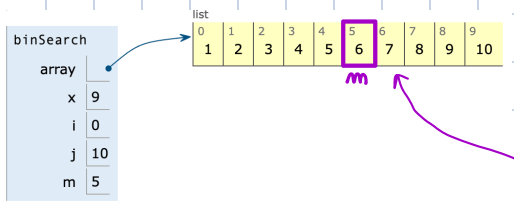
METODO DELL'ITERAZIONE

CON QUESTO METODO ANDIAMO A "SROTOLARE" LA RICORSIONE, OTTENENDO (NON PER FORZA) UNA SOMMATORIA DIPENDENTE DALLA DIMENSIONE m DEL PROBLEMA INIZIALE.

ANALISI BIN SEARCH (METODO ITERAZIONE)

algoritmo RicercaBinariaRic(array *L*, elem *x*, int *i*, int *j*) → intero

1. **if** (i>j) **then return** -1
2. m=⌊ (i+j)/2⌋
3. **if** (L[m]=*x*) **then return** m
4. **if** (L[m]>x) **then return** RicercaBinariaRic(L, *x*, i, m-1)
5. **else return** RicercaBinariaRic(L, *x*, m+1,j)

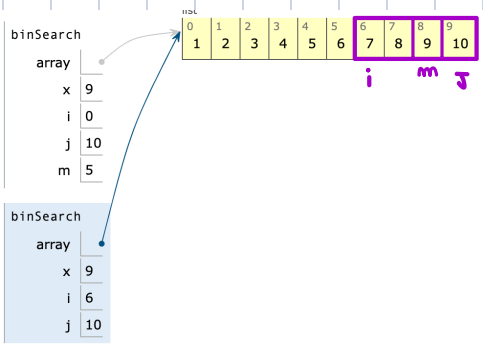


m = 5

L [m] = 6 (6 < 9 QUINDI STA NELLA PARTE DX PERCHÉ L[m] È LA METÀ)

RETURN RICERCABINARIARIC (L , x , m+1, J)

ARRAY — ELEM DA CERCARE — POSIZIONE DOPO LA METÀ, LA METÀ È m — ULTIMA POSIZ. DELL'ARRAY



m = 8

L [m] = 9

RETURN m

L'ALGORITMO PARTE CON i = INDEX 0 E CON J = POSIZIONE ULTIMA DELL'ARRAY, SOMMANDO i + J E DIVIDENDO PER 2 TROVO LA POSIZIONE A METÀ DELL'ARRAY

CONTROLLO SE L'ELEMENTO ALLA METÀ È IL NUMERO CHE STO CERCANDO, SE:

- È IL NUMERO ⟶ RETURN m
- SE L[m] > X ⟶ RICOMINCIO L'ALGORITMO FORNENDO COME POSIZIONE INIZIALE i = m+1

ELEMENTO ALLA METÀ

TENENDO $J=J$

OVVERO COME POS. INIZIALE IL PRIMO NUMERO DOPO LA METÀ

- SE $L[m] < X$ ⟶ RICOMINCIO L'ALGORITMO FORNENDO COME POSIZIONE INIZIALE $i=i$ QUINDI PARTENDO SEMPRE DAL PRIMO ELEMENTO, E $J=m-1$

ARRIVANDO QUINDI FINO ALL' ELEMENTO PRIMA DELLA METÀ

CONSIDERANDO $T(m) = T(m/2) + O(1)$ VADO A SPIOTOLARE L'EQUAZIONE DI RICORR.

$T(m) = T(m/2) + O(1)$

$T(m) = T(m/4) + O(1) + O(1)$

$T(m) = T(m/8) + O(1) + O(1) + O(1)$

...

$T(m) = T(m/2^i)$

↳ CHE VALORE POSSO DARE AD i TALE CHE $\boxed{m/2^i = 1}$?

↳ $\log_2 m$ ✓

QUINDI $T(m) = O(\log m)$

ANALISI FIBONACCI 2 (METODO ITERAZIONE)

algoritmo `fibonacci2`*(intero n)* → *intero*
  if (n≤2) **then return** 1
  else return `fibonacci2`(n-1) + `fibonacci2`(n-2)

$T(m) = T(m-1) + T(m-2) + O(1)$

$T(m) = T(m-2) + T(m-3) + O(1) + T(m-3) + T(m-4) + O(1)$

$T(m) = T(m-3) + T(m-4) + O(1) + T(m-4) + T(m-5) + O(1)$

...

COME POSSIAMO VEDERE ANDREMMO AVANTI ALL' INFINITO... PER QUESTO PER QUANTO RIGUARDA UN ALGORITMO DEL GENERE DOBBIAMO UTILIZZARE L' ALBERO DELLA RICORSIONE.

METODO ALBERO DELLA RICORSIONE

IDEA:

- DISEGNARE L'ALBERO DELLE CHIAMATE RICORSIVE INDICANDO LA DIMENSIONE DI OGNI NODO
- STIMARE IL TEMPO SPESO DA OGNI NODO DELL'ALBERO
- STIMARE IL TEMPO COMPLESSIVO "SOMMANDO" IL TEMPO SPESO DA OGNI NODO

SUGGERIMENTO 1: SE IL TEMPO SPESO DA OGNI NODO È COSTANTE, $T(m)$ È PROPORZIONALE AL NUMERO NODI.

SUGGERIMENTO 2: A VOLTE ANALIZZIAMO L'ALBERO PER LIVELLI ANALIZZANDO IL TEMPO SPESO SU OGNI LIVELLO & STIMANDO IL NUMERO DI LIVELLI

ESEMPIO:

$T(m) = T(m-1) + 1$
$T(1) = 1$

n
$n-1$
$n-2$
$\vdots$
$n-i$
$\vdots$
2
1

- OGNI NODO COSTA 1
- CI SONO m NODI

$T(m) = \Theta(m)$

ESEMPIO:

$T(m) = T(m-1) + m$
$T(1) = 1$

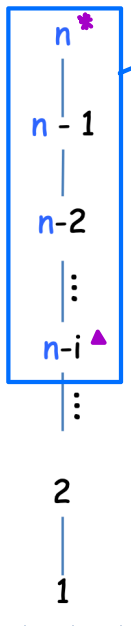
n
$n-1$
$n-2$
$\vdots$
$n-i$
$\vdots$
2
1

- OGNI NODO COSTA m *
- CI SONO m NODI

$T(m) = O(m^2)$

* IMMAGINE CHE L'OPERAZIONE $+m$ SIA UN FOR DA 1...N, QUINDI CI SAREBBE UN FOR AD OGNI CHIAMATA

NELL'ESEMPIO 2 VALE $\Theta(m^2)$, ABBIAMO DIMOSTRATO CHE NEL PEGGIORE DEI CASI ABBIAMO UN COSTO CHE È AL PIÙ m^2 QUINDI $O(m^2)$, ADESSO PER STABILIRE CHE È $\Theta(m^2)$ DOBBIAMO DIMOSTRARE CHE È ANCHE $\Omega(m^2)$, QUINDI:



PRENDIAMO COME ISTANZA PIÙ PICCOLA METÀ ALBERO, ABBIAMO QUINDI $m/2$ NODI.

IL COSTO DI OGNI NODO È $\geq m/2$, PERCHE?
METTIAMO CASO CHE PER OGNI NODO ABBIAMO UN CICLO FOR CHE VA DA 0 AL VALORE DI m, COME ABBIAMO GIÀ DETTO IL WORST CASE È ARRIVARE DA 0 A m *, MA IL CASO MINIMO È ARRIVARE DA 0 A $m/2$ ▲.

ABBIAMO QUINDI $m/2$ NODI PER UN COSTO MINIMO DI $m/2$ QUINDI:

$$\frac{m}{2} + \frac{m}{2} = \frac{m^2}{4}$$

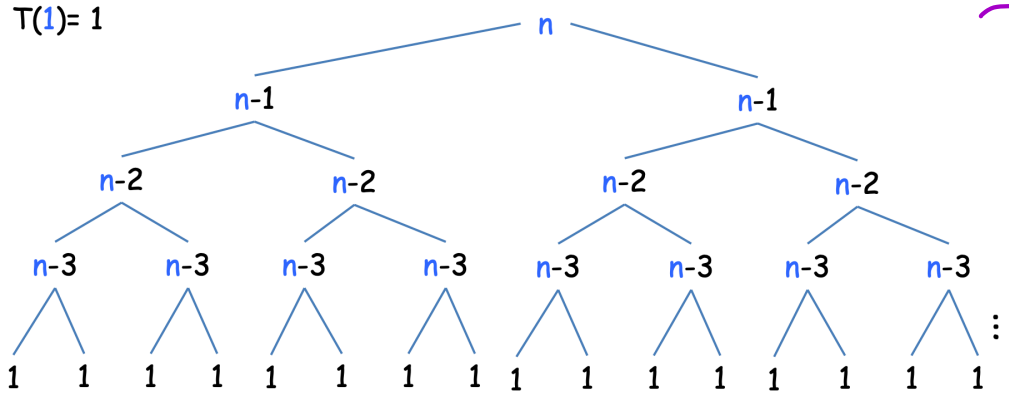
CHE SAREBBE $\frac{1}{4} \cdot m^2$ QUINDI IGNORO IL $\frac{\ }{4}$

QUINDI: $T(m) = \Omega(m^2)$ E DI CONSEGUENZA $T(m) = \Theta(m^2)$ ✓

ESEMPIO:

$T(n) = 2T(n-1) + 1$

$T(1) = 1$



OGNI NODO È UNA CHIAMATA

COSTO OGNI NODO: 1

ALTEZZA ALBERO: $h = m-1$ (CONTO GLI ARCHI)

DI NODI: $\sum_{i=0}^{h} 2^i = 2^{h+1} - 1$ (AVENDO $h = m-1$, 2^{h+1} È EQUIVALENTE A 2^m)

EQUIVALENTI

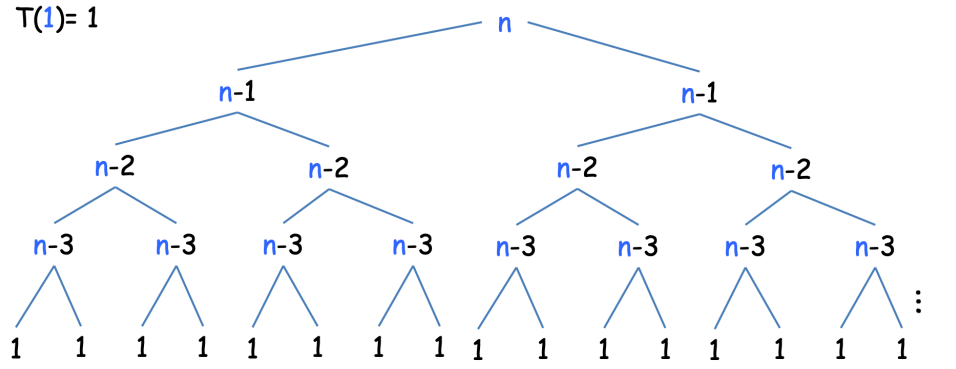
$$T(m) = 2^m - 1 = \Theta(2^m)$$ ✓

SAPENDO CHE OGNI NODO È UNA CHIAMATA RICORSIVA, NON CONTANDO LE FOGLIE VISTO CHE NON EFFETTUANO ALTRE CHIAMATE, IL COSTO DI "GENERAZIONE" DELL'ALBERO SARÀ APPUNTO IL COSTO DI OGNI NODO (CHE IN QUESTO CASO È 1), MOLTIPLICATO PER IL NUMERO DI CHIAMATE (OVVERO IL NUMERO DI NODI).

ESEMPIO:

$T(n) = 2T(n-1) + n$

$T(1) = 1$



COSTO OGNI NODO: n

ALTEZZA ALBERO: $h = n-1$

DI NODI: $\sum_{i=0}^{h} 2^i = 2^{h+1} - 1$

$T(n) = O(n \cdot 2^n)$ ✓

COSTO DI OGNI NODO MOLTIPLICATO PER IL NUMERO DI CHIAMATE (OVVERO IL NUMERO DI NODI)

ESEMPIO CON MAGGIORAZIONI

$T(n) = \underline{T(n-1)}_{*} + T(n-2) + 1$

$T(1) = 1$

→ POSSO UTILIZZARE UNA MAGGIORAZIONE PER FORNIRE UN UPPERBOUND

↓

$T(n) \leq R(n)$

$R(n) = 2 \cdot R(\underline{n-1}_{*}) + 1 \equiv R(n-1) + R(n-1) + 1$

HO MESSO $n-1$ PERCHÉ $n-1$ È SICURAMENTE MAGGIORE ANCHE DI $n-2$ QUINDI $n-1$ "COPRE" ANCHE $n-2$.

AVENDO $T(n-1) + T(n-2)$ SICURAMENTE UTILIZZANDO $2 \cdot T(n-1)$ AVRÒ UN UPPERBOUND

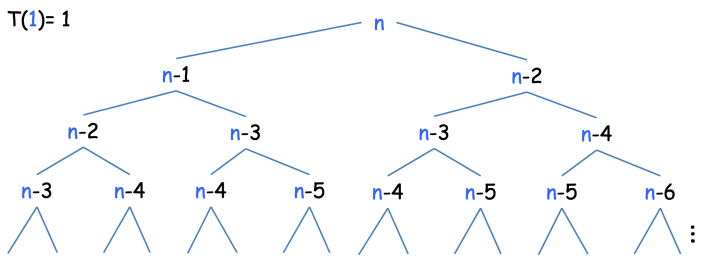
AVENDO CHE $R(m) = \Theta(2^m)$ CON $T(m) \leq R(m)$, QUESTO IMPLICA CHE $T(m) = O(2^m)$.

MA POSSO SAPERE SE VALE $T(m) = \Theta(2^m)$?

CON UNA FORMULA CHIUSA SAPPIAMO CHE:

$T(n) = T(n-1) + T(n-2) + 1$

$T(1) = 1$



albero chiamate ricorsive dell'algorito `Fibonacci2`!

quanto costa ogni nodo? ...uno

quanti nodi ha l'albero? $\Theta(\phi^n)$ ➡ $T(n) = \Theta(\phi^n)$

$[T(n) = o(2^n)]$

ϕ = NUMERO AUREO CHE VALE CIRCA 1.618

POSSIAMO SAPERE CHE LA COMPLESSITÀ DELL'ALBERO È $\Theta(\phi^m)$ DOVE ϕ È UN NUMERO < 2, QUINDI ESSENDO MINORE DI 2 POSSIAMO CONCLUDERE CHE:

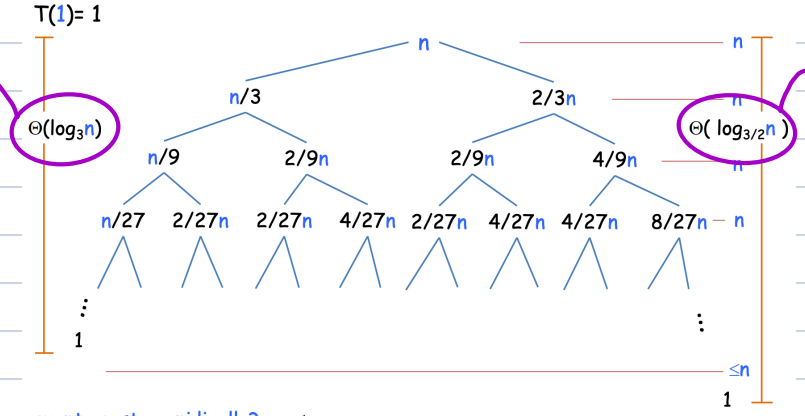
$T(m) = \Theta(\phi^m)$ ✓ OPPURE $T(m) = o(2^m)$

ESEMPIO

$T(n) = T(n/3) + T(2/3\, n) + n$

$T(1) = 1$

QUESTO RAMO FINISCE PRIMA DEL RAMO DX PERCHÉ ARRIVERÀ AD UN PUNTO IN CUI AVREMO $\frac{m}{m}$ ED ENTREREMO NEL CASO BASE.



QUESTO RAMO ARRIVERÀ AL CASO BASE PIÙ LENTAMENTE RISPETTO AL RAMO DI SX.

quanto costa ogni livello? $\leq n$

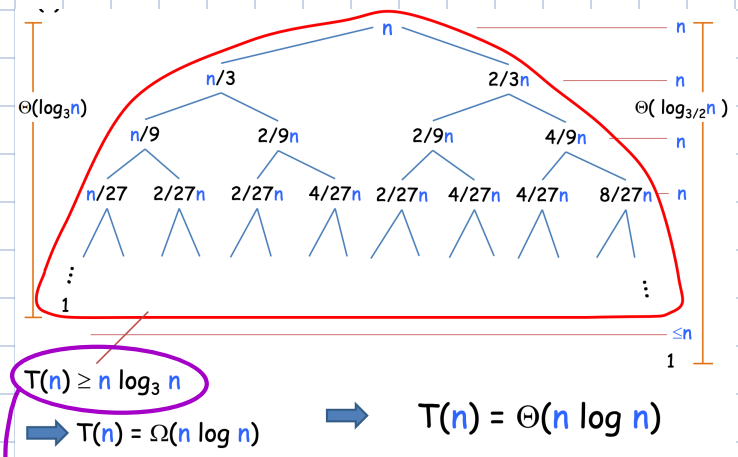
quanti livelli ha l'albero? $O(\log_{3/2} n)$ ➡ $T(n) = O(n \log n)$

vale $T(n) = \Theta(n \log n)$?

QUANTO COSTA OGNI NODO? $\leq m$

QUANTI LIVELLI HA L'ALBERO? $O(\log_{\frac{3}{2}} m)$

ABBIAMO QUINDI $T(n) = O(n \log n)$, MA VALE $\Theta(n \log n)$?



$\Rightarrow T(n) = \Omega(n \log n)$ $\Rightarrow$ $T(n) = \Theta(n \log n)$

IL LOWERBOUND È $T(n) \geq n \log_3 n$ DEL RAMO A SX, QUINDI:

$T(n) = \Omega(n \log n)$

ABBIAMO QUINDI $T(n) = O(n \log n)$ E $T(n) = \Omega(n \log n)$ QUINDI POSSIAMO CONCLUDERE CHE:

$T(n) = \Theta(n \log n)$ ✓

HACK CHRISTIAN

$2T(n-4) + n$

QUALCOSA MOLTIPLICA LA T

$\Theta(n \cdot \sqrt[4]{2^n})$

$T(n-1) + n^3$

LA T NON È MOLTIPLICATA

$\Theta(n^4)$

SE AL POSTO DI -1 ABBIAMO UN VALORE MINORE DI -1, AD ESEMPIO -2 o -3 ALLORA NON È Θ MA O

$T(n-3) + n^3$

LA T NON È MOLTIPLICATA

$O(n^4)$