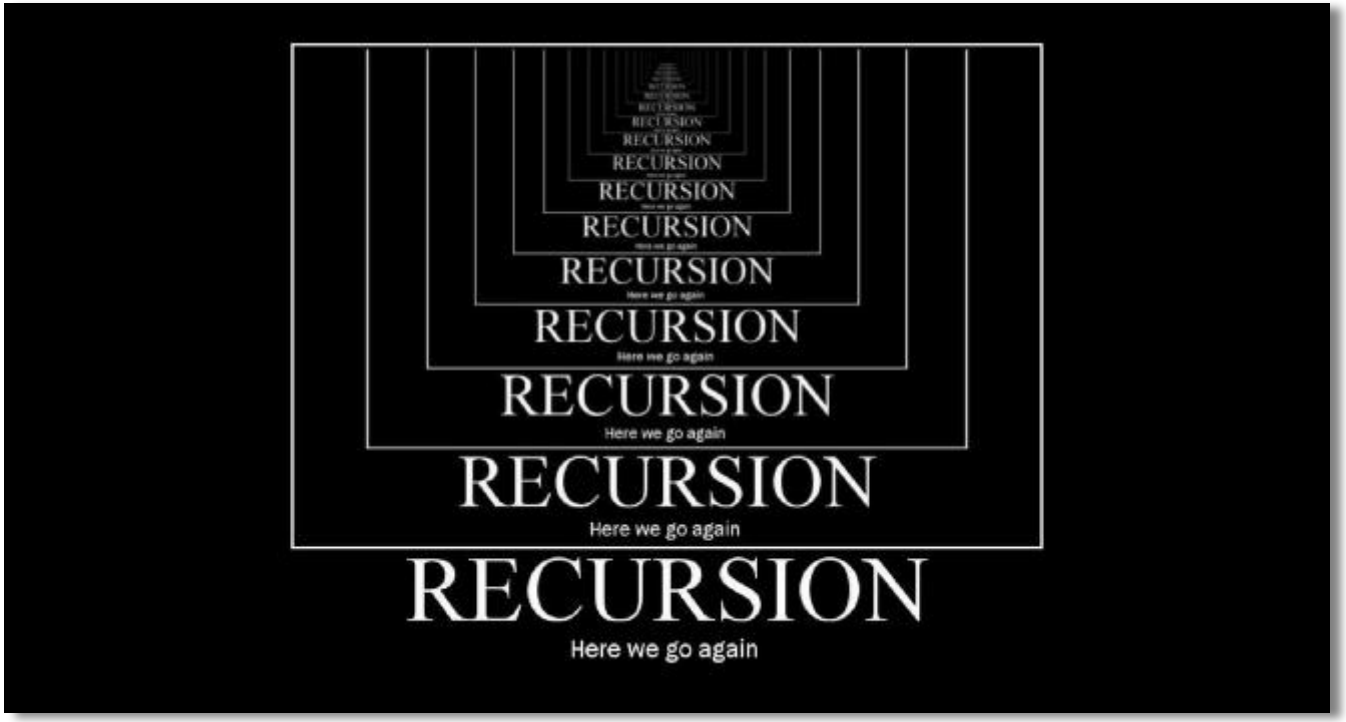
Ricorsione, Serie di Fibonacci, Bubble sort

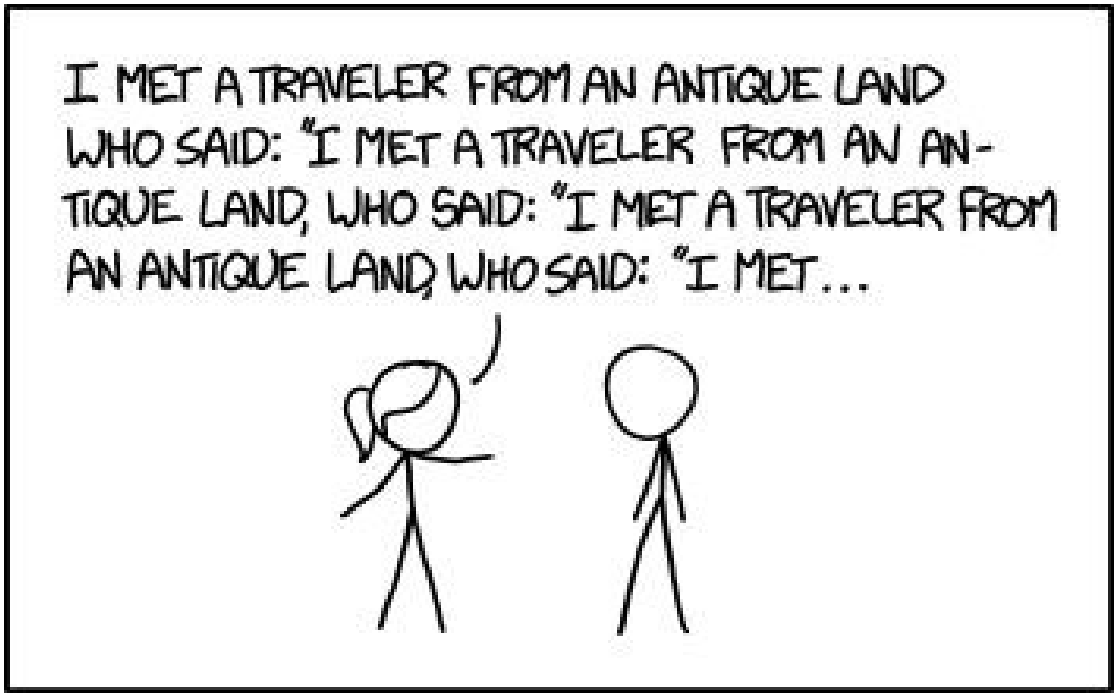
Dott. Emanuele Pio Barracchia

Ricorsione

- La **ricorsione** in informatica è un metodo che si usa quando la soluzione di un problema dipende dalle soluzioni di piccole istanze dello stesso problema
- La definizione di una funzione ricorsiva ha uno o più ***casi base***, cioè input per i quali la funzione produce un risultato senza richiamare se stessa
- Ad ogni esecuzione della funzione ricorsiva il problema in input è semplificato in modo tale che dopo un numero di passi riuscirà ad arrivare ad un ***caso base***



Ricorsione


I MET A TRAVELER FROM AN ANTIQUE LAND
WHO SAID: "I MET A TRAVELER FROM AN AN-
TIQUE LAND, WHO SAID: "I MET A TRAVELER FROM
AN ANTIQUE LAND, WHO SAID: "I MET...

Approccio ricorsivo Vs Approccio iterativo

- Per ogni soluzione ricorsiva esiste la corrispondente soluzione iterativa
- La ricorsione occupa più spazio in memoria ed è più lenta rispetto all’approccio iterativo
- Per alcuni problemi le soluzioni ricorsive sono spesso più semplici, più leggibili e più eleganti rispetto alle soluzioni iterative

Un esempio: il fattoriale di un numero

$n!$

- In matematica, si definisce fattoriale di un numero naturale il prodotto dei numeri interi positivi minori o uguali a tale numero.

- Inoltre il fattoriale di 0 è pari a 1.

- **Esercizio:** scrivere una funzione ricorsiva in C che, dato un numero, calcoli il fattoriale.
  - Tip: prima di tutto pensate a quale possa essere il *caso base.*

Un esempio: il fattoriale di un numero

Senza puntatore

```
long factorial(long n)
{
    if (n == 0) //passo base
        return 1;
    else
        return n*factorial(n-1);
}
```

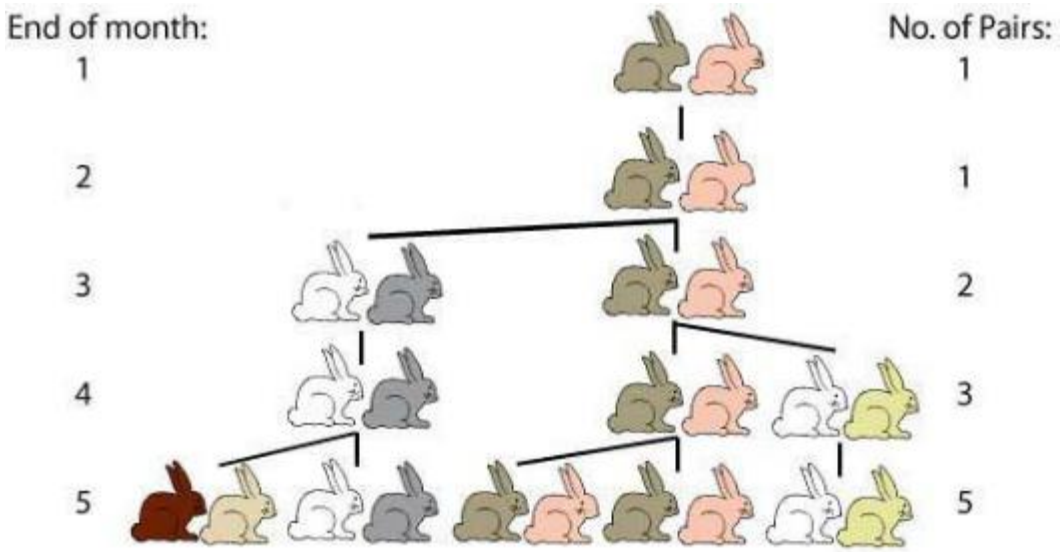
Con puntatore

```
long fact_point(long* pointer)
{
    if (*pointer == 0){
        return 1;
    }
    else if (*pointer == 1){  //passo base
        return *pointer;
    }
    else
    {
        long tmp = *pointer - 1;
        fact_point(&tmp);
        return *pointer *= tmp;
    }
}
```

Successione di Fibonacci - Origini

Fibonacci cercò un modo per descrivere la crescita di una popolazione di conigli. In particolare assunse per ipotesi che:

- Si dispone di una coppia di conigli appena nati
- Ogni coppia diventa fertile dopo un mese e dà alla luce una nuova coppia di conigli dopo un ulteriore mese
- Le coppie fertili danno alla luce una coppia di figli al mese



Successione di Fibonacci

- In matematica, la successione di Fibonacci è una successione di numeri interi positivi in cui ciascun numero è la somma dei due numeri precedenti.

- I primi due termini della successione sono, per definizione, $F_1 = 1$ e $F_2 = 1$

- La regola è la seguente:
  - $F_1 = 1$
  - $F_2 = 1$
  - $F_n = F_{n-1} + F_{n-2}$ (per ogni $n > 2$)

- **Esercizio:** scrivere una funzione ricorsiva in C che, dato un numero *i*, calcoli l'*i*-esimo numero della successione di Fibonacci.
  - Tip: prima di tutto pensate a quale possa essere il *caso base.*

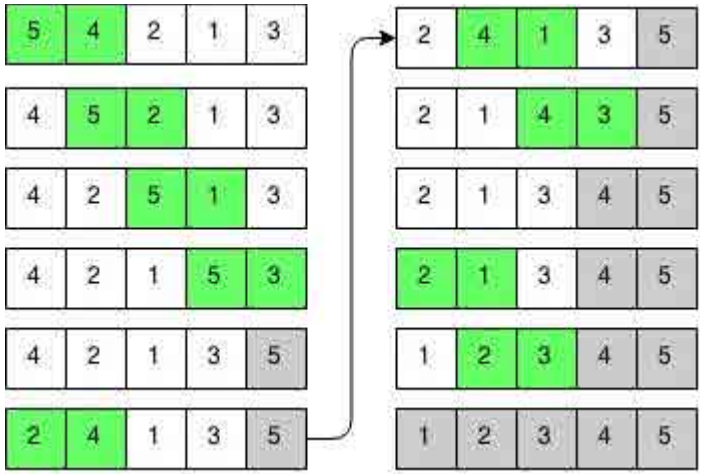
Successione di Fibonacci

```
long fibonacci(long n){
    //inizio passo base
    if (n == 0){
        return 0;
    } else if (n == 1){
        return 1;
    }
    //fine passo base
    else {
        return fibonacci(n-1) + fibonacci(n-2);
    }

    return 0;
}
```

Bubble sort

- In informatica il **Bubble sort**, o ordinamento a bolle, è un algoritmo di ordinamento che confronta ogni coppia di elementi adiacenti e li inverte nel caso in cui siano nell'ordine sbagliato.



- Il nome dell'algoritmo deriva dal modo in cui gli elementi vengono ordinati in questa lista: quelli più **"leggeri" "risalgono"** verso un'estremità, mentre quelli più **"pesanti" "affondano"** verso l'estremità opposta.

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

i = 0
j = 0

21	10	3	8	30

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

i = 1

j = 0

21	10	3	8	30

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

i = 1
j = 0

21	10	3	8	30

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

i = 1

j = 1

10	21	3	8	30

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

i = 1
j = 1

10	21	3	8	30

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

i = 1

j = 2

10	3	21	8	30

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

i = 1
j = 2

10	3	21	8	30

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

i = 1

j = 3

10	3	8	21	30

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

i = 1
j = 3

10	3	8	21	30

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

i = 2
j = 0

10	3	8	21	30

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

i = 2
j = 0

10	3	8	21	30

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

i = 2

j = 1

3	10	8	21	30


A FEW
MOMENTS LATER

Bubble sort - Algoritmo

Array iniziale:

21	10	3	8	30

Array dopo l'esecuzione dell'algoritmo **Bubble sort**:

3	8	10	21	30

Bubble sort - Algoritmo

Senza puntatore

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

Con puntatore

```
void bubblesort_pointer (int* pointer, int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (*(pointer+j) > *(pointer+j+1)){
                int temp = *(pointer+j);
                *(pointer+j) = *(pointer+j+1);
                *(pointer+j+1) = temp;
                sorted = 0;
            }
        }
    }
}
```

Domande?