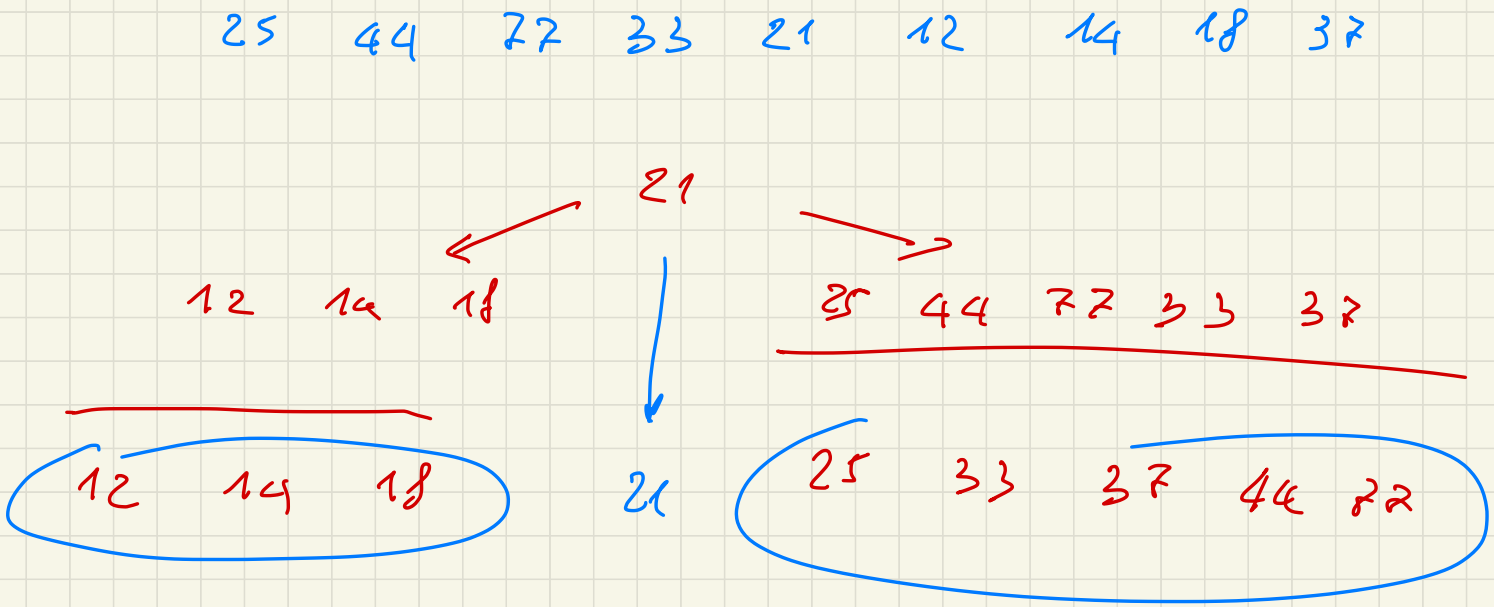
Algoritmi e Strutture Dati

Lezione 12

20 ottobre 2025

Quicksort


25 44 77 33 21 12 14 18 37
21
12 14 18
25 44 77 33 37
12 14 18
21
25 33 37 44 77

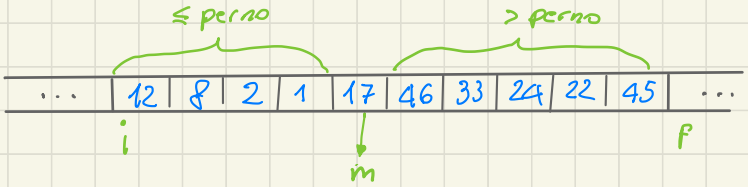
ALGORITMO quickSort (vettore A)

IF lunghezza di A > 1 THEN

- scegli un elemento x in A
- $B \leftarrow \{ y \in A \mid y < x \}$
- $C \leftarrow \{ y \in A \mid y > x \}$

  ← partizionamento
- quickSort(B)
- quickSort(C)
- $A \leftarrow B ; x ; C$

```
PROCEDURA quickSort (Array A, indice i, indice f)
    IF f - i > 1 THEN
        m ← partizione (A, i, f)
        quickSort (A, i, m)
        quickSort (A, m+1, f)
```



1 2 8 12 17 22 24 33 45 46

```
ALGORITMO quickSort (Array A[0 .. n-1])
    quickSort (A, 0, n)
```

QuickSort: numero di confronti $C(n)$

se $n \leq 1$ → 0 confronti

altrimenti

confronti per calcolare partizione — $C_{part}(n) = n$

+

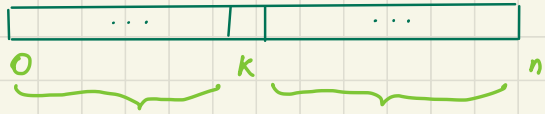
confronti per ordinare parte sx — $C(k)$

+

confronti per ordinare parte dx — $C(n-k-1)$

se il pivot finisce in posizione k

```
PROCEDURA quickSort (Array A, indice i, indice f)
  IF f - i > 1 THEN
    m ← partiziona (A, i, f)
    quickSort (A, i, m)
    quickSort (A, m+1, f)
```

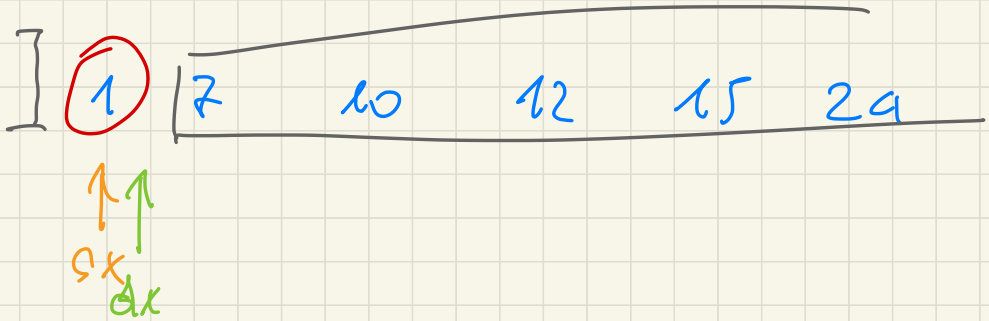


QuickSort: numero di confronti

CASO PEGGIORE $C(n) \approx \frac{n^2}{2}$ ← caso peggiore molto raro!

CASO MIGLIORE $C(n) \approx n \log_2 n$

CASO MEDIO $C(n) \approx 1.39\, n \log_2 n$



QuickSort: uso dello spazio

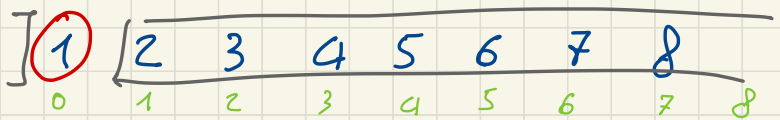
```
ALGORITMO quickSort (Array A[0.. n-1])
    quickSort(A, 0, n)
```

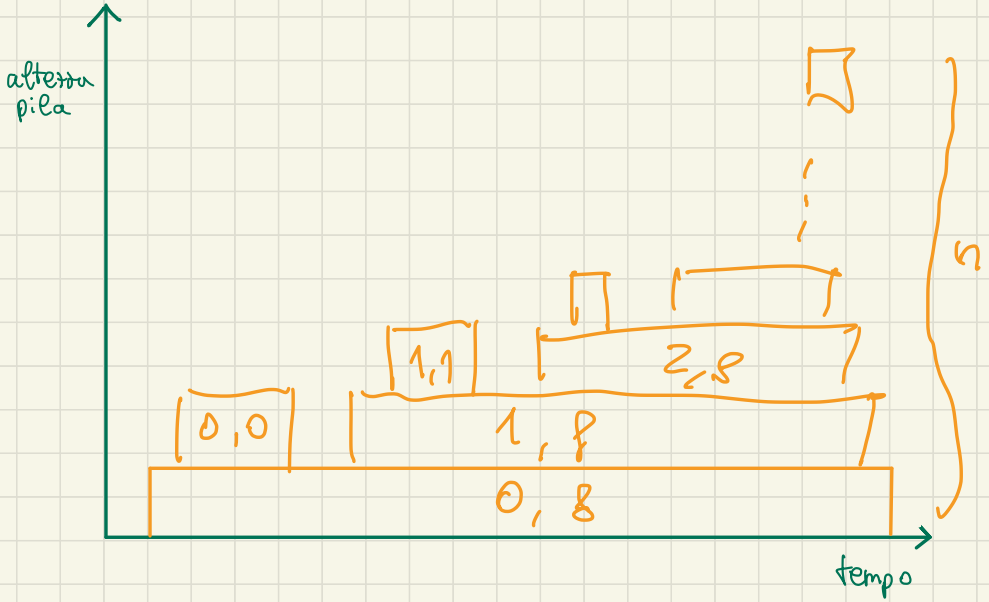
```
PROCEDURA quickSort (Array A, indice i, indice f)
    IF f - i > 1 THEN
        m ← partiziona(A, i, f)
        quickSort(A, i, m)
        quickSort(A, m+1, f)
```

```
ALGORITMO partiziona (Array A, indice i, indice f) → indice
    perno ← A[i]
    sx ← i, dx ← f
    WHILE sx < dx DO
        DO dx ← dx - 1 WHILE A[dx] > perno
        DO sx ← sx + 1 WHILE sx < dx AND A[sx] ≤ perno
        IF sx < dx THEN
            scambia A[sx] con A[dx]
    scambia A[i] con A[dx]
    RETURN dx
```

```
PROCEDURE quickSort (Array A, indice i, indice f)
    IF f - i > 1 THEN
        m ← partiziona (A, i, f)
        quickSort (A, i, m)
        quickSort (A, m+1, f)
```

[A] Array già ordinato

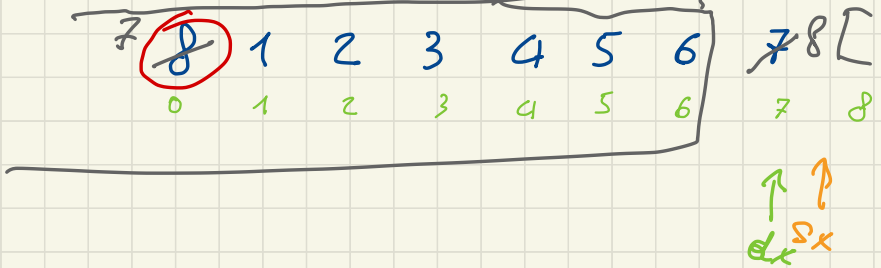


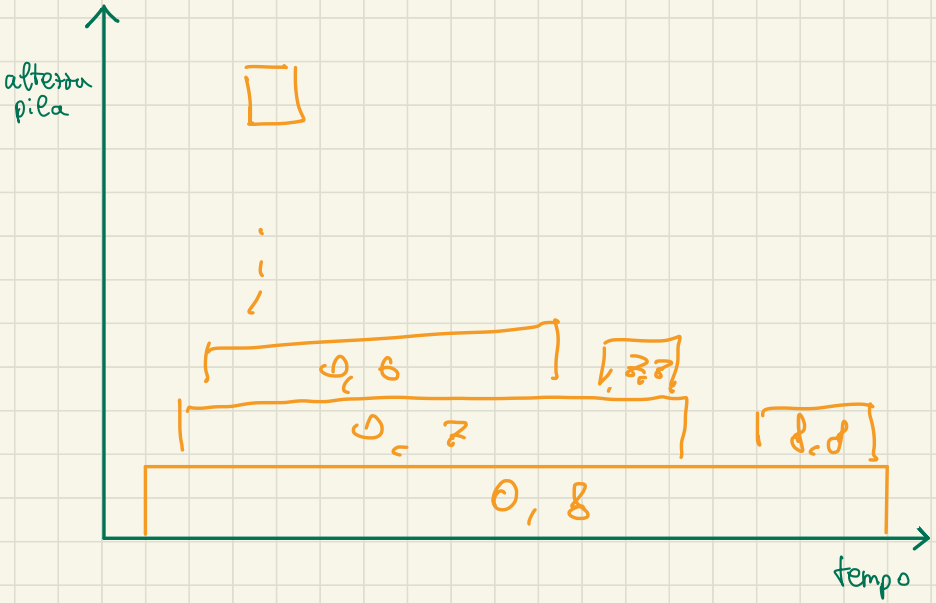



```
PROCEDURA quickSort (Array A, indice i, indice f)
  IF f - i > 1 THEN
     m ← partiziona (A, i, f)
     quickSort (A, i, m)
     quickSort (A, m+1, f)
```

altezza circa n

[B] Array quasi ordinato

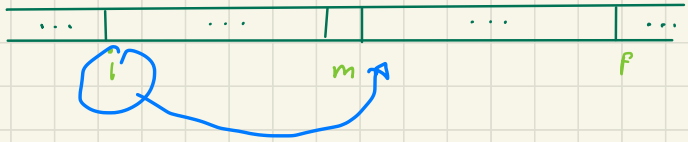





```
PROCEDURA quickSort (Array A, indice i, indice f)
    IF f-i > 1 THEN
        m ← partiziona (A, i, f)
        quickSort (A, i, m)
        quickSort (A, m+1, f)
```

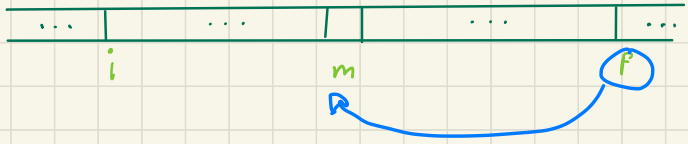
altezza pila $\simeq n$

```
PROCEDURA quickSort (Array A, indice i, indice f)
    IF f-i > 1 THEN
        m ← partizione (A, i, f)
        quickSort (A, i, m)
        quickSort (A, m+1, f)
```




```
PROCEDURA quickSort(Array A, indice i, indice f)
    WHILE f-i > 1 DO
        m ← partizione (A, i, f)
        quickSort (A, i, m)
        i ← m+1
```

```
PROCEDURE quickSort (Array A, indice i, indice f)
    IF f - i > 1 THEN
        m ← partition (A, i, f)
        quickSort (A, i, m)
        quickSort (A, m+1, f)
```




```
PROCEDURE quickSort (Array A, indice i, indice f)
    IF f - i > 1 THEN
        m ← partition (A, i, f)
        quickSort (A, m+1, f)
        quickSort (A, i, m)
```

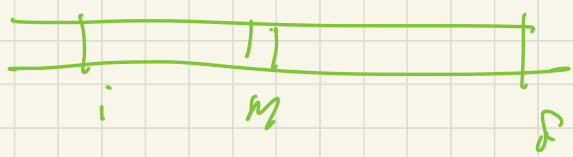
```
PROCEDURE quickSort (Array A, indice i, indice f)
    WHILE f - i > 1 DO
        m ← partition (A, i, f)
        quickSort (A, m+1, f)
        f ← m
```

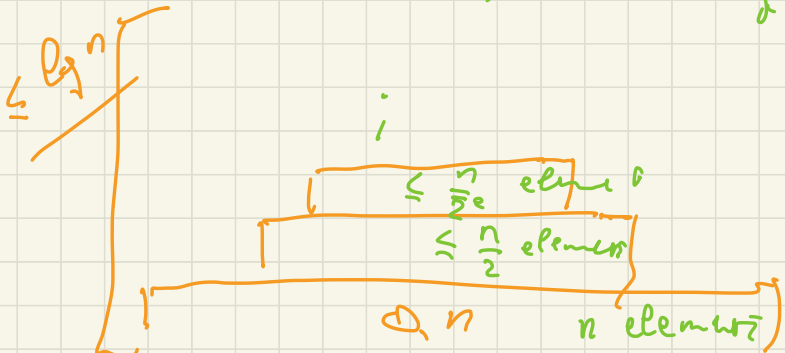
```
PROCEDURA quickSort (Array A, indice i, indice f)
    IF f-i > 1 THEN
        m ← partizione (A, i, f)
        quickSort (A, i, m)
        quickSort (A, m+1, f)
```

```
PROCEDURA quickSort (Array A, indice i, indice f)
    WHILE f-i > 1 DO
        m ← partizione (A, i, f)
        quickSort (A, i, m)
        i ← m+1
```

```
PROCEDURA quickSort (Array A, indice i, indice f)
    WHILE f-i > 1 DO
        m ← partizione (A, i, f)
        IF m-i < f-m THEN
            quickSort (A, i, m)
            i ← m+1
        ELSE
            quickSort (A, m+1, f)
            f ← m
```

```
PROCEDURA quickSort (Array A, indice i, indice f)
    WHILE f-i > 1 DO
        m ← partizione (A, i, f)
        quickSort (A, m+1, f)
        f ← m
```





Spazio

Spazio

- versione migliorata

  altezza stack $\Theta(\log n)$

  grandezza record attiv. $\Theta(1)$

  $\Theta(\log n)$

- versione base

  altezza stack $\Theta(n)$

  grandezza record attiv. $\Theta(1)$

  $\Theta(n)$

Tipi di dati

COSA

Tipo di una variabile

Attributo che specifica l'insieme di *valori* che la variabile può assumere e le relative *operazioni*

Esempio

Tipo Dizionario

Collezione di elementi ciascuno dei quali è caratterizzato da una *chiave*

Chiavi

Appartengono a un dominio *totalmente* ordinato

Operazioni tipiche

- inserimento
- ricerca
- cancellazione

Strutture dati

COME

Struttura dati

Specifica *organizzazione delle informazioni* che permette di realizzare e implementare un determinato tipo di dati

stesso tipo ≠ struttura dati

es DIZIONARIO

- array ordinato in base alla chiave
  RICERCA $\Theta(\log n)$
  INSERIMENTO $\Theta(n)$

- array non ordinato
  RICERCA $\Theta(n)$
  INSERIMENTO $\Theta(1)$

Collezioni: strutture indicizzate (array)

- Allocate in una *porzione contigua* di memoria
- Accesso mediante *indice* (posizione)
- Tempo di accesso *indipendente* dalla posizione del dato

LIMITAZIONE: capacità fissata

STRUTTURE STATICHE

Collezioni: strutture collegate

- Non è necessario allocare l'intera struttura in una porzione contigua di memoria
- Elementi collegati tra loro
- Passaggio da un elemento ad altri tramite collegamenti
- Varie tipologie di collegamento (liste, alberi, ...)

STRUTTURE DINAMICHE

Liste concatenate

Lista lineare

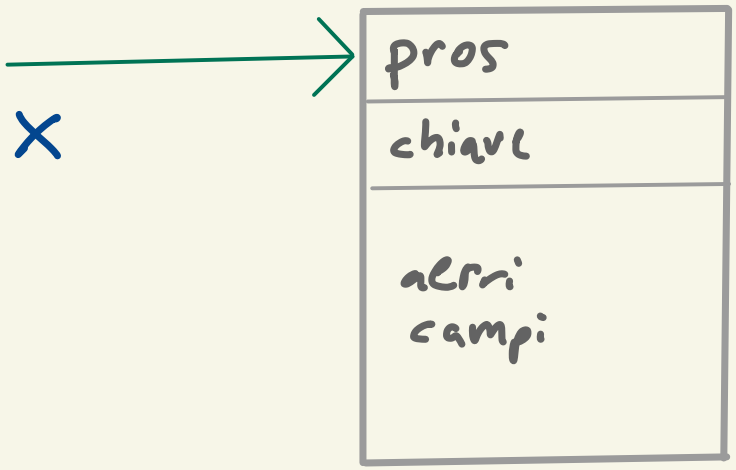
Insieme ordinato di *nodi* collegati linearmente uno dopo l'altro.

Ogni nodo contiene:

- un *dato* della collezione
  (in genere formato da un certo insieme di campi, tra cui uno funge da campo *chiave*)
- l'informazione per accedere al nodo successivo

Liste concatenate

Accesso ai nodi tramite riferimenti (puntatori)



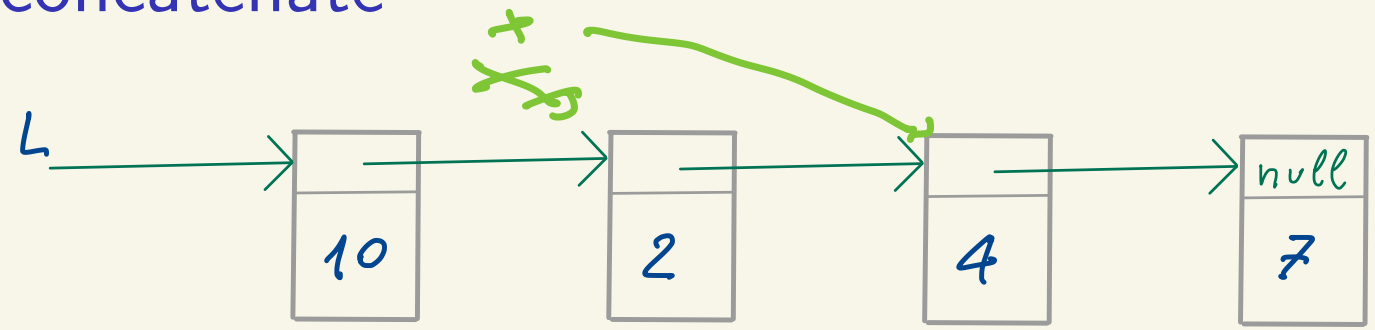
x ≡ riferimento al nodo

x.chiave ≡ campo chiave

x.pros ≡ riferimento a nodo successivo

null ≡ riferimento nullo

Liste concatenate

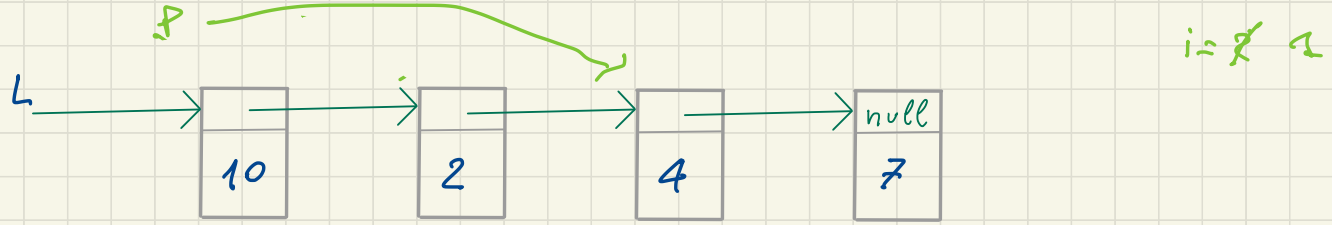


$x \leftarrow L.pos$

$x \leftarrow x.pos$

$y \leftarrow x.chiave$

Ricerca elemento in base alla posizione

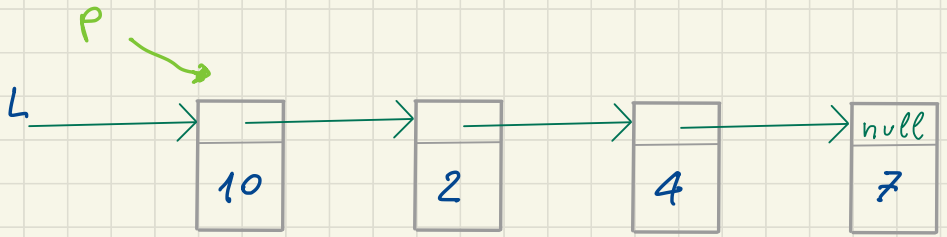


i = ~~2~~ 1

```
FUNZIONE elemento(Lista L, intero i) -> Nodo
    p ← L
    WHILE p ≠ null AND i > 0 DO
        p ← p.pros
        i ← i - 1
    RETURN p
```

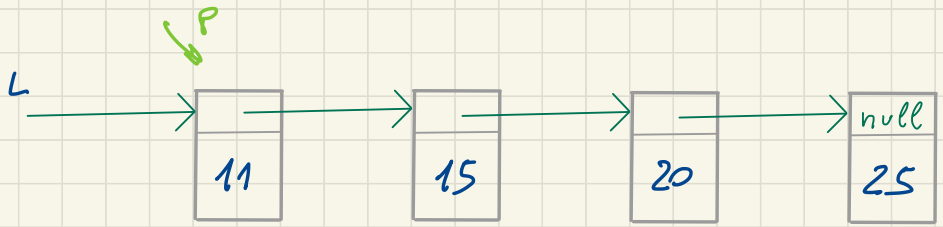
Ricerca elemento in base alla chiave

k [2,




```
FUNZIONE trova (Lista L, tipoChiave K) → Nodo
    p ← L
    WHILE p ≠ null AND p.chiave ≠ K DO
        p ← p.pros
    RETURN p
```

Ricerca elemento in base alla chiave in una lista ORDINATA




```
FUNZIONE trova (ListaOrdinata L, tipoChiave K) -> Nodo
    p ← L
    WHILE p ≠ null AND p.chiave < K DO
        p ← p.pros
    IF p = null OR p.chiave > K THEN
        RETURN null
    ELSE RETURN p
```

Inserimento in una lista ordinata