STRUTTURE DATI
-LISTE-

Stack - LIFO

Insieme dinamico dove l'elemento che viene rimosso è predeterminato
LIFO (Last In - First out): l'elemento inserito per ultimo viene cancellato

	1	2	3	4	5	6	7
S	15	6	2	9			

$S.top = 4$

```
STACK-EMPTY(S)
if S.top == 0
    return TRUE
else return FALSE
```

```
PUSH(S, x)
S.top = S.top + 1
S[S.top] = x
```

```
POP(S)
if STACK-EMPTY(S)
    error "underflow"
else S.top = S.top - 1
    return S[S.top + 1]
```

le precedenti funzioni hanno tutte complessità $O(1)$

Operazioni su Stack

- push = Inserisce un elemento sulla testa dello stack
- pop = Elimina un elemento sulla testa dello stack e ne libera la memoria
- is_empty = Controlla se è vuota

Code - FIFO

Le code utilizza la metodologia FIFO (First in - First out) ed è caratterizzato da head e tail.

- L'elemento aggiunto viene messo alla fine della lista (tail)
- L'elemento rimosso viene preso dalla testa della lista (head)

	1	2	3	4	5	6	7	8	9	10	11	12
Q							15	6	9	8	4	

$Q.head = 7$ $Q.tail = 12$

```
ENQUEUE(Q, x)
Q[Q.tail] = x
if Q.tail == Q.length
    Q.tail = 1
else Q.tail = Q.tail + 1
```

```
DEQUEUE(Q)
x = Q[Q.head]
if Q.head == Q.length
    Q.head = 1
else Q.head = Q.head + 1
return x
```

Operazioni su code

- enqueue = Inserisce un nuovo nodo alla fine della Coda
- dequeue = Elimina un elemento dalla testa

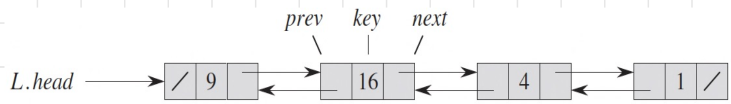
Liste concatenate

Gli elementi sono posizionati linearmente e l'ordine non è determinato da indici ma da un puntatore

puntatori: variabili il cui valore è un indirizzo di memoria

Una lista concatenata ha un attributo chiave key e altri due attributi next e prev e può avere più forme: può essere singolarmente o doppiamente concate.

- Lista singolarmente concatenata: si omette il puntatore prev



Se la lista è ordinata, il suo ordine lineare corrisponde all'ordine lineare delle chiavi memorizzate

- Elemento minimo: testa della lista
- Elemento massimo: coda della lista

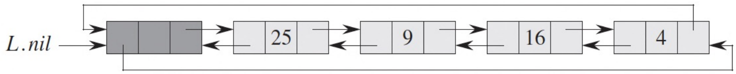
In una lista non ordinata gli elementi possono essere in qualsiasi posizione

```
LIST-INSERT(L, x)
x.next = L.head
if L.head ≠ NIL
    L.head.prev = x
L.head = x
x.prev = NIL
```

```
LIST-DELETE(L, x)
if x.prev ≠ NIL
    x.prev.next = x.next
else L.head = x.next
if x.next ≠ NIL
    x.next.prev = x.prev
```

```
LIST-SEARCH(L, k)
x = L.head
while x ≠ NIL and x.key ≠ k
    x = x.next
return x
```

In una lista circolare, il puntatore prev punta alla coda e il puntatore next punta alla testa



Sentinella: oggetto fittizio che ci consente di semplificare le condizioni al contorno, permette ad esempio di trasformare una lista doppiamente concatenata in una lista circolare doppiamente concatenata con sentinella

```
LIST-INSERT'(L, x)
x.next = L.nil.next
L.nil.next.prev = x
L.nil.next = x
x.prev = L.nil
```

```
LIST-DELETE'(L, x)
x.prev.next = x.next
x.next.prev = x.prev
```

```
LIST-SEARCH'(L, k)
x = L.nil.next
while x ≠ L.nil and x.key ≠ k
    x = x.next
return x
```

operazioni di insert - delete - search con l'uso delle sentinelle

Quicksort

Ordinamento in loco con:

caso pessimo: $\Theta(n^2)$

caso medio: $\Theta(n \lg n)$

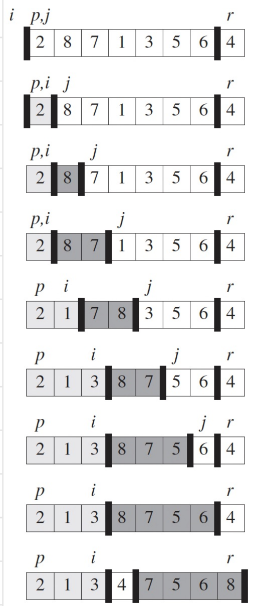
caso ottimo: $\Theta(n \lg n)$

pur essendo al caso peggiore $\Theta(n^2)$ è spesso utilizzato grazie alle sue due altre casistiche ottimali

Divide: partizionare l'array $A[p \mathinner{..} r]$ in due sottoarray $A[p \mathinner{..} q-1]$ e $A[q+1 \mathinner{..} r]$ (eventualmente vuoti) tali che ogni elemento di $A[p \mathinner{..} q-1]$ sia minore o uguale ad $A[q]$ che, a sua volta, è minore o uguale a ogni elemento di $A[q+1 \mathinner{..} r]$. Calcolare l'indice q come parte di questa procedura di partizionamento.

Impera: ordinare i due sottoarray $A[p \mathinner{..} q-1]$ e $A[q+1 \mathinner{..} r]$ chiamando ricorsivamente quicksort.

Combina: poiché i sottoarray sono già ordinati, non occorre alcun lavoro per combinarli: l'intero array $A[p \mathinner{..} r]$ è ordinato.




```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)
    QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        scambia A[i] con A[j]
scambia A[i + 1] con A[r]
return i + 1
```

con chiamata:

QUICKSORT(A, 1, A.length)