Titolo unità didattica: Algoritmi ricorsivi [14]

Titolo modulo : Esempi di algoritmi ricorsivi [03-T]

Sviluppo di versioni ricorsive di algoritmi basati sia sull'approccio incrementale sia sull'approccio divide et impera

Argomenti trattati:

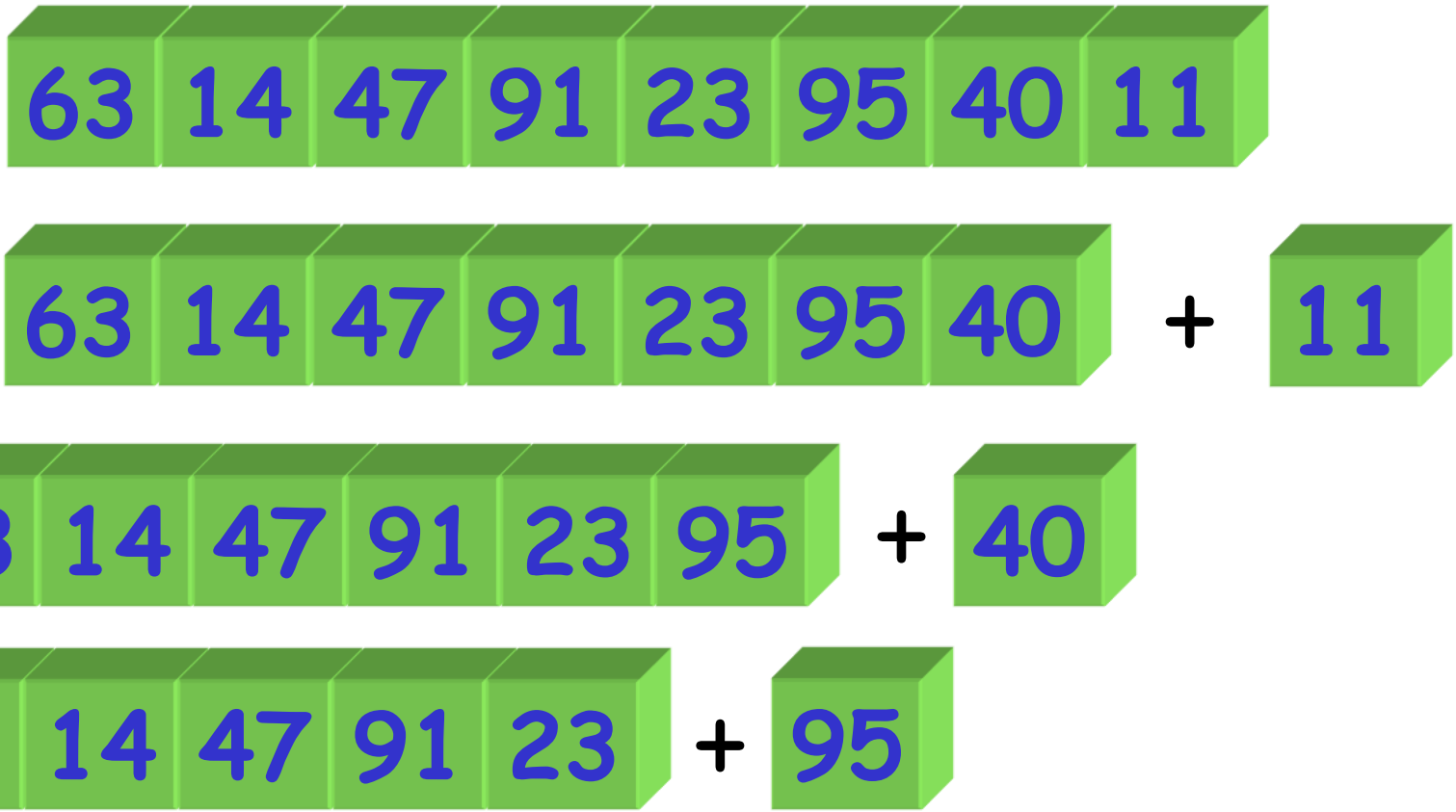
- ✓ algoritmi ricorsivi per la somma degli elementi di un array
- ✓ algoritmi ricorsivi per il massimo di un array
- ✓ algoritmo ricorsivo per il massimo comun divisore
- ✓ algoritmo ricorsivo elementare per il calcolo dell'n-simo numero di Fibonacci
- ✓ analisi della complessità dell'algoritmo ricorsivo elementare per il calcolo dell'n-simo numero di Fibonacci

Prerequisiti richiesti: P1-07-02-T, P1-13-03-T, P1-14-01-T

tecnica di programmazione ricorsiva per l'algoritmo incrementale di **somma** degli elementi di un array

y_k è la somma dei primi k elementi di a

$$y_k = y_{k-1} + a_k \qquad y_0 = 0$$



tail recursion

tecnica di programmazione ricorsiva per l'algoritmo incrementale di **somma** degli elementi di un array

y_k è la somma dei primi k elementi di a

$$y_k = y_{k-1} + a_k \qquad y_0 = 0$$

```
int somma_a_ricAI(int a[], int n) {
 if (n == 1)
/* soluzione del problema banale  */
        { return a[0]; }
     else
       /*  autoattivazione   */
        { return a[n-1] + somma_a_ricAI(a,n-1) ; }
}
```

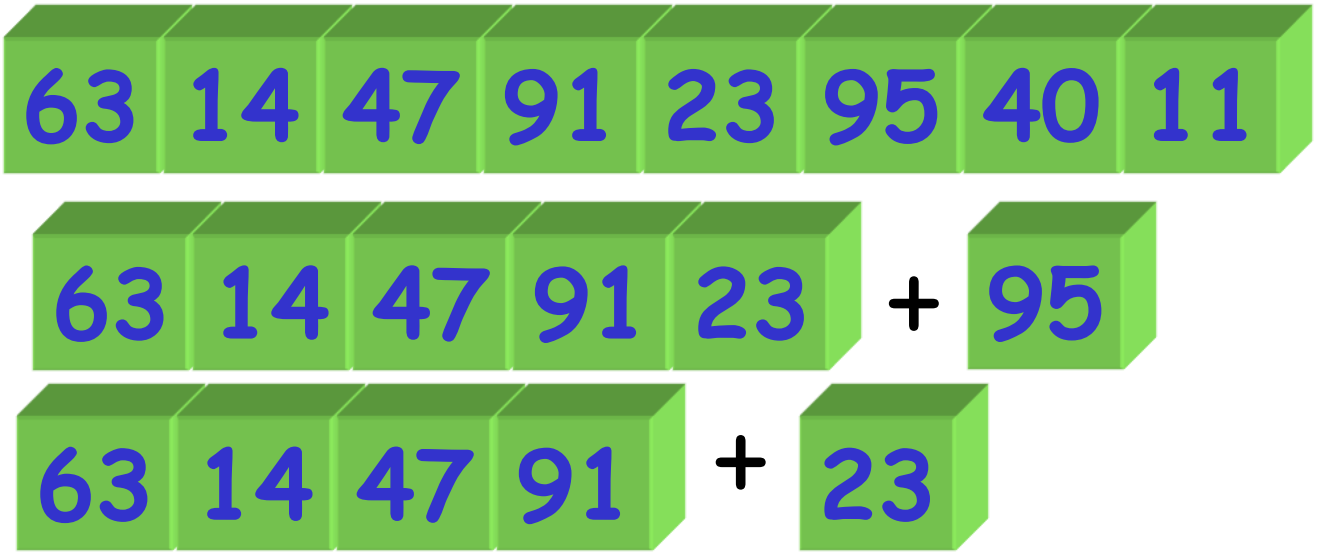
T(n) = n-1
somme

tecnica di programmazione ricorsiva per l'algoritmo incrementale di **somma** degli elementi di un array


```
40 + somma_ric_AI(a,6)
11 + somma_ric_AI(a,7)
```

tecnica di programmazione ricorsiva per l'algoritmo incrementale di **somma** degli elementi di un array

n=8




```
23 + somma_ric_AI(a,4)
95 + somma_ric_AI(a,5)
40 + somma_ric_AI(a,6)
11 + somma_ric_AI(a,7)
```

tecnica di programmazione ricorsiva per l'algoritmo incrementale di **somma** degli elementi di un array

n=8


```
63
14 + somma_ric_AI(a,1)
47 + somma_ric_AI(a,2)
91 + somma_ric_AI(a,3)
23 + somma_ric_AI(a,4)
95 + somma_ric_AI(a,5)
40 + somma_ric_AI(a,6)
11 + somma_ric_AI(a,7)
```

finora, non è stata fatta nessuna addizione !

tecnica di programmazione ricorsiva per l'algoritmo incrementale di **somma** degli elementi di un array

n=8


```
14 + 63
47 + somma_ric_AI(a,2)
91 + somma_ric_AI(a,3)
23 + somma_ric_AI(a,4)
95 + somma_ric_AI(a,5)
40 + somma_ric_AI(a,6)
11 + somma_ric_AI(a,7)
```

prima addizione

tecnica di programmazione ricorsiva per l'algoritmo incrementale di **somma** degli elementi di un array

n=8


```
47 + 77
91 + somma_ric_AI(a,3)
23 + somma_ric_AI(a,4)
95 + somma_ric_AI(a,5)
40 + somma_ric_AI(a,6)
11 + somma_ric_AI(a,7)
```

2° addizione

tecnica di programmazione ricorsiva per l'algoritmo incrementale di **somma** degli elementi di un array

n=8

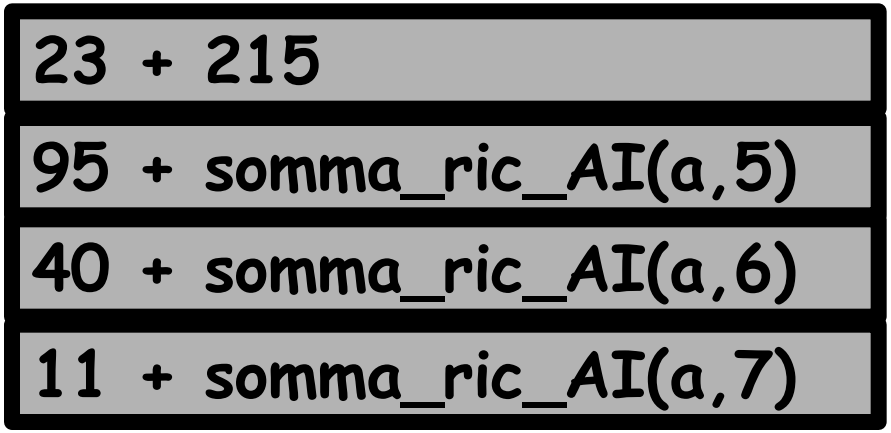

```
91 + 124
23 + somma_ric_AI(a,4)
95 + somma_ric_AI(a,5)
40 + somma_ric_AI(a,6)
11 + somma_ric_AI(a,7)
```

3° addizione

tecnica di programmazione ricorsiva per l'algoritmo incrementale di **somma** degli elementi di un array

n=8

quando lo stack diventa vuoto, l'algoritmo termina



4° addizione

tecnica di programmazione ricorsiva per l'algoritmo incrementale di **somma** degli elementi di un array

n=8

l'ordine temporale delle somme è

63+**14**

77+**47**

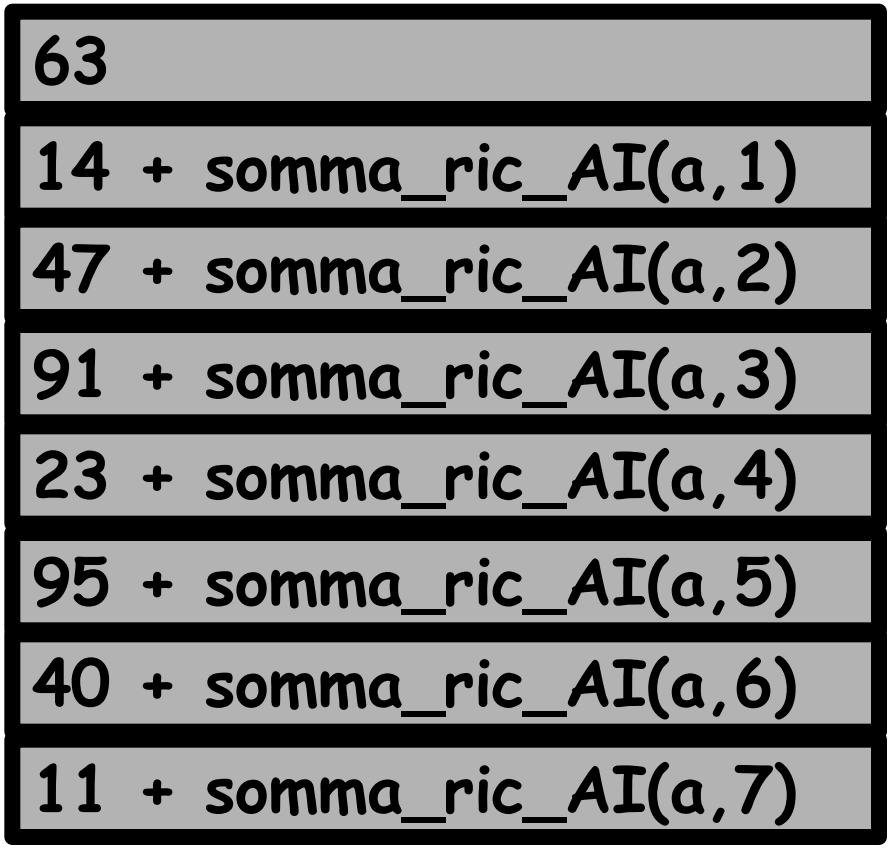
124+**91**

215+**23**

238+**95**

333+**40**

373+**11**



tecnica di programmazione ricorsiva per l'algoritmo incrementale di **somma** degli elementi di un array

n=8

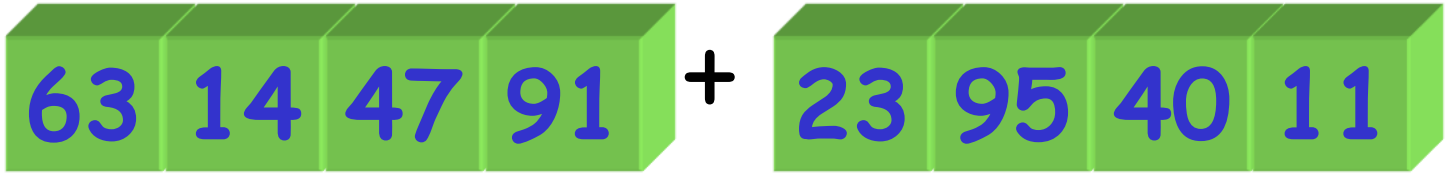
l'ordine temporale delle somme è
63+**14**
77+**47**
124+**91**
215+**23**
238+**95**
333+**40**
373+**11**

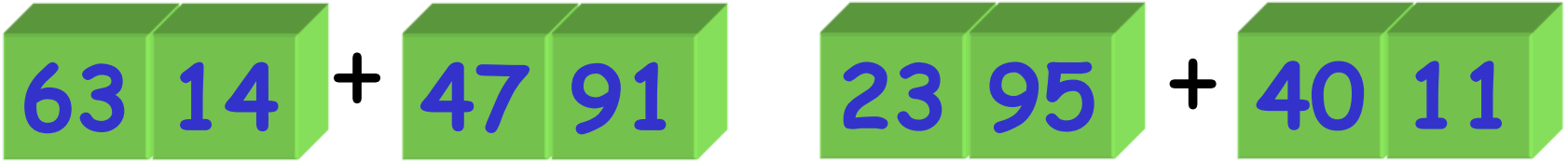
dal primo elemento verso l'ultimo

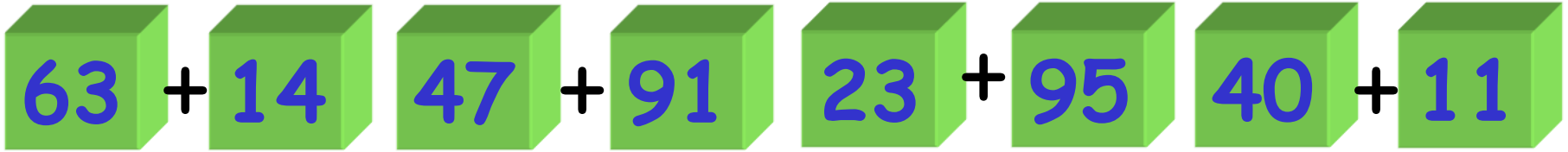
come modificare l'algoritmo per invertire l'ordine temporale delle somme?

dall'ultimo elemento verso il primo ?

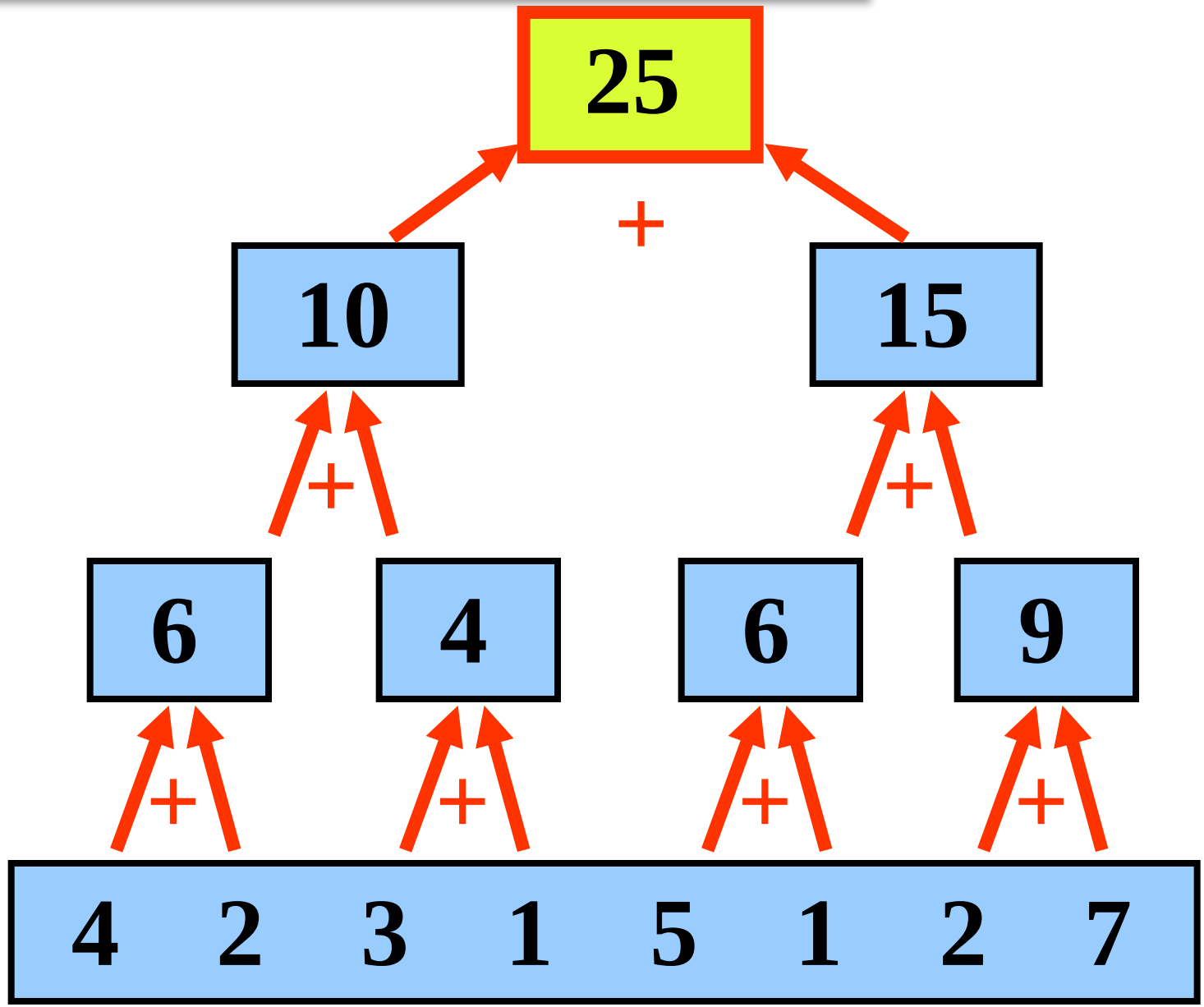
approccio **divide et impera** al problema della **somma** degli elementi di un array



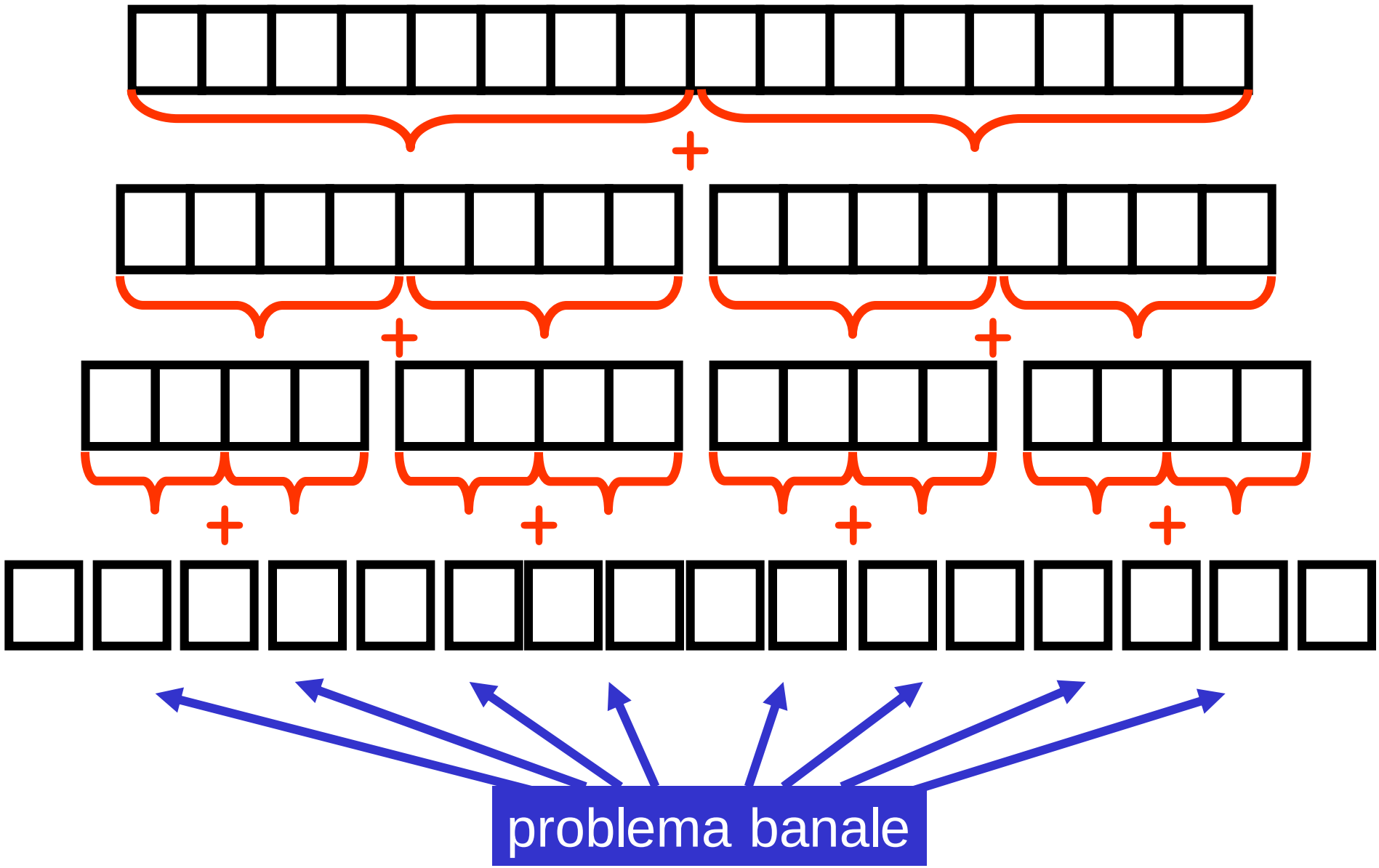




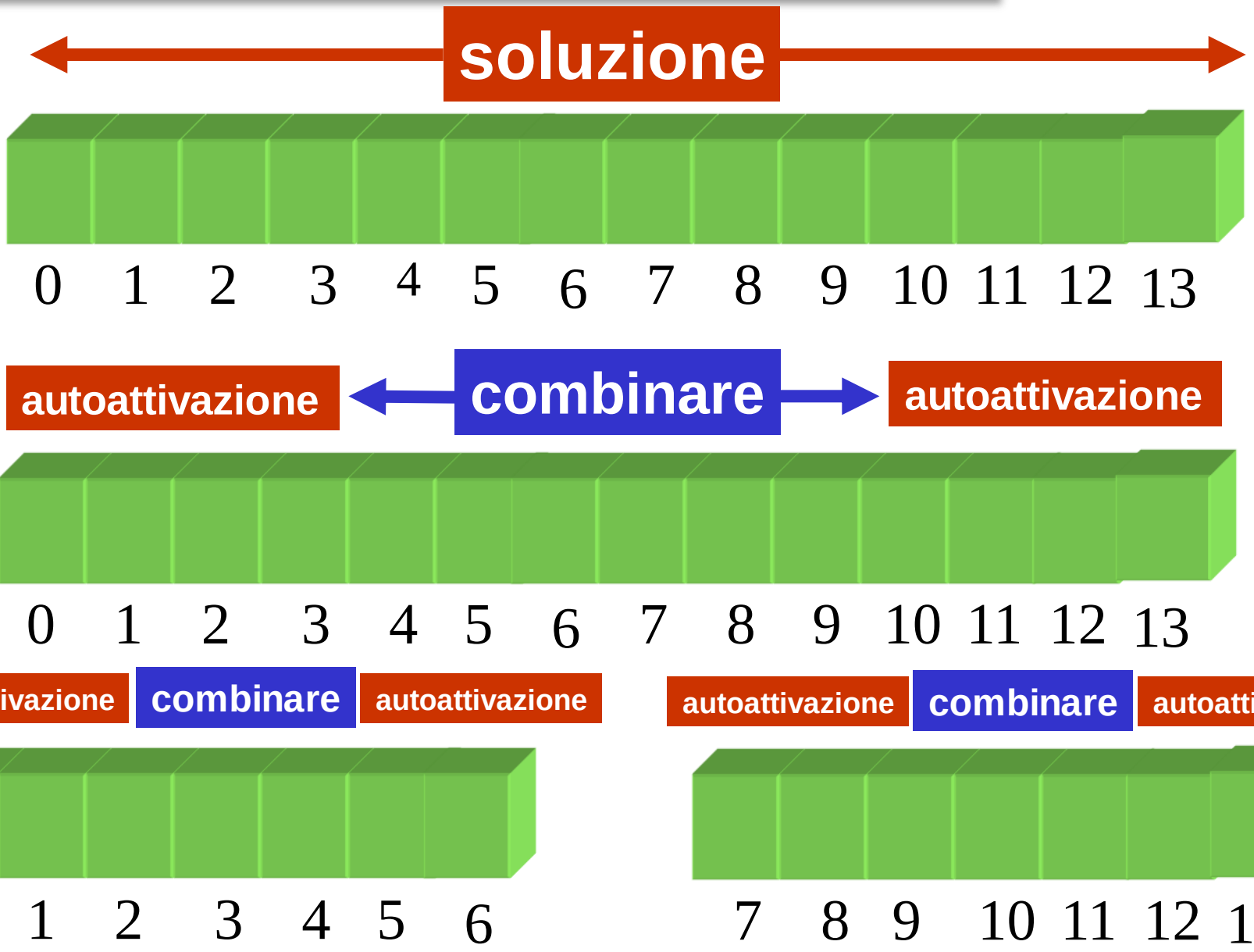
approccio **divide et impera** al problema della **somma** degli elementi di un array



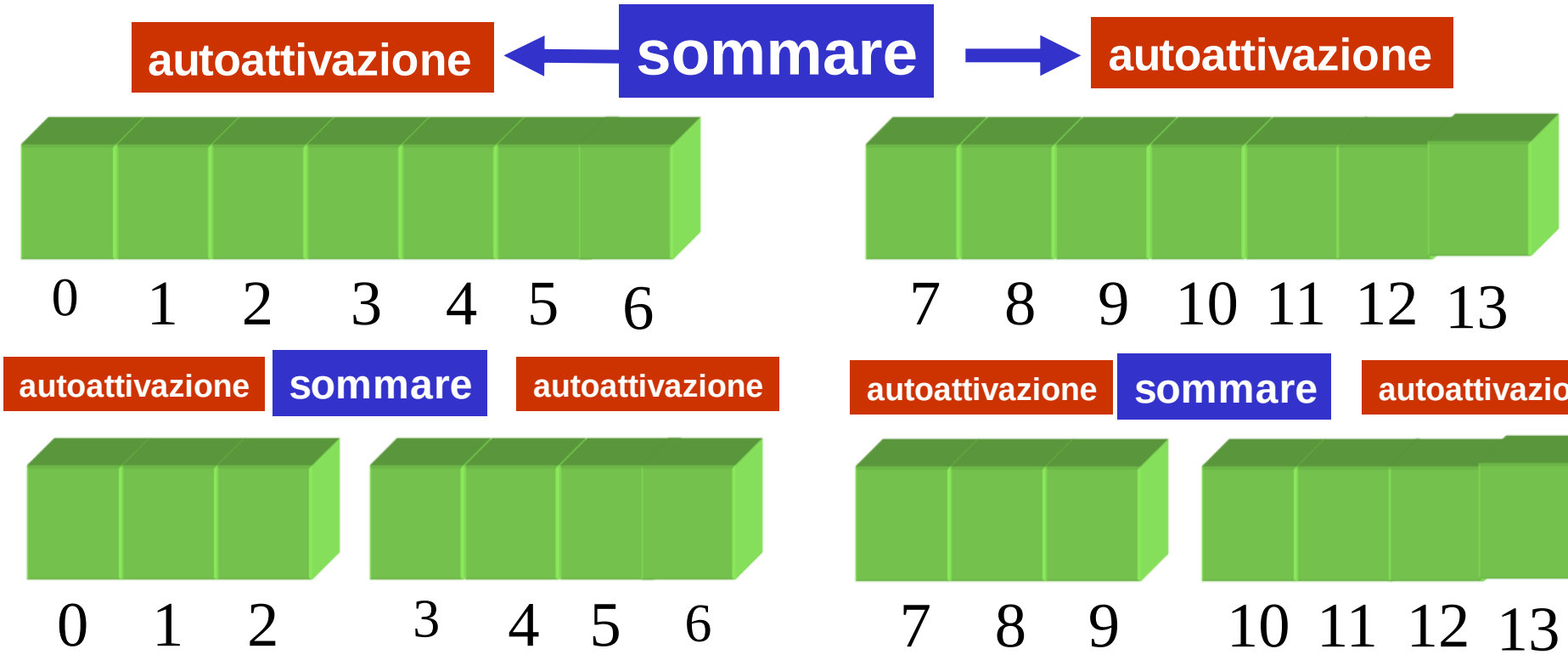
tecnica di programmazione ricorsiva per l'algoritmo divide et impera di **somma** degli elementi di un array



ricorsione e divide et impera



ricorsione e divide et impera



tecnica di programmazione ricorsiva per l'algoritmo divide et impera di **somma** degli elementi di un array

```
int somma_a_ricDI(int a[], int primo, int ultimo) {
 int mediano;
 if (primo == ultimo)
     /* soluzione del problema banale   */
        { return a[primo]; }
   else
    /*  autoattivazioni   */
        {
         mediano = (primo+ultimo)/2 ;
         return somma_a_ricDI(a,primo,mediano) + ...
                somma_a_ricDI(a,mediano+1,ultimo) ;
        }
}
```

versione 1

$T(n) = n-1$ somme

tecnica di programmazione ricorsiva per l'algoritmo divide et impera di **somma** degli elementi di un array

```
int somma_a_ricDI(int a[], int n) {
 int nmezzi;
 if (n == 1)
     /* soluzione del problema banale   */
        { return a[0]; }
   else
    /*  autoattivazioni   */
        {
         nmezzi = n/2 ;
         return somma_a_ricDI(a, nmezzi) + ...
                    somma_a_ricDI(a+nmezzi, n-nmezzi) ;
        }
}
```

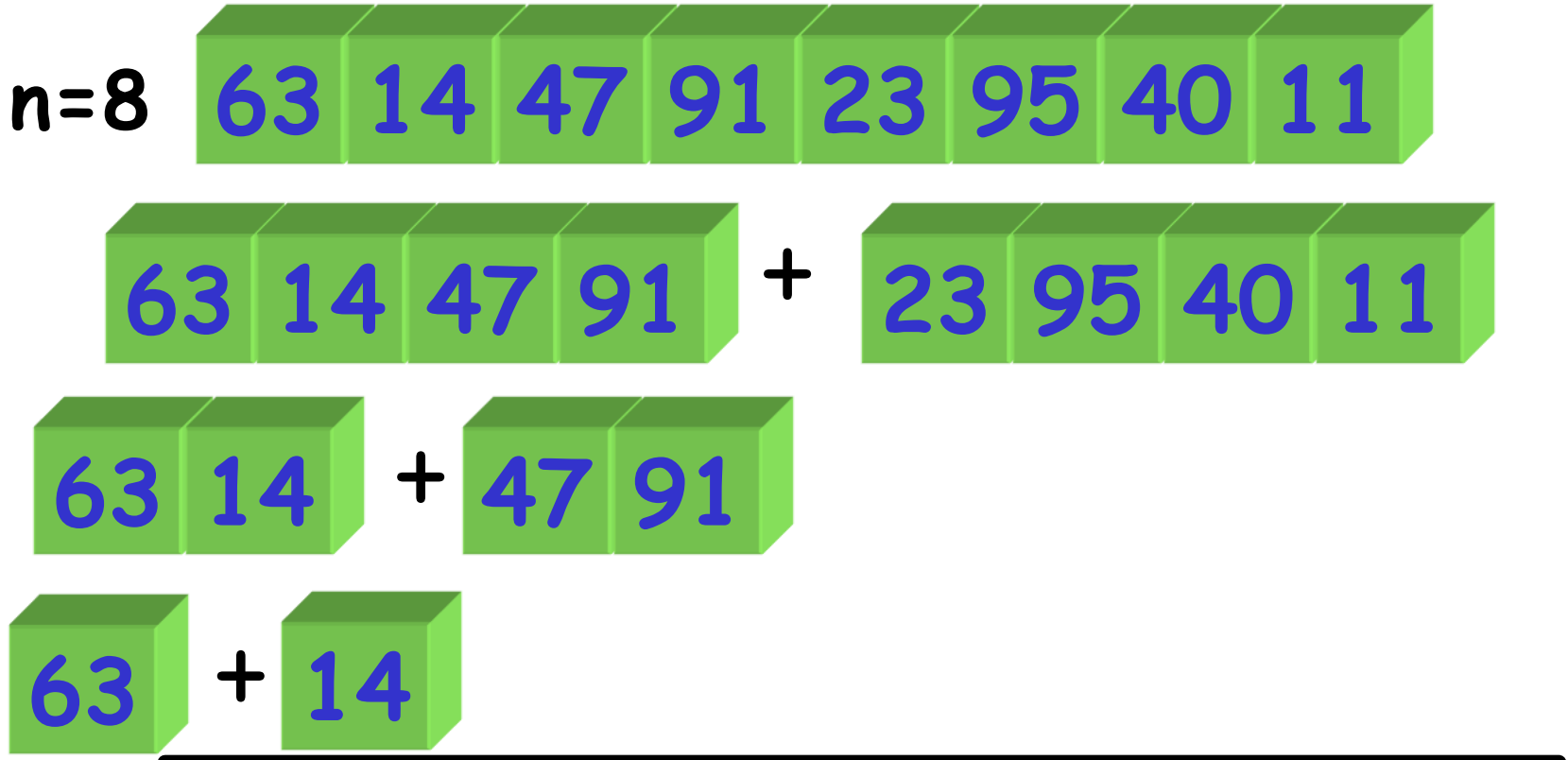
versione 2

$T(n) = n-1$ somme

indirizzo di inizio della porzione di destra

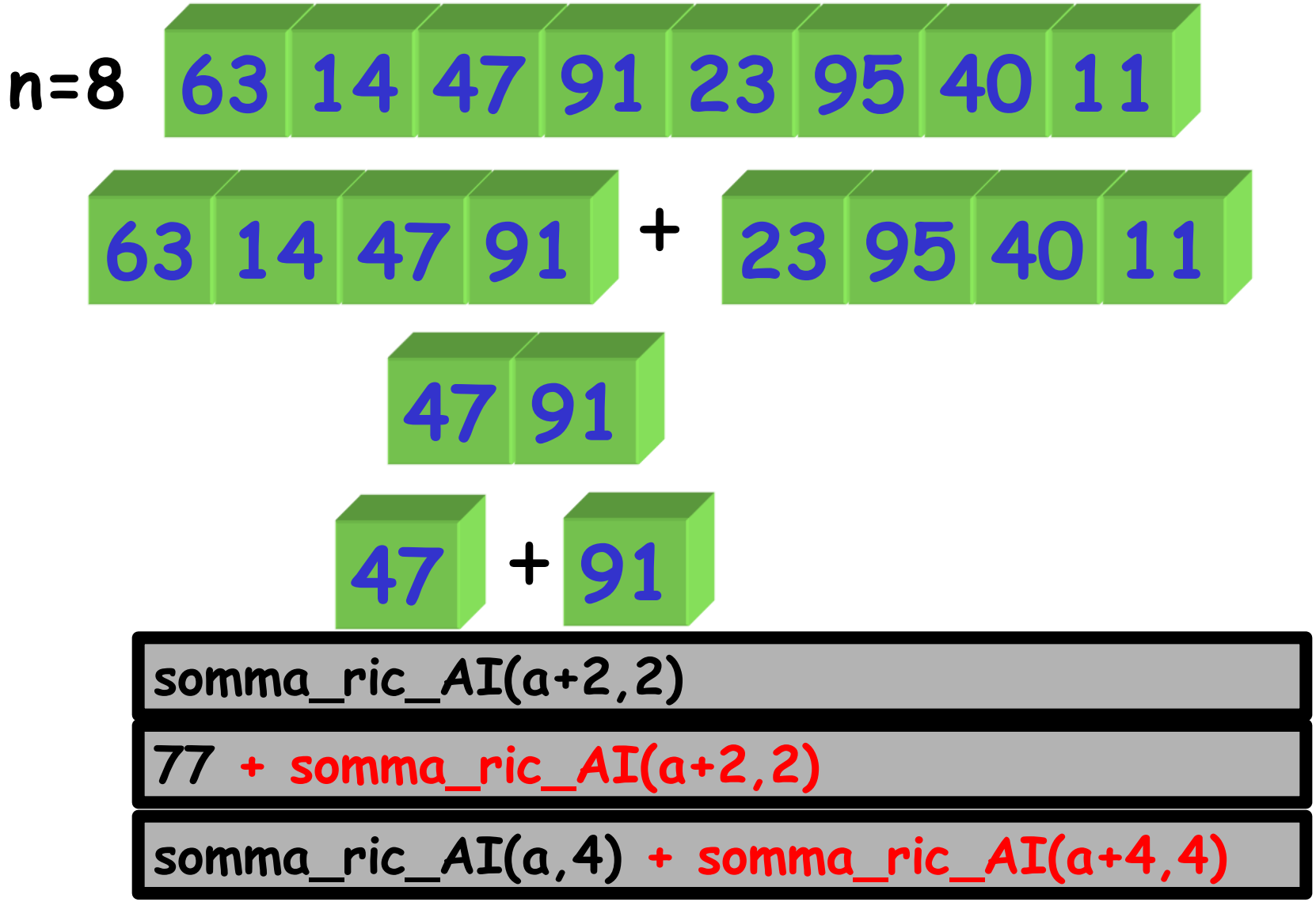
size della porzione di destra

tecnica di programmazione ricorsiva per l'algoritmo divide et impera di **somma** degli elementi di un array

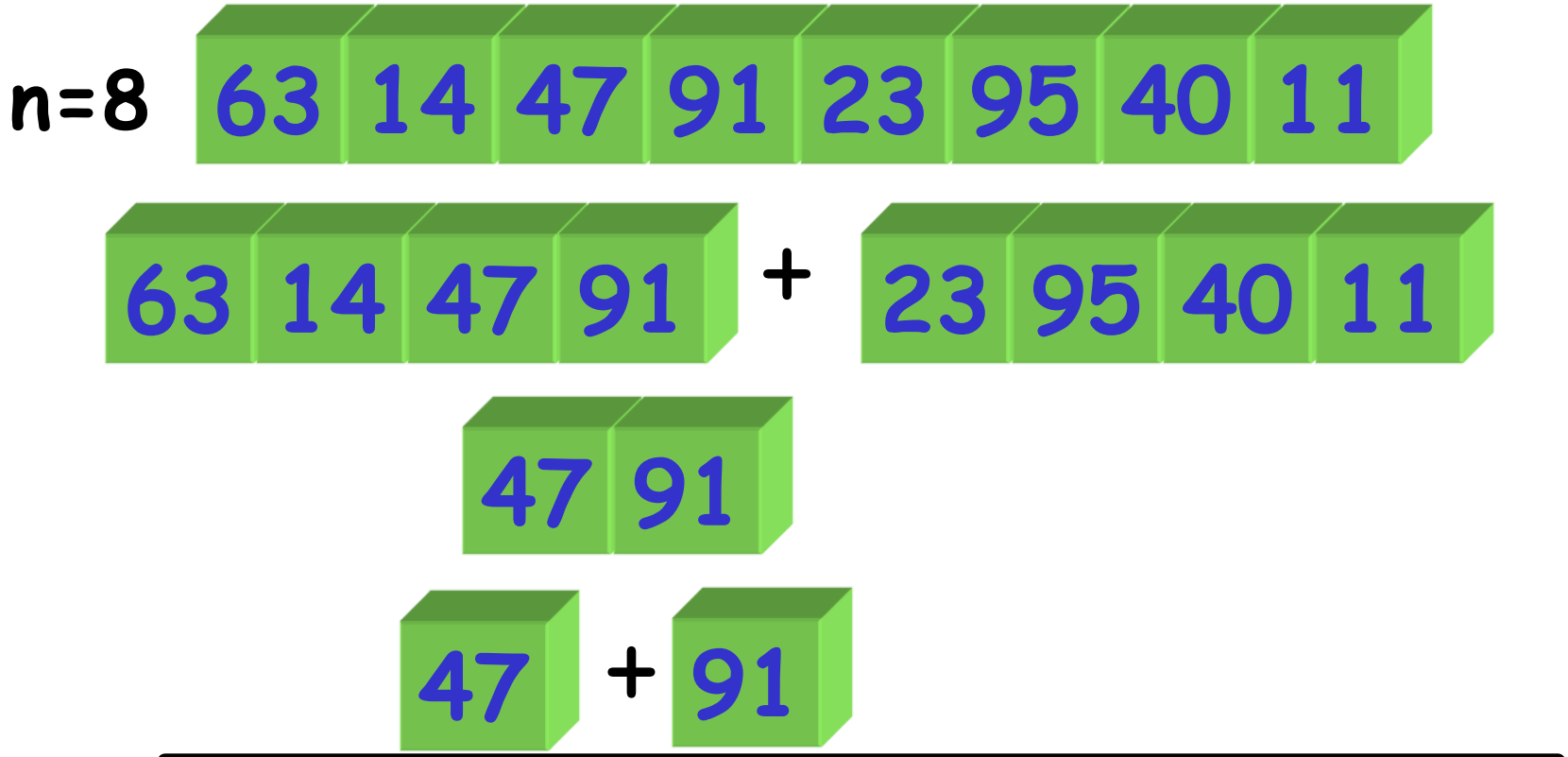



```
63 + 14
somma_ric_AI(a,2) + somma_ric_AI(a+2,2)
somma_ric_AI(a,4) + somma_ric_AI(a+4,4)
```

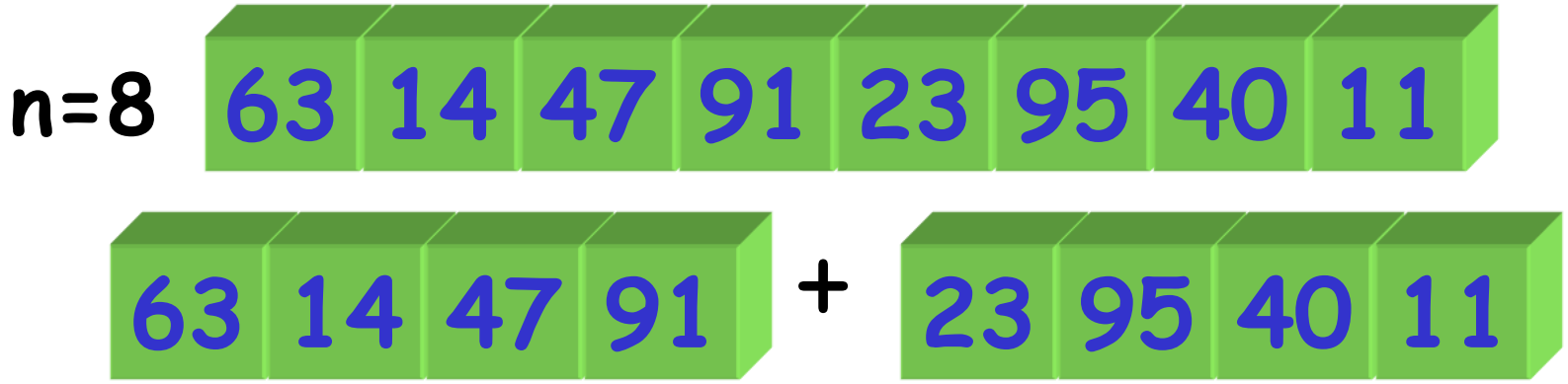
tecnica di programmazione ricorsiva per l'algoritmo divide et impera di **somma** degli elementi di un array



tecnica di programmazione ricorsiva per l'algoritmo divide et impera di **somma** degli elementi di un array

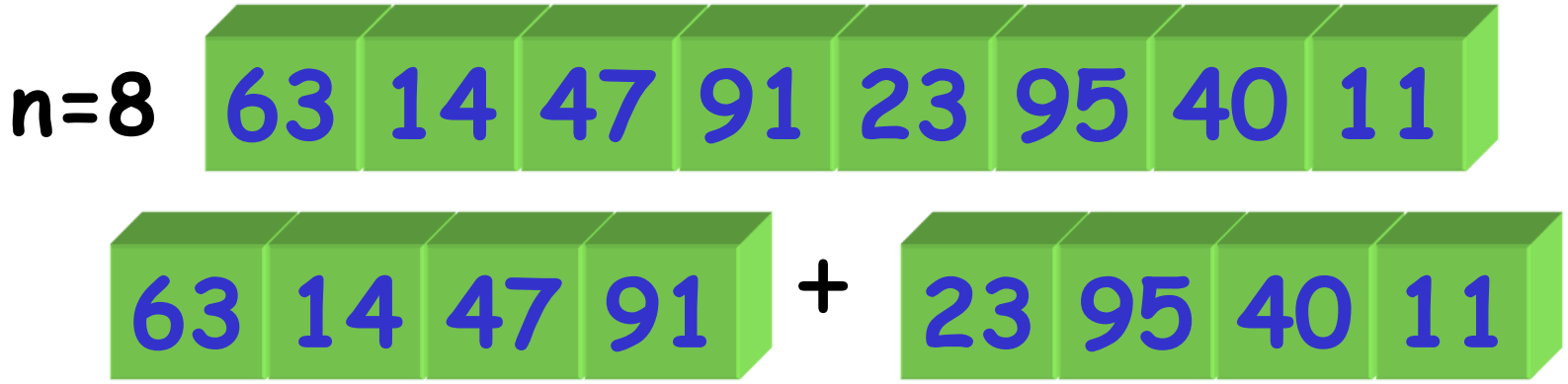


tecnica di programmazione ricorsiva per l'algoritmo divide et impera di **somma** degli elementi di un array



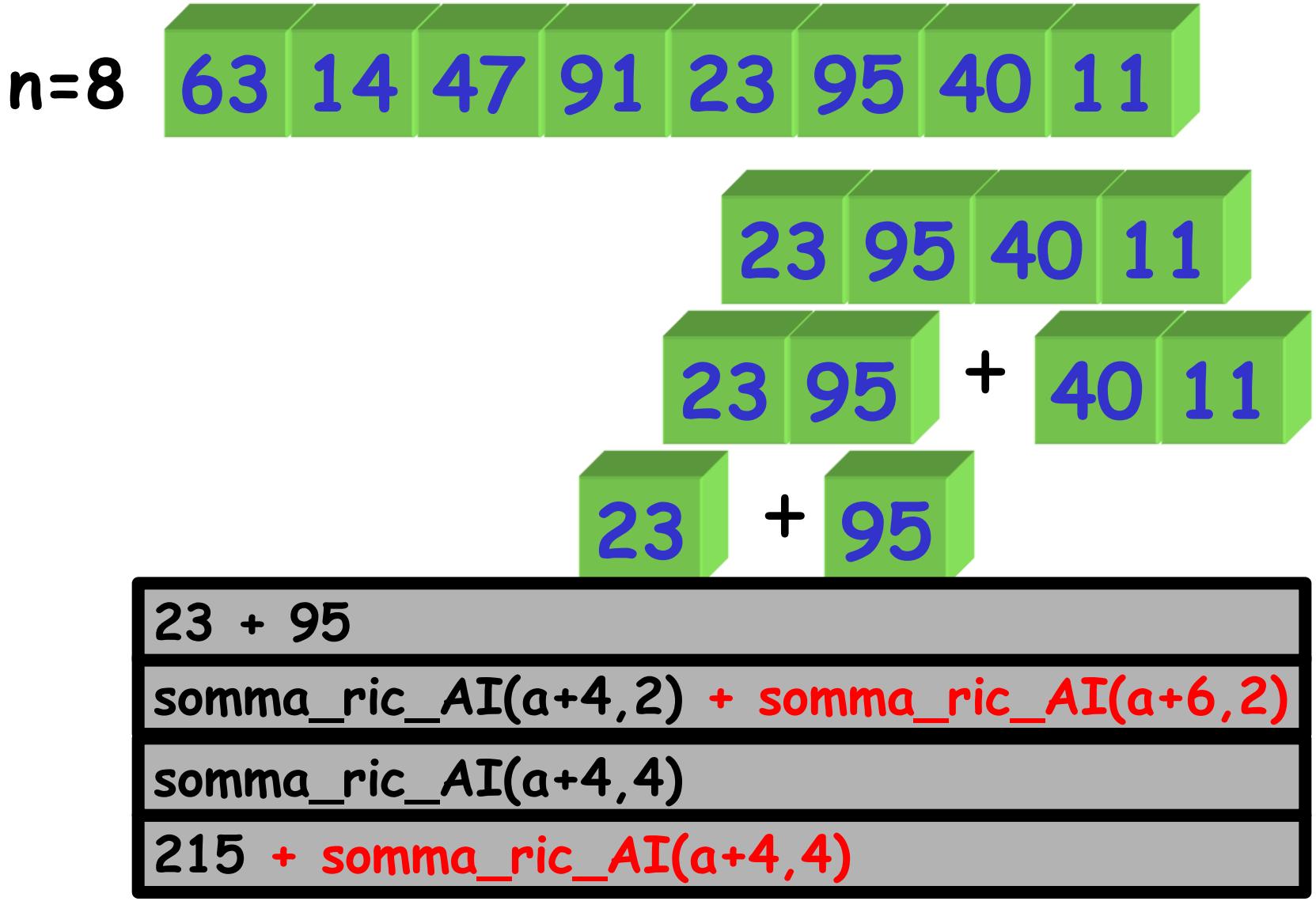

```
77 + 138
somma_ric_AI(a,4) + somma_ric_AI(a+4,4)
```

tecnica di programmazione ricorsiva per l'algoritmo divide et impera di **somma** degli elementi di un array

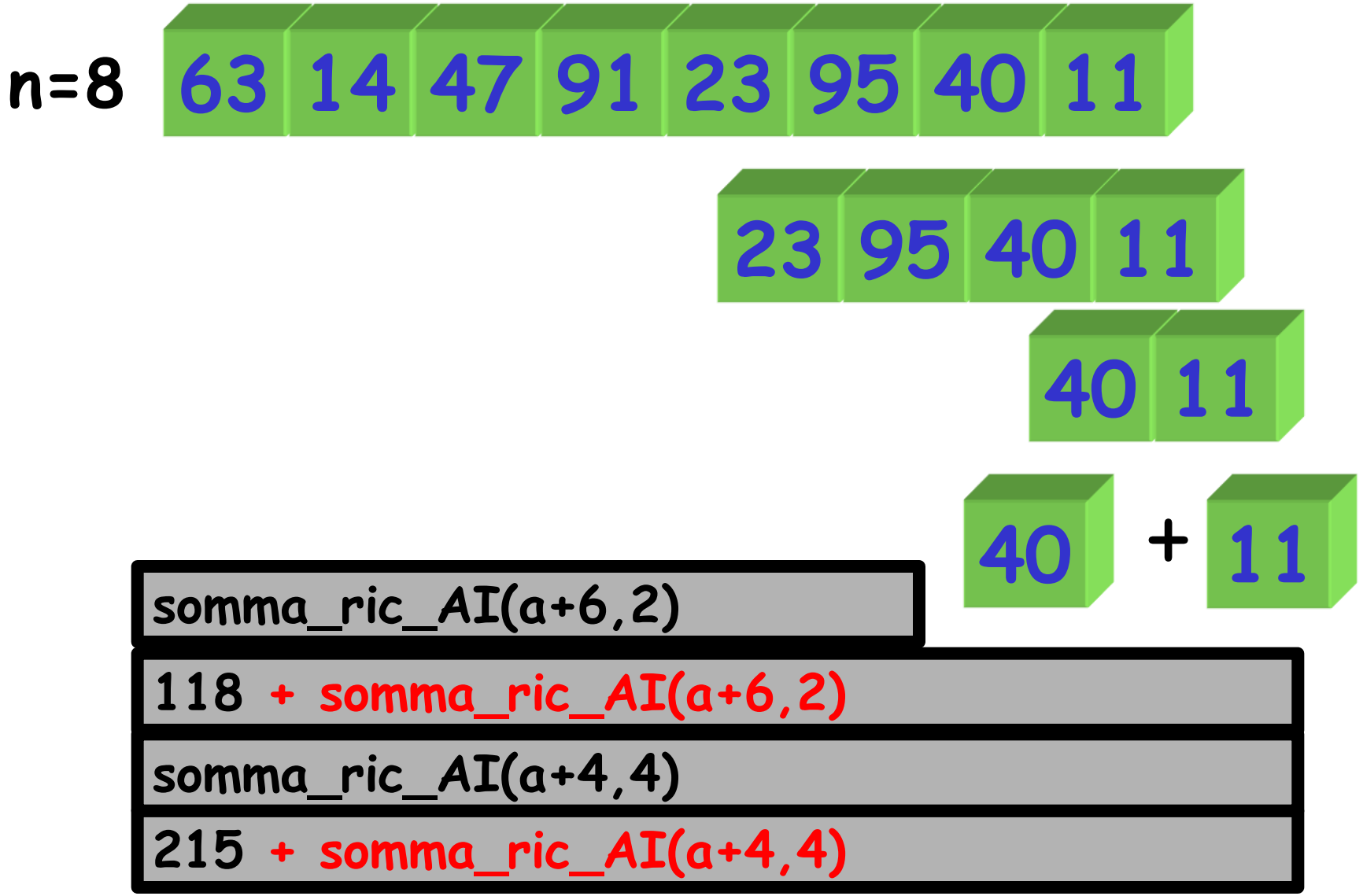



```
215 + somma_ric_AI(a+4,4)
```

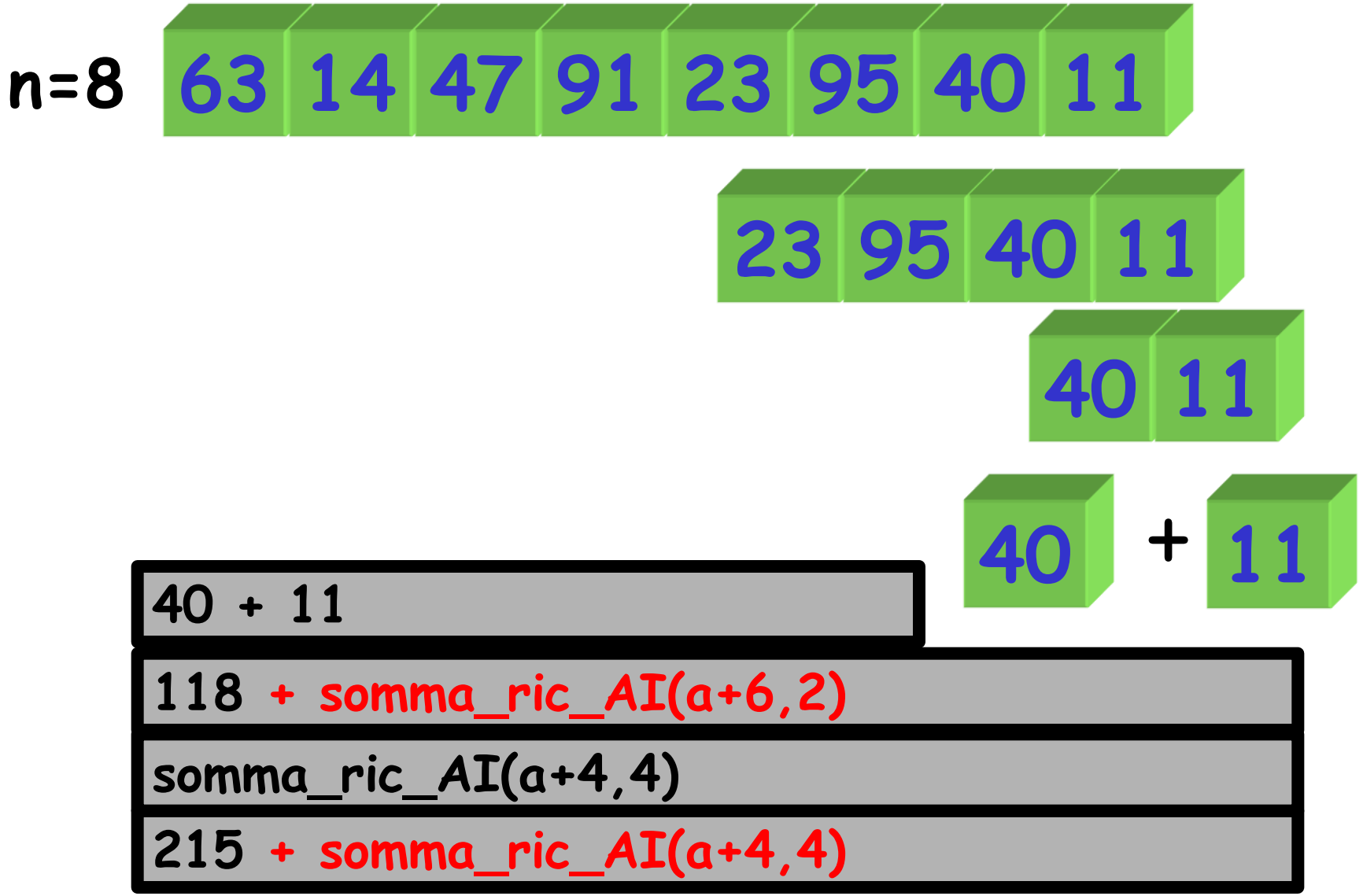
tecnica di programmazione ricorsiva per l'algoritmo divide et impera di **somma** degli elementi di un array



tecnica di programmazione ricorsiva per l'algoritmo divide et impera di **somma** degli elementi di un array

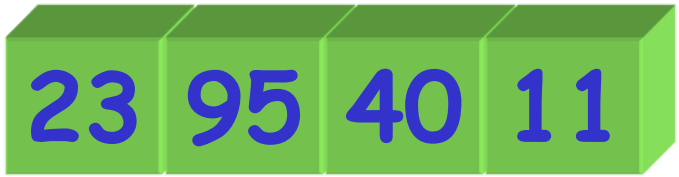


tecnica di programmazione ricorsiva per l'algoritmo divide et impera di **somma** degli elementi di un array



tecnica di programmazione ricorsiva per l'algoritmo divide et impera di **somma** degli elementi di un array

n=8




```
118 + 51
somma_ric_AI(a+4,4)
215 + somma_ric_AI(a+4,4)
```

tecnica di programmazione ricorsiva per l'algoritmo divide et impera di **somma** degli elementi di un array

n=8

169

215 + somma_ric_AI(a+4,4)

tecnica di programmazione ricorsiva per l'algoritmo divide et impera di **somma** degli elementi di un array

n=8

384

215 + 169

tecnica di programmazione ricorsiva per l'algoritmo incrementale di **massimo** degli elementi di un array

$$y_k = \max(y_{k-1}, a_k) \qquad y_1 = a_1$$

```
int massimo_a_ricAI(int a[], int n) {
 if (n == 1)
   /* soluzione del problema banale  */
       { return a[0]; }
    else
      /*    autoattivazione     */
     { return max(a[n-1],massimo_a_ricAI(a,n-1) ) ; }
}
```

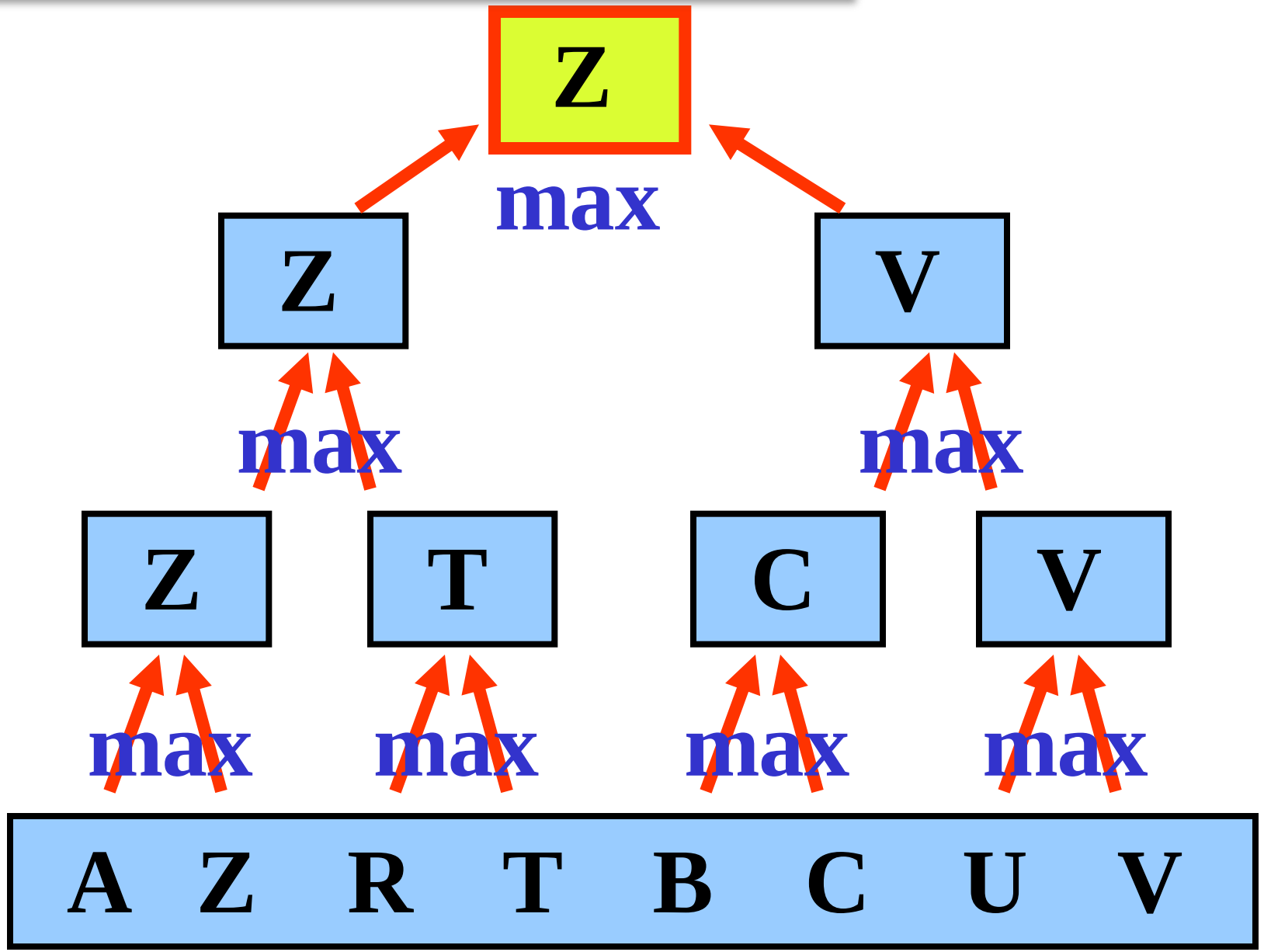
```
int max(int x, int y) {
if (x >= y)
        { return x; }
    else
        { return y; }
}
```

T(n) = n-1

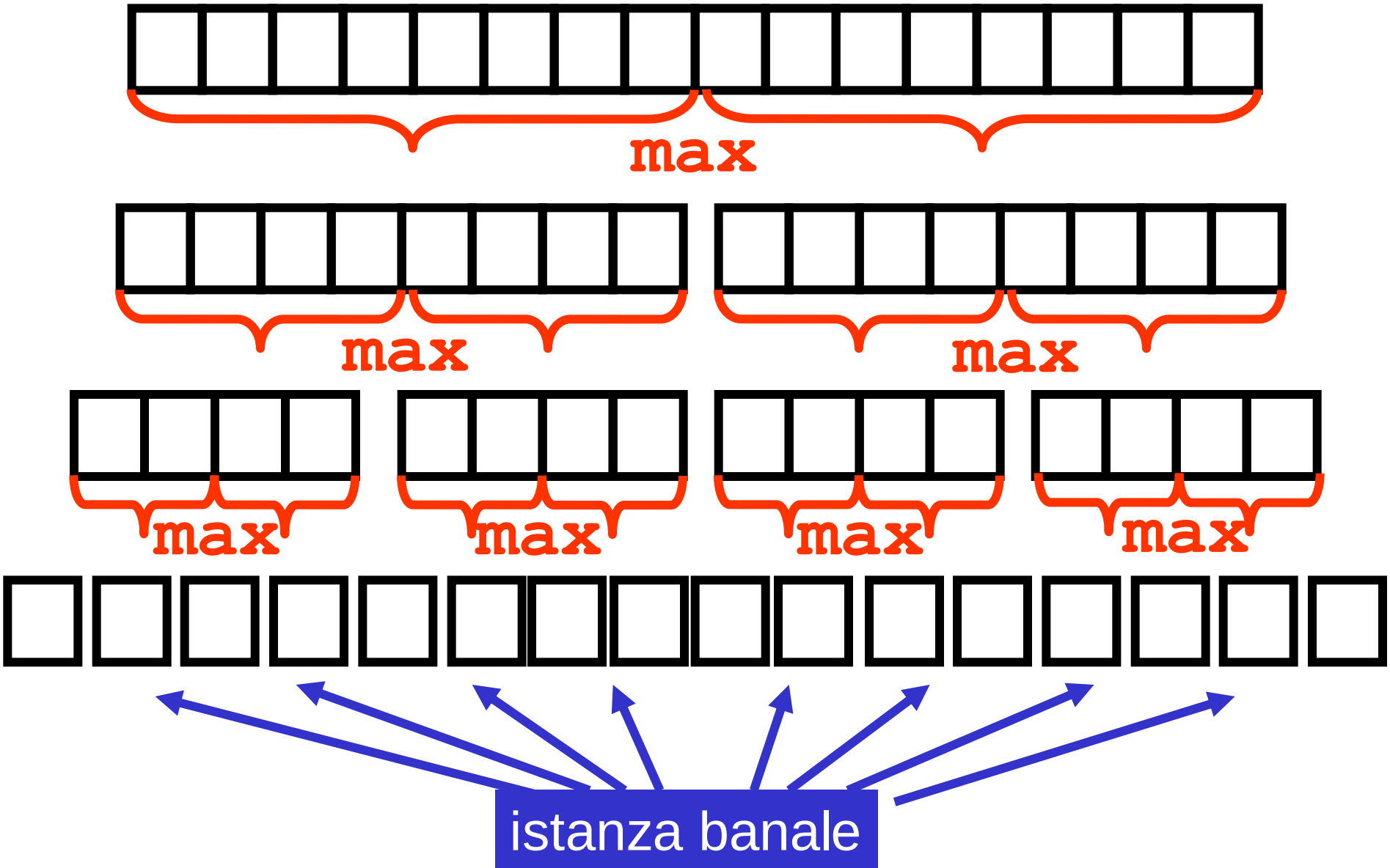
confronti

(tra elementi dell'array)

approccio **divide et impera** al problema del **massimo** degli elementi di un array



tecnica di programmazione ricorsiva per l'algoritmo divide et impera di **massimo** degli elementi di un array



tecnica di programmazione ricorsiva per l'algoritmo divide et impera di **massimo** degli elementi di un array

```
int massimo_a_ricDI(int a[], int primo, int ultimo) {
 int mediano;
 if (primo == ultimo)
     /* soluzione del problema banale   */
        { return a[primo]; }
   else
    /*  autoattivazioni   */
        {
         mediano = (primo+ultimo)/2 ;
         return max(massimo_a_ricDI(a,primo,mediano) , ...
                    massimo_a_ricDI(a,mediano+1,ultimo) ) ;
        }
}
```

versione 1

$T(n) = n-1$

confronti

(tra elementi dell'array)

tecnica di programmazione ricorsiva per l'algoritmo divide et impera di **massimo** degli elementi di un array

```
int massimo_a_ricDI(int a[], int n){
 int nmezzi;
 if (n == 1)
     /* soluzione del problema banale   */
        { return a[0]; }
   else
     /*  autoattivazioni   */
        {
         nmezzi = n/2 ;
         return max(massimo_a_ricDI(a, nmezzi) , ...
                    massimo_a_ricDI(a+nmezzi, n- nmezzi)) ;
        }
}
```

versione 2

$T(n) = n-1$ confronti (tra elementi dell'array)

indirizzo di inizio della porzione di destra

size della porzione di destra

problema della **somma** degli elementi di un array

	approccio incrementale	**approccio divide et impera**
tecnica di programmazione iterativa	somma_array(a,n)	somma_raddoppiamento(a,n)
tecnica di programmazione ricorsiva	somma_a_ricAI (a,primo,ultimo)	somma_a_ricDI (a,primo,ultimo)

problema del **massimo** degli elementi di un array

	approccio incrementale	**approccio divide et impera**
tecnica di programmazione iterativa	massimo_array(a,n)	max_raddoppiamento(a,n)
tecnica di programmazione ricorsiva	massimo_a_ricAI (a,primo,ultimo)	massimo_a_ricDI (a,primo,ultimo)

Massimo Comun Divisore
algoritmo di Euclide

il Massimo Comun Divisore (MCD) di due numeri interi è il più grande numero naturale che li divide con resto zero (cioè è un *divisore*)

siano a, b due numeri interi, il loro massimo comun divisore $\mathrm{MCD}(a,b)$ è calcolabile nei seguenti modi:

- si enumerano tutti i divisori di a
- si enumerano tutti i divisori di b
- $\mathrm{MCD}(a,b)$ è il più grande tra i divisori comuni

Massimo Comun Divisore
algoritmo di Euclide

il Massimo Comun Divisore (MCD) di due numeri interi è il più grande numero naturale che li divide con resto zero (cioè è un *divisore*)

siano a, b due numeri interi, il loro massimo comun divisore $\mathrm{MCD}(a,b)$ è calcolabile nei seguenti modi:

$T(n)$ esponenziale
n = numero di cifre di a,b

Esempio $a = 90$, $b = 84$

- divisori di 90: 1,2,3,5,**6**,9,10,15,18,30,45
- divisori di 84: 1,2,3,4,**6**,7,12,14,21,28,42
- $\mathrm{MCD}(90,84) = 6$

Massimo Comun Divisore
algoritmo di Euclide

il Massimo Comun Divisore (MCD) di due numeri interi è il più grande numero naturale che li divide con resto zero (cioè è un *divisore*)

siano a, b due numeri interi, il loro massimo comun divisore $\mathrm{MCD}(a,b)$ è calcolabile nei seguenti modi:

- si calcola la fattorizzazione in numeri primi di a
- si calcola la fattorizzazione in numeri primi di b
- $\mathrm{MCD}(a,b)$ è il prodotto dei fattori primi comuni, considerati con il minimo esponente

Massimo Comun Divisore
algoritmo di Euclide

il Massimo Comun Divisore (MCD) di due numeri interi è il più grande numero naturale che li divide con resto zero (cioè è un *divisore*)

siano a, b due numeri interi, il loro massimo comun divisore $\mathrm{MCD}(a,b)$ è calcolabile nei seguenti modi:

$T(n)$ esponenziale
n = numero di cifre di a,b

Esempio $a = 90$, $b = 84$

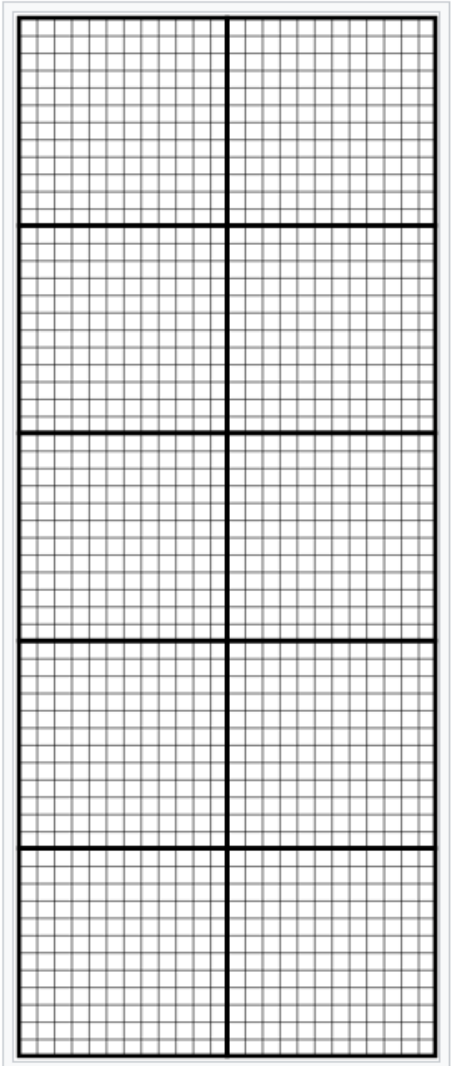
- fattori primi di 90: $2,3,3,5 \Rightarrow \mathbf{2},3^2,5$
- fattori primi di 84: $2,2,3,7 \Rightarrow 2^2,\mathbf{3},7$
- $\mathrm{MCD}(90,84) = \mathbf{2}*\mathbf{3} = 6$

Massimo Comun Divisore
algoritmo di Euclide

alcune proprietà

Un rettangolo axb può essere ricoperto con tessere quadrate di lunghezza c solo se c è un divisore comune di a e b

qual è la lunghezza della più grande tessera quadrata usabile per ricoprirlo **?**



Es: dato un rettangolo 24x60 la più grande tessera quadrata usabile per ricoprirlo è 12x12, dove MCD(60,24) = 12

Massimo Comun Divisore
algoritmo di Euclide

alcune proprietà

$$\mathrm{MCD}(a,b) \cdot \mathrm{mcm}(a,b) = |a \cdot b|$$

Es: $a = 90$, $b = 84$, $\mathrm{MCD}(90,84) = 6$, $\mathrm{mcm}(90,84) = 1260$

$a \cdot b = 7560$, $6 \cdot 1260 = 7560$

è il prodotto dei fattori primi comuni e non comuni, considerati una sola volta e con il massimo esponente

Massimo Comun Divisore
algoritmo di Euclide

alcune proprietà

$MCD(a,b) = MCD(a,a-b) = MCD(b,a-b)$

Es: a= 42, b=28, MCD(42,28) =14= MCD(42,14) =14
MCD(28,14)=14

$MCD(a,b) = MCD(b, a \bmod b)$, $a>b$

Es: a= 42, b=28, MCD(42,28) =14= MCD(28,14) =14

$MCD(b,0) = MCD(0, b) = |b|$

Massimo Comun Divisore
algoritmo di Euclide

$a > b$

versione iterativa

```
int mcd(int a, int b){
int r;
 r = a%b ; /* a mod b */
 while (r != 0) {
        a = b ;
        b = r ;
        r = a%b ;
 }
 return b ;
}
```

versione ricorsiva

```
int mcd(int a, int b){
int r;
 r = a%b ; /* a mod b */
 if (r == 0)
        return b;
    else
        return mcd(b,r);
}
```

$T(n) = O(n^2)$ al più

n = numero di cifre di a,b

tecnica di programmazione ricorsiva per l'algoritmo di **fibonacci**

$$y_k = y_{k-1} + y_{k-2}$$

$y_0 = 0$ $y_1 = 1$

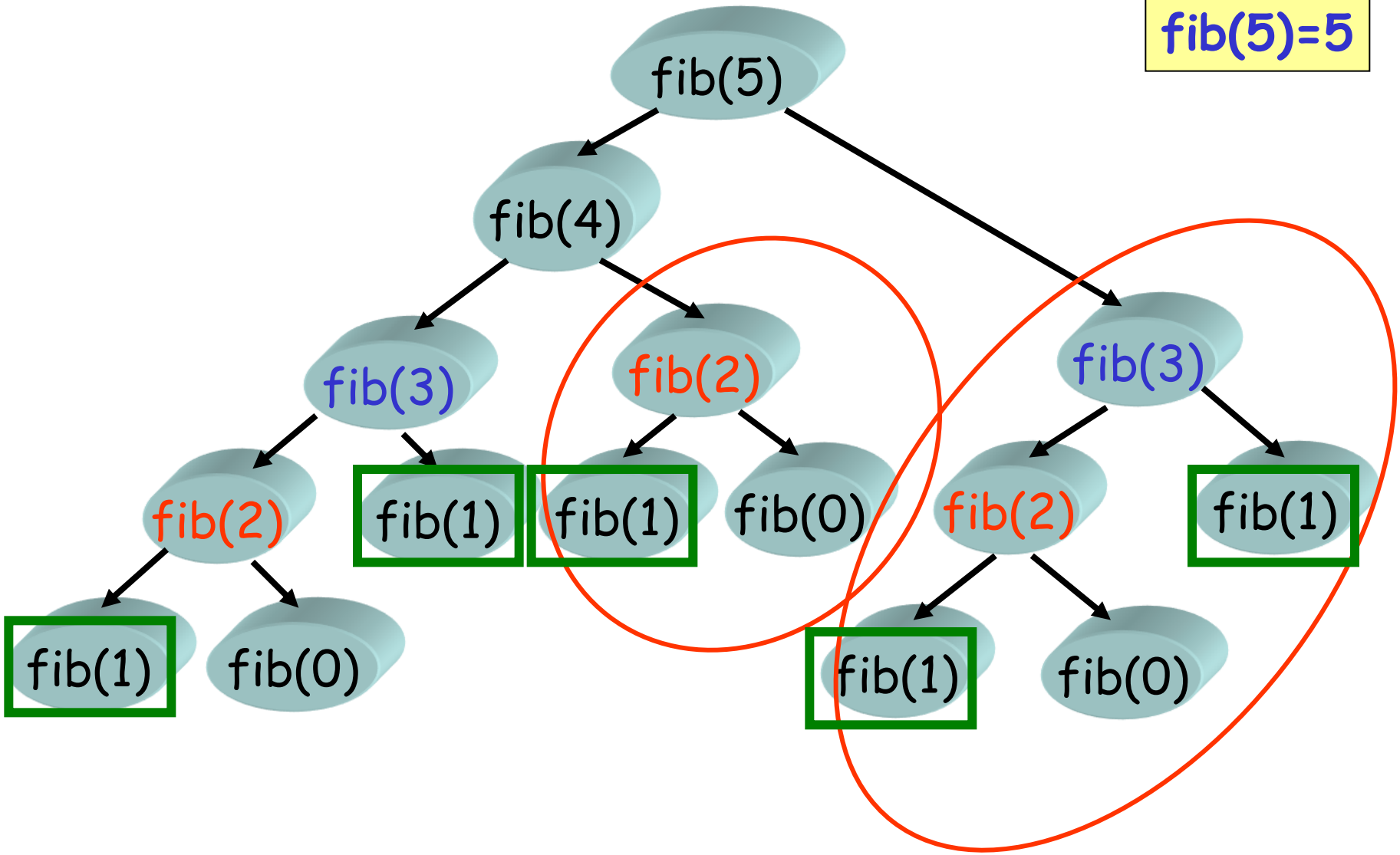
```
int fib_ric_el (int n) {
if (n == 1 || n == 0)
/* soluzione del problema banale   */
   { return n; }
  else
   /*   autoattivazioni     */
     { return fib_ric_el(n-1) + fib_ric_el(n-2) ; }
}
```

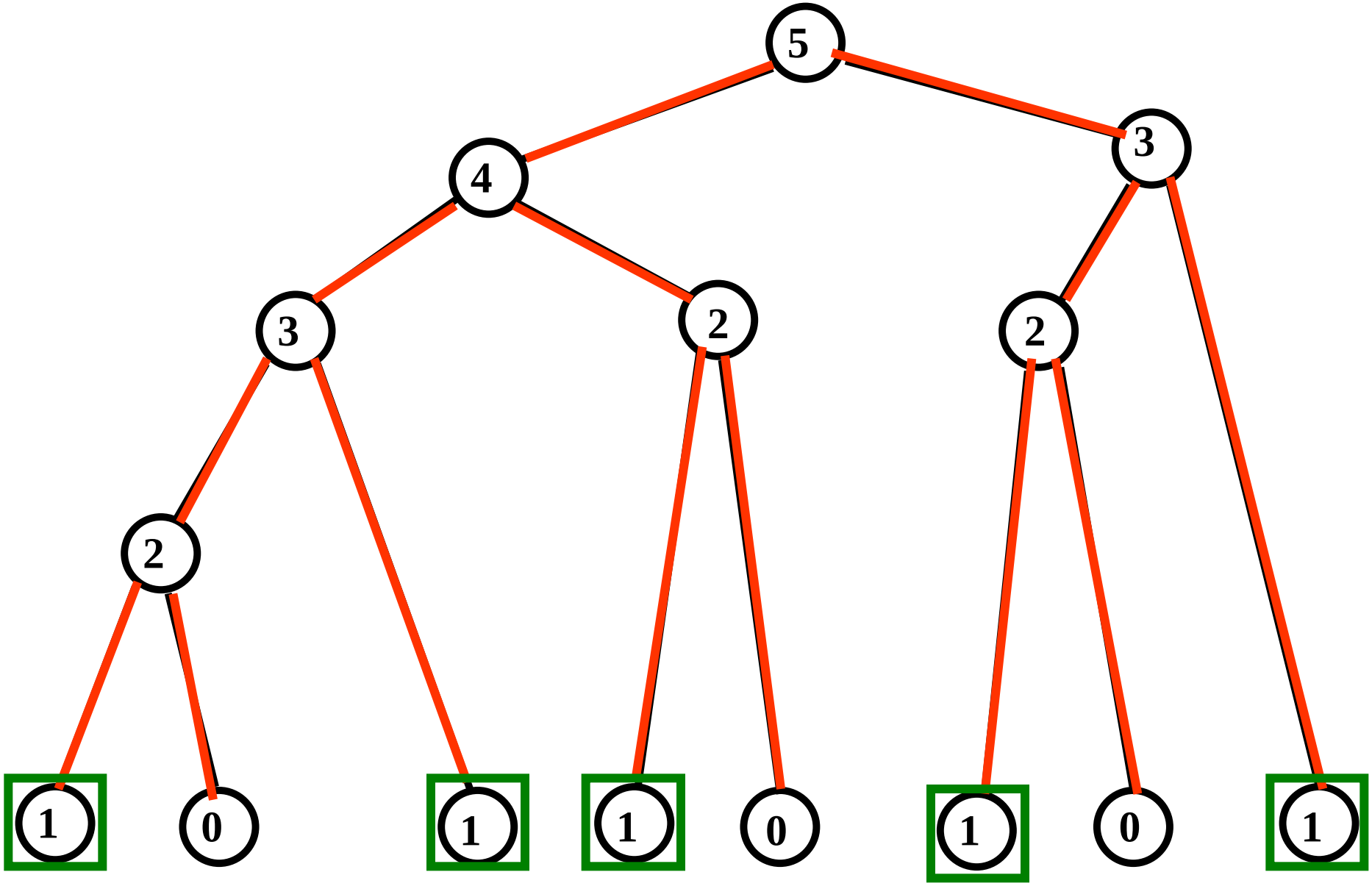
complessità di tempo ?

fib_ric_el: albero delle autoattivazioni per n=5

fib(1)=1
fib(0)=0

...
fib(5)=5





fib_ric_el: schema delle autoattivazioni

il valore dell'n-simo numero di Fibonacci f_n viene calcolato dalla function **fib_ric_el(n)** **sommando un numero opportuno** di volte la costante **1**

quante volte?

f_n volte 1

$T(n) = f_n - 1$ somme

$T(n) = O(\phi^n) = O(1.618^n)$ somme

sezione aurea

$$f_n = \frac{\phi^n - (1-\phi)^n}{\sqrt{5}}$$

$$f_n \approx \phi^n / 2.2$$

complessità esponenziale

l'algoritmo ricorsivo effettua più chiamate alla function **fib_ric_el** con il medesimo valore del parametro di input

la sequenza computazionale denotata da **fib_ric_el** **non è uguale** a quella dell'algoritmo **iterativo** **fibonacci**

fib_ric_el

$T(n) = O(1.6^n)$

somme

fibonacci

$T(n) = n-1$

somme

la sequenza computazionale denotata da
fib_ric_el **non è uguale**
a quella dell'algoritmo **iterativo** **fibonacci**

fib_ric_el

$T(n) = O(1.6^n)$
somme

complessità esponenziale

è possibile sviluppare un algoritmo ricorsivo per fibonacci di complessità lineare ?