# 5. Divide And Conquer I

- Mergesort e Relazioni di ricorrenza
- Esempi di progettazione D&I
- Ricerca in una matrice
- Moltiplicazione di interi
- Contare inversioni

## Divide-et-Impera (Divide and conquer)

**Definizione attribuita a Giulio Cesare**

Divide-et-Impera.

- suddividi il problema in sottoproblemi
  - problemi dello stesso tipo su un'istanza più piccola
- risolvi ogni sottoproblema ricorsivamente
  - applicando la stessa strategia risolutiva
- dalle soluzioni ai sottoproblemi costruisci una soluzione al problema di partenza.

Esempio tipico: algoritmo mergesort

- Divide un'istanza di taglia $n$ in due sottoproblemi di taglia $n/2$
- risolve i due problemi ricorsivamente.
- Combina le due soluzioni in una soluzione globale in tempo lineare.

Consequence.

- Brute force: $\Theta(n^2)$.
- Divide-and-conquer: $\Theta(n \log n)$.

Tipicamente: miglioriamo la complessità di un algoritmo già polinomiale

Altro esempio: **maxsubarray**



## Problema dell'ordinamento

Problema. Data una lista di $n$ elementi di uno spazio totalmente ordinato, mettili in ordine crescente.

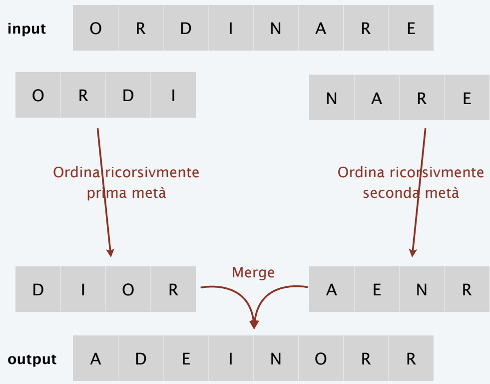
Esempi:

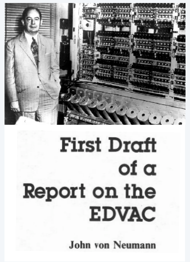
- lista di numeri:
  - 5, 4, 10, 20, 15, 12 ⇨ 4, 5, 10, 12, 15, 20
- lettere
  - A, C, Z, D, G, H, U ⇨ A, C, D, G, H, U, Z
- parole
  - casa, abaco, algortmo, telefono, musica
    ⇨ abaco, algoritmo, casa, telefono



## Mergesort

- Se la lista è un solo elemento, è ordinata
- altrimenti
  - Ordina ricorsivamente la prima metà dell'array.
  - Ordina ricorsivamente la seconda metà dell'array.
  - "Fondi" (merge) le due metà in un tutto ordinato.



## Operazione di Fusione

Scopo. fonde due liste ordinate $A$ e $B$ in una lista ordinata $C$.

- Scorri $A$ e $B$ da sinistra a destra.
- Confronta $a_i$ e $b_j$.
- se $a_i \leq b_j$, aggiungi $a_i$ a $C$ ($\leq$ ogni elemento rimanente in $B$).
- se $a_i > b_j$, aggiungi $b_j$ a $C$ ($<$ ogni elemento rimanente in $A$).

lista ordinata A

| 3 | 7 | 10 | $a_i$ | 18 |
|---|---|---|---|---|

lista ordinata B

| 2 | 11 | $b_j$ | 17 | 23 |
|---|---|---|---|---|

Running time: O(|A|+|B|)

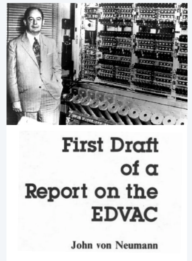
fusione per ottenere la lista ordinata C

| 2 | 3 | 7 | 10 | 11 | | | | | |
|---|---|---|---|---|---|---|---|---|---|



## Mergesort

- Se la lista è un solo elemento, è ordinata
- altrimenti
  - Ordina ricorsivamente la prima metà dell'array.
  - Ordina ricorsivamente la seconda metà dell'array.
  - "Fondi" (merge) le due metà in un tutto ordinato.




input: O R D I N A R E

O R D I — N A R E

Divide

Ordina ricorsivmente prima metà

Ordina ricorsivmente seconda metà

D I O R — A E N R

Merge O(n)

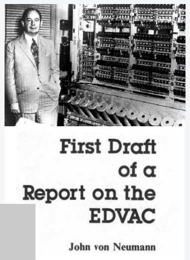
Impera

output: A D E I N O R R



## Mergesort

- Se la lista è un solo elemento, è ordinata
- altrimenti
  - Ordina ricorsivamente la prima metà dell'array.
  - Ordina ricorsivamente la seconda metà dell'array.
  - "Fondi" (merge) le due metà in un tutto ordinato.




input: O R D I N A R E

```
Merge-sort(A[1],A[2],...,A[n])
{
  if(n = 1)
      return // ritorna la lista così come è
  else
      Merge-sort(A[1],...,A[n/2])
      Merge-sort(A[n/2+1],...,A[n])
      Merge(A[1...n/2], A[n/2+1...n])
}
```

output: A D E I N O R R



## Mergesort

- Se la lista è un solo elemento, è ordinata
- altrimenti
  - Ordina ricorsivamente la prima metà dell'array.
  - Ordina ricorsivamente la seconda metà dell'array.
  - "Fondi" (merge) le due metà in un tutto ordinato.


Come analizziamo un tale algoritmo?

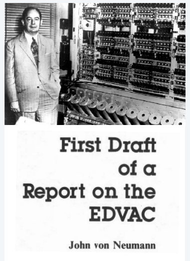
input: O R D I N A R E

```
Merge-sort(A[1],A[2],...,A[n])
{
  if(n = 1)
      return // ritorna la lista così come è         } ← O(1)
  else
      Merge-sort(A[1],...,A[n/2])      ← ???
      Merge-sort(A[n/2+1],...,A[n])    ←
      Merge(A[1...n/2], A[n/2+1...n])  ← O(n)
}
```

output: A D E I N O R R

## Mergesort

- Se la lista è un solo elemento, è ordinata
- altrimenti
  - Ordina ricorsivamente la prima metà dell'array.
  - Ordina ricorsivamente la seconda metà dell'array.
  - "Fondi" (merge) le due metà in un tutto ordinato.




input O R D I N A R E

O R D I    N A R E

O R    D I    N A    R E

O    R    D    I    N    A    R    E

O R    D I    A N    E R

D I O R    A E N R

output A D E I N O R R



## Relazioni di ricorrenza

Denotiamo con $T(n)$ il massimo numero di confronti su un input di taglia $n$

```
Merge-sort(A[1],A[2],...,A[n])
{
  if(n = 1)
      return // ritorna la lista così come è    } 0
  else
      Merge-sort(A[1],...,A[n/2])      T(n/2)
      Merge-sort(A[n/2+1],...,A[n])    T(n/2)
      Merge(A[1...n/2], A[n/2+1...n])   n
}
```

$$T(n) = \begin{cases} 0 & \text{se } n = 1 \\ T(n/2) + T(n/2) + n & \text{altrimenti} \end{cases}$$



## Relazioni di ricorrenza

Def. (per il nostro uso) un'espressione che definisce la quantità di risorse usate da un algoritmo su un'istanza di una data taglia in funzione della stessa quantità per specifiche istanze di taglia inferiore.

Es. $T(n)$ = max #confronti fatti da mergesort su una lista di taglia $\leq n$.

Ricorrenza di mergesort.

$$T(n) = \begin{cases} 0 & \text{se } n = 1 \\ T(n/2) + T(n/2) + n & \text{altrimenti} \end{cases}$$

Soluzione. $T(n)$ è $O(n \log_2 n)$.

Come valutiamo una ricorrenza?

- vediamo ora vari esempi di ricorrenze e di come risolverle



## Relazioni di ricorrenza

$$T(n) = \begin{cases} b & \text{se } n = n_0 \\ aT(f(n)) + g(n) & \text{altrimenti} \end{cases}$$

Parametri della ricorrenza

- $n_0$ = taglia del caso base
- $b$ tempo di esecuzione per il caso base (costante)
- $a$ numero di volte che si effettua una chiamata ricorsiva
- $f(n)$ taglia delle istanze risolte nelle chiamate ricorsive
- $g(n)$ il tempo di calcolo non incluso nelle chiamate ricorsive

## Relazioni di ricorrenza - Esempio 1

```
Pluto(n)
{
  if(n < 3)
      fai qualcosa in tempo costante // caso base
  else
      Pluto(n/3)
      for i = 1 to n
          fai qualcosa in tempo costante
      Pluto(n/3)
}
```

$$T(n) = \begin{cases} b & \text{se } n < 3 \\ 2T(n/3) + d\,n & \text{altrimenti} \end{cases}$$



## Relazioni di ricorrenza - Esempio 2

```
Topolino(A[1],...,A[n])
{
  if(n < 11)
      fai qualcosa in tempo costante // caso base
  else
      for i = 0 to 4
        Topolino(A[i+1],...,A[i+n/2])
      for i = 1 to n
        for j = 1 to n
          qualcosa in tempo costante
}
```

$$T(n) = \begin{cases} b & \text{se } n < 11 \\ 5T(n/2) + d\,n^2 & \text{altrimenti} \end{cases}$$



## Relazioni di ricorrenza - Esempio 3

```
Minnie(n)
{
  if(n = 1)
      print(ciao) // caso base
  else
      for i = 1 to n-1
          Minnie(n-i)
          print(Basta!)

}
```

$$T(n) = \begin{cases} b & \text{se } n = 1 \\ T(1) + T(2) + \ldots + T(n-1) + b\,n & \text{altrimenti} \end{cases}$$



## Relazioni di ricorrenza - Esempio 4

```
Min-max(A[1],...,A[n])
{
  if(n ≤ 2)
      return(min(A[1],A[n]), max(A[1],A[n]) // caso base
  else
      (A,B) ← Min-max(A[1],...,A[n/3]
      (C,D) ← Min-max(A[n/3+1],...,A[n]
      m ← min(A,C)
      M ← max(B,D)
      return(m, M)
}
```

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

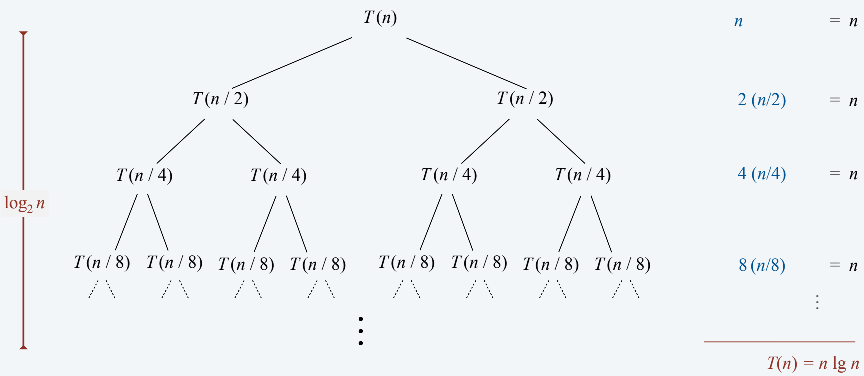
## Risoluzione di ricorrenze: Albero di ricorsione

Proposizione. Se $T(n)$ soddisfa la seguente ricorrenza, allora $T(n) = n \log_2 n$.

assumiamo n potenza di 2

$$T(n) = \begin{cases} 0 & \text{se } n = 1 \\ 2\,T(n/2) + n & \text{altrimenti} \end{cases}$$

Dim 1.






## Risoluzione per sostituzione iterativa

Proposizione. Se $T(n)$ soddisfa la seguente ricorrenza, allora $T(n) = n \log_2 n$.

assumiamo n potenza di 2

$$T(n) = \begin{cases} 0 & \text{se } n = 1 \\ 2\,T(n/2) + n & \text{altrimenti} \end{cases}$$

Dim 2.

- Sostituiamo l'espressione della ricorrenza

$$\begin{aligned} T(n) &= n + 2\,T(n/2) \\ &= n + 2(n/2 + 2T(n/4)) = n + n + 4T(n/4) \\ &= 2n + 4(n/4 + 2T(n/8)) = 2n + n + 8T(n/8) \\ &= 3n + 8T(n/8) \\ &= \ldots = i\,n + 2^i\,T(n/2^i) \\ &= \ldots = (\log(n))\,n + 2^{\log(n)}\,\underbrace{T(n/2^{\log(n)})}_{=T(n/n) = T(1) = 0} = n \log n\,. \quad \blacksquare \end{aligned}$$



## Un criterio generale

Teorema. Per $a, b, c, d, k$ costanti, la ricorrenza

$$T(n) = \begin{cases} b & \text{se } n \le n_0 \\ a\,T(n/c) + d\,n^k & \text{altrimenti} \end{cases}$$

ha soluzione

$$T(n) = \begin{cases} O(n^k) & \text{se } a < c^k \\ O(n^k \log n) & \text{se } a = c^k \\ O(n^{\log_c a}) & \text{se } a > c^k \end{cases}$$



## Un criterio generale - Esempio 1: ricorrenza di Pluto(n)

Teorema. Per $a, b, c, d, k$ costanti, la ricorrenza

$$T(n) = \begin{cases} b & \text{se } n \le n_0 \\ a\,T(n/c) + d\,n^k & \text{altrimenti} \end{cases}$$

ha soluzione

$$T(n) = \begin{cases} O(n^k) & \text{se } a < c^k \\ O(n^k \log n) & \text{se } a = c^k \\ O(n^{\log_c a}) & \text{se } a > c^k \end{cases}$$

T(n) : running time di Pluto(n)

$$T(n) = \begin{cases} b & \text{se } n < 3 \\ 2T(n/3) + d\,n & \text{altrimenti} \end{cases}$$

a = 2, c = 3, k = 1

a = 2

$c^k = 3^1 = 3$

$a < c^k$ quindi $T(n) = O(n^k) = O(n)$

## Un criterio generale - Esempio 2: ricorrenza di Topolino(A[1]...A[n])

Teorema. Per $a, b, c, d, k$ costanti, la ricorrenza

$$T(n) = \begin{cases} b & \text{se } n \le n_0 \\ a\,T(n/c) + d\,n^k & \text{altrimenti} \end{cases}$$

ha soluzione

$$T(n) = \begin{cases} O(n^k) & \text{se } a < c^k \\ O(n^k \log n) & \text{se } a = c^k \\ O(n^{\log_c a}) & \text{se } a > c^k \end{cases}$$

T(n) : running time di Topolino(A[1]...A[n])

$$T(n) = \begin{cases} b & \text{se } n < 3 \\ 5T(n/2) + d\,n^2 & \text{altrimenti} \end{cases}$$

a = 5, c = 2, k = 2

$a = 5$
$c^k = 2^2 = 4$
$a > c^k$ quindi $T(n) = O(n^{\log_2 5}) = O(n^{2.32})$



## Un criterio generale - Esempio 3: ricorrenza di mergesort

Teorema. Per $a, b, c, d, k$ costanti, la ricorrenza

$$T(n) = \begin{cases} b & \text{se } n \le n_0 \\ a\,T(n/c) + d\,n^k & \text{altrimenti} \end{cases}$$

ha soluzione

$$T(n) = \begin{cases} O(n^k) & \text{se } a < c^k \\ O(n^k \log n) & \text{se } a = c^k \\ O(n^{\log_c a}) & \text{se } a > c^k \end{cases}$$

T(n) : running time di mergesort

$$T(n) = \begin{cases} 0 & \text{se } n < 2 \\ 2T(n/2) + n & \text{altrimenti} \end{cases}$$

a = 2, c = 2, k = 1

$a = 2$
$c^k = 2^1 = 2$
$a = c^k$ quindi $T(n) = O(n^k \log n)$
$= O(n\log(n))$



## DIVIDE ET IMPERA

- Mergesort e Relazioni di ricorrenza
- Esempi di progettazione D&I
- Ricerca in una matrice
- Moltiplicazione di interi
- Contare inversioni

## Ricerca in una matrice

Input:
- una matrice A quadrata di n*n di interi (assumiamo n = potenza di 2)
  - le righe sono ordinate in senso crescente da sinistra a destra
  - le colonne sono ordinate in senso crescente dall'alto verso il basso
- un intero ***x***

Output:
- la posizione in A di ***x*** se esiste
- NO altrimenti

Esempio. n=8, x= 72

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 9 | 18 | 30 | 39 | 60 | 84 | 93 |
| 15 | 16 | 24 | 33 | 57 | 65 | 87 | 120 |
| 19 | 21 | 40 | 45 | 58 | 69 | 90 | 123 |
| 20 | 28 | 46 | 53 | 63 | 81 | 92 | 125 |
| 29 | 34 | 52 | 75 | 78 | 89 | 99 | 126 |
| 42 | 56 | 62 | 85 | 88 | 94 | 101 | 128 |
| 44 | 64 | 72 | 86 | 91 | 97 | 102 | 150 |
| 61 | 66 | 96 | 100 | 103 | 105 | 111 | 171 |

Ricerca esaustiva. Prova tutte le caselle: $\Theta(n^2)$

e con Divide et Impera?

## Ricerca in una matrice

Input:
- una matrice A quadrata di n*n di interi (assumiamo n = potenza di 2)
  - le righe sono ordinate in senso crescente da sinistra a destra
  - le colonne sono ordinate in senso crescente dall'alto verso il basso
- un intero ***x***

Output:
- la posizione in A di ***x*** se esiste
- NO altrimenti

Esempio. n=8, x= 72

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 9 | 18 | 30 | 39 | 60 | 84 | 93 |
| 15 | 16 | 24 | 33 | 57 | 65 | 87 | 120 |
| 19 | 21 | 40 | 45 | 58 | 69 | 90 | 123 |
| 20 | 28 | 46 | 53 | 63 | 81 | 92 | 125 |
| 29 | 34 | 52 | 75 | 78 | 89 | 99 | 126 |
| 42 | 56 | 62 | 85 | 88 | 94 | 101 | 128 |
| 44 | 64 | 72 | 86 | 91 | 97 | 102 | 150 |
| 61 | 66 | 96 | 100 | 103 | 105 | 111 | 171 |



## Ricerca in una matrice

Input:
- una matrice A quadrata di n*n di interi (assumiamo n = potenza di 2)
  - le righe sono ordinate in senso crescente da sinistra a destra
  - le colonne sono ordinate in senso crescente dall'alto verso il basso
- un intero ***x***

Output:
- la posizione in A di ***x*** se esiste
- NO altrimenti

Esempio. n=8, x= 72

Confronto x con A[6,6]= 94
- x < A[6,6] quindi....

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 9 | 18 | 30 | 39 | 60 | 84 | 93 |
| 15 | 16 | 24 | 33 | 57 | 65 | 87 | 120 |
| 19 | 21 | 40 | 45 | 58 | 69 | 90 | 123 |
| 20 | 28 | 46 | 53 | 63 | 81 | 92 | 125 |
| 29 | 34 | 52 | 75 | 78 | 89 | 99 | 126 |
| 42 | 56 | 62 | 85 | 88 | 94 | 101 | 128 |
| 44 | 64 | 72 | 86 | 91 | 97 | 102 | 150 |
| 61 | 66 | 96 | 100 | 103 | 105 | 111 | 171 |



## Ricerca in una matrice

Input:
- una matrice A quadrata di n*n di interi (assumiamo n = potenza di 2)
  - le righe sono ordinate in senso crescente da sinistra a destra
  - le colonne sono ordinate in senso crescente dall'alto verso il basso
- un intero ***x***

Output:
- la posizione in A di ***x*** se esiste
- NO altrimenti

Esempio. n=8, x= 72

Confronto x con A[6,6]= 94
- x < A[6,6] quindi....

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 9 | 18 | 30 | 39 | 60 | 84 | 93 |
| 15 | 16 | 24 | 33 | 57 | 65 | 87 | 120 |
| 19 | 21 | 40 | 45 | 58 | 69 | 90 | 123 |
| 20 | 28 | 46 | 53 | 63 | 81 | 92 | 125 |
| 29 | 34 | 52 | 75 | 78 | 89 | 99 | 126 |
| 42 | 56 | 62 | 85 | 88 | 94 | 101 | 128 |
| 44 | 64 | 72 | 86 | 91 | 97 | 102 | 150 |
| 61 | 66 | 96 | 100 | 103 | 105 | 111 | 171 |



## Ricerca in una matrice

Input:
- una matrice A quadrata di n*n di interi (assumiamo n = potenza di 2)
  - le righe sono ordinate in senso crescente da sinistra a destra
  - le colonne sono ordinate in senso crescente dall'alto verso il basso
- un intero ***x***

Output:
- la posizione in A di ***x*** se esiste
- NO altrimenti

Esempio. n=8, x= 72

Confronto x con A[6,6]= 94
- x < A[6,6] quindi....

Confronto x con A[3,3]= 40
- x > A[3,3] quindi....

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 9 | 18 | 30 | 39 | 60 | 84 | 93 |
| 15 | 16 | 24 | 33 | 57 | 65 | 87 | 120 |
| 19 | 21 | 40 | 45 | 58 | 69 | 90 | 123 |
| 20 | 28 | 46 | 53 | 63 | 81 | 92 | 125 |
| 29 | 34 | 52 | 75 | 78 | 89 | 99 | 126 |
| 42 | 56 | 62 | 85 | 88 | 94 | 101 | 128 |
| 44 | 64 | 72 | 86 | 91 | 97 | 102 | 150 |
| 61 | 66 | 96 | 100 | 103 | 105 | 111 | 171 |

## Ricerca in una matrice

Input:

- una matrice A quadrata di n*n di interi (assumiamo n = potenza di 2)
  - le righe sono ordinate in senso crescente da sinistra a destra
  - le colonne sono ordinate in senso crescente dall'alto verso il basso
- un intero ***x***

Output:

- la posizione in A di ***x*** se esiste
- NO altrimenti

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 9 | 18 | 30 | 39 | 60 | 84 | 93 |
| 15 | 16 | 24 | 33 | 57 | 65 | 87 | 120 |
| 19 | 21 | 40 | 45 | 58 | 69 | 90 | 123 |
| 20 | 28 | 46 | 53 | 63 | 81 | 92 | 125 |
| 29 | 34 | 52 | 75 | 78 | 89 | 99 | 126 |
| 42 | 56 | 62 | 85 | 88 | 94 | 101 | 128 |
| 44 | 64 | **72** | 86 | 91 | 97 | 102 | 150 |
| 61 | 66 | 96 | 100 | 103 | 105 | 111 | 171 |

Esempio. n=8, x= 72

Confronto x con A[6,6]= 94

- x < A[6,6] quindi....

Confronto x con A[3,3]= 40

- x > A[3,3] quindi....



## Ricerca in una matrice

Input:

- una matrice A quadrata di n*n di interi (assumiamo n = potenza di 2)
  - le righe sono ordinate in senso crescente da sinistra a destra
  - le colonne sono ordinate in senso crescente dall'alto verso il basso
- un intero ***x***

Output:

- la posizione in A di ***x*** se esiste
- NO altrimenti

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 9 | 18 | 30 | 39 | 60 | 84 | 93 |
| 15 | 16 | 24 | 33 | 57 | 65 | 87 | 120 |
| 19 | 21 | 40 | 45 | 58 | 69 | 90 | 123 |
| 20 | 28 | 46 | 53 | 63 | 81 | 92 | 125 |
| 29 | 34 | 52 | 75 | 78 | 89 | 99 | 126 |
| 42 | 56 | 62 | 85 | 88 | 94 | 101 | 128 |
| 44 | 64 | **72** | 86 | 91 | 97 | 102 | 150 |
| 61 | 66 | 96 | 100 | 103 | 105 | 111 | 171 |

Esempio. n=8, x= 72

Confronto x con A[6,6]= 94

- x < A[6,6] quindi....

Confronto x con A[3,3]= 40

- x > A[3,3] quindi....



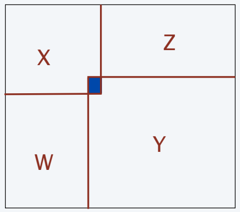
## Ricerca in una matrice

Input:

- una matrice A quadrata di n*n di interi (assumiamo n = potenza di 2)
  - le righe sono ordinate in senso crescente da sinistra a destra
  - le colonne sono ordinate in senso crescente dall'alto verso il basso
- un intero ***x***

Output:

- la posizione in A di ***x*** se esiste
- NO altrimenti

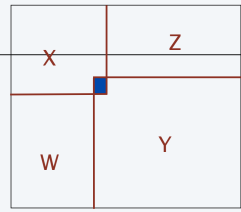



In generale. un confronto mi permette di dividere il problema in 4 e di eliminare una di queste 4 parti

- se x < ■ allora x non può essere in Y
- se x > ■ allora x non può essere in X



## Ricerca in una matrice

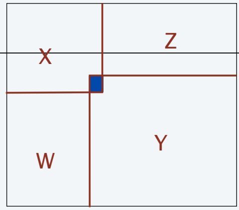



In generale. un confronto mi permette di dividere il problema in 4 e di eliminare una di queste 4 parti

- se x < ■
  - allora x non può essere in Y
- se x > ■
  - allora x non può essere in X

```
Cerca(A[r_i...r_f; c_i...c_f], x)
  if(r_i=r_f e c_i = c_f) //la dimensione è 1x1
      if(A[r_i,c_i] = x) return (r_i,c_i)
  else
        q_r = (r_i + r_f)/2; q_c = (c_i + c_f)/2;
        Z ← A[r_i...q_r-1, q_c+1...c_f]
        W ← A[q_r+1...r_f, c_i...q_c-1]
        X ← A[r_i...q_r, c_i...q_c]
        Y ← A[q_r...r_f, q_c...c_f]
        Cerca(Z, x); Cerca(W, x)
        if x < A[q_r,q_c]
          Cerca(X, x);
        else
          Cerca(Y, x);
```

## Ricerca in una matrice




In generale. un confronto mi permette di dividere il problema in 4 e di eliminare una di queste 4 parti

- se x < ■
  - allora x non può essere in Y
- se x > ■
  - allora x non può essere in X

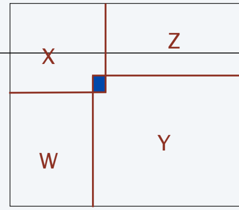
```
Cerca(A[ri...rf; ci...cf], x)
  if(ri=rf e ci = cf) //la dimensione è 1x1
      if(A[ri,ci] = x) return (ri,ci)
  else
        qr = (ri + rf)/2; qc = (ci + cf)/2;
        Z ← A[ri...qr-1, qc+1...cf]
        W ← A[qr+1...rf, ci...qc-1]
        X ← A[ri...qr, ci...qc]
        Y ← A[qr...rf, qc...cf]
        Cerca(Z, x); Cerca(W, x)
        if x < A[qr,qc]
          Cerca(X, x);
        else
          Cerca(Y, x);
```

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

$$T(n) = O(n^{\log 3}) = O(n^{1.58})$$

possiamo far meglio?

## Ricerca in una matrice




In generale. un confronto mi permette di dividere il problema in 4 e di eliminare una di queste 4 parti

- se x < ■
  - allora x non può essere in Y
- se x > ■
  - allora x non può essere in X

```
Cerca(A[ri...rf; ci...cf], x)
  if(ri=rf e ci = cf) //la dimensione è 1x1
      if(A[ri,ci] = x) return (ri,ci)
  else
        qr = (ri + rf)/2; qc = (ci + cf)/2;
        Z ← A[ri...qr-1, qc+1...cf]
        W ← A[qr+1...rf, ci...qc-1]
        X ← A[ri...qr, ci...qc]
        Y ← A[qr...rf, qc...cf]
        Cerca(Z, x); Cerca(W, x)
        if x < A[qr,qc]
          Cerca(X, x);
        else
          Cerca(Y, x);
```

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

$$T(n) = O(n^{\log 3}) = O(n^{1.58})$$

possiamo far meglio?

## Ricerca in una matrice

In generale, un confronto mi permette di dividere il problema in 4 e di eliminare una di queste 4 parti

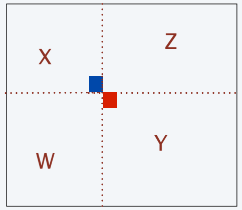
- se x < ■
  - allora x non può essere in Y
- se x > ■
  - allora x non può essere in X

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

$$T(n) = O(n^{\log 3}) = O(n^{1.58})$$

possiamo far meglio?

Supponiamo di fare una serie di confronti tali che ci permettono di individuare due caselle adiacenti come nella fiugura




Se scopro che

■ < x < ■

cosa posso concludere ?



## Ricerca in una matrice

In generale, un confronto mi permette di dividere il problema in 4 e di eliminare una di queste 4 parti

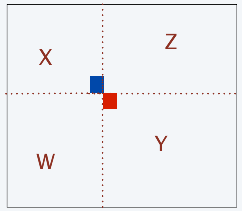
- se x < ■
  - allora x non può essere in Y
- se x > ■
  - allora x non può essere in X

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

$$T(n) = O(n^{\log 3}) = O(n^{1.58})$$

possiamo far meglio?

Supponiamo di fare una serie di confronti tali che ci permettono di individuare due caselle adiacenti come nella fiugura




- x > ■ allora x non può essere in X
- x < ■ allora x non può essere in Y
  - elimino due sottomatrici
  - il caso “peggiore” è se ■ ■ sono al centro
  - ■ ■ posso trovarli con ricerca binaria sulla diagonale : O(log n) confronti

$$T(n) \leq \begin{cases} 1 & \text{se } n < 1 \\ 2T(n/2) + \log n & \text{altrimenti} \end{cases} = O(n)$$

## Divide et Impera



## Moltiplicazione di interi

Moltiplicazione. Dati due interi ad $n$-cifre $a$ e $b$, calcola $a \times b$.
L'algoritmo delle scuole elementari. $\Theta(n^2)$ moltiplicazioni elementari.

| | | | |
|---|---|---|---|
| | 3 | 6 | × |
| | 2 | 3 | |
| 1 | 0 | 8 | |
| 7 | 2 | | |
| 8 | 2 | 8 | |

Domanda. Possiamo usare un numero inferiore di moltiplicazioni elementari?



## Moltiplicazione di interi (binario)

Moltiplicazione. Dati due interi ad $n$-cifre $a$ e $b$, calcola $a \times b$.
L'algoritmo delle scuole elementari. $\Theta(n^2)$ moltiplicazioni elementari.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| | | | | | | | | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| | | | | | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| | | | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | |
| | | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | | |
| | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | | | |
| | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | | | | |
| | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | | | | | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | | | | |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

In Binario. Moltiplicazione elementare = Bit operation (AND).
Domanda. Possiamo usare meno di $\Theta(n^2)$ bit-operations?



## Divide et Impera - dividi in sottoproblemi, risolvi, componi

Primo tentativo: due interi ad $n$-cifre $x$ e $y$:

- Dividi $x$ e $y$ in due parti: n/2 cifre più significative e n/2 cifre meno significative.
- Moltiplica i quattro interi a $n/2$-cifre, ricorsivamente.
- ricomponi per ottenere il risultato.

$$x = x_n\, x_{n-1} \ldots x_{n/2+1} \mid x_{n/2} \ldots x_1 \qquad y = y_n\, y_{n-1} \ldots y_{n/2+1} \mid y_{n/2} \ldots y_1$$

$$a = x_n\, x_{n-1} \ldots x_{n/2+1} \qquad b = x_{n/2} \ldots x_1 \qquad c = y_n\, y_{n-1} \ldots y_{n/2+1} \qquad d = y_{n/2} \ldots y_1$$

Calcola quattro prodotti: $\underset{1}{ac}$ ; $\underset{2}{bc}$; $\underset{3}{ad}$; $\underset{4}{bd}$

$$x*y = (10^{n/2}\, a + b)\,(10^{n/2}\, c + d) \;=\; 10^n \underset{1}{ac} \;+\; 10^{n/2}\,(\underset{2}{bc} + \underset{3}{ad}) \;+\; \underset{4}{bd}$$

Es. $x = \underbrace{12}_{a}\underbrace{46}_{b}$ $\quad y = \underbrace{27}_{c}\underbrace{65}_{d}$

$$x*y = (12*100+46) * (27*100+65)$$
$$= (\underbrace{12*27}_{a\,c}*10000)+(\underbrace{12*65}_{a\,d}+\underbrace{46*27}_{b\,c})*100+\underbrace{46*65}_{b\,d}$$

## Divide-and-conquer multiplication

MULTIPLY($x = x_n \ldots x_1$, $y = y_n \ldots y_1$, $n$)

IF ($n = 1$)
  RETURN $x \times y$.
ELSE
  $m \leftarrow n / 2$.
  $a \leftarrow x_n \ldots x_{n/2+1}$;  $b \leftarrow x_{n/2} \ldots x_1$.
  $c \leftarrow y_n \ldots y_{n/2+1}$;  $d \leftarrow y_{n/2} \ldots y_1$.
  $e \leftarrow$ MULTIPLY($a$, $c$, $m$).
  $f \leftarrow$ MULTIPLY($b$, $d$, $m$).
  $g \leftarrow$ MULTIPLY($b$, $c$, $m$).
  $h \leftarrow$ MULTIPLY($a$, $d$, $m$).
  RETURN $10^{2m}\, e + 10^m\, (g + h) + f$.

Esempio: x = 1246  y = 2765  n = 4

m = 2
a = 12  b = 46
c = 27  d = 65

x*y = 1246 * 2765
= (12*100 + 46) * (27*100+65)
= 12*27*10⁴+ (12*65+46*27)*10²+ 46*65
= a*c *10⁴ + (a*d + b*c)*10² + b*d

$$T(n) = \underbrace{4T(n/2)}_{\text{chiamate ricorsive}} + \underbrace{\Theta(n)}_{\text{scomposizione e ricomposizione}} \Rightarrow T(n) = \Theta(n^2)$$

Ops! Non abbiamo guadagnato nulla!



## Divide et Impera - dividi in sottoproblemi, risolvi, componi

$x = x_n\, x_{n-1} \ldots x_{n/2+1} \mid x_{n/2} \ldots x_1$    $y = y_n\, y_{n-1} \ldots y_{n/2+1} \mid y_{n/2} \ldots y_1$

$a = x_n\, x_{n-1} \ldots x_{n/2+1}$   $b = x_{n/2} \ldots x_1$   $c = y_n\, y_{n-1} \ldots y_{n/2+1}$   $d = y_{n/2} \ldots y_1$

Non è necessario calcolare quattro prodotti: $ac$ ; $bc$; $ad$; $bd$
1 2 3 4

x*y $= (10^{n/2}\, a + b)\, (10^{n/2}\, c + d) = 10^n\, ac + 10^{n/2}\, (bc + ad) + bd$
1 2 3 4

questo termine

$(bc + ad) = (b+a)(c+d) - ac - bd$



## Divide et Impera - dividi in sottoproblemi, risolvi, componi

$x = x_n\, x_{n-1} \ldots x_{n/2+1} \mid x_{n/2} \ldots x_1$    $y = y_n\, y_{n-1} \ldots y_{n/2+1} \mid y_{n/2} \ldots y_1$

$a = x_n\, x_{n-1} \ldots x_{n/2+1}$   $b = x_{n/2} \ldots x_1$   $c = y_n\, y_{n-1} \ldots y_{n/2+1}$   $d = y_{n/2} \ldots y_1$

Non è necessario calcolare quattro prodotti: $ac$ ; $bc$; $ad$; $bd$
1 2 3 4

x*y $= (10^{n/2}\, a + b)\, (10^{n/2}\, c + d) = 10^n\, ac + 10^{n/2}\, (bc + ad) + bd$
1 2 3 4

questo termine

$(bc + ad) = (b+a)(c+d) - ac - bd$

x*y $= (10^{n/2}\, a + b)\, (10^{n/2}\, c + d) = 10^n\, ac + 10^{n/2}\, ((b+a)(c+d) - ac - bd) + bd$
1 3 2



## Algoritmo di Karatsuba

KARATSUBA-MULTIPLY($x = x_n \ldots x_1$, $y = y_n \ldots y_1$, $n$)

IF ($n = 1$)
  RETURN $x \times y$.
ELSE
  $m \leftarrow n / 2$.
  $a \leftarrow x_n \ldots x_{n/2+1}$;  $b \leftarrow x_{n/2} \ldots x_1$.
  $c \leftarrow y_n \ldots y_{n/2+1}$;  $d \leftarrow y_{n/2} \ldots y_1$.
  $e \leftarrow$ KARATSUBA-MULTIPLY($a$, $c$, $m$).
  $f \leftarrow$ KARATSUBA-MULTIPLY($b$, $d$, $m$).
  $g \leftarrow$ KARATSUBA-MULTIPLY($a + b$, $c + d$, $m$).
  RETURN $10^n\, e + 10^m\, (g - e - f) + f$.

Proposizione. L'algoritmo di Karatsuba richiede $O(n^{1.585})$ operazioni per moltiplicare due interi a $n$-cifre.
Dim. Applichiamo ancora il caso 3 del criterio generale

$$T(n) = 3\, T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n^{\lg 3}) = O(n^{1.585}).$$

## Algoritmo di Karatsuba

Esempio: x = 1246 y = 2785 n = 4

m = 2
a = 12 b = 46
c = 27 d = 65

e = a*c = 12*46 f = b*d = 27*65
g = (a+b)*(c+d) = 58*92

x*y = 1246 * 2765
= (12*100 + 46) * (27*100+85)
= 12*27*10^4+ (12*85+46*27)*10^2+ 46*85
= a*c *10^4 + (g - e - f)*10^2 + b*d

KARATSUBA-MULTIPLY($x = x_n \ldots x_1$, $y = y_n \ldots y_1$, $n$)

IF ($n = 1$)
  RETURN $x \times y$.
ELSE
  $m \leftarrow n / 2$.
  $a \leftarrow x_n \ldots x_{n/2+1}$; $b \leftarrow x_{n/2} \ldots x_1$.
  $c \leftarrow y_n \ldots y_{n/2+1}$; $d \leftarrow y_{n/2} \ldots y_1$.
  $e \leftarrow$ KARATSUBA-MULTIPLY($a, c, m$).
  $f \leftarrow$ KARATSUBA-MULTIPLY($b, d, m$).
  $g \leftarrow$ KARATSUBA-MULTIPLY($a + b, c + d, m$).
  RETURN $10^n e + 10^m (g - e - f) + f$.

Proposizione. L'algoritmo di Karatsuba richiede $O(n^{1.585})$ operazioni per moltiplicare due interi a $n$-cifre.
Dim. Applichiamo ancora il caso 3 del criterio generale

$$T(n) = 3\,T(n/2) + \Theta(n) \quad \Rightarrow \quad T(n) = \Theta(n^{\lg 3}) = O(n^{1.585}).$$