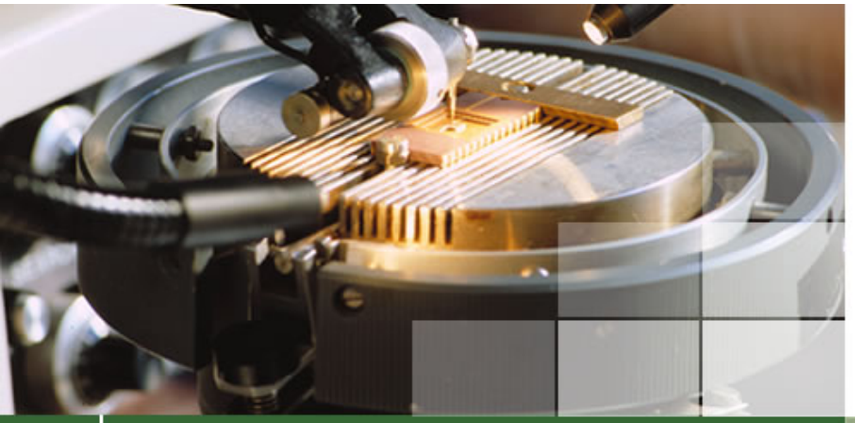


POLITECNICO DI MILANO

Dipartimento di
Elettronica e Informazione

MATLAB: funzioni e ricorsione

Matteo Ferroni

matteo.ferroni@polimi.it

14/12/2018

Esercizio: Funzioni e stringhe

Scrivere una funzione che riceve in input una stringa e restituisce **true** se è *palindroma*, **false** altrimenti.

Scrivere anche un esempio di chiamata della funzione.

Esercizio: Funzioni e stringhe

palindroma.m

```
1
2
3 -
4
5
6 -
7 -
8 -
9 -
10 -
```

Esercizio: Funzioni e stringhe

palindroma.m

```
1   function ris=palindroma(stringa)
2       % Ricavo la stringa invertita
3   -
4
5
6   -
7   -
8   -
9   -
10  -
```

Esercizio: Funzioni e stringhe

palindroma.m

```
1   function ris=palindroma(stringa)
2       % Ricavo la stringa invertita
3 -     stringa_r = stringa(end:-1:1);
4
5       % Se le stringhe sono uguali, la stringa palindroma
6 -
7 -
8 -
9 -
10 -
```

Esercizio: Funzioni e stringhe

palindroma.m

```
1   function ris=palindroma(stringa)
2     % Ricavo la stringa invertita
3 -   stringa_r = stringa(end:-1:1);
4
5     % Se le stringhe sono uguali, la stringa palindroma
6 -   if all(stringa == stringa_r)
7 -     ris=true;
8 -   else
9 -     ris=false;
10 -  end
```

Scrivere anche un esempio di chiamata della funzione.

Esercizio: Funzioni e stringhe

palindroma.m

```
1   function ris=palindroma(stringa)
2     % Ricavo la stringa invertita
3 -   stringa_r = stringa(end:-1:1);
4
5     % Se le stringhe sono uguali, la stringa palindroma
6 -   if all(stringa == stringa_r)
7 -     ris=true;
8 -   else
9 -     ris=false;
10 -  end
```

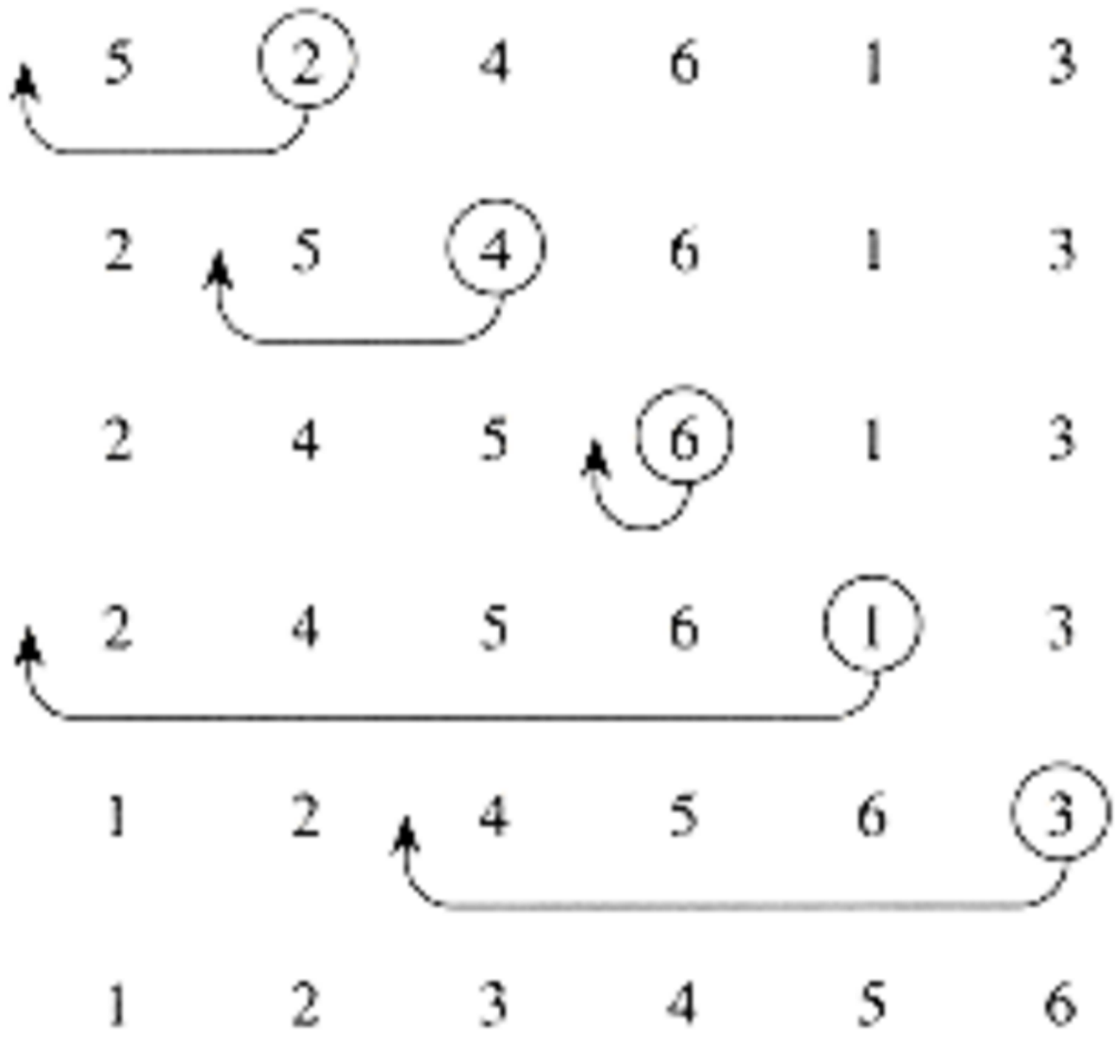
Scrivere anche un esempio di chiamata della funzione.

```
palindroma('ingegni')

palindroma('itopinonavevanonipoti')
```

Esercizio: Maschere di bit (ordinamento v2)

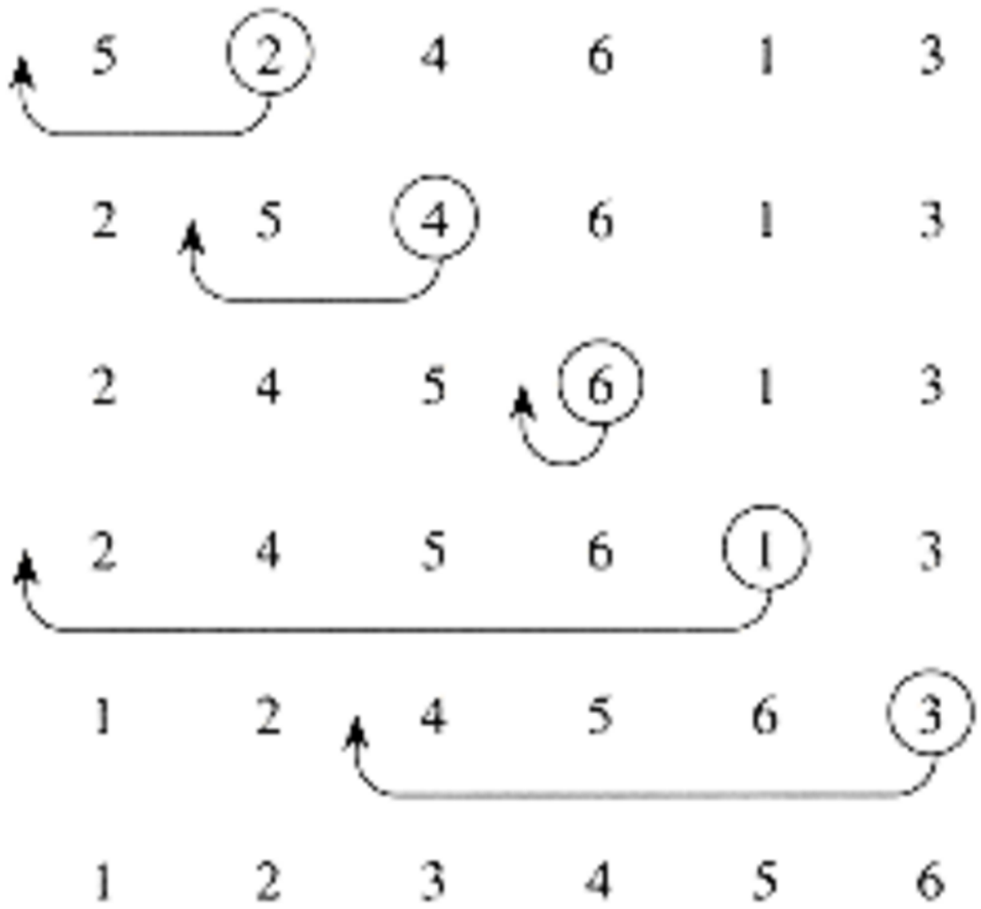
- Si ordini un array di n elementi facendo uso delle istruzioni messe a disposizione da matlab.

Algoritmo di ordinamento: “Insertion sort”
(https://it.wikipedia.org/wiki/Insertion_sort)



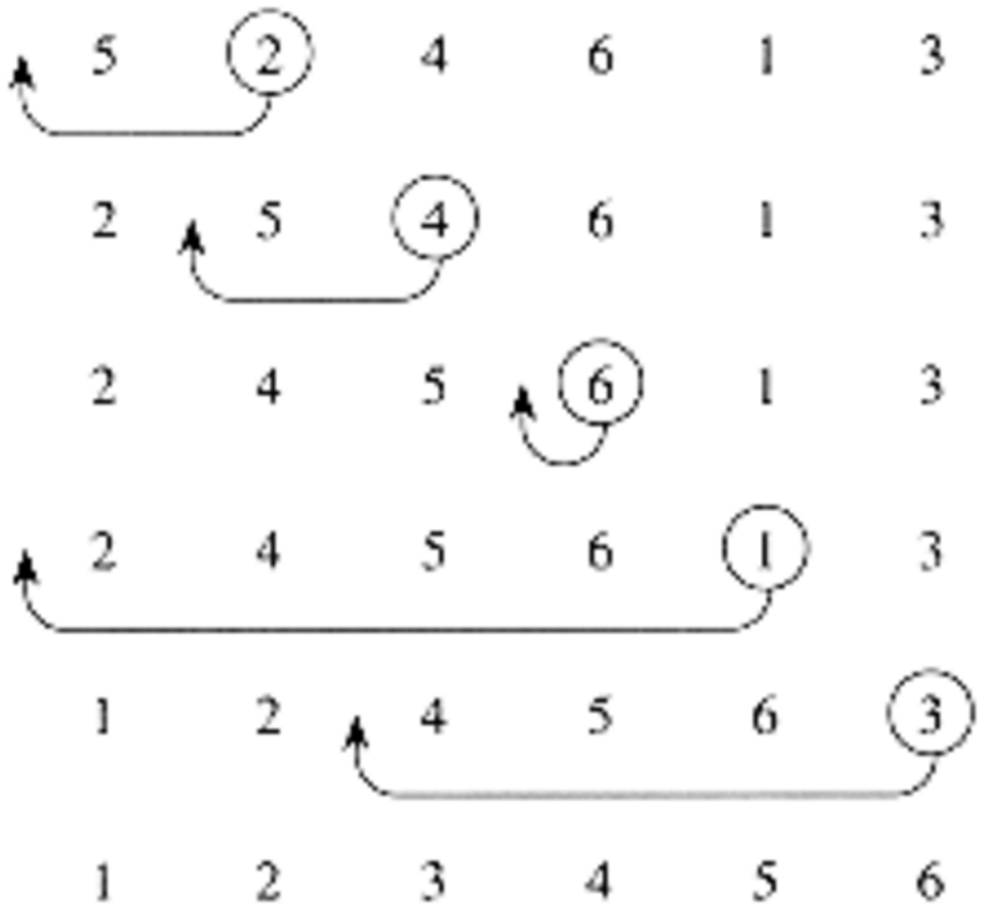
Esercizio: Maschere di bit (ordinamento v2)

```
a = [5 2 4 6 1 3]
```



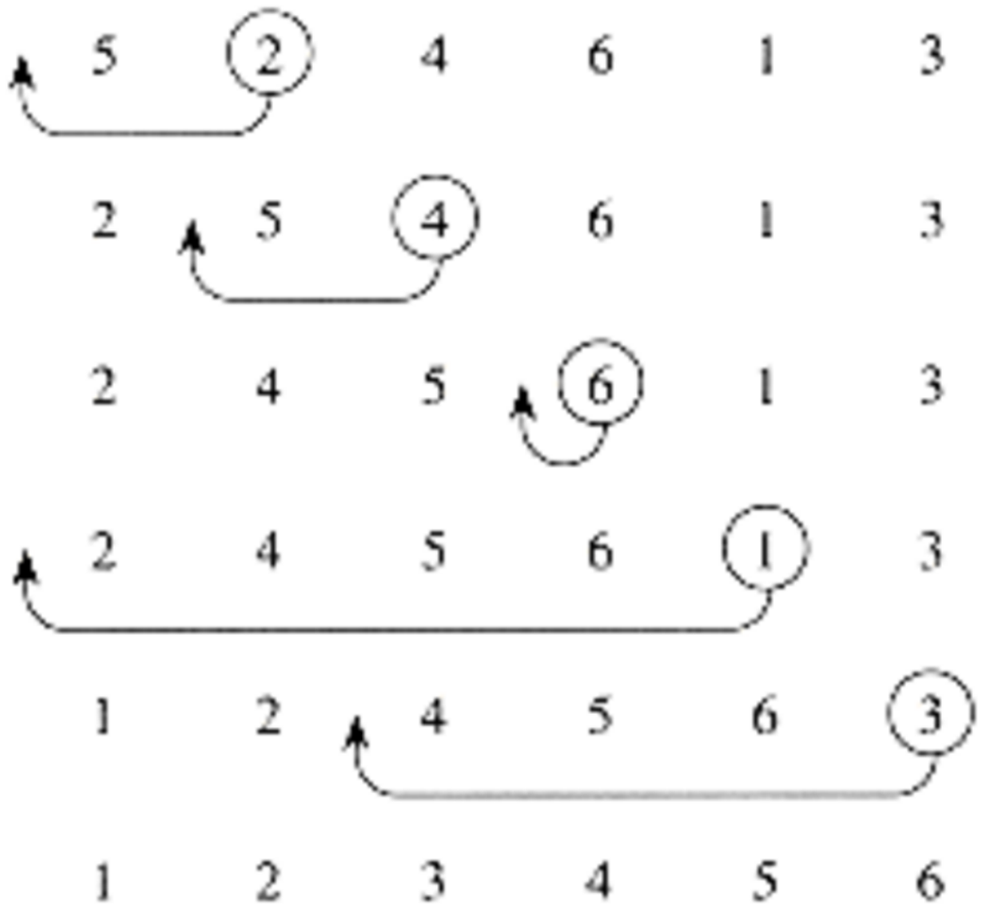
Esercizio: Maschere di bit (ordinamento v2)

```
a = [5 2 4 6 1 3]

a_ord = a(1);
```



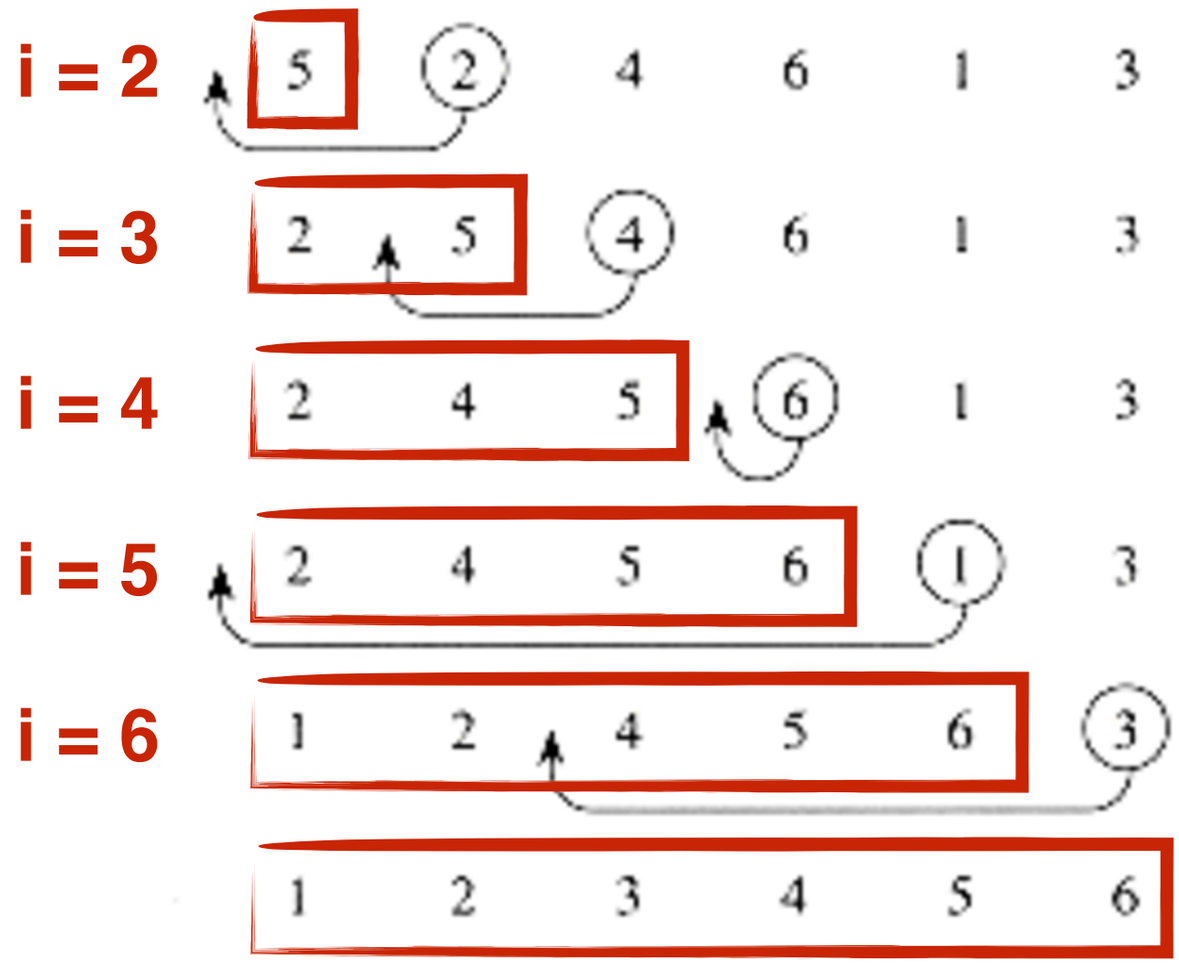
Esercizio: Maschere di bit (ordinamento v2)

```
a = [5 2 4 6 1 3]

a_ord = a(1);
for i=2:length(a)
```



Esercizio: Maschere di bit (ordinamento v2)

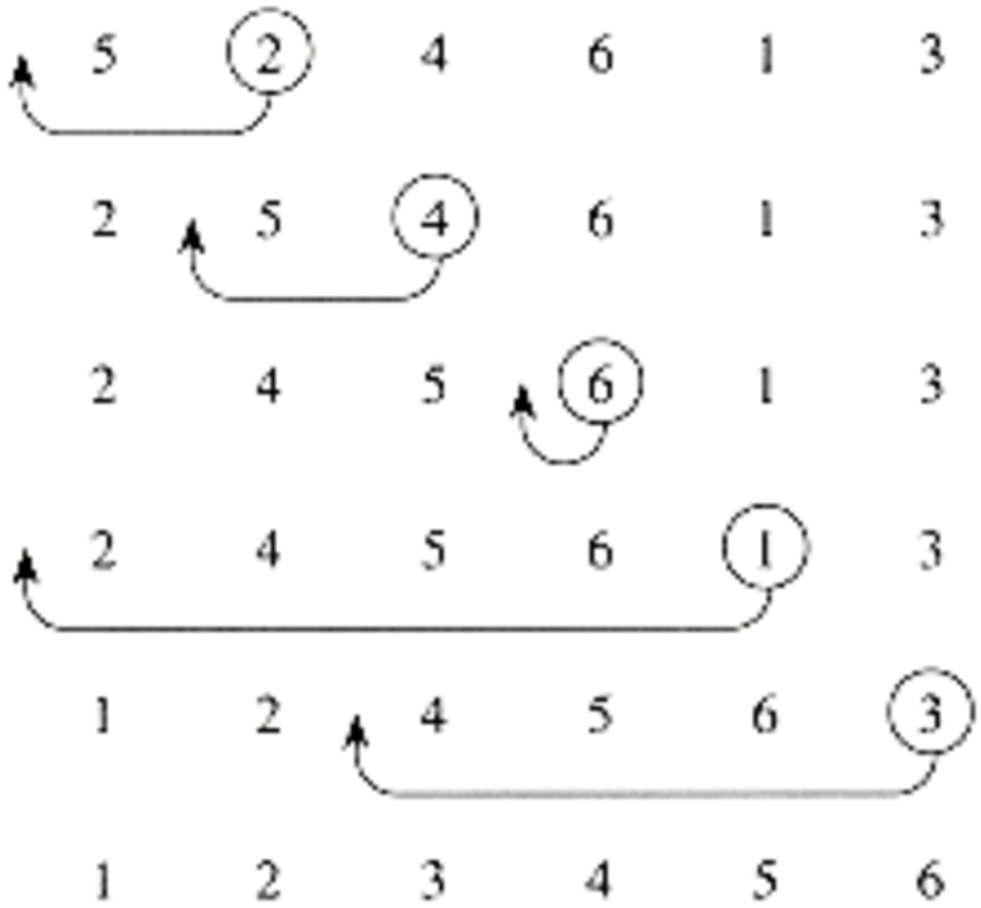
```
a = [5 2 4 6 1 3]

a_ord = a(1);
for i=2:length(a)
    a_ord = inserisci(a_ord, a(i));
end

a_ord
```



Esercizio: Maschere di bit (ordinamento v2)

```
a = [5 2 4 6 1 3]

a_ord = a(1);
for i=2:length(a)
    a_ord = inserisci(a_ord,a(i));
end

a_ord
```

```
function v_ord = inserisci(v,e)
    v_ord = [v(v<=e) e v(v>e)];
```



Ripasso: **maschera**, selezione, concatenazione

```
function v_ord = inserisci(v,e)
    v_ord = [v(v<=e) e v(v>e)];
```

maschera

Vettore di **bit**:
[0 0 **1** 0 … 0 **1 1 1** 0 0]

1 se: **v(i) <= e**
0 altrimenti

Vettore di **bit**:
[**1 1** 0 **1** … **1** 0 0 0 **1 1**]

1 se: **v(i) > e**
0 altrimenti

Ripasso: maschera, **selezione**, concatenazione

```
function v_ord = inserisci(v,e)
    v_ord = [v(v<=e) e v(v>e)];
```

selezione

v([0 0 **1** 0 … 0 **1 1 1** 0 0]**)** **v(**[**1 1** 0 **1** … **1** 0 0 0 **1 1**]**)**

Crea un **vettore**
che contiene gli **elementi** di **v**
solo nelle **posizioni degli 1**

Ripasso: maschera, selezione, **concatenazione**

```
function v_ord = inserisci(v,e)
    v_ord = [v(v<=e) e v(v>e)];
```

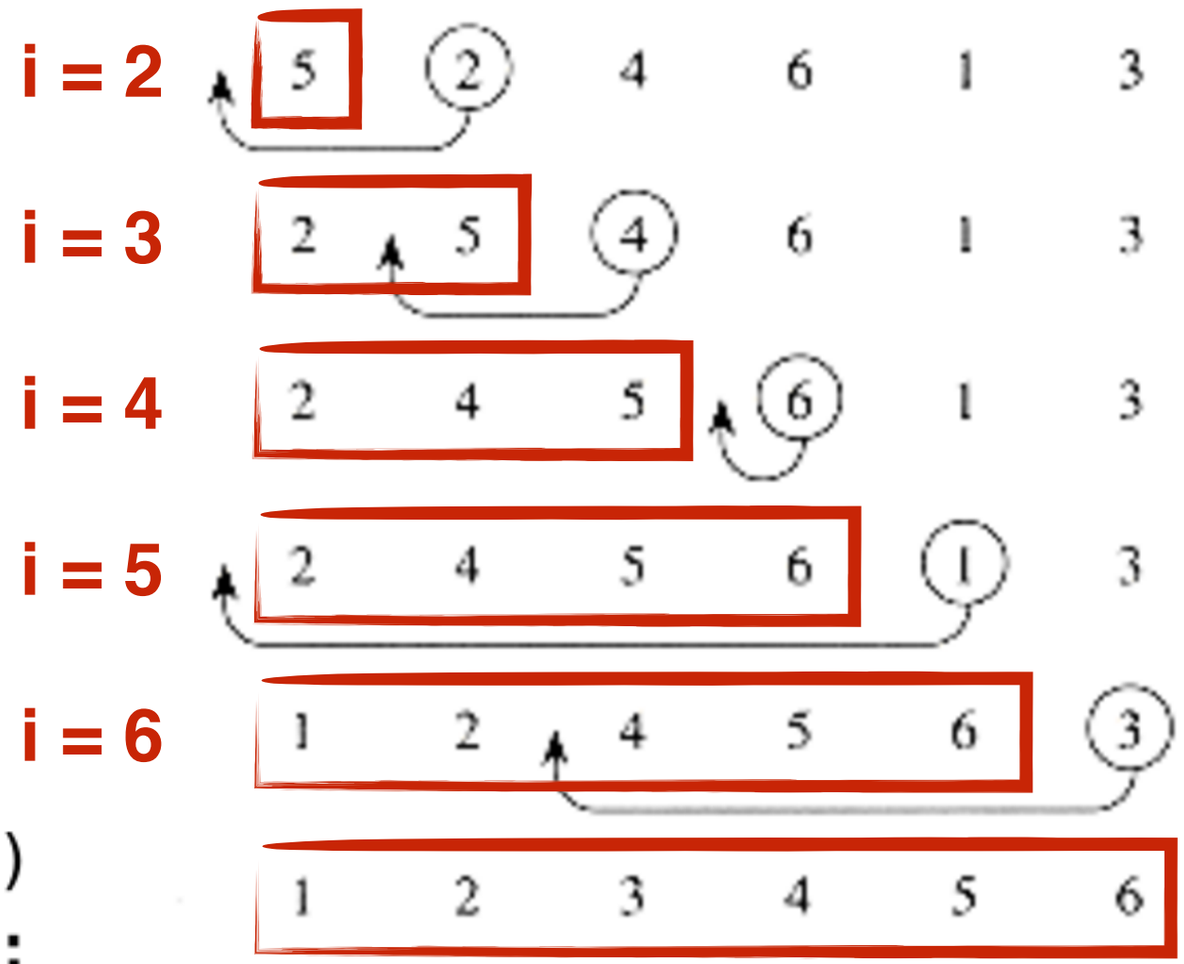
concatenazione

[**array 1** **array 2** **array 3**]

Crea un **vettore**
che contiene gli **elementi** dei **vettori**
nell'ordine specificato

Esercizio: Maschere di bit (ordinamento v2)

```
a = [5 2 4 6 1 3]

a_ord = a(1);
for i=2:length(a)
    a_ord = inserisci(a_ord,a(i));
end

a_ord
```




```
function v_ord = inserisci(v,e)
    v_ord = [v(v<=e) e v(v>e)];
```

Esercizio: Funzioni e divisori

Scrivere una funzione che riceva in ingresso un numero **N** e restituisca un array contenente **tutti i suoi divisori** (ad eccezione di 1 ed N stesso).

Nel caso N non abbia divisori, la funzione deve restituire l'array vuoto.

Esercizio: Funzioni e divisori

```
N = input ('Inserire un numero: ');
```

Esercizio: Funzioni e divisori

```
N = input ('Inserire un numero: ');
```

```
divisori(N)
```

Esercizio: Funzioni e divisori

```
N = input ('Inserire un numero: ');
```

```
mat2str(divisori(N))
```

Esercizio: Funzioni e divisori

```
N = input ('Inserire un numero: ');

disp (['I divisori di ' num2str(N) ' sono ' mat2str(divisori(N))]);
```

Esercizio: Funzioni e divisori

```
N = input ('Inserire un numero: ');

disp (['I divisori di ' num2str(N) ' sono ' mat2str(divisori(N))]);
```

```
  function v = divisori(N)

end
```

Esercizio: Funzioni e divisori

```
N = input ('Inserire un numero: ');

disp (['I divisori di ' num2str(N) ' sono ' mat2str(divisori(N))]);
```

```
  function v = divisori(N)
  %Possibili divisori
  candidati = 2:N-1;
```

```
end
```

Esercizio: Funzioni e divisori

```
N = input ('Inserire un numero: ');

disp (['I divisori di ' num2str(N) ' sono ' mat2str(divisori(N))]);
```

```
  function v = divisori(N)
  %Possibili divisori
  candidati = 2:N-1;

  %Calcoliamo i divisori
  idx = mod(N,candidati)==0;



  end
```

Esercizio: Funzioni e divisori

```
N = input ('Inserire un numero: ');

disp (['I divisori di ' num2str(N) ' sono ' mat2str(divisori(N))]);
```

```
  function v = divisori(N)
  %Possibili divisori
  candidati = 2:N-1;
```

```
  %Calcoliamo i divisori
  idx = mod(N,candidati)==0;
```

```
      v = candidati(idx);
```

```
  end
```

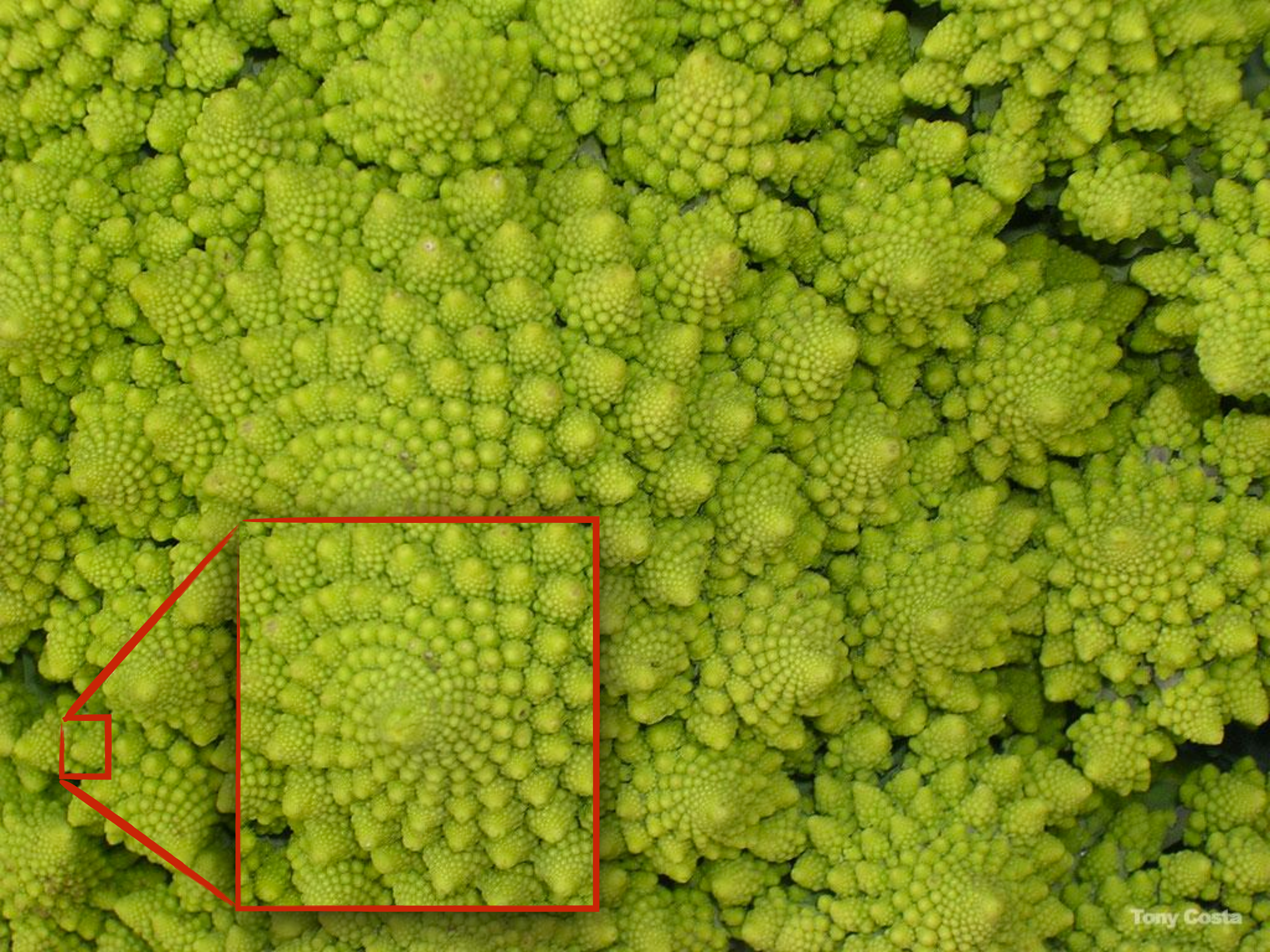
Esercizio: Funzioni e divisori

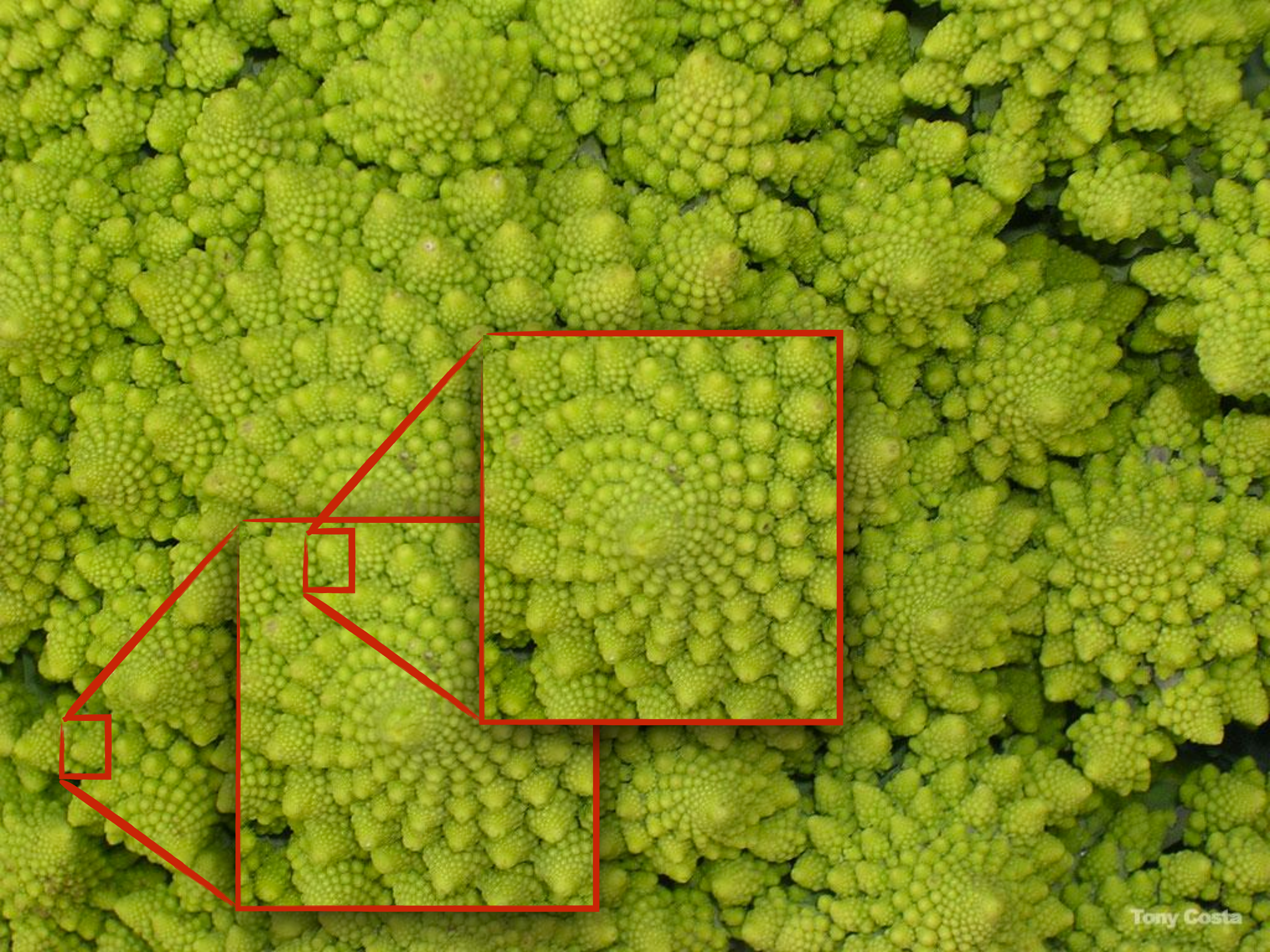
```
N = input ('Inserire un numero: ');

disp (['I divisori di ' num2str(N) ' sono ' mat2str(divisori(N))]);
```

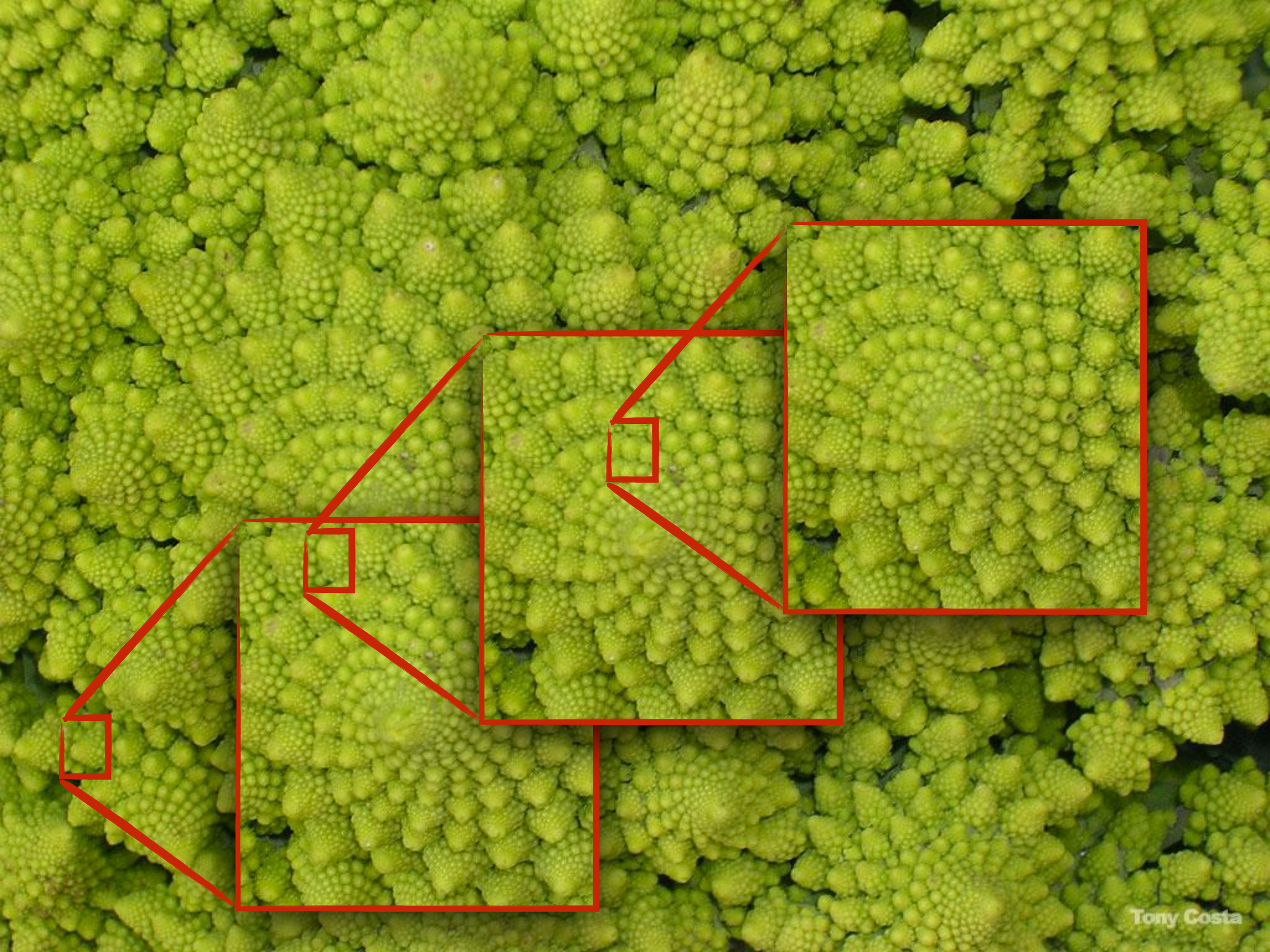
```
  function v = divisori(N)
  %Possibili divisori
  candidati = 2:N-1;

  %Calcoliamo i divisori
  idx = mod(N,candidati)==0;

  % se ci sono divisori elimino i candidati superflui
  if any(idx)
      v = candidati(idx);
  else
      v=[];
  end
```

…hai detto “ricorsione”?


Tony Costa


Tony Costa


Tony Costa

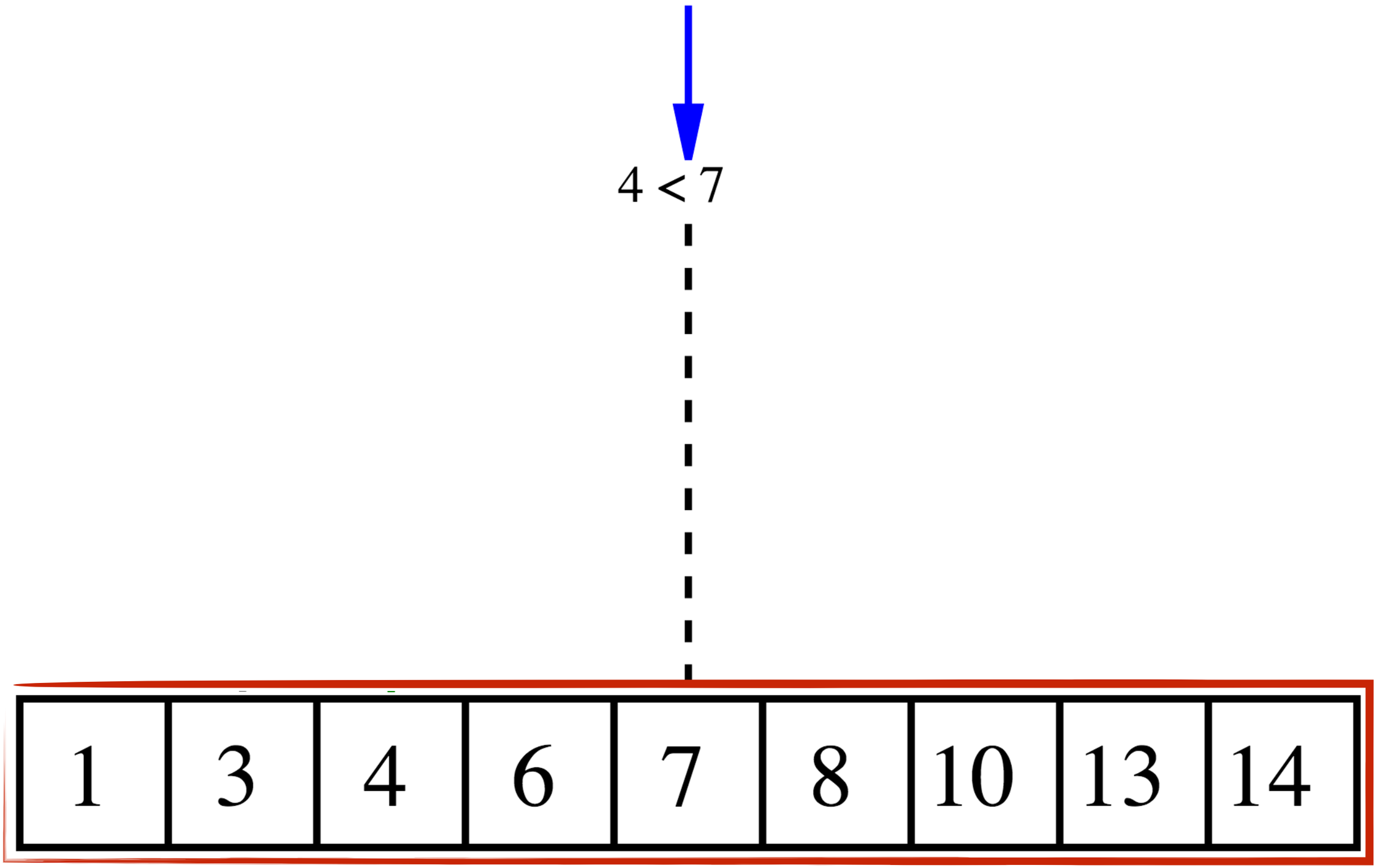

Tony Costa

Esercizio: ricerca binaria (ricorsiva)

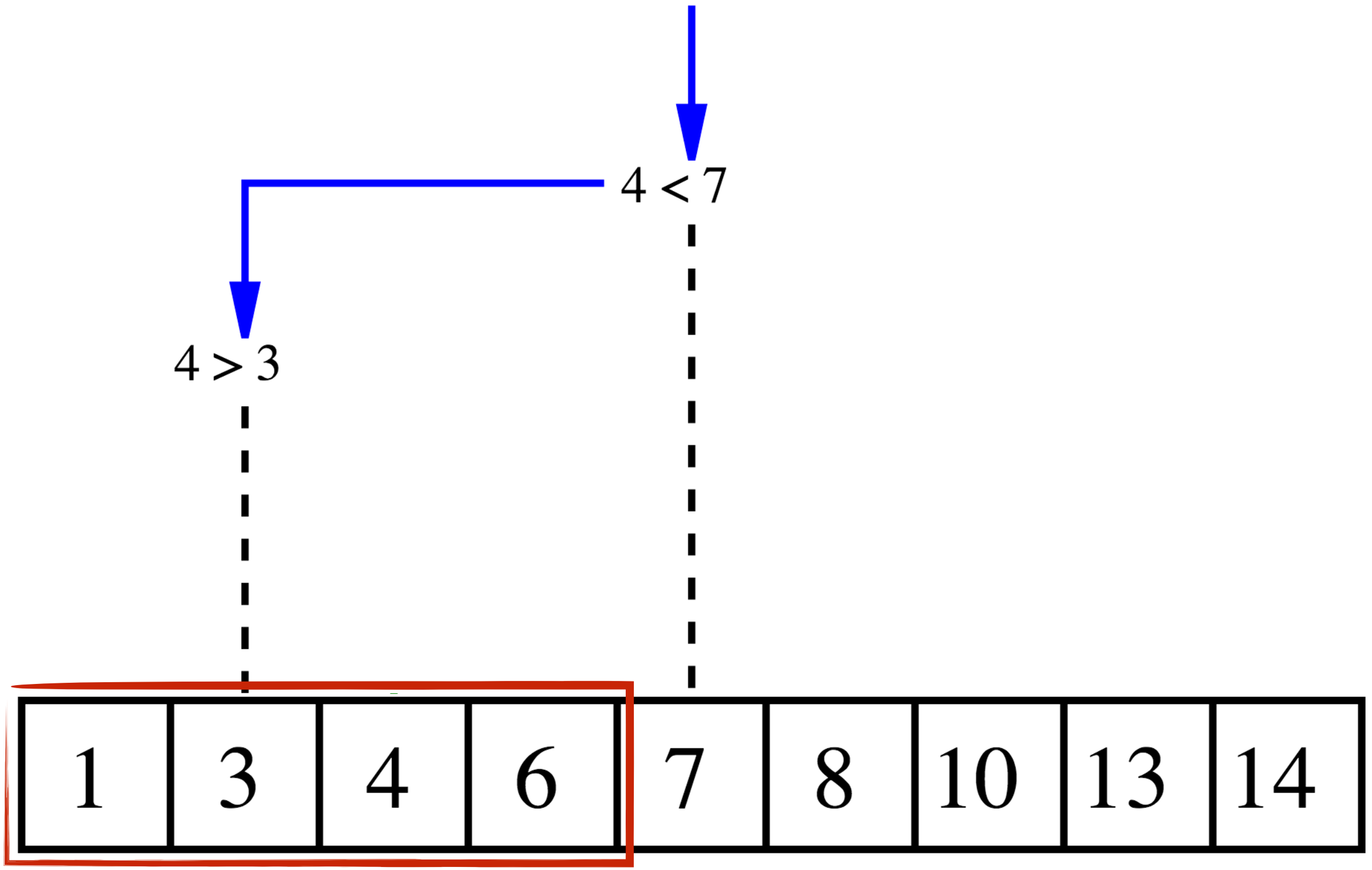
- Effettuare una **ricerca binaria** di un valore all'interno di un array ordinato. Il valore F é intero e l'array a é di interi.

*RICERCA BINARIA: di volta in volta cerco nel **punto medio** di un intervallo che si dimezza in lunghezza ad ogni invocazione dell'algoritmo, **tenendo** opportunamente **la parte destra o sinistra** a seconda che il **punto medio fosse a destra o a sinistra** del valore **F***

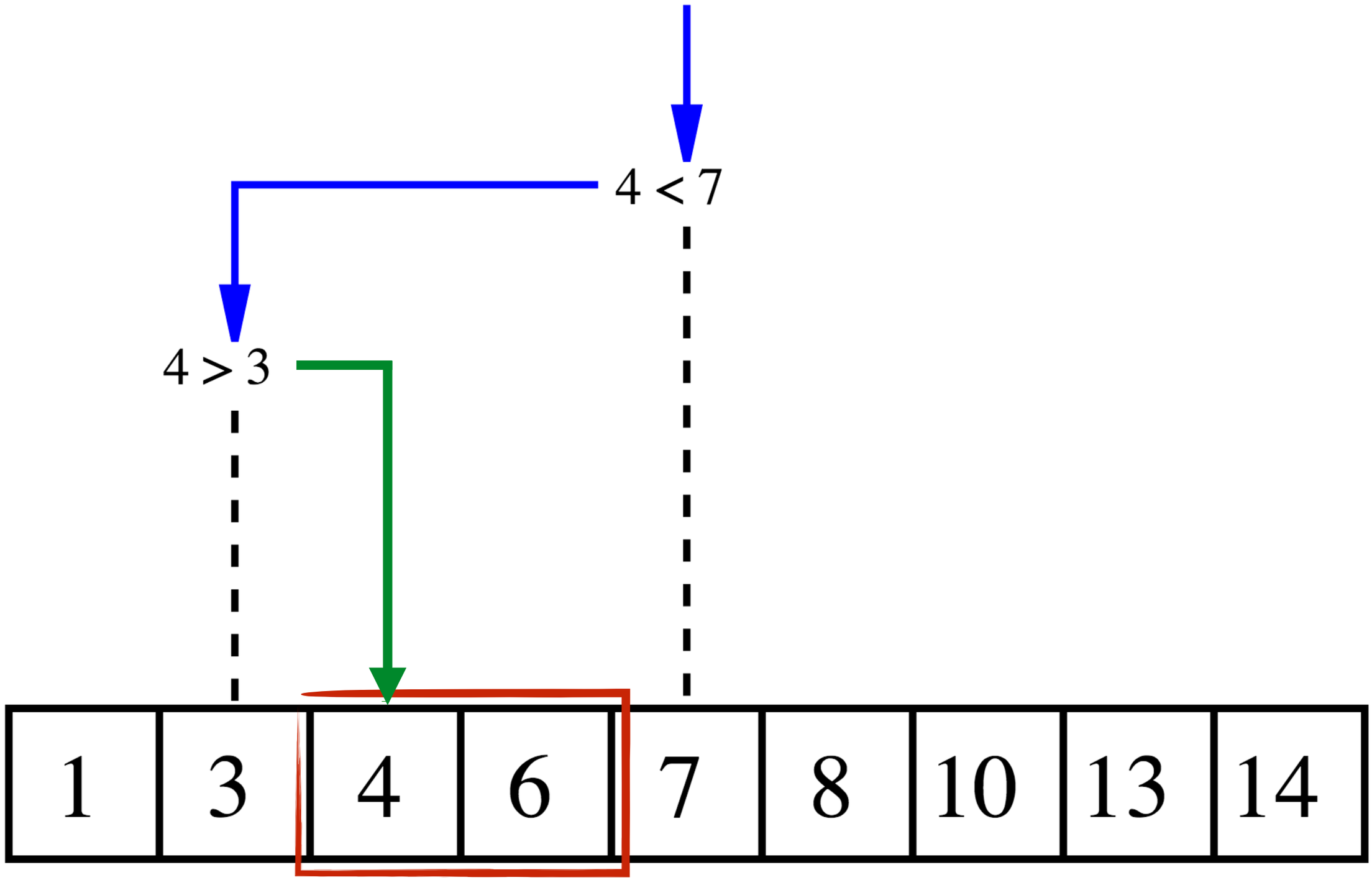
Esercizio: ricerca binaria (ricorsiva)



Esercizio: ricerca binaria (ricorsiva)



Esercizio: ricerca binaria (ricorsiva)



Esercizio: ricerca binaria (ricorsiva)

```
function val = ricercaBinaria(v, cercato, limSinistra, limDestra)
```

Esercizio: ricerca binaria (ricorsiva)

```
function val = ricercaBinaria(v, cercato, limSinistra, limDestra)

    elementoMedio = floor((limDestra + limSinistra)/2);
```

Esercizio: ricerca binaria (ricorsiva)

```
function val = ricercaBinaria(v, cercato, limSinistra, limDestra)

    elementoMedio = floor((limDestra + limSinistra)/2);

    if (v(elementoMedio) == cercato)
        val = elementoMedio;
```

Esercizio: ricerca binaria (ricorsiva)

```
function val = ricercaBinaria(v, cercato, limSinistra, limDestra)

    elementoMedio = floor((limDestra + limSinistra)/2);

    if (v(elementoMedio) == cercato)
        val = elementoMedio;
    else

        val = ricercaBinaria(v, cercato, limSinistra, limDestra);

end
```

Esercizio: ricerca binaria (ricorsiva)

```
function val = ricercaBinaria(v, cercato, limSinistra, limDestra)

    elementoMedio = floor((limDestra + limSinistra)/2);

    if (v(elementoMedio) == cercato)
        val = elementoMedio;
    else
        if (v(elementoMedio) > cercato)
            limDestra = elementoMedio;



        val = ricercaBinaria(v, cercato, limSinistra, limDestra);

end
```

Esercizio: ricerca binaria (ricorsiva)

```
function val = ricercaBinaria(v, cercato, limSinistra, limDestra)

    elementoMedio = floor((limDestra + limSinistra)/2);

    if (v(elementoMedio) == cercato)
        val = elementoMedio;
    else
```

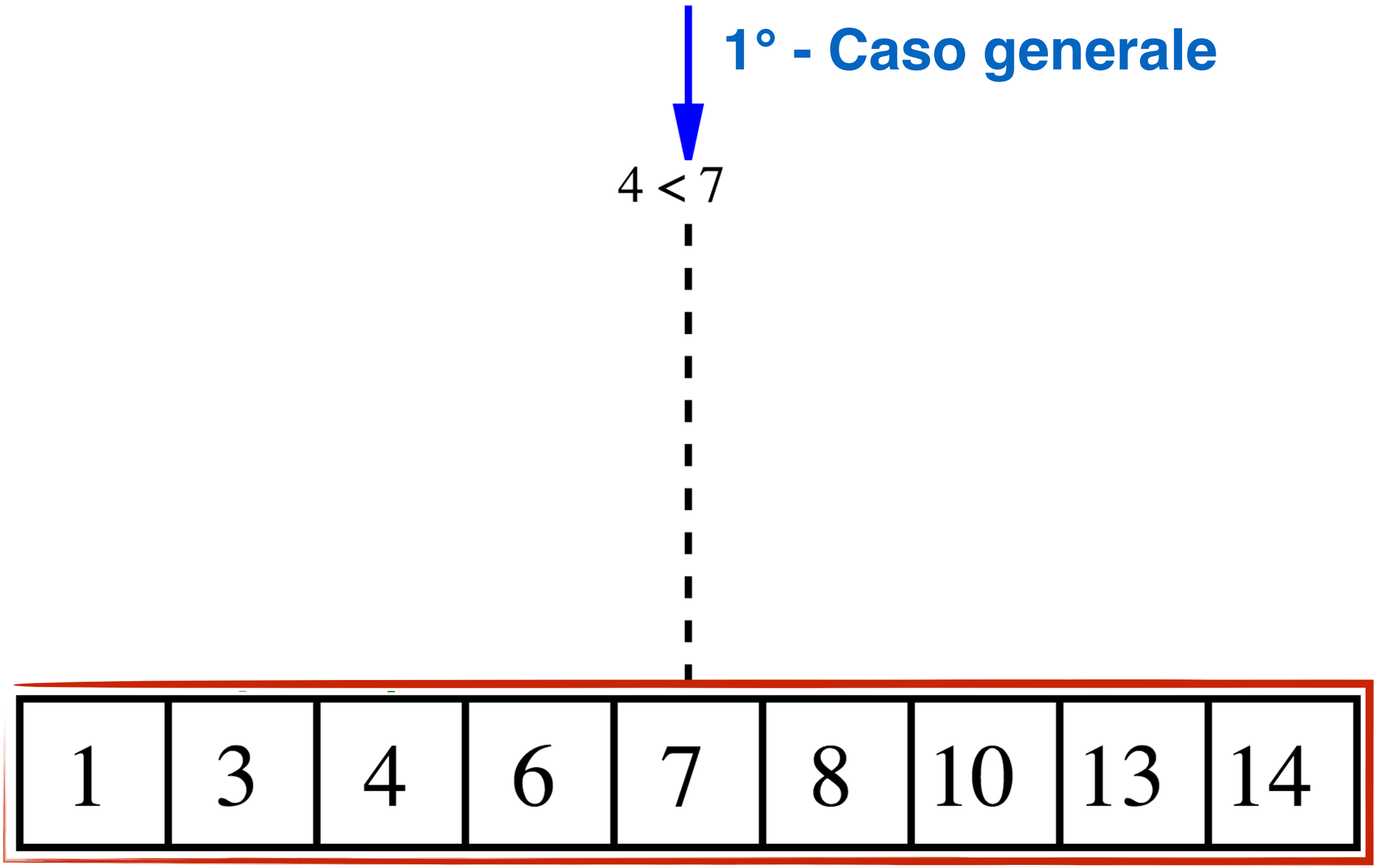
Caso particolare

```
    if (v(elementoMedio) > cercato)
        limDestra = elementoMedio;
    else
        limSinistra = elementoMedio;
    end

    val = ricercaBinaria(v, cercato, limSinistra, limDestra);
```

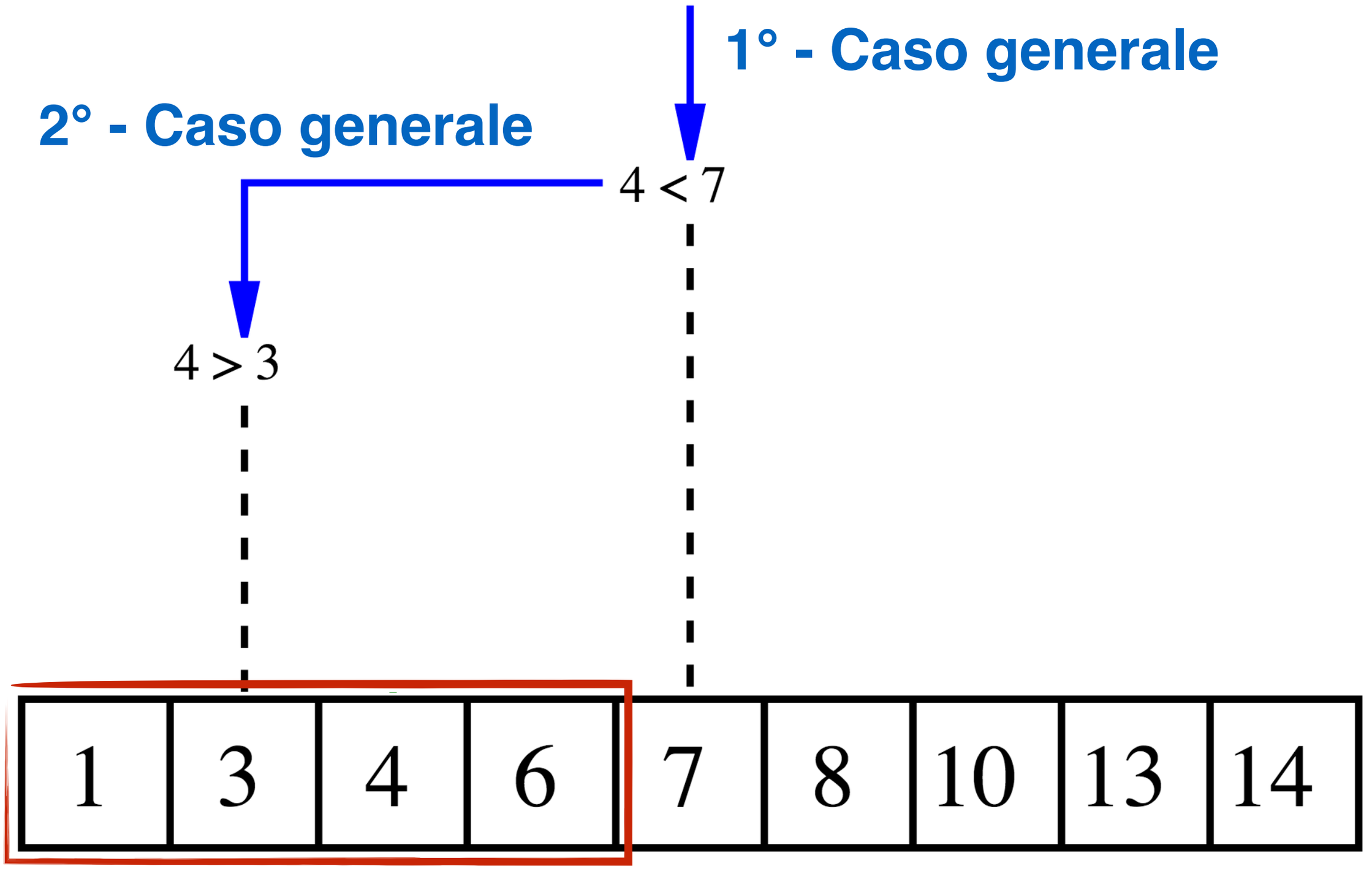
Caso generale

```
end
```

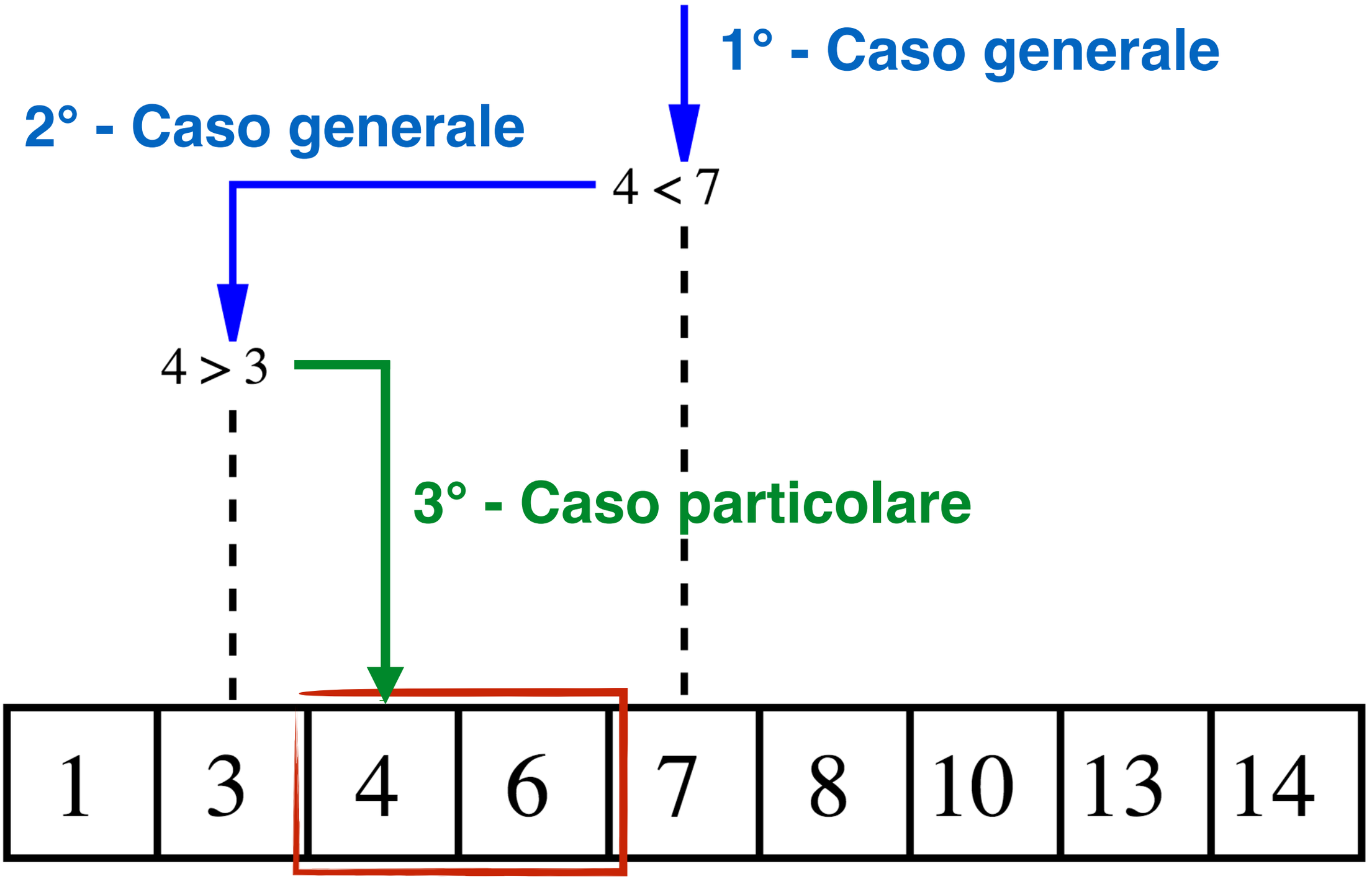
Esercizio: ricerca binaria (ricorsiva)



Esercizio: ricerca binaria (ricorsiva)



Esercizio: ricerca binaria (ricorsiva)



Esercizio: ricerca binaria (ricorsiva)

```
function val = ricercaBinaria(v, cercato, limSinistra, limDestra)

    elementoMedio = floor((limDestra + limSinistra)/2);

    if (v(elementoMedio) == cercato)
        val = elementoMedio;
    else
```

Caso particolare

```
    if (v(elementoMedio) > cercato)
        limDestra = elementoMedio;
    else
        limSinistra = elementoMedio;
    end

    val = ricercaBinaria(v, cercato, limSinistra, limDestra);
```

Caso generale

```
end
```

Sottomatrici, da un altro punto di vista

- Creare una matrice di dimensione **2n*2n** che nel suo centro contiene un quadrato 2x2 che contenente il valore 1 e, andando verso l'esterno, i valori 2, 3, .. fino a **n** nella cornice più esterna.

Esempio:

$$\begin{bmatrix}
6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\
6 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 6 \\
6 & 5 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 3 & 3 & 3 & 3 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 2 & 2 & 2 & 2 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 2 & 1 & 1 & 2 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 2 & 1 & 1 & 2 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 2 & 2 & 2 & 2 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 3 & 3 & 3 & 3 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 5 & 6 \\
6 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 6 \\
6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6
\end{bmatrix}$$

Sottomatrici, versione 1

```
function [M] = sottoMatr(n)
```

$$\begin{bmatrix}
6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\
6 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 6 \\
6 & 5 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 3 & 3 & 3 & 3 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 2 & 2 & 2 & 2 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 2 & 1 & 1 & 2 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 2 & 1 & 1 & 2 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 2 & 2 & 2 & 2 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 3 & 3 & 3 & 3 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 5 & 6 \\
6 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 6 \\
6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6
\end{bmatrix}$$

Sottomatrici, versione 1

```
function [M] = sottoMatr(n)

    if(n == 1)
        M = ones(2,2);
```

$$\begin{bmatrix}
6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\
6 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 6 \\
6 & 5 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 3 & 3 & 3 & 3 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 2 & 2 & 2 & 2 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 2 & 1 & 1 & 2 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 2 & 1 & 1 & 2 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 2 & 2 & 2 & 2 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 3 & 3 & 3 & 3 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 5 & 6 \\
6 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 6 \\
6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6
\end{bmatrix}$$

Sottomatrici, versione 1

```
function [M] = sottoMatr(n)

    if(n == 1)
        M = ones(2,2);
    else
        M = sottoMatr(n-1);
```

$$\begin{bmatrix}
6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\
6 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 6 \\
6 & 5 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 3 & 3 & 3 & 3 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 2 & 2 & 2 & 2 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 2 & 1 & 1 & 2 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 2 & 1 & 1 & 2 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 2 & 2 & 2 & 2 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 3 & 3 & 3 & 3 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 5 & 6 \\
6 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 6 \\
6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6
\end{bmatrix}$$

Sottomatrici, versione 1

```
function [M] = sottoMatr(n)

    if(n == 1)
        M = ones(2,2);
    else
        M = sottoMatr(n-1);
        r = ones(1, size(M, 2)) * n;
        M = [r; M ; r];
```

6	6	6	6	6	6	6	6	6	6	6	6
6	5	5	5	5	5	5	5	5	5	5	6
6	5	4	4	4	4	4	4	4	4	5	6
6	5	4	3	3	3	3	3	3	4	5	6
6	5	4	3	2	2	2	2	3	4	5	6
6	5	4	3	2	1	1	2	3	4	5	6
6	5	4	3	2	1	1	2	3	4	5	6
6	5	4	3	2	2	2	2	3	4	5	6
6	5	4	3	3	3	3	3	3	4	5	6
6	5	4	4	4	4	4	4	4	4	5	6
6	5	5	5	5	5	5	5	5	5	5	6
6	6	6	6	6	6	6	6	6	6	6	6

Sottomatrici, versione 1

```
function [M] = sottoMatr(n)

    if(n == 1)
        M = ones(2,2);
    else
        M = sottoMatr(n-1);
        r = ones(1, size(M, 2)) * n;
        M = [r; M ; r];
        c = ones(size(M, 1), 1) * n;
        M = [c M c];
    end
```

6	6	6	6	6	6	6	6	6	6	6	6
6	5	5	5	5	5	5	5	5	5	5	6
6	5	4	4	4	4	4	4	4	4	5	6
6	5	4	3	3	3	3	3	3	4	5	6
6	5	4	3	2	2	2	2	3	4	5	6
6	5	4	3	2	1	1	2	3	4	5	6
6	5	4	3	2	1	1	2	3	4	5	6
6	5	4	3	2	2	2	2	3	4	5	6
6	5	4	3	3	3	3	3	3	4	5	6
6	5	4	4	4	4	4	4	4	4	5	6
6	5	5	5	5	5	5	5	5	5	5	6
6	6	6	6	6	6	6	6	6	6	6	6

Sottomatrici, versione 2

$$\begin{bmatrix}
6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\
6 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 6 \\
6 & 5 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 3 & 3 & 3 & 3 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 2 & 2 & 2 & 2 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 2 & 1 & 1 & 2 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 2 & 1 & 1 & 2 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 2 & 2 & 2 & 2 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 3 & 3 & 3 & 3 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 5 & 6 \\
6 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 6 \\
6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6
\end{bmatrix}$$

```
function [matrRis]=sottoMatrici2(n)
```

Sottomatrici, versione 2

$$\begin{bmatrix}
6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\
6 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 6 \\
6 & 5 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 3 & 3 & 3 & 3 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 2 & 2 & 2 & 2 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 2 & 1 & 1 & 2 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 2 & 1 & 1 & 2 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 2 & 2 & 2 & 2 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 3 & 3 & 3 & 3 & 3 & 3 & 4 & 5 & 6 \\
6 & 5 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 5 & 6 \\
6 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 6 \\
6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6
\end{bmatrix}$$

```
function [matrRis]=sottoMatrici2(n)

    if n==1
        matrRis=ones(2);
```

Sottomatrici, versione 2

6	6	6	6	6	6	6	6	6	6	6	6
6	5	5	5	5	5	5	5	5	5	5	6
6	5	4	4	4	4	4	4	4	4	5	6
6	5	4	3	3	3	3	3	3	4	5	6
6	5	4	3	2	2	2	2	3	4	5	6
6	5	4	3	2	1	1	2	3	4	5	6
6	5	4	3	2	1	1	2	3	4	5	6
6	5	4	3	2	2	2	2	3	4	5	6
6	5	4	3	3	3	3	3	3	4	5	6
6	5	4	4	4	4	4	4	4	4	5	6
6	5	5	5	5	5	5	5	5	5	5	6
6	6	6	6	6	6	6	6	6	6	6	6

```
function [matrRis]=sottoMatrici2(n)

    if n==1
        matrRis=ones(2);
    else
        matrRis=n*ones(2*n);
```

Sottomatrici, versione 2

6	6	6	6	6	6	6	6	6	6	6	6
6	5	5	5	5	5	5	5	5	5	5	6
6	5	4	4	4	4	4	4	4	4	5	6
6	5	4	3	3	3	3	3	3	4	5	6
6	5	4	3	2	2	2	2	3	4	5	6
6	5	4	3	2	1	1	2	3	4	5	6
6	5	4	3	2	1	1	2	3	4	5	6
6	5	4	3	2	2	2	2	3	4	5	6
6	5	4	3	3	3	3	3	3	4	5	6
6	5	4	4	4	4	4	4	4	4	5	6
6	5	5	5	5	5	5	5	5	5	5	6
6	6	6	6	6	6	6	6	6	6	6	6

```
function [matrRis]=sottoMatrici2(n)

    if n==1
        matrRis=ones(2);
    else
        matrRis=n*ones(2*n);
        matrRis(2:2*n-1,2:2*n-1)=sottoMatrici2(n-1);
    end
```

Esercizio: calcolo derivata (ricorsiva)

- Rappresentiamo un **polinomio** con un **vettore** contenente i suoi **coefficienti**, dal termine di grado massimo a quello di grado minimo.

Si noti che un polinomio di grado n corrisponde a un vettore di lunghezza n+1

*Esempio: **3**x^4 + **5**x^2 + **2**x + **7** (grado 4) corrisponde al vettore [**3 0 5 2 7**] (lunghezza 5)*

- Scrivere una funzione ricorsiva di nome **derivata** che:
  1. Riceva in ingresso un vettore che rappresenta un polinomio e un valore n
  2. Restituisca un vettore che rappresenta la derivata n-esima del polinomio

- Per calcolare la **derivata prima** del polinomio applicare la comune regola di derivazione per i polinomi. **Caso particolare**
- Calcolare la **derivata n-esima** come la **derivata prima della derivata (n-1)-esima** **Caso generale**

Esercizio: calcolo derivata (ricorsiva)

```
function[der] = derivata (pol, n)
```

Esercizio: calcolo derivata (ricorsiva)

```
function[der] = derivata (pol, n)

% Caso base: calcolo della derivata prima
if n==1
```

```
else

    % Passo ricorsivo: derivata (n-1)-esima della derivata prima

end
```

Esercizio: calcolo derivata (ricorsiva)

```
function[der] = derivata (pol, n)

% Caso base: calcolo della derivata prima
if n==1

else

    % Passo ricorsivo: derivata (n-1)-esima della derivata prima
    der = derivata(derivata(pol, n-1), 1)

end
```

Esercizio: calcolo derivata (ricorsiva)

```
function[der] = derivata (pol, n)

% Caso base: calcolo della derivata prima
if n==1

    % Vettore con gli esponenti dei singoli monomi
    % Es. per 2x^3 + x^2 + 3 --> esp = [3 2 1]
    esp = [length(pol)-1 : -1 : 1]

else

    % Passo ricorsivo: derivata (n-1)-esima della derivata prima
    der = derivata(derivata(pol, n-1), 1)

end
```

Esercizio: calcolo derivata (ricorsiva)

```
function[der] = derivata (pol, n)

% Caso base: calcolo della derivata prima
if n==1

    % Vettore con gli esponenti dei singoli monomi
    % Es. per 2x^3 + x^2 + 3 --> esp = [3 2 1]
    esp = [length(pol)-1 : -1 : 1]

    % Calcolo della derivata prima:
    % Es. per il polinomio precedente:
    % [3 2 1].*[2 1 0] = [6 2 0]
    der = esp.*pol(1:length(pol)-1)
```

Caso particolare

```
else

    % Passo ricorsivo: derivata (n-1)-esima della derivata prima
    der = derivata(derivata(pol, n-1), 1)
```

Caso generale

```
end
```

Esercizio: calcolo derivata (ricorsiva)

```
% Passo ricorsivo: derivata (n-1)-esima della derivata prima
der = derivata(derivata(pol, n-1), 1)
```

5x^5 + **4**x^4 + **3**x^3 + **2**x^2 + **1**x^1 + **1** *(grado 5)*

corrisponde al vettore [**5 4 3 2 1 1**] *(lunghezza 6)*

derivata(pol, 3)

der = derivata(**derivata(pol, 2)**, 1)

derivata(pol, 2)

der = derivata(**derivata(pol, 1)**, 1)

derivata(pol, 1)

esp = [5 4 3 2 1]

der = **[25 16 9 4 1]**

Esercizio: calcolo derivata (ricorsiva)

```
% Passo ricorsivo: derivata (n-1)-esima della derivata prima
der = derivata(derivata(pol, n-1), 1)
```

5x^5 + **4**x^4 + **3**x^3 + **2**x^2 + **1**x^1 + **1** *(grado 5)*

corrisponde al vettore [**5 4 3 2 1 1**] *(lunghezza 6)*

derivata(pol, 3)

der = derivata(**derivata(pol, 2)**, 1)

derivata(pol, 2)

der = derivata([25 16 9 4 1], 1)

derivata([25 16 9 4 1], 1)

esp = [4 3 2 1]

der = **[100 48 18 4]**

Esercizio: calcolo derivata (ricorsiva)

```
% Passo ricorsivo: derivata (n-1)-esima della derivata prima
der = derivata(derivata(pol, n-1), 1)
```

5x^5 + **4**x^4 + **3**x^3 + **2**x^2 + **1**x^1 + **1** *(grado 5)*

corrisponde al vettore [**5 4 3 2 1 1**] *(lunghezza 6)*

derivata(pol, 3)

der = derivata(**derivata(pol, 2)**, 1)

derivata(pol, 2)

der = **[100 48 18 4]**

Esercizio: calcolo derivata (ricorsiva)

```
% Passo ricorsivo: derivata (n-1)-esima della derivata prima
der = derivata(derivata(pol, n-1), 1)
```

5x^5 + **4**x^4 + **3**x^3 + **2**x^2 + **1**x^1 + **1** *(grado 5)*

corrisponde al vettore [**5 4 3 2 1 1**] *(lunghezza 6)*

derivata(pol, 3)

der = **derivata([100 48 18 4], 1)**

derivata([100 48 18 4], 1)

esp = [3 2 1]

der = **[300 96 18]**

Esercizio: calcolo derivata (ricorsiva)

```
% Passo ricorsivo: derivata (n-1)-esima della derivata prima
der = derivata(derivata(pol, n-1), 1)
```

5x^5 + **4**x^4 + **3**x^3 + **2**x^2 + **1**x^1 + **1** *(grado 5)*

corrisponde al vettore [**5 4 3 2 1 1**] *(lunghezza 6)*

derivata(pol, 3)

der = **[300 96 18]**