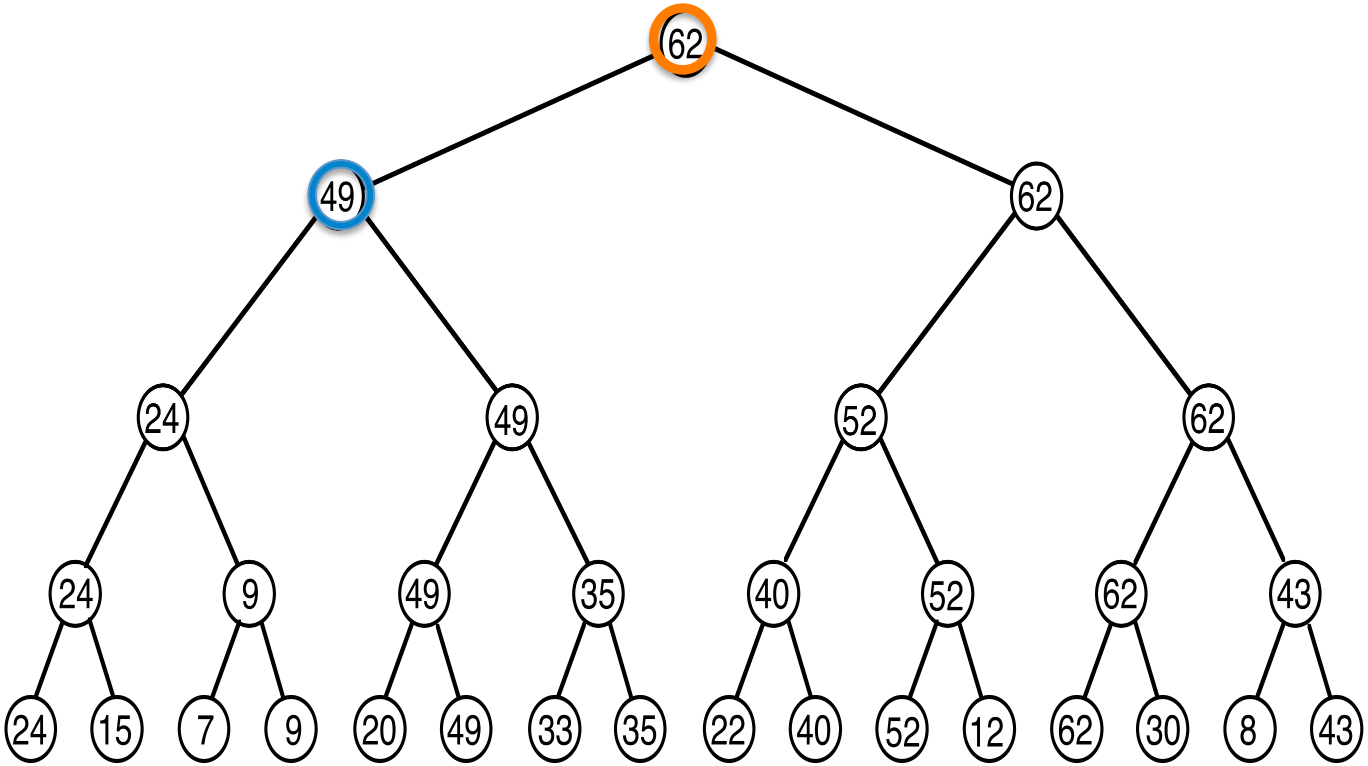

62
49
62
24
49
52
62
24
9
49
35
40
52
62
43
24
15
7
9
20
49
33
35
22
40
52
12
62
30
8
43

Posto $n = 2^t$, gli elementi si confrontano in coppie e restano in gara $n/2$ vincitori dopo ill primo turno, $n/4$ dopo il secondo turno, fino alla finale tra 2 partecipanti. Il numero di confronti $\sum_{i=0}^{t-1} 2^i = 2^t - 1 = n - 1$

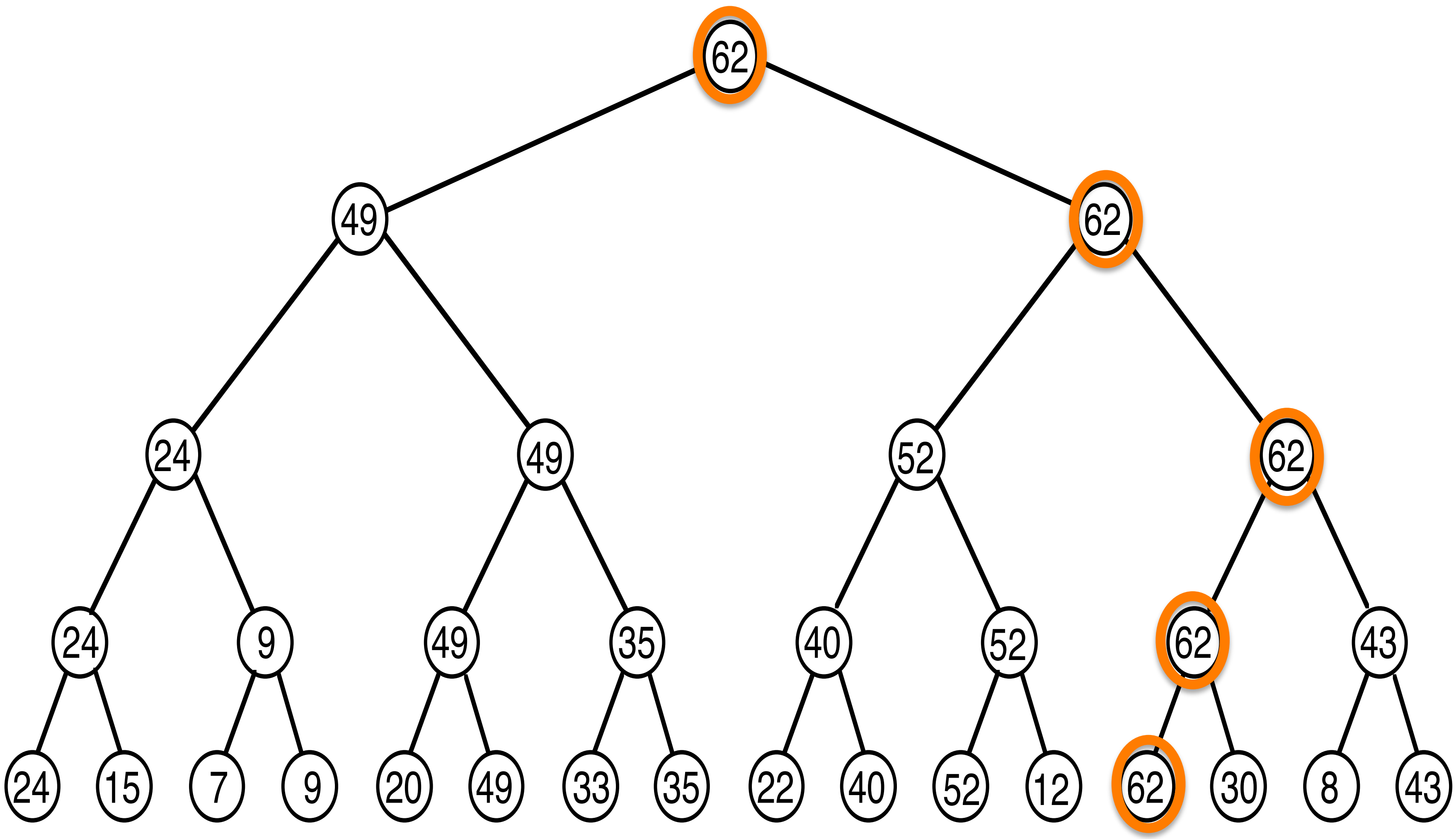
I due algoritmi sono entrambi ottimi anche se eseguono confronti diversi tra elementi

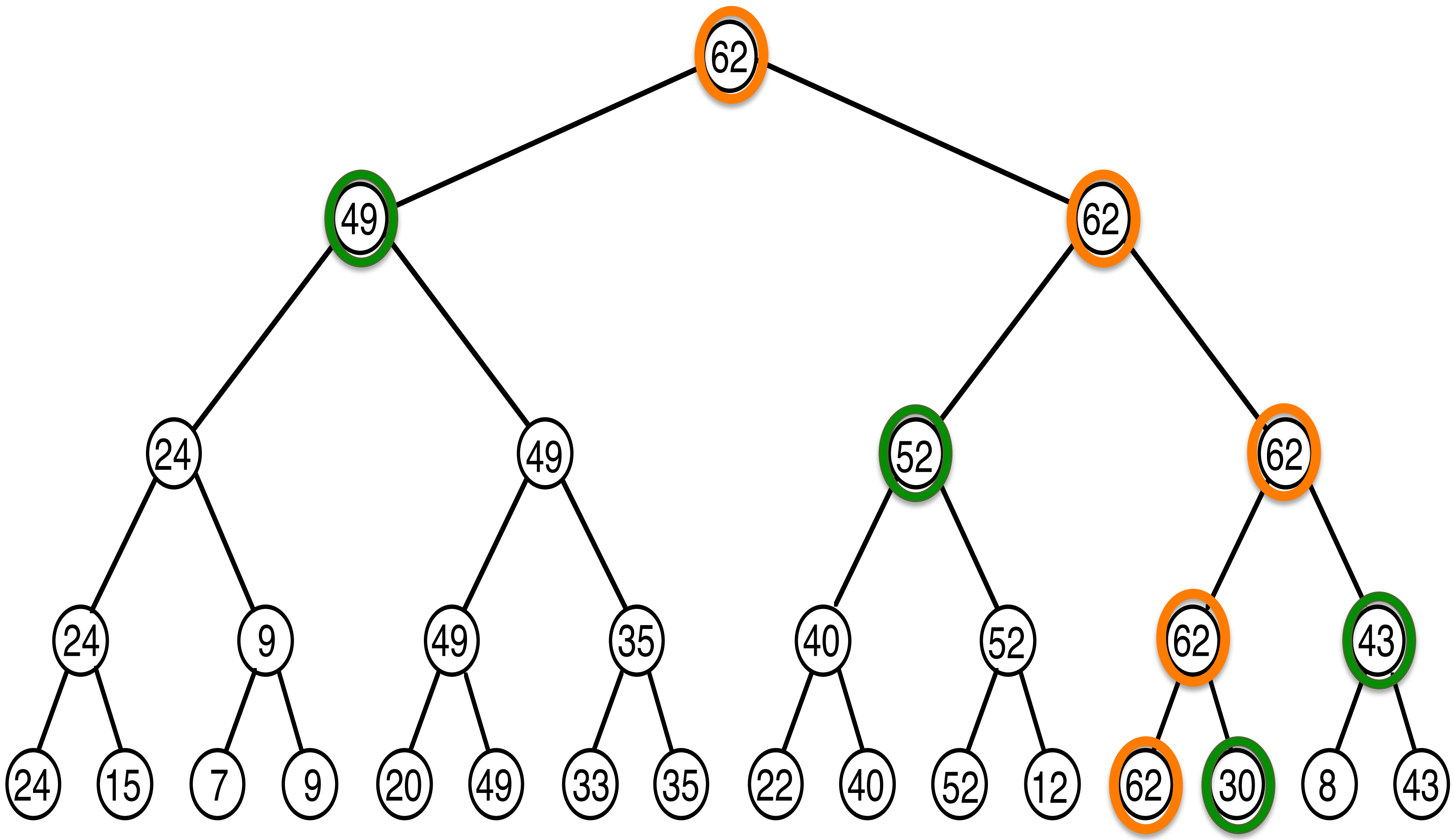
Poniamoci ora una nuova domanda sull'albero del torneo

Lewis Carroll, St. James' Gazette, 1883



49 è entrato in finale con 62, ma è veramente il secondo?


62
49
62
24
49
52
62
24
9
49
35
40
52
62
43
24
15
7
9
20
49
33
35
22
40
52
12
62
30
8
43



62
49
62
24
49
52
62
24
9
49
35
40
52
62
43
24
15
7
9
20
49
33
35
22
40
52
12
62
30
8
43

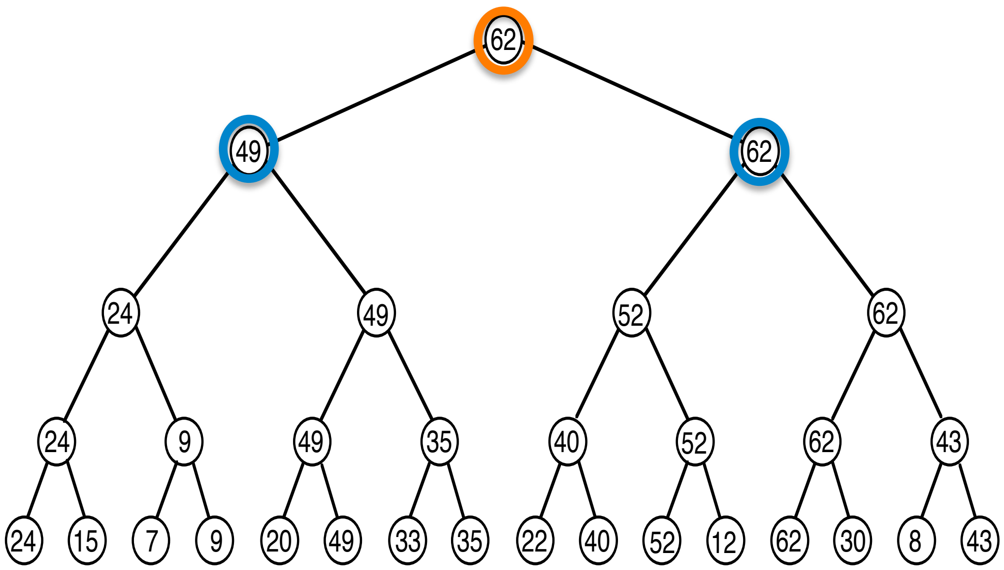
Il secondo deve essere cercato tra i candidati che hanno "perso" in un confronto con il massimo

Il numero di questi candidati è pari all'altezza $\log_2 n$ dell'albero (4 in questo caso). Per determinare il secondo classificato si può fare un altro torneo tra questi candidati con $\log_2 n - 1$ confronti, quindi $n + \log_2 n - 2$ confronti in totale (18 in questo caso)

Questo algoritmo è ottimo, ma dimostrarlo non è affatto semplice

Sull'albero del torneo abbiamo indicato le operazioni di confronto nell'ordine in cui vengono eseguite. Ma come si definisce "semplicemente" un algoritmo per un arbitratio numero n di elementi?

Il problema, e l'algoritmo di soluzione, possono avere una definizione ricorsiva.



Il problema e il suo algoritmo possono essere definiti e codificati osservando che il campione è il vincitore tra i vincitori delle due metà del tabellone.

Un procedimento ricorsivo su n dati chiama se stesso su sottoinsiemi di tali dati, finché questi sottoinsiemi raggiungono una dimensione costante (e piccola) per cui il problema è risolto in modo diretto.

L'algoritmo ricorsivo del torneo si codifica come segue

MASSIMO-RICOR(A, i, j, m)

// Ricerca ricorsiva del massimo in un sotto-vettore di A ***tra le posizioni*** i, j ***//***

input A, i, j; **output** m;

if $(j - i == 1)$ $\{$**if** $(A[i] < A[j])$ $m = A[j]$

else $m = A[i]$; **return** $m\}$;

$k = \lfloor (i + j)/2 \rfloor$;

MASSIMO-RICOR(A, i, k, m_1);

MASSIMO-RICOR$(A, k + 1, j, m_2)$;

if $(m_1 < m_2)$ $m = m_2$ **else** $m = m_1$;

Il calcolo si innesca con la "chiamata" esterna MASSIMO-RICOR$(A, 0, n-1, m)$ **sull'intero vettore** A**.**

Il numero $C(n)$ di confronti tra elementi è soggetto a una *relazione di ricorrenza* che si ottiene da un esame del programma:

$$C(n) = 1, \qquad \text{per } n = 2;$$

$$C(n) = 2\,C(n/2) + 1, \quad \text{per } n > 2.$$

La soluzione è *C(n) = n-1* (Vedi dispense)

Ponendo $n = 2^t$ si ha:

Dispense

$$
\begin{aligned}
C(2^t) &= 2\,C(2^{t-1}) + 1 \\
&= 2\,(2\,C(2^{t-2}) + 1) + 1 \\
&= 4\,C(2^{t-2}) + 2 + 1 \\
&= 4\,(2\,C(2^{t-3}) + 1) + 2 + 1 \\
&= 8\,C(2^{t-3}) + 4 + 2 + 1 \\
&= \cdots\cdots\cdots \\
&= 2^{t-1}(C(2^{t-(t-1)}) + 2^{t-2} + 2^{t-3} + \ldots + 1 \\
&= 2^{t-1} + 2^{t-2} + 2^{t-3} + \ldots + 1 \\
&= 2^t - 1 = n - 1.
\end{aligned}
$$

Ricerca di un elemento *k* in un insieme di *n* elementi

Memorizziamo gli elementi nelle celle A[0] . . . A[n-1] di un vettore A[0…n] ove la cella A[n] è usata per controllo.

Un algoritmo *iterativo* elementare di ricerca, detto RICERCA-SEQUENZIALE, scandisce gli elementi del vettore.

RICERCA-SEQUENZIALE(k, A, r)

input k, A; **output** r;

$A[n] = k$;

$i = 0$;

while $(k \neq A[i])$ $i = i + 1$;

if $(i \leq n - 1)$ $r = i$ **else** $r = -1$;

Il risultato $r = -1$ indica che k non è contenuto nell'insieme

L'algoritmo si arresta dopo $i + 1$ ripetizioni del ciclo se k appare in $A[i]$ con $i \leq n-1$, altrimenti richiede $n+1$ iterazioni se k non è contenuto nell'insieme.

Poiché ogni iterazione richiede tempo costante, il tempo $T(n)$ è di ordine $O(n)$.

Se gli elementi appaiono in A senza alcuna regola un limite inferiore per il problema è $\Omega(n)$ perché tutti gli elementi devono ovviamente essere esaminati almeno una volta. Dunque l'algoritmo RICERCA-SEQUENZIALE è ottimo.

Ben diverso è il caso in cui gli elementi siano ordinati in ordine crescente (o alfabetico).

Il nuovo algoritmo, detto ricerca binaria, si definisce spontaneamente in modo ricorsivo

RICERCA-BINARIA(k, A, i, j, r) ricerca l'elemento k
nella porzione di A tra gli indici i , j .
r è l'output e indica la posizione di k in A.
Il calcolo si innesca con la chiamata esterna
RICERCA-BINARIA(k, A, 0, n-1, r) sull'intero vettore A.

```
RICERCA-BINARIA(k, A, i, j, r)
  input k, A, i, j; output r;
  if (i > j) {r = −1; stop;}
  else {m = ⌊(i + j)/2⌋;
    if (k == A[m]) {r = m; stop;}
    if (k < A[m])
      RICERCA-BINARIA(k, A, i, m−1, r)
    else
      RICERCA-BINARIA(k, A, m+1, j, r);
      }
```

Calcolo della complessità: il tempo *T(n)* è espresso dalla relazione di ricorrenza:

$$T(n) = c_1, \textbf{ con } c_1 \textbf{ costante,} \quad \textbf{per } n = 1;$$

$$T(n) \leq T(n/2) + c_2, \textbf{ con } c_2 \textbf{ costante,}$$

$$\textbf{per } n > 1.$$

La soluzione è $T(n) = c_1 + c_2 \log_2 n$ *(Vedi dispense)*

Quindi *T(n) di ordine O(log n)*

Nell'ipotesi semplificativa che sia $n = 2^t$ per un opportuno intero t abbiamo:

$$
\begin{aligned}
T(n) &\leq T(n/2) + c_2 \\
&= (T(n/4) + c_2) + c_2 = T(n/4) + 2c_2 \\
&= (T(n/8) + c_2) + 2c_2 = T(n/8) + 3c_2 \\
&= \ldots\ldots\ldots \\
&= (T(n/2^t) + c_2) + (t-1)c_2 = T(1) + tc_2 \\
&= c_1 + tc_2.
\end{aligned}
$$

Valutazione di un limite inferiore se il vettore A è ordinato

Il confronto tra k e un elemento $A[i]$ può generare le tre risposte $k<A[i]$, $k=A[i]$, $k>A[i]$, dunque dopo t confronti si aprono al massimo 3^t percorsi di computazione.

Le soluzioni si trovano nelle foglie di un albero ternario con 3^t foglie.
Esse sono $n+1$ (k è il primo, secondo, . . . , n-esimo elemento dell'insieme, o non è presente) e abbiamo $3^t \geq n+1$ dunque $t \geq \lceil \log_3 n \rceil$.

Il numero di confronti t è di ordine $\Omega(\log n)$. Poiché questo limite inferiore coincide con il limite superiore $O(\log n)$ di RICERCA-BINARIA, questo algoritmo è ottimo.