Corso di Informatica A.A. 2009-2010

Lezione 7

Algoritmi e loro proprietà

- Efficienza rispetto al tempo
- Efficienza rispetto allo spazio

Efficienza degli algoritmi

Una volta determinato un algoritmo corretto per la soluzione di un problema, occorre preoccuparsi della sua efficienza, ovvero di come esso gestisca le risorse

tempo (numero di operazioni necessarie alla soluzione) e

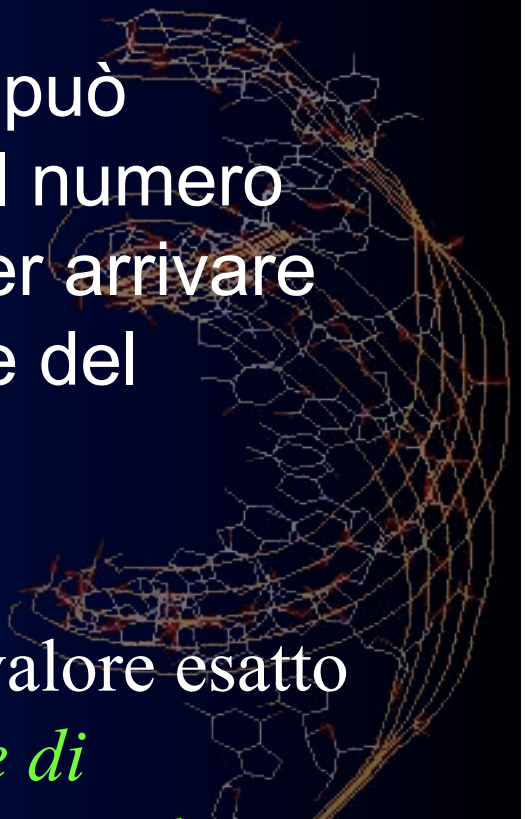
spazio (memoria occorrente per trovare la soluzione)

che saranno in quantità finita per qualunque elaboratore reale.

. . . . proseguendo

Un criterio per la valutazione dell'efficienza di un algoritmo e' di grande utilita' in quanto dobbiamo assicurarci la possibilita' di operare confronti tra diversi algoritmi, *indipendentemente* dal *linguaggio* con cui saranno implementati e dalla "*potenza*" della macchina su cui saranno eseguiti.

Valutare l'efficienza rispetto al tempo



L'efficienza rispetto alla risorsa tempo può essere valutata in generale contando il numero di operazioni richieste dall'algoritmo per arrivare alla soluzione del problema in funzione del numero *n* dei dati in ingresso.

In questa valutazione non è importante il valore esatto quanto la *dipendenza funzionale* e *l'ordine di grandezza*. Si parlerà allora di algoritmi o *complessita'* $O(n)$, $O(n^2)$, $O(\log n)$, $O(2^n)$ etc a seconda degli andamenti.

L’algoritmo di ricerca sequenziale...

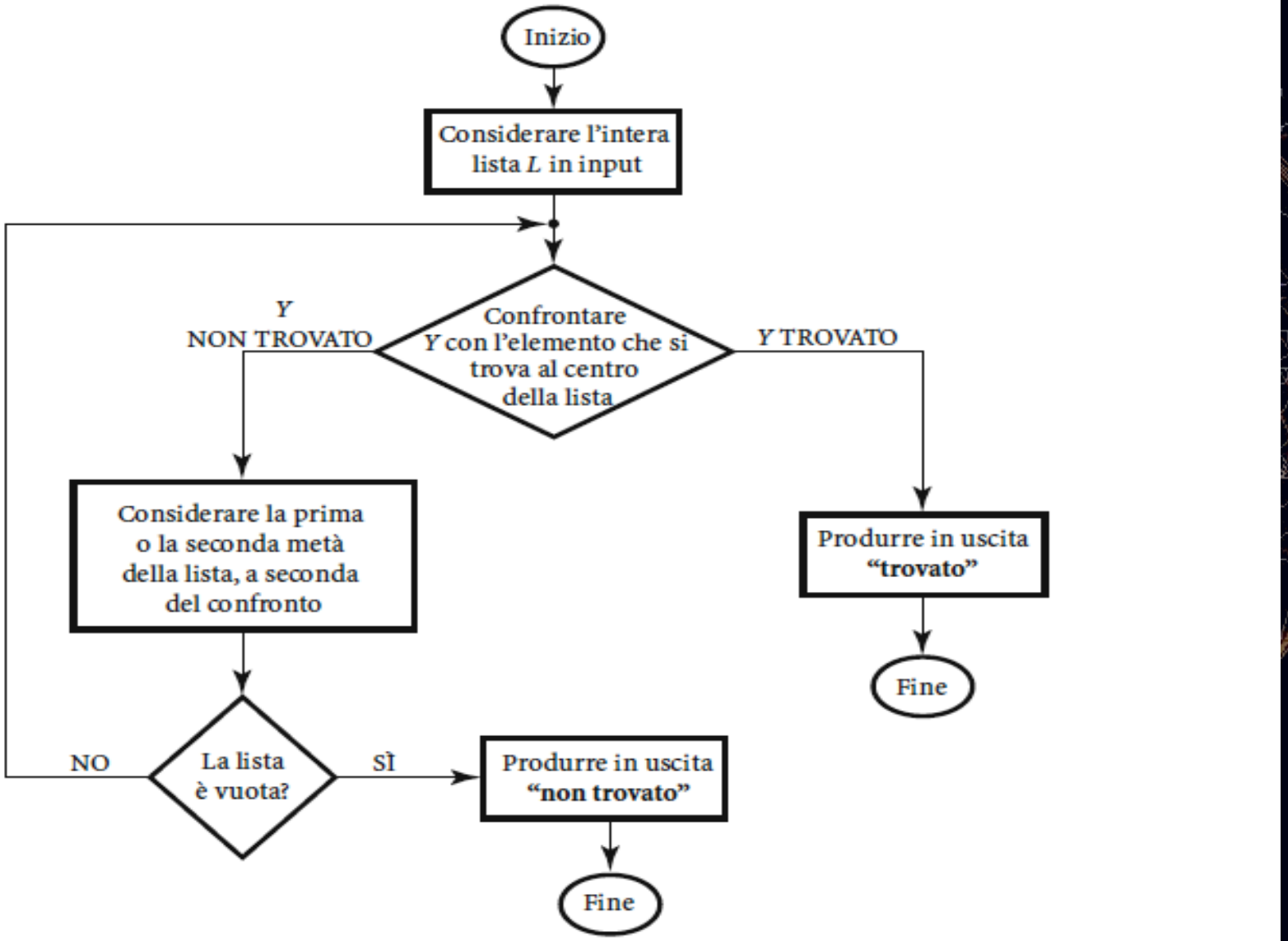
Supponiamo di voler cercare un nome in una rubrica telefonica contenente 100 nomi.

Il primo algoritmo che viene in mente potrebbe essere:

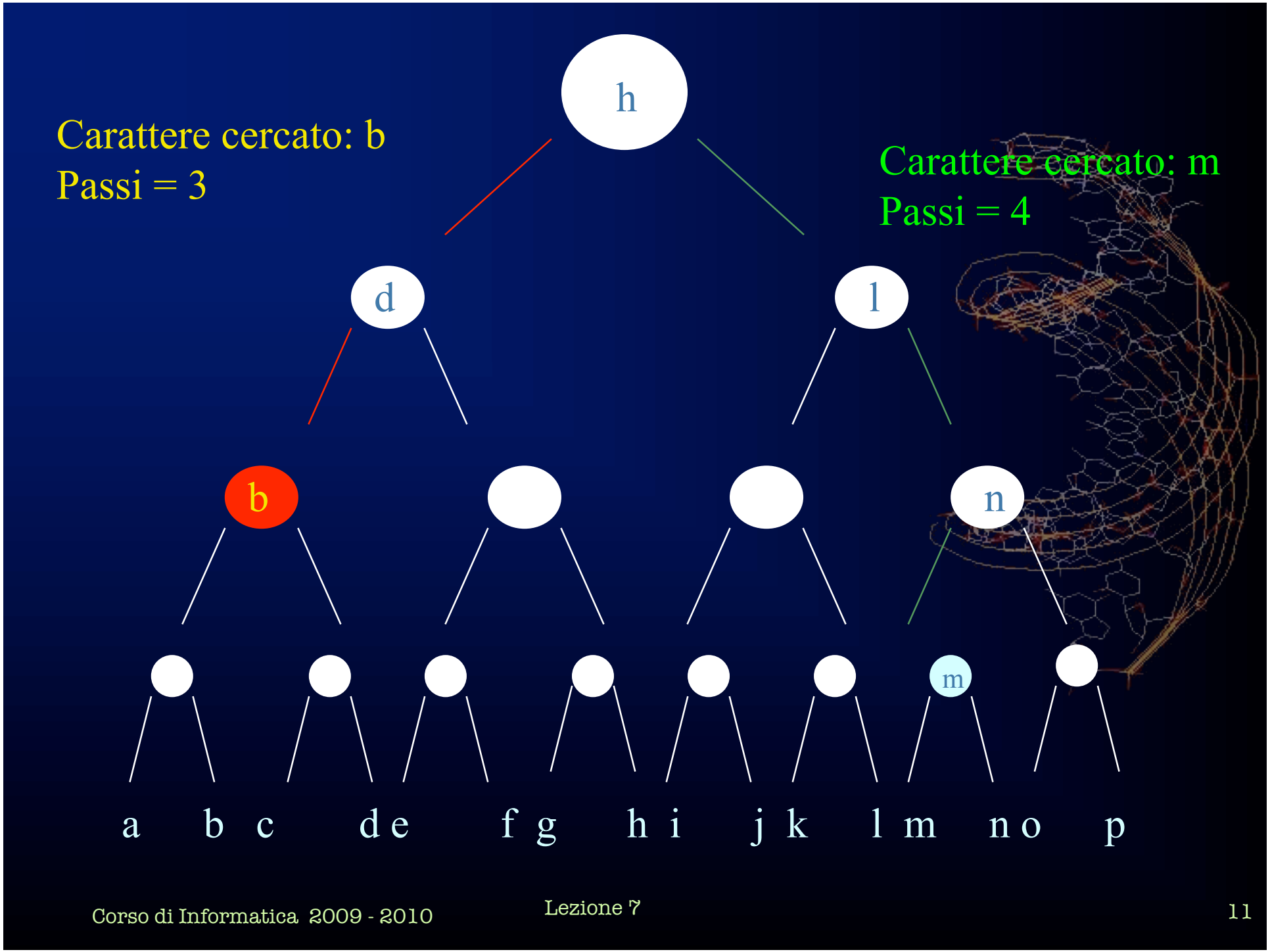
1. Acquisisci $nome_1 \ldots .. nome_{100}$
2. Acquisisci $tel_1 \ldots\ldots tel_{100}$
3. Acquisisci *nome cercato*
4. Poni *trovato* = falso
5. Poni $i = 1$
6. Ripeti finché *trovato* diventa vero o $i > 100$
   7. Se $nome_i$ = *nome cercato* allora
      8. Stampa tel_i
      9. Poni *trovato* = vero
      Altrimenti
      10. Incrementa i di 1
11. Fine del ciclo
12. Se *trovato* = falso allora
    13. Stampa messaggio 'Nome non in elenco'
14. Fermati

…e quello di ricerca binaria

L’algoritmo di ricerca sequenziale è semplice e corretto... ma non sfrutta il fatto che la rubrica è ordinata !


Inizio
Considerare l'intera lista L in input
Confrontare Y con l'elemento che si trova al centro della lista
Y NON TROVATO
Y TROVATO
Considerare la prima o la seconda metà della lista, a seconda del confronto
Produrre in uscita "trovato"
Fine
La lista è vuota?
NO
SÌ
Produrre in uscita "non trovato"
Fine

1. Acquisisci $nome_1 \ldots .nome_{100}$
2. Acquisisci $tel_1 \ldots \ldots tel_{100}$
3. Acquisisci *nome cercato*
4. Poni *trovato = falso*
5. Poni *inizio* = 1 e *fine* = 100
6. Ripeti finché *trovato* diventa vero o *fine<inizio*
    7. Poni *i* a (*inizio*+fine)/2
    8. Se $nome_i$ = *nome cercato* allora
        9. Stampa tel_i
        10. Poni *trovato = vero*
    Altrimenti
        Se *nome cercato* precede alfabeticamente $nome_i$
            11. Poni *fine = i -1*
        altrimenti (*nome cercato* segue $nome_i$)
            12. Poni *inizio = i +1*
13. Fine del ciclo
14. Se trovato = falso allora
    15.Stampa messaggio ‘ Nome non in elenco’
16. Fermati


Carattere cercato: b
Passi = 3
Carattere cercato: m
Passi = 4
h
d
l
b
n
m
a b c d e f g h i j k l m n o p

(1) Allen

(2) Baley

(3) Boyer

(4) Casy

(5) Davis

(6) Davison

(7) Glen

(8) Greer

(9) Haley

(10) Hanson

(11) Harrison -

(12) Lister

(13) Mendel

(14) Morgenstern

(15) Patton

(16) Perkins

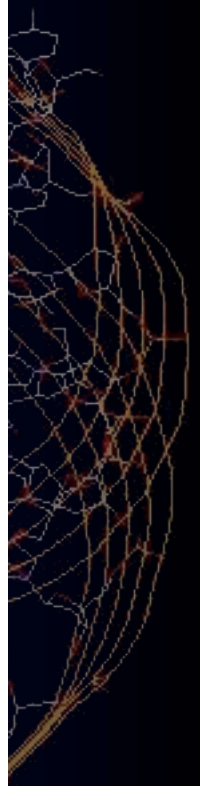
(17) Quinn

(18) Reed

(19) Schmidt

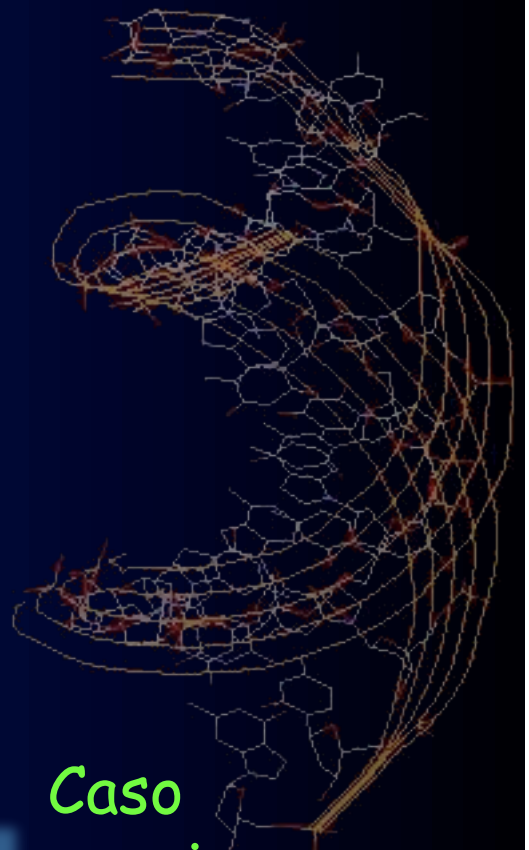
(20) Woolf

Inizio



Ricerca Binaria: Complessita'

Passi (confronti)	Numero di dati per passo
1	n
2	n /2
3	$n/2/2 = n/2^2$
4	$n/2^3$
.....	.....
k	$n/2^{(k-1)}$
.....	.....
s	$1 = n/2^{(s-1)}$



Caso peggiore

$2^{(s-1)} = n \; ; \; s-1 = \log_2 n \; ; \; s = \log_2 n + 1$ ⇒ $O(\log_2 n)$

Confronto fra ricerca sequenziale e ricerca binaria

	CRAY T3E-900 1360 processori in parallelo	Pentium Pro 200
Costo	30 Milioni €	~ 50 €
Operazione/sec	$7*10^{11}$	$7*10^{7}$
Algoritmo	Ricerca sequenziale	ricerca binaria

Elenco di Napoli (10^6 abitanti): **Pentium: 7 volte più veloce**

Elenco di New York ($2*10^7$ abitanti): **Pentium: 120 volte più veloce**

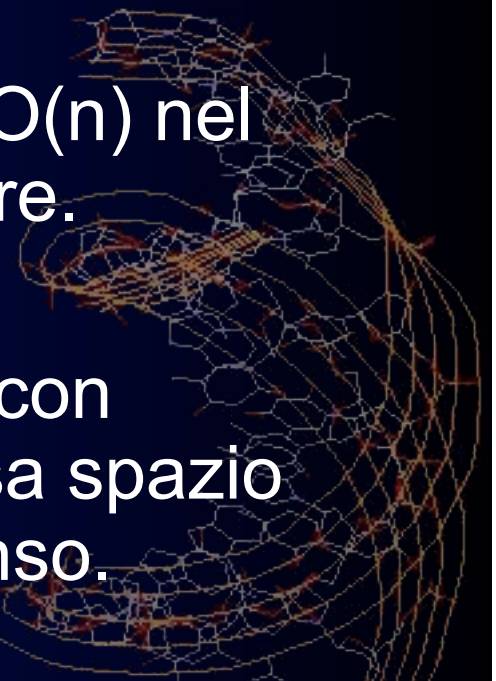
Ordini di grandezza

- gli algoritmi di complessita' di tipo polinomiale $O(n^k)$ producono una soluzione *trattabile* ad un problema;
- gli algoritmi di complessita' di tipo esponenziale $O(k^n)$ producono una soluzione *intrattabile*

$7 \cdot 10^7$ op. al secondo

	10	20	50	60
n	10^{-7} s	$2 \cdot 10^{-7}$ s	$5 \cdot 10^{-7}$ s	$6 \cdot 10^{-7}$ s
n^2	10^{-6} s	$4 \cdot 10^{-6}$ s	$25 \cdot 10^{-6}$ s	$36 \cdot 10^{-6}$ s
2^n	10^{-5} s	$1.6 \cdot 10^{-2}$ s	$1.6 \cdot 10^{7}$ s	$1.6 \cdot 10^{10}$ s

Efficienza rispetto allo spazio



- Tutti gli algoritmi mostrati finora sono O(n) nel numero di celle di memoria da utilizzare.
- In alcuni casi (ordinamento di elenchi con ricopiature e spostamenti etc) la risorsa spazio può essere utilizzata in modo più intenso.
- In generale esiste un trade-off tempo - spazio : la soluzione più efficiente per la risorsa tempo tende a utilizzare maggiormente la risorsa spazio e viceversa.

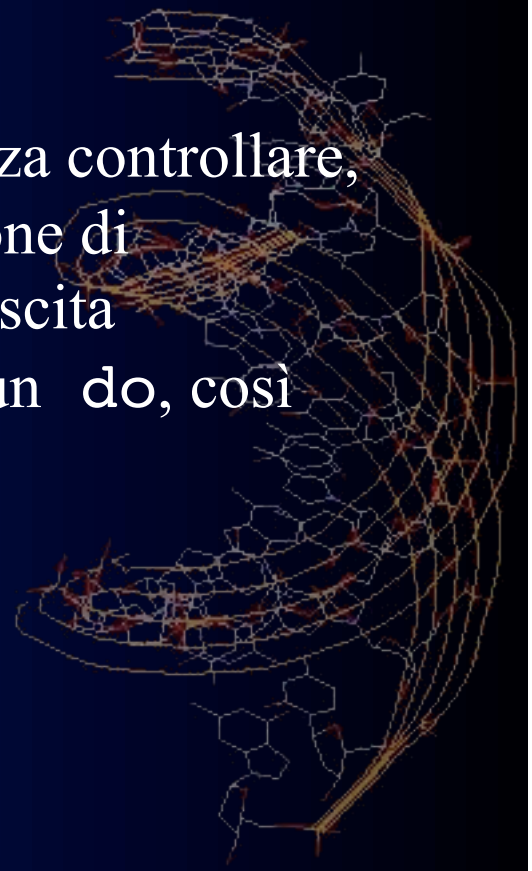
Riassumendo...

- Abbiamo definito gli algoritmi e visto alcuni esempi. Ci siamo preoccupati delle proprietà formali degli algoritmi.
- Abbiamo visto che gli algoritmi possono sempre essere ricondotti ad operazioni sequenziali-condizionali-iterative
- Abbiamo trovato dei criteri per studiare la correttezza e l'efficienza degli algoritmi.
- Abbiamo imparato che algoritmi di tipo esponenziale sono di fatto inutilizzabili.


Il linguaggio C

L'istruzione `break`

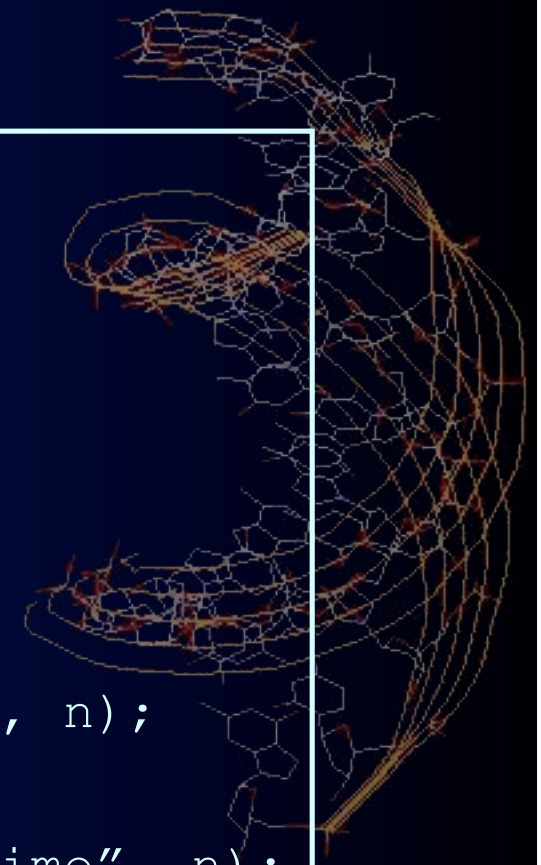
Talvolta può essere utile uscire da un ciclo senza controllare, all'inizio o alla fine dell'iterazione, la condizione di terminazione. L'istruzione `break` provoca l'uscita incondizionata da un `for`, un `while` oppure un `do`, così come consente di uscire da uno `switch`.



```
for ( count = 0; count < 10; count++ )
{
    if ( count == 5 )
        break;
}
```

Esempio

```
for (i=n-1; i>=2; i--)
{
resto = n%i;
if(resto == 0)
      break;
}
if(i==1)
       printf("%d è un numero primo", n);
else
       printf("%d non è un numero primo", n);
```

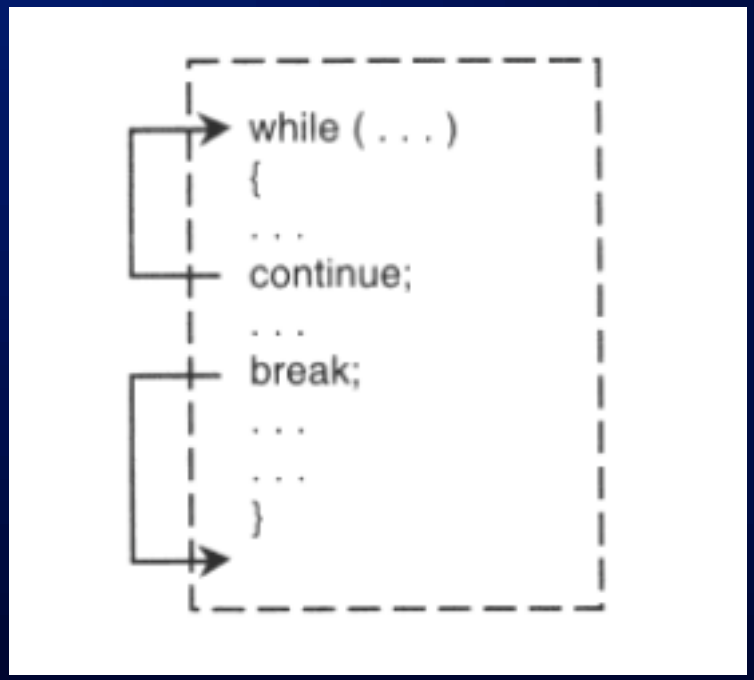


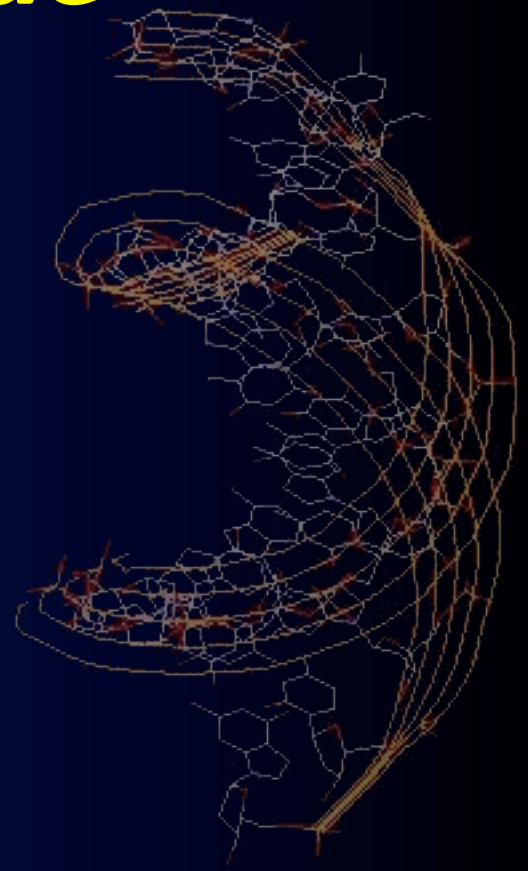
L'istruzione `continue`

L'istruzione `continue` è simile al `break` ma forza l'inizio della successiva iterazione di un `for`, un `while` oppure un `do`.

```
int x;
printf("Printing only the even numbers from 1 to 10\n");
for( x = 1; x <= 10; x++ )
{
   if( x % 2 != 0 )     /* See if the number is NOT even */
      continue;        /* Get next instance x */
   printf( "\n%d", x );
}
```

Break e continue





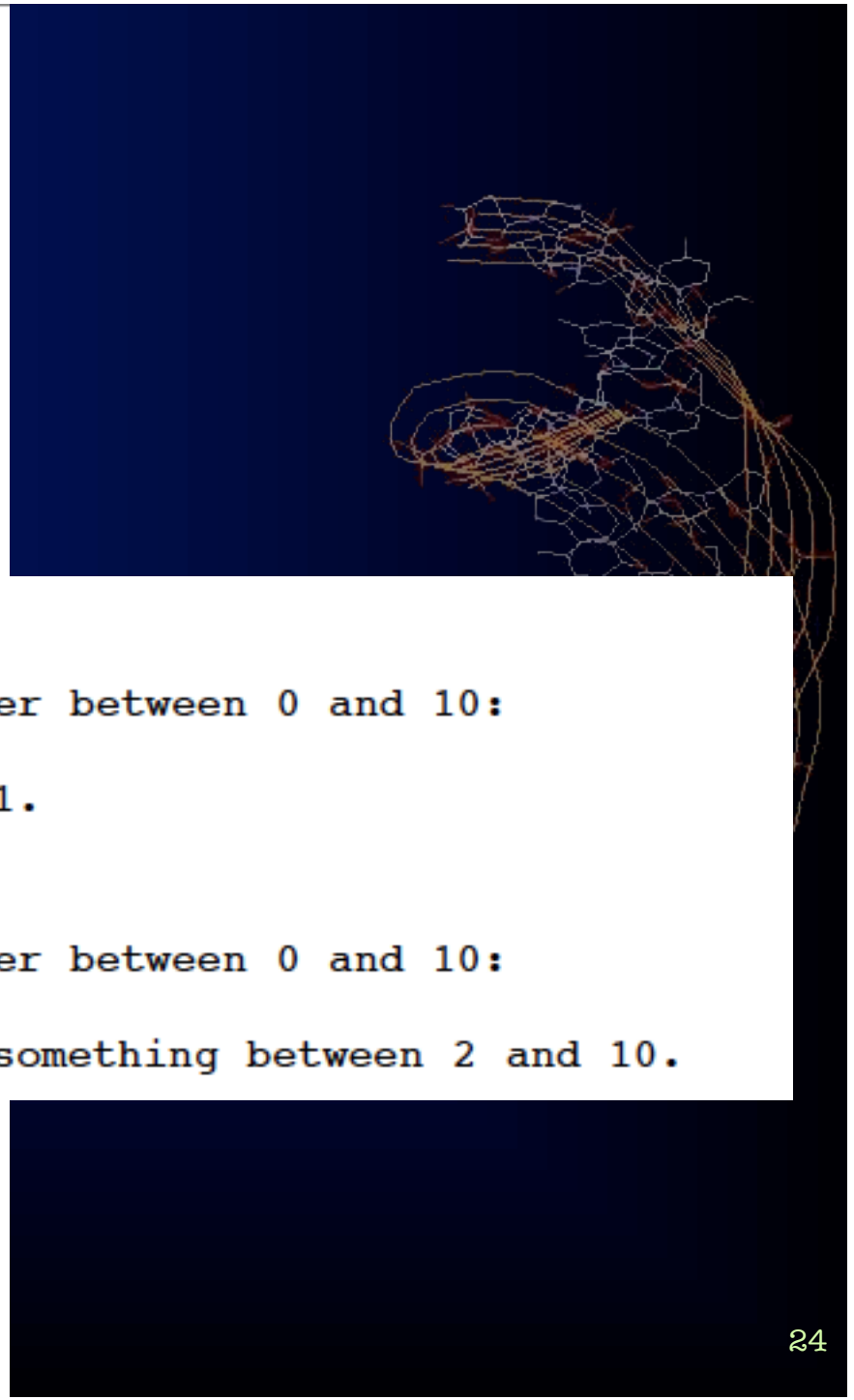
L'istruzione goto

L'istruzione `goto` causa un salto incondizionato a una istruzione, individuata da un'etichetta, che si trova in un altro punto della funzione corrente

```
goto location;
```

```
location: a C statement;
```

```c
/* Demonstrates the goto statement */

#include <stdio.h>

int main()
{
int n;

start:
 puts("Enter a number between 0 and 10: ");
 scanf("%d", &n);

 if (n < 0 ||n > 10 )
 goto start;
 else if (n == 0)
 goto location0;
 else if (n == 1)
 goto location1;
 else
 goto location2;

 location0:
 puts("You entered 0.\n");
 goto end;

 location1:
 puts("You entered 1.\n");
 goto end;

 location2:
 puts("You entered something between 2 and 10.\n");

 end:
 return 0;
 }
```



INPUT/OUTPUT:

```
Enter a number between 0 and 10:
1
You entered 1.
```

INPUT/OUTPUT:

```
Enter a number between 0 and 10:
9
You entered something between 2 and 10.
```

L'istruzione `goto`

L'utilizzo dell'istruzione `goto` va evitato in quanto è un metodo primitivo per alterare il flusso di controllo

È sempre possibile scrivere il codice necessario utilizzando altri costrutti.

Un programmatore che utilizza diffusamente l'istruzione `goto` rende ben presto incomprensibile il programma

SPAGHETTI CODING

La funzione `exit()`

Un programma in C normalmente termina quando l'esecuzione raggiunge la parentesi chiusa che indica la fine della funzione `main()`

- È tuttavia possibile terminare l'esecuzione di un programma utilizzando la funzione di libreria `exit()`

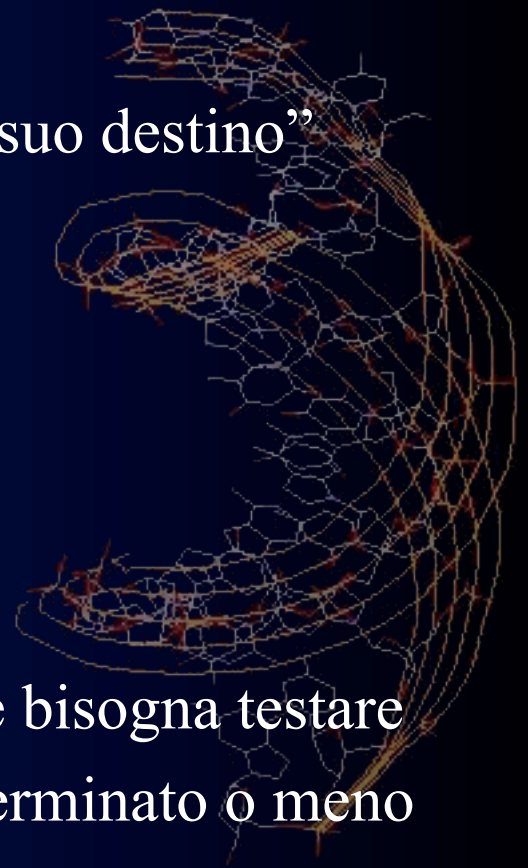
Per utilizzare la funzione `exit()` è necessario includere il file di header `stdlib.h`

- La funzione `exit()` ammette un unico argomento di tipo `int` che indica il successo (0) o il fallimento (1) del programma

Cicli infiniti

Per ciclo infinito si intende un ciclo che "lasciato al suo destino" girerebbe per sempre

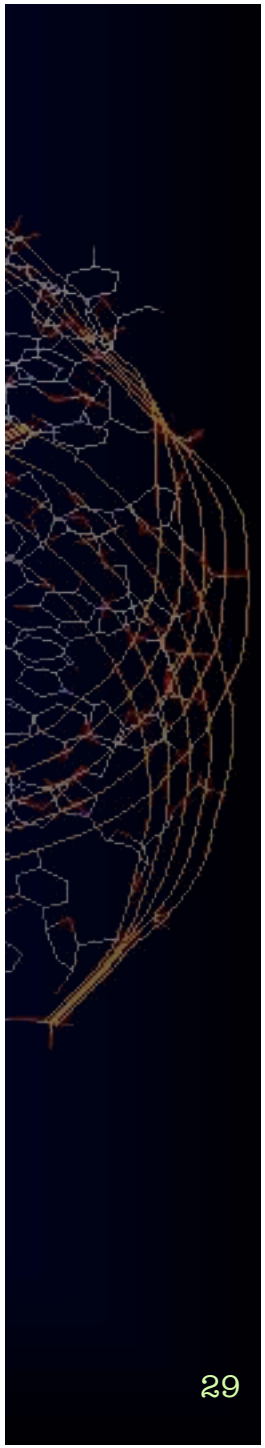
```
while(1)
{
    /* additional code goes here */
}
```



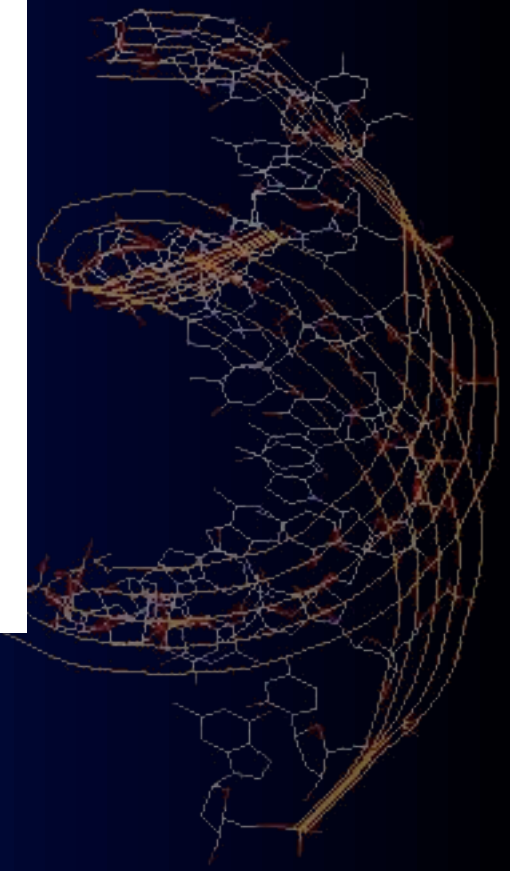
L'utilizzo di un ciclo infinito può essere utile allorché bisogna testare numerose condizioni per determinare se un ciclo va terminato o meno

```
/* Demonstrates using an infinite loop and the switch */
/* statement to implement a menu system. */
#include <stdio.h>
#include <stdlib.h>

#define DELAY 150000

int menu(void);
void delay(void);

int main()
{

   while (1)
   {
         /* Get user's selection and branch based on the input. */

         switch(menu())
         {
            case 1:
                 {
                   puts("n\Executing task A.");
                   delay();
                   break;
                 }
            case 2:
                 {
                   puts("\nExecuting task B.");
```

```
                    delay();
                    break;
                }
            case 3:
                {
                    puts ("\nExecuting task C.");
                    delay(0);
                    break;
                }
            case 4:
                {
                    puts("\nExecuting task D.");
                    delay();
                    break;
                }
            case 5:     /* Exit program. */
                {
                    puts("\nExiting program now . . . \n");
                    delay();
                    exit;
                }
            default:
                {
                    puts("\nInvalid choice, try again.");
                    delay();
                }
        }   /* End of switch */
    }       /* End of while  */
    return 0;
}
```



```
/* Displays a menu and inputs user's selection. */
int menu(void)
{
     int reply;

     puts("\nEnter 1 for task A.");
     puts("Enter 2 for task B.");
     puts("Enter 3 for task C.");
     puts("Enter 4 for task D.");
     puts("Enter 5 to exit program.");

     scanf("%d", &reply);

     return reply;
}
```



```
void delay( void )
{
     long x;
     for( x =0; x < DELAY; x++ )
          ;
}
```

Operatore Condizionale: ? :

espressione1 **?** *espressione2* **:** *espressione 3*;

Viene prima valutata l'espressione1; se risulta *vera*, viene valutata l'espressione2 il cui valore diviene il valore della espressione condizionale. Se invece l'espressione1 risulta falsa, l'espressione3 viene valutata ed il suo valore diviene il valore dell'espressione condizionale.

```
max = (a > b)? a: b;
```

```
if(a > b)
    max = a;
else
    max = b;
```

Operatore Condizionale: ? :

L'operatore condizionale può essere utilizzato quando non è possibile usare un'istruzione `if` come ad esempio all'interno di una chiamata a un'altra funzione

```
printf( "The larger value is %d", ((x > y) ? x : y) );
```