Lezione 10

Sulle equazioni di ricorrenza:

Sulle equazioni di ricorrenza:

Supponiamo di avere la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} d & \text{se } n \leq 1 \\ T(n/2) + T(n/3) + \Theta(n) & \text{altrimenti.} \end{cases}$$

Sulle equazioni di ricorrenza:

Supponiamo di avere la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} d & \text{se } n \leq 1 \\ T(n/2) + T(n/3) + \Theta(n) & \text{altrimenti.} \end{cases}$$

Vogliamo provare che $T(n) = O(n)$, ovvero che

$$\exists c > 0 \text{ tale che } T(n) \leq cn, \text{ per } n \text{ sufficientemente grande}$$

Sulle equazioni di ricorrenza:

Supponiamo di avere la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} d & \text{se } n \leq 1 \\ T(n/2) + T(n/3) + \Theta(n) & \text{altrimenti.} \end{cases}$$

Vogliamo provare che $T(n) = O(n)$, ovvero che

$$\exists c > 0 \text{ tale che } T(n) \leq cn, \text{ per } n \text{ sufficientemente grande}$$

Procediamo per induzione.

Sulle equazioni di ricorrenza:

Supponiamo di avere la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} d & \text{se } n \leq 1 \\ T(n/2) + T(n/3) + \Theta(n) & \text{altrimenti.} \end{cases}$$

Vogliamo provare che $T(n) = O(n)$, ovvero che

$$\exists c > 0 \text{ tale che } T(n) \leq cn, \text{ per } n \text{ sufficientemente grande}$$

Procediamo per induzione. Assumiamo l'esistenza di una costante c per cui $T(k) \leq ck$, per tutti i valori di $k < n$, e proviamo che $T(n) \leq cn$.

Sulle equazioni di ricorrenza:

Supponiamo di avere la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} d & \text{se } n \leq 1 \\ T(n/2) + T(n/3) + \Theta(n) & \text{altrimenti.} \end{cases}$$

Vogliamo provare che $T(n) = O(n)$, ovvero che

$$\exists c > 0 \text{ tale che } T(n) \leq cn, \text{ per } n \text{ sufficientemente grande}$$

Procediamo per induzione. Assumiamo l'esistenza di una costante c per cui $T(k) \leq ck$, per tutti i valori di $k < n$, e proviamo che $T(n) \leq cn$.

$$\begin{aligned} T(n) &= T(n/2) + T(n/3) + \Theta(n) \\ &\leq T(n/2) + T(n/3) + an \qquad \text{(per qualche costante } a\text{)} \end{aligned}$$

Sulle equazioni di ricorrenza:

Supponiamo di avere la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} d & \text{se } n \leq 1 \\ T(n/2) + T(n/3) + \Theta(n) & \text{altrimenti.} \end{cases}$$

Vogliamo provare che $T(n) = O(n)$, ovvero che

$\exists c > 0$ tale che $T(n) \leq cn$, per n sufficientemente grande

Procediamo per induzione. Assumiamo l'esistenza di una costante c per cui $T(k) \leq ck$, per tutti i valori di $k < n$, e proviamo che $T(n) \leq cn$.

$$\begin{aligned} T(n) &= T(n/2) + T(n/3) + \Theta(n) \\ &\leq T(n/2) + T(n/3) + an \qquad \text{(per qualche costante } a\text{)} \\ &\leq c(n/2) + c(n/3) + an \qquad \text{(dall'ipotesi induttiva)} \end{aligned}$$

Sulle equazioni di ricorrenza:

Supponiamo di avere la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} d & \text{se } n \leq 1 \\ T(n/2) + T(n/3) + \Theta(n) & \text{altrimenti.} \end{cases}$$

Vogliamo provare che $T(n) = O(n)$, ovvero che

$\exists c > 0$ tale che $T(n) \leq cn$, per n sufficientemente grande

Procediamo per induzione. Assumiamo l'esistenza di una costante c per cui $T(k) \leq ck$, per tutti i valori di $k < n$, e proviamo che $T(n) \leq cn$.

$$\begin{aligned} T(n) &= T(n/2) + T(n/3) + \Theta(n) \\ &\leq T(n/2) + T(n/3) + an \qquad \text{(per qualche costante } a\text{)} \\ &\leq c(n/2) + c(n/3) + an \qquad \text{(dall'ipotesi induttiva)} \\ &= c\left(\frac{5n}{6}\right) + an \end{aligned}$$

Sulle equazioni di ricorrenza:

Supponiamo di avere la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} d & \text{se } n \leq 1 \\ T(n/2) + T(n/3) + \Theta(n) & \text{altrimenti.} \end{cases}$$

Vogliamo provare che $T(n) = O(n)$, ovvero che

$\exists c > 0$ tale che $T(n) \leq cn$, per n sufficientemente grande

Procediamo per induzione. Assumiamo l'esistenza di una costante c per cui $T(k) \leq ck$, per tutti i valori di $k < n$, e proviamo che $T(n) \leq cn$.

$$\begin{aligned} T(n) &= T(n/2) + T(n/3) + \Theta(n) \\ &\leq T(n/2) + T(n/3) + an \qquad \text{(per qualche costante } a\text{)} \\ &\leq c(n/2) + c(n/3) + an \qquad \text{(dall'ipotesi induttiva)} \\ &= c\left(\frac{5n}{6}\right) + an = cn - c\left(\frac{n}{6}\right) + an \end{aligned}$$

Sulle equazioni di ricorrenza:

Supponiamo di avere la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} d & \text{se } n \leq 1 \\ T(n/2) + T(n/3) + \Theta(n) & \text{altrimenti.} \end{cases}$$

Vogliamo provare che $T(n) = O(n)$, ovvero che

$\exists c > 0$ tale che $T(n) \leq cn$, per n sufficientemente grande

Procediamo per induzione. Assumiamo l'esistenza di una costante c per cui $T(k) \leq ck$, per tutti i valori di $k < n$, e proviamo che $T(n) \leq cn$.

$$\begin{aligned} T(n) &= T(n/2) + T(n/3) + \Theta(n) \\ &\leq T(n/2) + T(n/3) + an \qquad \text{(per qualche costante } a\text{)} \\ &\leq c(n/2) + c(n/3) + an \qquad \text{(dall'ipotesi induttiva)} \\ &= c\left(\frac{5n}{6}\right) + an = cn - c\left(\frac{n}{6}\right) + an \\ &= cn - \left(\frac{cn}{6} - an\right) \end{aligned}$$

Sulle equazioni di ricorrenza:

Supponiamo di avere la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} d & \text{se } n \leq 1 \\ T(n/2) + T(n/3) + \Theta(n) & \text{altrimenti.} \end{cases}$$

Vogliamo provare che $T(n) = O(n)$, ovvero che

$\exists c > 0$ tale che $T(n) \leq cn$, per n sufficientemente grande

Procediamo per induzione. Assumiamo l'esistenza di una costante c per cui $T(k) \leq ck$, per tutti i valori di $k < n$, e proviamo che $T(n) \leq cn$.

$$\begin{aligned} T(n) &= T(n/2) + T(n/3) + \Theta(n) \\ &\leq T(n/2) + T(n/3) + an \qquad \text{(per qualche costante } a\text{)} \\ &\leq c(n/2) + c(n/3) + an \qquad \text{(dall'ipotesi induttiva)} \\ &= c\left(\frac{5n}{6}\right) + an = cn - c\left(\frac{n}{6}\right) + an \\ &= cn - \left(\frac{cn}{6} - an\right) \leq cn \qquad \text{(purchè uno scelga } c > 6a\text{).} \end{aligned}$$

Esercizio 1

Input: Matrice di numeri distinti `a[i][j]`, `i=1...n`, `j=1...n`, in cui **ogni** riga e colonna è ordinata in senso crescente, numero `x`.

Esercizio 1

Input: Matrice di numeri distinti `a[i][j]`, `i=1...n`, `j=1...n`, in cui **ogni** riga e colonna è ordinata in senso crescente, numero `x`.
Output: Coppia `(i,j)` se `a[i][j]=x`, "`non c'è`", altrimenti

Esercizio 1

Input: Matrice di numeri distinti `a[i][j]`, `i=1...n`, `j=1...n`, in cui **ogni** riga e colonna è ordinata in senso crescente, numero `x`.
Output: Coppia `(i,j)` se `a[i][j]=x`, "`non c'è`", altrimenti

Esercizio 1

Input: Matrice di numeri distinti `a[i][j]`, `i=1...n`, `j=1...n`, in cui **ogni** riga e colonna è ordinata in senso crescente, numero `x`.
Output: Coppia `(i,j)` se `a[i][j]=x`, "`non c'è`", altrimenti

$x = 13$

$$\begin{pmatrix} 1 & 6 & 11 & 16 & 21 & 26 \\ 2 & 7 & 12 & 17 & 22 & 27 \\ 3 & 8 & 13 & 18 & 23 & 28 \\ 4 & 9 & 14 & 19 & 24 & 29 \\ 5 & 10 & 15 & 20 & 25 & 30 \end{pmatrix}$$

Prima idea

Confrontiamo l'elemento `x` con l'elemento centrale della matrice `a`.

Prima idea

Confrontiamo l'elemento `x` con l'elemento centrale della matrice `a`. Possono accadere due casi:

1. `x` è $\leq$ dell'elemento centrale `c=a[n/2][n/2]` della matrice `a`.

Prima idea

Confrontiamo l'elemento `x` con l'elemento centrale della matrice `a`. Possono accadere due casi:

1. `x` è $\leq$ dell'elemento centrale `c=a[n/2][n/2]` della matrice `a`.

$$\mathtt{a} = \begin{bmatrix} . & \dots & \dots & . & \dots & \dots & . \\ \vdots & \ddots & \dots & \vdots & \iddots & \dots & \vdots \\ . & \dots & \dots & \mathtt{c} & > \mathtt{c} & \dots & > \mathtt{c} \\ \vdots & \iddots & \dots & > \mathtt{c} & > \mathtt{c} & \dots & > \mathtt{c} \\ . & \dots & \dots & > \mathtt{c} & > \mathtt{c} & \dots & > \mathtt{c} \end{bmatrix}$$

Prima idea

Confrontiamo l'elemento `x` con l'elemento centrale della matrice `a`. Possono accadere due casi:

1. `x` è $\leq$ dell'elemento centrale `c=a[n/2][n/2]` della matrice `a`.

$$\mathtt{a} = \begin{bmatrix} . & \dots & \dots & . & \dots & \dots & . \\ \vdots & \ddots & \dots & \vdots & \iddots & \dots & \vdots \\ . & \dots & \dots & \mathtt{c} & > \mathtt{c} & \dots & > \mathtt{c} \\ \vdots & \iddots & \dots & > \mathtt{c} & > \mathtt{c} & \dots & > \mathtt{c} \\ . & \dots & \dots & > \mathtt{c} & > \mathtt{c} & \dots & > \mathtt{c} \end{bmatrix}$$

Sappiamo allora che `x` non può apparire nel quadrante in basso a destra della matrice `a`,

Prima idea

Confrontiamo l'elemento `x` con l'elemento centrale della matrice `a`. Possono accadere due casi:

1. `x` è $\leq$ dell'elemento centrale `c=a[n/2][n/2]` della matrice `a`.

$$\mathtt{a} = \begin{bmatrix} . & \dots & \dots & . & \dots & \dots & . \\ \vdots & \ddots & \dots & \vdots & \iddots & \dots & \vdots \\ . & \dots & \dots & \mathtt{c} & > \mathtt{c} & \dots & > \mathtt{c} \\ \vdots & \iddots & \dots & > \mathtt{c} & > \mathtt{c} & \dots & > \mathtt{c} \\ . & \dots & \dots & > \mathtt{c} & > \mathtt{c} & \dots & > \mathtt{c} \end{bmatrix}$$

Sappiamo allora che `x` non può apparire nel quadrante in basso a destra della matrice `a`, per cui ricorreremo nella **restante** parte della matrice `a`, ovvero in 3 matrici di dimensione $n/2$ ciascuna.

Secondo Caso

2. `x` è $>$ dell'elemento centrale `c=a[n/2][n/2]` della matrice `a`.

Secondo Caso

2. `x` è $>$ dell'elemento centrale `c=a[n/2][n/2]` della matrice `a`.

$$\mathtt{a} = \begin{bmatrix} < \mathtt{c} & < \mathtt{c} & \dots & < \mathtt{c} & \dots & \dots & . \\ \vdots & \ddots & \dots & \vdots & \iddots & \dots & \vdots \\ < \mathtt{c} & < \mathtt{c} & \dots & \mathtt{c} & . & \dots & . \\ \vdots & \iddots & \dots & . & . & \dots & . \\ . & \dots & \dots & . & . & \dots & . \end{bmatrix}$$

Secondo Caso

2. `x` è $>$ dell'elemento centrale `c=a[n/2][n/2]` della matrice `a`.

$$\mathtt{a} = \begin{bmatrix} < \mathtt{c} & < \mathtt{c} & \dots & < \mathtt{c} & \dots & \dots & . \\ \vdots & \ddots & \dots & \vdots & \iddots & \dots & \vdots \\ < \mathtt{c} & < \mathtt{c} & \dots & \mathtt{c} & . & \dots & . \\ \vdots & \iddots & \dots & . & . & \dots & . \\ . & \dots & \dots & . & . & \dots & . \end{bmatrix}$$

Sappiamo allora che `x` non può apparire nel quadrante in alto a sinistra della matrice `a`,

Secondo Caso

2. `x` è $>$ dell'elemento centrale `c=a[n/2][n/2]` della matrice `a`.

$$\mathtt{a} = \begin{bmatrix} < \mathtt{c} & < \mathtt{c} & \dots & < \mathtt{c} & \dots & \dots & . \\ \vdots & \ddots & \dots & \vdots & \iddots & \dots & \vdots \\ < \mathtt{c} & < \mathtt{c} & \dots & \mathtt{c} & . & \dots & . \\ \vdots & \iddots & \dots & . & . & \dots & . \\ . & \dots & \dots & . & . & \dots & . \end{bmatrix}$$

Sappiamo allora che `x` non può apparire nel quadrante in alto a sinistra della matrice `a`, per cui ricorreremo nella restante parte della matrice `a`, ovvero in 3 matrici di dimensione $n/2$ ciascuna.

La ricorsione termina quando la matrice in cui stiamo cercando è composta da un unico elemento.

La ricorsione termina quando la matrice in cui stiamo cercando è composta da un unico elemento.

Si ricorre sempre in una parte di matrice composta da 3 matrici di dimensione $n/2 \times n/2$, per cui avremo che il tempo di esecuzione $T(n)$ dell'algoritmo soddisfa la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} d & \text{se } n \leq 1 \\ 3T(n/2) + 1 & \text{altrimenti.} \end{cases}$$

La ricorsione termina quando la matrice in cui stiamo cercando è composta da un unico elemento.

Si ricorre sempre in una parte di matrice composta da 3 matrici di dimensione $n/2 \times n/2$, per cui avremo che il tempo di esecuzione $T(n)$ dell'algoritmo soddisfa la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} d & \text{se } n \leq 1 \\ 3T(n/2) + 1 & \text{altrimenti.} \end{cases}$$

Usando i risultati generali visti nelle precedenti lezioni, sappiamo che $T(n) = O(n^{\log_2 3})$

Possiamo fare meglio?

Possiamo fare meglio?

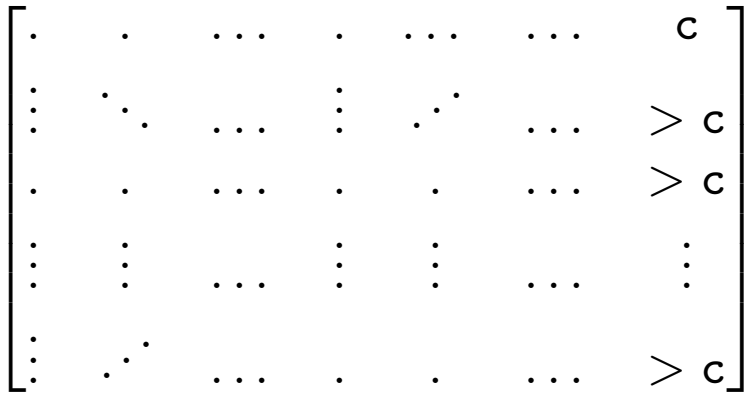
Sì.

Possiamo fare meglio?

Sì. Confrontiamo `x` con l'elemento `c=a[1][n]` nella riga `1` e colonna `n` della matrice `a`.

Possiamo fare meglio?

Sì. Confrontiamo `x` con l'elemento `c=a[1][n]` nella riga `1` e colonna `n` della matrice `a`.



Possiamo fare meglio?

Sì. Confrontiamo `x` con l'elemento `c=a[1][n]` nella riga `1` e colonna `n` della matrice `a`.

$$\begin{bmatrix} \cdot & \cdot & \cdots & \cdot & \cdots & \cdots & c \\ \vdots & \ddots & \cdots & \vdots & \iddots & \cdots & > c \\ \cdot & \cdot & \cdots & \cdot & \cdot & \cdots & > c \\ \vdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots \\ \vdots & \iddots & \cdots & \cdot & \cdot & \cdots & > c \end{bmatrix}$$

Sappiamo che

- se `x<c` allora `x` **non** può apparire nell'ultima colonna di `a`, (in quanto in essa compaiono sicuramente elementi ancora più grandi di `c=a[1][n]`,

Possiamo fare meglio?

Sì. Confrontiamo `x` con l'elemento `c=a[1][n]` nella riga `1` e colonna `n` della matrice `a`.

$$\begin{bmatrix} \cdot & \cdot & \cdots & \cdot & \cdots & \cdots & c \\ \vdots & \ddots & \cdots & \vdots & \iddots & \cdots & > c \\ \cdot & \cdot & \cdots & \cdot & \cdot & \cdots & > c \\ \vdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots \\ \vdots & \iddots & \cdots & \cdot & \cdot & \cdots & > c \end{bmatrix}$$

Sappiamo che

- se `x<c` allora `x` **non** può apparire nell'ultima colonna di `a`, (in quanto in essa compaiono sicuramente elementi ancora più grandi di `c=a[1][n]`, per cui ricorreremo nella restante parte della matrice `a`.

Possiamo fare meglio?

Sì. Confrontiamo `x` con l'elemento `c=a[1][n]` nella riga `1` e colonna `n` della matrice `a`.

$$\begin{bmatrix} \cdot & \cdot & \cdots & \cdot & \cdots & \cdots & c \\ \vdots & \ddots & \cdots & \vdots & \iddots & \cdots & > c \\ \cdot & \cdot & \cdots & \cdot & \cdot & \cdots & > c \\ \vdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots \\ \vdots & \iddots & \cdots & \cdot & \cdot & \cdots & > c \end{bmatrix}$$

Sappiamo che

- se `x<c` allora `x` **non** può apparire nell'ultima colonna di `a`, (in quanto in essa compaiono sicuramente elementi ancora più grandi di `c=a[1][n]`, per cui ricorreremo nella restante parte della matrice `a`.
- se `x>c`, possiamo senz'altro escludere da future ricerche la prima riga di `a`.

L'algoritmo:

```
CercaInMatrice(a,x)
1.   i=1, j=n
```

L'algoritmo:

```
CercaInMatrice(a,x)
1.  i=1, j=n
2.  while (i<n+1&&j>=1)
```

L'algoritmo:

```
CercaInMatrice(a,x)
1.   i=1, j=n
2.   while (i<n+1&&j>=1)
3.      if(a[i][j]==x){
4.       Return(i,j)
```

L'algoritmo:

```
CercaInMatrice(a,x)
1.   i=1, j=n
2.   while (i<n+1&&j>=1)
3.      if(a[i][j]==x){
4.       Return(i,j)
     }
5.      if(x<a[i][j]){
```

L'algoritmo:

```
CercaInMatrice(a,x)
1.   i=1, j=n
2.   while (i<n+1&&j>=1)
3.      if(a[i][j]==x){
4.       Return(i,j)
     }
5.      if(x<a[i][j]){
6.       j=j-1
```

L'algoritmo:

```
CercaInMatrice(a,x)
1.   i=1, j=n
2.   while (i<n+1&&j>=1)
3.      if(a[i][j]==x){
4.       Return(i,j)
     }
5.      if(x<a[i][j]){
6.       j=j-1
7.       } else { i=i+1 }
   }
```

L'algoritmo:

```
CercaInMatrice(a,x)
1.   i=1, j=n
2.   while (i<n+1&&j>=1)
3.      if(a[i][j]==x){
4.       Return(i,j)
     }
5.      if(x<a[i][j]){
6.       j=j-1
7.       } else { i=i+1 }
   }
Return(``non c'è '')
```

L'algoritmo:

```
CercaInMatrice(a,x)
1.   i=1, j=n
2.   while (i<n+1&&j>=1)
3.      if(a[i][j]==x){
4.       Return(i,j)
     }
5.      if(x<a[i][j]){
6.       j=j-1
7.       } else { i=i+1 }
   }
Return(‘‘non c'è ’’)
```

Complessità : Ad ogni passo o si elimina un'intera colonna oppure si elimina un'intera riga, per cui dopo n passi abbiamo sicuramente terminato.

L'algoritmo:

```
CercaInMatrice(a,x)
1.  i=1, j=n
2.  while (i<n+1&&j>=1)
3.     if(a[i][j]==x){
4.       Return(i,j)
     }
5.     if(x<a[i][j]){
6.       j=j-1
7.       } else { i=i+1 }
  }
Return(‘‘non c'è ’’)
```

Complessità : Ad ogni passo o si elimina un'intera colonna oppure si elimina un'intera riga, per cui dopo n passi abbiamo sicuramente terminato. Poichè ogni passo ha un costo costante, si ha che $T(n) = \Theta(n)$ nel caso peggiore.

```
CercaInMatrice(a,x)
1.  i=1, j=n
2.  while (i<n+1&&j>=1)
3.     if(a[i][j]==x){
4.       Return(i,j)
    }
5.     if(x<a[i][j]){
6.       j=j-1
7.       } else { i=i+1 }
  }
Return(‘‘non c’è ’’)
```

```
CercaInMatrice(a,x)
1.  i=1, j=n
2.  while (i<n+1&&j>=1)
3.     if(a[i][j]==x){
4.       Return(i,j)
     }
5.     if(x<a[i][j]){
6.       j=j-1
7.       } else { i=i+1 }
  }
Return(‘‘non c’è ’’)
```

$x = 13$

$$\begin{pmatrix} 1 & 6 & 11 & 16 & 21 & 26 \\ 2 & 7 & 12 & 17 & 22 & 27 \\ 3 & 8 & 13 & 18 & 23 & 28 \\ 4 & 9 & 14 & 19 & 24 & 29 \\ 5 & 10 & 15 & 20 & 25 & 30 \end{pmatrix}$$

```
CercaInMatrice(a,x)
1.  i=1, j=n
2.  while (i<n+1&&j>=1)
3.     if(a[i][j]==x){
4.       Return(i,j)
     }
5.     if(x<a[i][j]){
6.       j=j-1
7.       } else { i=i+1 }
  }
Return(‘‘non c’è ’’)
```

$$\begin{pmatrix} 1 & 6 & 11 & 16 & 21 & 26 \\ 2 & 7 & 12 & 17 & 22 & 27 \\ 3 & 8 & 13 & 18 & 23 & 28 \\ 4 & 9 & 14 & 19 & 24 & 29 \\ 5 & 10 & 15 & 20 & 25 & 30 \end{pmatrix}$$

$x = 13$

$$\begin{pmatrix} 1 & 6 & 11 & 16 & 21 & 26 \\ 2 & 7 & 12 & 17 & 22 & 27 \\ 3 & 8 & 13 & 18 & 23 & 28 \\ 4 & 9 & 14 & 19 & 24 & 29 \\ 5 & 10 & 15 & 20 & 25 & 30 \end{pmatrix}$$

```
CercaInMatrice(a,x)
    i=1, j=n
    while (i<n+1&&j>=1)
       if(a[i][j]==x){
        Return(i,j)
    }
       if(x<a[i][j]){
        j=j-1
        } else { i=i+1 }
  }
Return(‘‘non c’è ’’)
```

$x = 13$

$$\begin{pmatrix} 1 & 6 & 11 & 16 & 21 & 26 \\ 2 & 7 & 12 & 17 & 22 & 27 \\ 3 & 8 & {\color{red}13} & 18 & 23 & 28 \\ 4 & 9 & 14 & 19 & 24 & 29 \\ 5 & 10 & 15 & 20 & 25 & 30 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 6 & 11 & 16 & 21 & {\color{blue}26} \\ 2 & 7 & 12 & 17 & 22 & {\color{blue}27} \\ 3 & 8 & {\color{red}13} & 18 & 23 & {\color{blue}28} \\ 4 & 9 & 14 & 19 & 24 & {\color{blue}29} \\ 5 & 10 & 15 & 20 & 25 & {\color{blue}30} \end{pmatrix}$$

$$\begin{pmatrix} 1 & 6 & 11 & 16 & {\color{blue}21} & {\color{blue}26} \\ 2 & 7 & 12 & 17 & {\color{blue}22} & {\color{blue}27} \\ 3 & 8 & {\color{red}13} & 18 & {\color{blue}23} & {\color{blue}28} \\ 4 & 9 & 14 & 19 & {\color{blue}24} & {\color{blue}29} \\ 5 & 10 & 15 & 20 & {\color{blue}25} & {\color{blue}30} \end{pmatrix}$$

```
CercaInMatrice(a,x)
    i=1, j=n
    while (i<n+1&&j>=1)
       if(a[i][j]==x){
        Return(i,j)
      }
       if(x<a[i][j]){
        j=j-1
        } else { i=i+1 }
  }
Return(‘‘non c’è ’’)
```

$x = 13$

$$\begin{pmatrix} 1 & 6 & 11 & 16 & 21 & 26 \\ 2 & 7 & 12 & 17 & 22 & 27 \\ 3 & 8 & \color{red}{13} & 18 & 23 & 28 \\ 4 & 9 & 14 & 19 & 24 & 29 \\ 5 & 10 & 15 & 20 & 25 & 30 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 6 & 11 & 16 & 21 & \color{blue}{26} \\ 2 & 7 & 12 & 17 & 22 & \color{blue}{27} \\ 3 & 8 & \color{red}{13} & 18 & 23 & \color{blue}{28} \\ 4 & 9 & 14 & 19 & 24 & \color{blue}{29} \\ 5 & 10 & 15 & 20 & 25 & \color{blue}{30} \end{pmatrix}$$

$$\begin{pmatrix} 1 & 6 & 11 & 16 & \color{blue}{21} & \color{blue}{26} \\ 2 & 7 & 12 & 17 & \color{blue}{22} & \color{blue}{27} \\ 3 & 8 & \color{red}{13} & 18 & \color{blue}{23} & \color{blue}{28} \\ 4 & 9 & 14 & 19 & \color{blue}{24} & \color{blue}{29} \\ 5 & 10 & 15 & 20 & \color{blue}{25} & \color{blue}{30} \end{pmatrix}$$

$$\begin{pmatrix} 1 & 6 & 11 & \color{blue}{16} & \color{blue}{21} & \color{blue}{26} \\ 2 & 7 & 12 & \color{blue}{17} & \color{blue}{22} & \color{blue}{27} \\ 3 & 8 & \color{red}{13} & \color{blue}{18} & \color{blue}{23} & \color{blue}{28} \\ 4 & 9 & 14 & \color{blue}{19} & \color{blue}{24} & \color{blue}{29} \\ 5 & 10 & 15 & \color{blue}{20} & \color{blue}{25} & \color{blue}{30} \end{pmatrix}$$

```
CercaInMatrice(a,x)
    i=1, j=n
    while (i<n+1&&j>=1)
       if(a[i][j]==x){
        Return(i,j)
    }
       if(x<a[i][j]){
        j=j-1
        } else { i=i+1 }
  }
Return(‘‘non c’è ’’)
```

$x = 13$

$$\begin{pmatrix} 1 & 6 & 11 & 16 & 21 & 26 \\ 2 & 7 & 12 & 17 & 22 & 27 \\ 3 & 8 & \color{red}{13} & 18 & 23 & 28 \\ 4 & 9 & 14 & 19 & 24 & 29 \\ 5 & 10 & 15 & 20 & 25 & 30 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 6 & 11 & 16 & 21 & \color{blue}{26} \\ 2 & 7 & 12 & 17 & 22 & \color{blue}{27} \\ 3 & 8 & \color{red}{13} & 18 & 23 & \color{blue}{28} \\ 4 & 9 & 14 & 19 & 24 & \color{blue}{29} \\ 5 & 10 & 15 & 20 & 25 & \color{blue}{30} \end{pmatrix}$$

$$\begin{pmatrix} 1 & 6 & 11 & 16 & \color{blue}{21} & \color{blue}{26} \\ 2 & 7 & 12 & 17 & \color{blue}{22} & \color{blue}{27} \\ 3 & 8 & \color{red}{13} & 18 & \color{blue}{23} & \color{blue}{28} \\ 4 & 9 & 14 & 19 & \color{blue}{24} & \color{blue}{29} \\ 5 & 10 & 15 & 20 & \color{blue}{25} & \color{blue}{30} \end{pmatrix}$$

$$\begin{pmatrix} 1 & 6 & 11 & \color{blue}{16} & \color{blue}{21} & \color{blue}{26} \\ 2 & 7 & 12 & \color{blue}{17} & \color{blue}{22} & \color{blue}{27} \\ 3 & 8 & \color{red}{13} & \color{blue}{18} & \color{blue}{23} & \color{blue}{28} \\ 4 & 9 & 14 & \color{blue}{19} & \color{blue}{24} & \color{blue}{29} \\ 5 & 10 & 15 & \color{blue}{20} & \color{blue}{25} & \color{blue}{30} \end{pmatrix}$$

$$\begin{pmatrix} 1 & 6 & \color{green}{11} & \color{blue}{16} & \color{blue}{21} & \color{blue}{26} \\ 2 & 7 & 12 & \color{blue}{17} & \color{blue}{22} & \color{blue}{27} \\ 3 & 8 & \color{red}{13} & \color{blue}{18} & \color{blue}{23} & \color{blue}{28} \\ 4 & 9 & 14 & \color{blue}{19} & \color{blue}{24} & \color{blue}{29} \\ 5 & 10 & 15 & \color{blue}{20} & \color{blue}{25} & \color{blue}{30} \end{pmatrix}$$

```
CercaInMatrice(a,x)
   i=1, j=n
   while (i<n+1&&j>=1)
      if(a[i][j]==x){
       Return(i,j)
   }
      if(x<a[i][j]){
       j=j-1
       } else { i=i+1 }
 }
Return(‘‘non c’è ’’)
```

$x = 13$

$$\begin{pmatrix} 1 & 6 & 11 & 16 & 21 & 26 \\ 2 & 7 & 12 & 17 & 22 & 27 \\ 3 & 8 & \color{red}{13} & 18 & 23 & 28 \\ 4 & 9 & 14 & 19 & 24 & 29 \\ 5 & 10 & 15 & 20 & 25 & 30 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 6 & 11 & 16 & 21 & \color{blue}{26} \\ 2 & 7 & 12 & 17 & 22 & \color{blue}{27} \\ 3 & 8 & \color{red}{13} & 18 & 23 & \color{blue}{28} \\ 4 & 9 & 14 & 19 & 24 & \color{blue}{29} \\ 5 & 10 & 15 & 20 & 25 & \color{blue}{30} \end{pmatrix}$$

$$\begin{pmatrix} 1 & 6 & 11 & 16 & \color{blue}{21} & \color{blue}{26} \\ 2 & 7 & 12 & 17 & \color{blue}{22} & \color{blue}{27} \\ 3 & 8 & \color{red}{13} & 18 & \color{blue}{23} & \color{blue}{28} \\ 4 & 9 & 14 & 19 & \color{blue}{24} & \color{blue}{29} \\ 5 & 10 & 15 & 20 & \color{blue}{25} & \color{blue}{30} \end{pmatrix}$$

$$\begin{pmatrix} 1 & 6 & 11 & \color{blue}{16} & \color{blue}{21} & \color{blue}{26} \\ 2 & 7 & 12 & \color{blue}{17} & \color{blue}{22} & \color{blue}{27} \\ 3 & 8 & \color{red}{13} & \color{blue}{18} & \color{blue}{23} & \color{blue}{28} \\ 4 & 9 & 14 & \color{blue}{19} & \color{blue}{24} & \color{blue}{29} \\ 5 & 10 & 15 & \color{blue}{20} & \color{blue}{25} & \color{blue}{30} \end{pmatrix}$$

$$\begin{pmatrix} 1 & 6 & \color{green}{11} & \color{blue}{16} & \color{blue}{21} & \color{blue}{26} \\ 2 & 7 & 12 & \color{blue}{17} & \color{blue}{22} & \color{blue}{27} \\ 3 & 8 & \color{red}{13} & \color{blue}{18} & \color{blue}{23} & \color{blue}{28} \\ 4 & 9 & 14 & \color{blue}{19} & \color{blue}{24} & \color{blue}{29} \\ 5 & 10 & 15 & \color{blue}{20} & \color{blue}{25} & \color{blue}{30} \end{pmatrix}$$

$$\begin{pmatrix} 1 & 6 & \color{green}{11} & \color{blue}{16} & \color{blue}{21} & \color{blue}{26} \\ 2 & 7 & \color{green}{12} & \color{blue}{17} & \color{blue}{22} & \color{blue}{27} \\ 3 & 8 & \color{red}{13} & \color{blue}{18} & \color{blue}{23} & \color{blue}{28} \\ 4 & 9 & 14 & \color{blue}{19} & \color{blue}{24} & \color{blue}{29} \\ 5 & 10 & 15 & \color{blue}{20} & \color{blue}{25} & \color{blue}{30} \end{pmatrix}$$

```
CercaInMatrice(a,x)
1.  i=1, j=n
2.  while (i<n+1&&j>=1)
3.     if(a[i][j]==x){
4.      Return(i,j)
    }
5.     if(x<a[i][j]){
6.      j=j-1
7.      } else { i=i+1 }
  }
Return(‘‘non c’è ’’)
```

$x = 13$

$$\begin{pmatrix} 1 & 6 & 11 & 16 & 21 & 26 \\ 2 & 7 & 12 & 17 & 22 & 27 \\ 3 & 8 & \textcolor{red}{13} & 18 & 23 & 28 \\ 4 & 9 & 14 & 19 & 24 & 29 \\ 5 & 10 & 15 & 20 & 25 & 30 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 6 & 11 & 16 & 21 & \textcolor{blue}{26} \\ 2 & 7 & 12 & 17 & 22 & \textcolor{blue}{27} \\ 3 & 8 & \textcolor{red}{13} & 18 & 23 & \textcolor{blue}{28} \\ 4 & 9 & 14 & 19 & 24 & \textcolor{blue}{29} \\ 5 & 10 & 15 & 20 & 25 & \textcolor{blue}{30} \end{pmatrix}$$

$$\begin{pmatrix} 1 & 6 & 11 & 16 & \textcolor{blue}{21} & \textcolor{blue}{26} \\ 2 & 7 & 12 & 17 & \textcolor{blue}{22} & \textcolor{blue}{27} \\ 3 & 8 & \textcolor{red}{13} & 18 & \textcolor{blue}{23} & \textcolor{blue}{28} \\ 4 & 9 & 14 & 19 & \textcolor{blue}{24} & \textcolor{blue}{29} \\ 5 & 10 & 15 & 20 & \textcolor{blue}{25} & \textcolor{blue}{30} \end{pmatrix}$$

$$\begin{pmatrix} 1 & 6 & 11 & \textcolor{blue}{16} & \textcolor{blue}{21} & \textcolor{blue}{26} \\ 2 & 7 & 12 & \textcolor{blue}{17} & \textcolor{blue}{22} & \textcolor{blue}{27} \\ 3 & 8 & \textcolor{red}{13} & \textcolor{blue}{18} & \textcolor{blue}{23} & \textcolor{blue}{28} \\ 4 & 9 & 14 & \textcolor{blue}{19} & \textcolor{blue}{24} & \textcolor{blue}{29} \\ 5 & 10 & 15 & \textcolor{blue}{20} & \textcolor{blue}{25} & \textcolor{blue}{30} \end{pmatrix}$$

$$\begin{pmatrix} 1 & 6 & \textcolor{green}{11} & \textcolor{blue}{16} & \textcolor{blue}{21} & \textcolor{blue}{26} \\ 2 & 7 & 12 & \textcolor{blue}{17} & \textcolor{blue}{22} & \textcolor{blue}{27} \\ 3 & 8 & \textcolor{red}{13} & \textcolor{blue}{18} & \textcolor{blue}{23} & \textcolor{blue}{28} \\ 4 & 9 & 14 & \textcolor{blue}{19} & \textcolor{blue}{24} & \textcolor{blue}{29} \\ 5 & 10 & 15 & \textcolor{blue}{20} & \textcolor{blue}{25} & \textcolor{blue}{30} \end{pmatrix}$$

$$\begin{pmatrix} 1 & 6 & \textcolor{green}{11} & \textcolor{blue}{16} & \textcolor{blue}{21} & \textcolor{blue}{26} \\ 2 & 7 & \textcolor{green}{12} & \textcolor{blue}{17} & \textcolor{blue}{22} & \textcolor{blue}{27} \\ 3 & 8 & \textcolor{red}{13} & \textcolor{blue}{18} & \textcolor{blue}{23} & \textcolor{blue}{28} \\ 4 & 9 & 14 & \textcolor{blue}{19} & \textcolor{blue}{24} & \textcolor{blue}{29} \\ 5 & 10 & 15 & \textcolor{blue}{20} & \textcolor{blue}{25} & \textcolor{blue}{30} \end{pmatrix}$$

Esercizio 2

Data un vettore numerico `a=a[1]...a[n]`, diremo che un elemento di `a` è maggioritario se e solo se esso compare almeno $\lfloor n/2 \rfloor + 1$ volte in `a`.

Esercizio 2

Data un vettore numerico `a=a[1]...a[n]`, diremo che un elemento di `a` è maggioritario se e solo se esso compare almeno $\lfloor n/2 \rfloor + 1$ volte in `a`.

Il problema algoritmico è :
Input: `a=a[1]...a[n]`
Output: `True`, se `a` contiene un elemento maggioritario, `False` altrimenti.

Esercizio 2

Data un vettore numerico `a=a[1]...a[n]`, diremo che un elemento di `a` è maggioritario se e solo se esso compare almeno $\lfloor n/2 \rfloor + 1$ volte in `a`.

Il problema algoritmico è :
Input: `a=a[1]...a[n]`
Output: `True`, se `a` contiene un elemento maggioritario, `False` altrimenti.

Esempio: `a=1 1 3 5 1 1 2 1 5 1` ha come elemento maggioritario `1`

Esercizio 2

Data un vettore numerico `a=a[1]...a[n]`, diremo che un elemento di `a` è maggioritario se e solo se esso compare almeno $\lfloor n/2 \rfloor + 1$ volte in `a`.

Il problema algoritmico è :
Input: `a=a[1]...a[n]`
Output: `True`, se `a` contiene un elemento maggioritario, `False` altrimenti.

Esempio: `a=1 1 3 5 1 1 2 1 5 1` ha come elemento maggioritario `1`

`a=4 1 3 5 1 1 2 1 5 1` non ha elementi maggioritari

Prima idea

Ordina il vettore a

Prima idea

Ordina il vettore a (cosicchè elementi che appaiono più volte in a sono adesso **consecutivi**)

Prima idea

Ordina il vettore a (cosicchè elementi che appaiono più volte in a sono adesso **consecutivi**) scorrilo da sinistra a destra incrementando un contatore quando si incontrano elementi uguali (e **riportandolo** ad 1 nel caso opposto).

Prima idea

Ordina il vettore a (cosicchè elementi che appaiono più volte in a sono adesso **consecutivi**) scorrilo da sinistra a destra incrementando un contatore quando si incontrano elementi uguali (e **riportandolo** ad 1 nel caso opposto).

Restituisci `True` appena il contatore supera $n/2$, restituisci `False` se usciamo dal ciclo senza aver trovato un elemento maggioritario.

Prima idea

Ordina il vettore a (cosicchè elementi che appaiono più volte in a sono adesso **consecutivi**) scorrilo da sinistra a destra incrementando un contatore quando si incontrano elementi uguali (e **riportandolo** ad 1 nel caso opposto).

Restituisci `True` appena il contatore supera $n/2$, restituisci `False` se usciamo dal ciclo senza aver trovato un elemento maggioritario.

```
CercaMaggioritario(a)
1. Ordina(a)
```

Prima idea

Ordina il vettore a (cosicchè elementi che appaiono più volte in a sono adesso **consecutivi**) scorrilo da sinistra a destra incrementando un contatore quando si incontrano elementi uguali (e **riportandolo** ad 1 nel caso opposto).

Restituisci `True` appena il contatore supera $n/2$, restituisci `False` se usciamo dal ciclo senza aver trovato un elemento maggioritario.

```
CercaMaggioritario(a)
1. Ordina(a)
2. last=a[1]; cont=1
```

Prima idea

Ordina il vettore a (cosicchè elementi che appaiono più volte in a sono adesso **consecutivi**) scorrilo da sinistra a destra incrementando un contatore quando si incontrano elementi uguali (e **riportandolo** ad 1 nel caso opposto).

Restituisci `True` appena il contatore supera $n/2$, restituisci `False` se usciamo dal ciclo senza aver trovato un elemento maggioritario.

```
CercaMaggioritario(a)
1. Ordina(a)
2. last=a[1]; cont=1
3. for(i=2;i<n+1;i=i+1){
4.    if(a[i]==last){
5.      cont=cont+1
```

Prima idea

Ordina il vettore a (cosicchè elementi che appaiono più volte in a sono adesso **consecutivi**) scorrilo da sinistra a destra incrementando un contatore quando si incontrano elementi uguali (e **riportandolo** ad 1 nel caso opposto).

Restituisci `True` appena il contatore supera $n/2$, restituisci `False` se usciamo dal ciclo senza aver trovato un elemento maggioritario.

```
CercaMaggioritario(a)
1. Ordina(a)
2. last=a[1]; cont=1
3. for(i=2;i<n+1;i=i+1){
4.    if(a[i]==last){
5.      cont=cont+1
6.      if (cont>n/2){
7.      return True
```

Prima idea

Ordina il vettore a (cosicchè elementi che appaiono più volte in a sono adesso **consecutivi**) scorrilo da sinistra a destra incrementando un contatore quando si incontrano elementi uguali (e **riportandolo** ad 1 nel caso opposto).

Restituisci `True` appena il contatore supera $n/2$, restituisci `False` se usciamo dal ciclo senza aver trovato un elemento maggioritario.

```
CercaMaggioritario(a)
1. Ordina(a)
2. last=a[1]; cont=1
3. for(i=2;i<n+1;i=i+1){
4.    if(a[i]==last){
5.      cont=cont+1
6.      if (cont>n/2){
7.      return True
8.      } else {
```

Prima idea

Ordina il vettore a (cosicchè elementi che appaiono più volte in a sono adesso **consecutivi**) scorrilo da sinistra a destra incrementando un contatore quando si incontrano elementi uguali (e **riportandolo** ad 1 nel caso opposto).

Restituisci `True` appena il contatore supera $n/2$, restituisci `False` se usciamo dal ciclo senza aver trovato un elemento maggioritario.

```
CercaMaggioritario(a)
1. Ordina(a)
2. last=a[1]; cont=1
3. for(i=2;i<n+1;i=i+1){
4.    if(a[i]==last){
5.      cont=cont+1
6.      if (cont>n/2){
7.      return True
8.      } else {
9.      last=a[i]
10.      cont=1
   }   }
```

Prima idea

Ordina il vettore a (cosicchè elementi che appaiono più volte in a sono adesso **consecutivi**) scorrilo da sinistra a destra incrementando un contatore quando si incontrano elementi uguali (e **riportandolo** ad 1 nel caso opposto).

Restituisci `True` appena il contatore supera $n/2$, restituisci `False` se usciamo dal ciclo senza aver trovato un elemento maggioritario.

```
CercaMaggioritario(a)
1. Ordina(a)
2. last=a[1]; cont=1
3. for(i=2;i<n+1;i=i+1){
4.    if(a[i]==last){
5.      cont=cont+1
6.      if (cont>n/2){
7.      return True
8.      } else {
9.      last=a[i]
10.      cont=1
   }   }
return False
```

Prima idea

Ordina il vettore a (cosicchè elementi che appaiono più volte in a sono adesso **consecutivi**) scorrilo da sinistra a destra incrementando un contatore quando si incontrano elementi uguali (e **riportandolo** ad 1 nel caso opposto).

Restituisci `True` appena il contatore supera $n/2$, restituisci `False` se usciamo dal ciclo senza aver trovato un elemento maggioritario.

```
CercaMaggioritario(a)
1. Ordina(a)
2. last=a[1]; cont=1
3. for(i=2;i<n+1;i=i+1){
4.    if(a[i]==last){
5.      cont=cont+1
6.      if (cont>n/2){
7.      return True
8.      } else {
9.      last=a[i]
10.      cont=1
   }   }
return False
```

Esempio: `a=4 1 3 5 1 1 2 1 5 1`

Prima idea

Ordina il vettore a (cosicchè elementi che appaiono più volte in a sono adesso **consecutivi**) scorrilo da sinistra a destra incrementando un contatore quando si incontrano elementi uguali (e **riportandolo** ad 1 nel caso opposto).

Restituisci `True` appena il contatore supera $n/2$, restituisci `False` se usciamo dal ciclo senza aver trovato un elemento maggioritario.

```
CercaMaggioritario(a)
1. Ordina(a)
2. last=a[1]; cont=1
3. for(i=2;i<n+1;i=i+1){
4.    if(a[i]==last){
5.      cont=cont+1
6.      if (cont>n/2){
7.      return True
8.      } else {
9.      last=a[i]
10.      cont=1
   }   }
return False
```

Esempio: a=4 1 3 5 1 1 2 1 5 1$\rightarrow$ a=1 1 1 1 1 2 3 4 5 5

Prima idea

Ordina il vettore a (cosicchè elementi che appaiono più volte in a sono adesso **consecutivi**) scorrilo da sinistra a destra incrementando un contatore quando si incontrano elementi uguali (e **riportandolo** ad 1 nel caso opposto).

Restituisci `True` appena il contatore supera $n/2$, restituisci `False` se usciamo dal ciclo senza aver trovato un elemento maggioritario.

```
CercaMaggioritario(a)
1. Ordina(a)
2. last=a[1]; cont=1
3. for(i=2;i<n+1;i=i+1){
4.    if(a[i]==last){
5.      cont=cont+1
6.      if (cont>n/2){
7.      return True
8.      } else {
9.      last=a[i]
10.      cont=1
   }   }
return False
```

Esempio: a=4 1 3 5 1 1 2 1 5 1$\rightarrow$ a=1 1 1 1 1 2 3 4 5 5
Esempio: a=1 1 3 5 1 1 2 1 5 1

Prima idea

Ordina il vettore a (cosicchè elementi che appaiono più volte in a sono adesso **consecutivi**) scorrilo da sinistra a destra incrementando un contatore quando si incontrano elementi uguali (e **riportandolo** ad 1 nel caso opposto).

Restituisci `True` appena il contatore supera $n/2$, restituisci `False` se usciamo dal ciclo senza aver trovato un elemento maggioritario.

```
CercaMaggioritario(a)
1. Ordina(a)
2. last=a[1]; cont=1
3. for(i=2;i<n+1;i=i+1){
4.    if(a[i]==last){
5.      cont=cont+1
6.      if (cont>n/2){
7.      return True
8.      } else {
9.      last=a[i]
10.      cont=1
   }   }
return False
```

Esempio: a=4 1 3 5 1 1 2 1 5 1→ a=1 1 1 1 1 2 3 4 5 5
Esempio: a=1 1 3 5 1 1 2 1 5 1→ a=1 1 1 1 1 1 2 3 5 5

Prima idea

Ordina il vettore a (cosicchè elementi che appaiono più volte in a sono adesso **consecutivi**) scorrilo da sinistra a destra incrementando un contatore quando si incontrano elementi uguali (e **riportandolo** ad 1 nel caso opposto).

Restituisci `True` appena il contatore supera $n/2$, restituisci `False` se usciamo dal ciclo senza aver trovato un elemento maggioritario.

```
CercaMaggioritario(a)
1. Ordina(a)
2. last=a[1]; cont=1
3. for(i=2;i<n+1;i=i+1){
4.    if(a[i]==last){
5.      cont=cont+1
6.      if (cont>n/2){
7.      return True
8.      } else {
9.      last=a[i]
10.      cont=1
   }   }
return False
```

Esempio: a=4 1 3 5 1 1 2 1 5 1$\rightarrow$ a=1 1 1 1 1 2 3 4 5 5
Esempio: a=1 1 3 5 1 1 2 1 5 1$\rightarrow$ a=1 1 1 1 1 1 2 3 5 5
Complessità: $T(n) = \Theta(n \log n) + n$

Prima idea

Ordina il vettore a (cosicchè elementi che appaiono più volte in a sono adesso **consecutivi**) scorrilo da sinistra a destra incrementando un contatore quando si incontrano elementi uguali (e **riportandolo** ad 1 nel caso opposto).

Restituisci `True` appena il contatore supera $n/2$, restituisci `False` se usciamo dal ciclo senza aver trovato un elemento maggioritario.

```
CercaMaggioritario(a)
1. Ordina(a)
2. last=a[1]; cont=1
3. for(i=2;i<n+1;i=i+1){
4.    if(a[i]==last){
5.      cont=cont+1
6.      if (cont>n/2){
7.      return True
8.      } else {
9.      last=a[i]
10.      cont=1
   }   }
return False
```

Esempio: a=4 1 3 5 1 1 2 1 5 1$\rightarrow$ a=1 1 1 1 1 2 3 4 5 5
Esempio: a=1 1 3 5 1 1 2 1 5 1$\rightarrow$ a=1 1 1 1 1 1 2 3 5 5
Complessità: $T(n) = \Theta(n \log n) + n = \Theta(n \log n)$.

Possiamo fare meglio?

Possiamo fare meglio?

Sì.

Possiamo fare meglio?

Sì. **Osservazione**: se esiste un elemento maggioritario, esso necessariamente comparirà anche nella posizione $n/2$ della versione ordinata di a,

Possiamo fare meglio?

Sì. **Osservazione**: se esiste un elemento maggioritario, esso necessariamente comparirà anche nella posizione $n/2$ della versione ordinata di a, (esempio: a=1 1 3 5 1 1 2 1 5 1 $\rightarrow$ a=1 1 1 1 1 1 2 3 5 5)

Possiamo fare meglio?

Sì. **Osservazione**: se esiste un elemento maggioritario, esso necessariamente comparirà anche nella posizione $n/2$ della versione ordinata di a, (esempio: a=1 1 3 5 1 1 2 1 5 1 $\rightarrow$ a=1 1 1 1 1 1 2 3 5 5) ovvero esso è **un elemento mediano**.

Possiamo fare meglio?

Sì. **Osservazione**: se esiste un elemento maggioritario, esso necessariamente comparirà anche nella posizione $n/2$ della versione ordinata di a, (esempio: a=1 1 3 5 1 1 2 1 5 1 $\rightarrow$ a=1 1 1 1 1 1 2 3 5 5) ovvero esso è **un elemento mediano**. Di conseguenza, basterà cercare l'elemento mediano di a e verificare se esso compare o meno almeno $\lceil n/2 \rceil + 1$ volte in a.

Possiamo fare meglio?

Sì. **Osservazione**: se esiste un elemento maggioritario, esso necessariamente comparirà anche nella posizione $n/2$ della versione ordinata di a, (esempio: a=1 1 3 5 1 1 2 1 5 1 $\rightarrow$ a=1 1 1 1 1 1 2 3 5 5) ovvero esso è **un elemento mediano**. Di conseguenza, basterà cercare l'elemento mediano di a e verificare se esso compare o meno almeno $\lceil n/2 \rceil + 1$ volte in a. (Esempio: a=4 1 3 5 1 1 2 1 5 1 $\rightarrow$ a=1 1 1 1 1 2 3 4 5 5)

Possiamo fare meglio?

Sì. **Osservazione**: se esiste un elemento maggioritario, esso necessariamente comparirà anche nella posizione $n/2$ della versione ordinata di a, (esempio: a=1 1 3 5 1 1 2 1 5 1 $\rightarrow$ a=1 1 1 1 1 1 2 3 5 5) ovvero esso è **un elemento mediano**. Di conseguenza, basterà cercare l'elemento mediano di a e verificare se esso compare o meno almeno $\lceil n/2 \rceil + 1$ volte in a. (Esempio: a=4 1 3 5 1 1 2 1 5 1 $\rightarrow$ a=1 1 1 1 1 2 3 4 5 5)

```
CercaMaggioritario2(a)
1. m=QuickSelect(a,1,n/2,n)
```

Possiamo fare meglio?

Sì. **Osservazione**: se esiste un elemento maggioritario, esso necessariamente comparirà anche nella posizione $n/2$ della versione ordinata di a, (esempio: a=1 1 3 5 1 1 2 1 5 1 $\rightarrow$ a=1 1 1 1 1 1 2 3 5 5) ovvero esso è **un elemento mediano**. Di conseguenza, basterà cercare l'elemento mediano di a e verificare se esso compare o meno almeno $\lceil n/2 \rceil + 1$ volte in a. (Esempio: a=4 1 3 5 1 1 2 1 5 1 $\rightarrow$ a=1 1 1 1 1 2 3 4 5 5)

```
CercaMaggioritario2(a)
1. m=QuickSelect(a,1,n/2,n)
2. cont=0
```

Possiamo fare meglio?

Sì. **Osservazione**: se esiste un elemento maggioritario, esso necessariamente comparirà anche nella posizione $n/2$ della versione ordinata di a, (esempio: a=1 1 3 5 1 1 2 1 5 1 $\rightarrow$ a=1 1 1 1 1 1 2 3 5 5) ovvero esso è **un elemento mediano**. Di conseguenza, basterà cercare l'elemento mediano di a e verificare se esso compare o meno almeno $\lceil n/2 \rceil + 1$ volte in a. (Esempio: a=4 1 3 5 1 1 2 1 5 1 $\rightarrow$ a=1 1 1 1 1 2 3 4 5 5)

```
CercaMaggioritario2(a)
1. m=QuickSelect(a,1,n/2,n)
2. cont=0
3. for(i=1;i<n+1;i=i+1){
```

Possiamo fare meglio?

Sì. **Osservazione**: se esiste un elemento maggioritario, esso necessariamente comparirà anche nella posizione $n/2$ della versione ordinata di a, (esempio: a=1 1 3 5 1 1 2 1 5 1 $\rightarrow$ a=1 1 1 1 1 1 2 3 5 5) ovvero esso è **un elemento mediano**. Di conseguenza, basterà cercare l'elemento mediano di a e verificare se esso compare o meno almeno $\lceil n/2 \rceil + 1$ volte in a. (Esempio: a=4 1 3 5 1 1 2 1 5 1 $\rightarrow$ a=1 1 1 1 1 2 3 4 5 5)

```
CercaMaggioritario2(a)
1. m=QuickSelect(a,1,n/2,n)
2. cont=0
3. for(i=1;i<n+1;i=i+1){
4.    if(a[i]==a[m]){
```

Possiamo fare meglio?

Sì. **Osservazione**: se esiste un elemento maggioritario, esso necessariamente comparirà anche nella posizione $n/2$ della versione ordinata di a, (esempio: a=1 1 3 5 1 1 2 1 5 1 $\rightarrow$ a=1 1 1 1 1 1 2 3 5 5) ovvero esso è **un elemento mediano**. Di conseguenza, basterà cercare l'elemento mediano di a e verificare se esso compare o meno almeno $\lceil n/2 \rceil + 1$ volte in a. (Esempio: a=4 1 3 5 1 1 2 1 5 1 $\rightarrow$ a=1 1 1 1 1 2 3 4 5 5)

```
CercaMaggioritario2(a)
1. m=QuickSelect(a,1,n/2,n)
2. cont=0
3. for(i=1;i<n+1;i=i+1){
4.    if(a[i]==a[m]){
5.      cont=cont+1
```

Possiamo fare meglio?

Sì. **Osservazione**: se esiste un elemento maggioritario, esso necessariamente comparirà anche nella posizione $n/2$ della versione ordinata di a, (esempio: a=1 1 3 5 1 1 2 1 5 1 $\rightarrow$ a=1 1 1 1 1 1 2 3 5 5) ovvero esso è **un elemento mediano**. Di conseguenza, basterà cercare l'elemento mediano di a e verificare se esso compare o meno almeno $\lceil n/2 \rceil + 1$ volte in a. (Esempio: a=4 1 3 5 1 1 2 1 5 1 $\rightarrow$ a=1 1 1 1 1 2 3 4 5 5)

```
CercaMaggioritario2(a)
1. m=QuickSelect(a,1,n/2,n)
2. cont=0
3. for(i=1;i<n+1;i=i+1){
4.    if(a[i]==a[m]){
5.      cont=cont+1
   }
6.    if (cont>n/2){
7.      return True
```

Possiamo fare meglio?

Sì. **Osservazione**: se esiste un elemento maggioritario, esso necessariamente comparirà anche nella posizione $n/2$ della versione ordinata di a, (esempio: a=1 1 3 5 1 1 2 1 5 1 $\rightarrow$ a=1 1 1 1 1 1 2 3 5 5) ovvero esso è **un elemento mediano**. Di conseguenza, basterà cercare l'elemento mediano di a e verificare se esso compare o meno almeno $\lceil n/2 \rceil + 1$ volte in a. (Esempio: a=4 1 3 5 1 1 2 1 5 1 $\rightarrow$ a=1 1 1 1 1 2 3 4 5 5)

```
CercaMaggioritario2(a)
1. m=QuickSelect(a,1,n/2,n)
2. cont=0
3. for(i=1;i<n+1;i=i+1){
4.    if(a[i]==a[m]){
5.      cont=cont+1
   }
6.    if (cont>n/2){
7.      return True
8.      } else {
9.      return False
}
```

Possiamo fare meglio?

Sì. **Osservazione**: se esiste un elemento maggioritario, esso necessariamente comparirà anche nella posizione $n/2$ della versione ordinata di a, (esempio: a=1 1 3 5 1 1 2 1 5 1 $\rightarrow$ a=1 1 1 1 1 1 2 3 5 5) ovvero esso è **un elemento mediano**. Di conseguenza, basterà cercare l'elemento mediano di a e verificare se esso compare o meno almeno $\lceil n/2 \rceil + 1$ volte in a. (Esempio: a=4 1 3 5 1 1 2 1 5 1 $\rightarrow$ a=1 1 1 1 1 2 3 4 5 5)

```
CercaMaggioritario2(a)
1. m=QuickSelect(a,1,n/2,n)
2. cont=0
3. for(i=1;i<n+1;i=i+1){
4.    if(a[i]==a[m]){
5.      cont=cont+1
   }
6.    if (cont>n/2){
7.      return True
8.      } else {
9.      return False
}
```

Tempo medio $T(n) = \Theta(n)$.

Esercizio 3:

Input: array `a=a[1]...a[n]` contenente valori interi ordinati in senso non decrescente (possono essere presenti valori duplicati), intero `x` .

Esercizio 3:

Input: array `a=a[1]...a[n]` contenente valori interi ordinati in senso non decrescente (possono essere presenti valori duplicati), intero `x` .
Output: l'indice (la posizione) della **prima** occorrenza di `x` in `a=a[1]...a[n]`, oppure `n+1` se il valore `x` non è presente in `a=a[1]...a[n]`.

Esercizio 3:

Input: array `a=a[1]...a[n]` contenente valori interi ordinati in senso non decrescente (possono essere presenti valori duplicati), intero `x` .
Output: l'indice (la posizione) della **prima** occorrenza di `x` in `a=a[1]...a[n]`, oppure `n+1` se il valore `x` non è presente in `a=a[1]...a[n]`.

Ad esempio, se `a=[1,3,4,4,4,5,6,6]` e `x=4`, l'algoritmo deve restituire 3, in quanto `a[3]` è la prima occorrenza del valore `4`.

Esercizio 3:

Input: array `a=a[1]...a[n]` contenente valori interi ordinati in senso non decrescente (possono essere presenti valori duplicati), intero `x` .
Output: l'indice (la posizione) della **prima** occorrenza di `x` in `a=a[1]...a[n]`, oppure `n+1` se il valore `x` non è presente in `a=a[1]...a[n]`.

Ad esempio, se `a=[1,3,4,4,4,5,6,6]` e `x=4`, l'algoritmo deve restituire 3, in quanto `a[3]` è la prima occorrenza del valore 4. Se `x=6` l'algoritmo deve restituire 7, in quanto `a[7]` è la prima occorrenza del valore 6.

L'idea:

Vogliamo un algoritmo `CERCAPRIMAOCCORRENZA(A,x,i,j)` che restituisce l'indice della **prima** occorrenza del valore `x` all'interno del sottovettore ordinato `a[i]...a[j]`.

L'idea:

Vogliamo un algoritmo `CERCAPRIMAOCCORRENZA(A,x,i,j)` che restituisce l'indice della **prima** occorrenza del valore `x` all'interno del sottovettore ordinato `a[i]...a[j]`.

I parametri di input dell'algoritmo devono soddisfare le seguenti precondizioni (*oltre* al fatto che `a[i]...a[j]` deve essere ordinato in senso non decrescente):

- $\forall$ `k=1,..., i-1` vale che `a[k]<x`
- $\forall$ `k=j+1,...,n` vale che `a[k]`$\geq$`x`

L'idea:

Vogliamo un algoritmo `CERCAPRIMAOCCORRENZA(A,x,i,j)` che restituisce l'indice della **prima** occorrenza del valore `x` all'interno del sottovettore ordinato `a[i]...a[j]`.

I parametri di input dell'algoritmo devono soddisfare le seguenti precondizioni (*oltre* al fatto che `a[i]...a[j]` deve essere ordinato in senso non decrescente):

- $\forall$ `k=1,..., i-1` vale che `a[k]<x`
- $\forall$ `k=j+1,...,n` vale che `a[k]`$\geq$`x`

Date le precondizioni sopra e detta `m` la posizione centrale del sottovettore `a[i]...a[j]`, il valore `m` **è il risultato cercato** se `A[m]==x`

L'idea:

Vogliamo un algoritmo `CERCAPRIMAOCCORRENZA(A,x,i,j)` che restituisce l'indice della **prima** occorrenza del valore `x` all'interno del sottovettore ordinato `a[i]...a[j]`.

I parametri di input dell'algoritmo devono soddisfare le seguenti precondizioni (*oltre* al fatto che `a[i]...a[j]` deve essere ordinato in senso non decrescente):

- $\forall$ `k=1,..., i-1` vale che `a[k]<x`
- $\forall$ `k=j+1,...,n` vale che `a[k]`$\geq$`x`

Date le precondizioni sopra e detta `m` la posizione centrale del sottovettore `a[i]...a[j]`, il valore `m` **è il risultato cercato** se `A[m]==x` **e vale una** delle seguenti condizioni:

1. siamo all'inizio del sottovettore, cioè vale `(m==i)`,

L'idea:

Vogliamo un algoritmo `CERCAPRIMAOCCORRENZA(A,x,i,j)` che restituisce l'indice della **prima** occorrenza del valore `x` all'interno del sottovettore ordinato `a[i]...a[j]`.

I parametri di input dell'algoritmo devono soddisfare le seguenti precondizioni (*oltre* al fatto che `a[i]...a[j]` deve essere ordinato in senso non decrescente):

- $\forall$ `k=1,..., i-1` vale che `a[k]<x`
- $\forall$ `k=j+1,...,n` vale che `a[k]`$\geq$`x`

Date le precondizioni sopra e detta `m` la posizione centrale del sottovettore `a[i]...a[j]`, il valore `m` **è il risultato cercato** se `A[m]==x` **e vale una** delle seguenti condizioni:

1. siamo all'inizio del sottovettore, cioè vale `(m==i)`, **oppure**
2. l'elemento precedente `a[m-1]` ha valore diverso da `x`.

L'idea:

Vogliamo un algoritmo `CERCAPRIMAOCCORRENZA(A,x,i,j)` che restituisce l'indice della **prima** occorrenza del valore `x` all'interno del sottovettore ordinato `a[i]...a[j]`.

I parametri di input dell'algoritmo devono soddisfare le seguenti precondizioni (*oltre* al fatto che `a[i]...a[j]` deve essere ordinato in senso non decrescente):

- $\forall$ `k=1,..., i-1` vale che `a[k]<x`
- $\forall$ `k=j+1,...,n` vale che `a[k]`$\geq$`x`

Date le precondizioni sopra e detta `m` la posizione centrale del sottovettore `a[i]...a[j]`, il valore `m` **è il risultato cercato** se `A[m]==x` **e vale una** delle seguenti condizioni:

1. siamo all'inizio del sottovettore, cioè vale `(m==i)`, **oppure**
2. l'elemento precedente `a[m-1]` ha valore diverso da `x`.

Se nessuna delle due condizioni vale, la ricerca prosegue ricorsivamente su una delle due metà del sottovettore `a[i]...a[j]`, **in modo tale** da mantenere valida la precondizione di cui sopra.

L'idea:

Vogliamo un algoritmo `CERCAPRIMAOCCORRENZA(A,x,i,j)` che restituisce l'indice della **prima** occorrenza del valore `x` all'interno del sottovettore ordinato `a[i]...a[j]`.

I parametri di input dell'algoritmo devono soddisfare le seguenti precondizioni (*oltre* al fatto che `a[i]...a[j]` deve essere ordinato in senso non decrescente):
- $\forall$ `k=1,..., i-1` vale che `a[k]<x`
- $\forall$ `k=j+1,...,n` vale che `a[k]`$\geq$`x`

Date le precondizioni sopra e detta `m` la posizione centrale del sottovettore `a[i]...a[j]`, il valore `m` **è il risultato cercato** se `A[m]==x` **e vale una** delle seguenti condizioni:
1. siamo all'inizio del sottovettore, cioè vale `(m==i)`, **oppure**
2. l'elemento precedente `a[m-1]` ha valore diverso da `x`.

Se nessuna delle due condizioni vale, la ricerca prosegue ricorsivamente su una delle due metà del sottovettore `a[i]...a[j]`, **in modo tale** da mantenere valida la precondizione di cui sopra.
Come fare a mantenere la precondizione lo vediamo direttamente nello pseudocodice.

L'algoritmo:

```
CERCAPRIMAOCCORRENZA(A,x,i,j)
1.  IF(i>j) {
2.        Return n+1
```

L'algoritmo:

```
CERCAPRIMAOCCORRENZA(A,x,i,j)
1.  IF(i>j) {
2.       Return n+1
3.     } ELSE {
4.       m=(i+j)/2
```

L'algoritmo:

```
CERCAPRIMAOCCORRENZA(A,x,i,j)
1.  IF(i>j) {
2.       Return n+1
3.      } ELSE {
4.       m=(i+j)/2
5.       IF((a[m]==x)&&(m==i||a[m]≠a[m-1])){
6.       Return m
```

L'algoritmo:

```
CERCAPRIMAOCCORRENZA(A,x,i,j)
1.  IF(i>j) {
2.       Return n+1
3.      } ELSE {
4.       m=(i+j)/2
5.       IF((a[m]==x)&&(m==i||a[m]≠a[m-1])){
6.       Return m
7.      } ELSE {
8.         IF((a[m]>=x){
```

L'algoritmo:

```
CERCAPRIMAOCCORRENZA(A,x,i,j)
1.  IF(i>j) {
2.       Return n+1
3.      } ELSE {
4.       m=(i+j)/2
5.       IF((a[m]==x)&&(m==i||a[m]≠a[m-1])){
6.       Return m
7.      } ELSE {
8.         IF((a[m]>=x){
9.            Return CERCAPRIMAOCCORRENZA(A,x,i,m-1)
```

L'algoritmo:

```
CERCAPRIMAOCCORRENZA(A,x,i,j)
1.  IF(i>j) {
2.       Return n+1
3.      } ELSE {
4.       m=(i+j)/2
5.       IF((a[m]==x)&&(m==i||a[m]≠a[m-1])){
6.       Return m
7.      } ELSE {
8.         IF((a[m]>=x){
9.            Return CERCAPRIMAOCCORRENZA(A,x,i,m-1)
10.           } ELSE {
11.            Return CERCAPRIMAOCCORRENZA(A,x,m+1,j)
     }
    }
```

L'algoritmo:

```
CERCAPRIMAOCCORRENZA(A,x,i,j)
1.  IF(i>j) {
2.       Return n+1
3.      } ELSE {
4.       m=(i+j)/2
5.       IF((a[m]==x)&&(m==i||a[m]≠a[m-1])){
6.       Return m
7.      } ELSE {
8.         IF((a[m]>=x){
9.            Return CERCAPRIMAOCCORRENZA(A,x,i,m-1)
10.           } ELSE {
11.            Return CERCAPRIMAOCCORRENZA(A,x,m+1,j)
     }
    }
```

Complessità: $T(n) = T(n/2) + d$

L'algoritmo:

```
CERCAPRIMAOCCORRENZA(A,x,i,j)
1.  IF(i>j) {
2.       Return n+1
3.      } ELSE {
4.       m=(i+j)/2
5.       IF((a[m]==x)&&(m==i||a[m]≠a[m-1])){
6.       Return m
7.      } ELSE {
8.         IF((a[m]>=x){
9.            Return CERCAPRIMAOCCORRENZA(A,x,i,m-1)
10.           } ELSE {
11.            Return CERCAPRIMAOCCORRENZA(A,x,m+1,j)
     }
    }
```

Complessità: $T(n) = T(n/2) + d \Rightarrow T(n) = \Theta(\log n)$.