

Inserimento in una lista ordinata

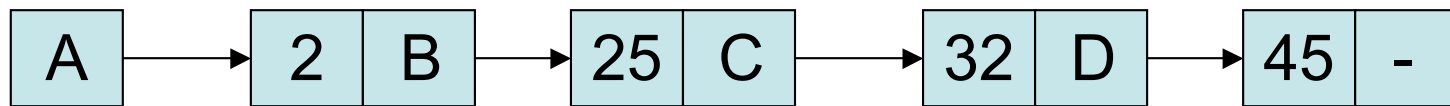
Vogliamo inserire un nuovo elemento in una lista in cui gli elementi sono memorizzati in ordine crescente:

Passo 1: creare un nuovo nodo della lista

Passo 2: trovare il punto di inserimento

Passo 3: realizzare l'inserimento
modificare i link in modo opportuno

Esempio di inserimento

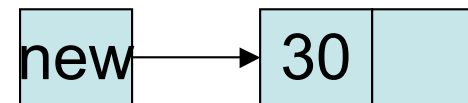


Poichè $A \rightarrow \text{elem} = 2 < \text{val} = 30$,

$B \rightarrow \text{elem} = 25 < \text{val}$ ma

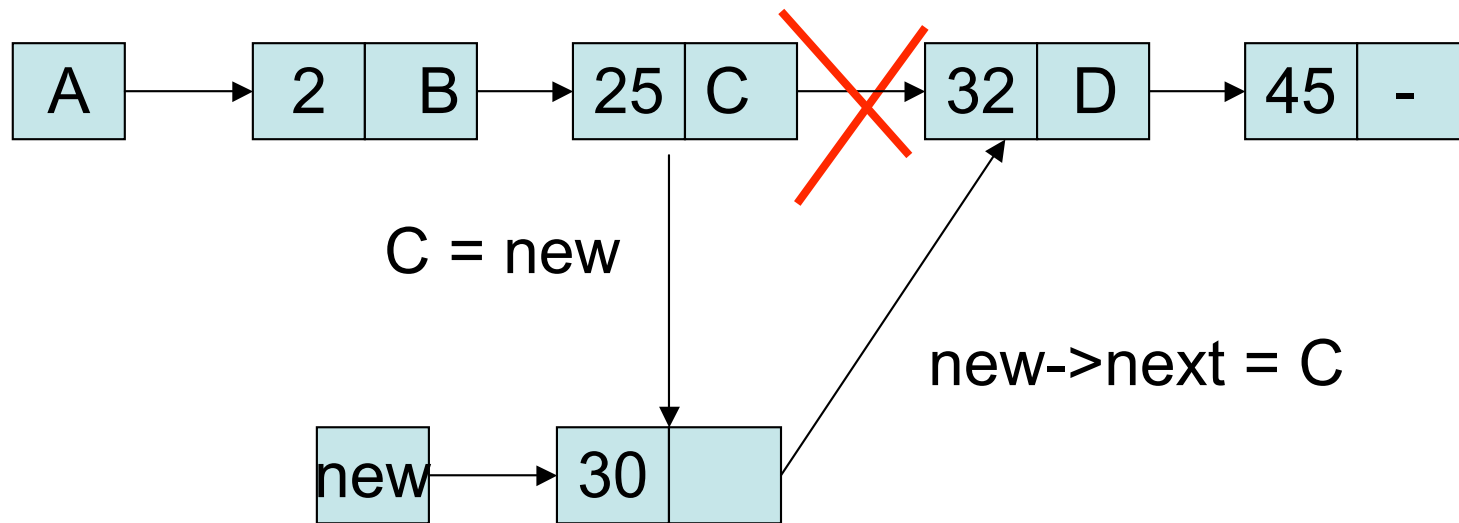
$C \rightarrow \text{elem} = 32 \geq \text{val}$

il punto giusto è tra C e D.



Creiamo il nodo da inserire:

Esempio inserimento 2



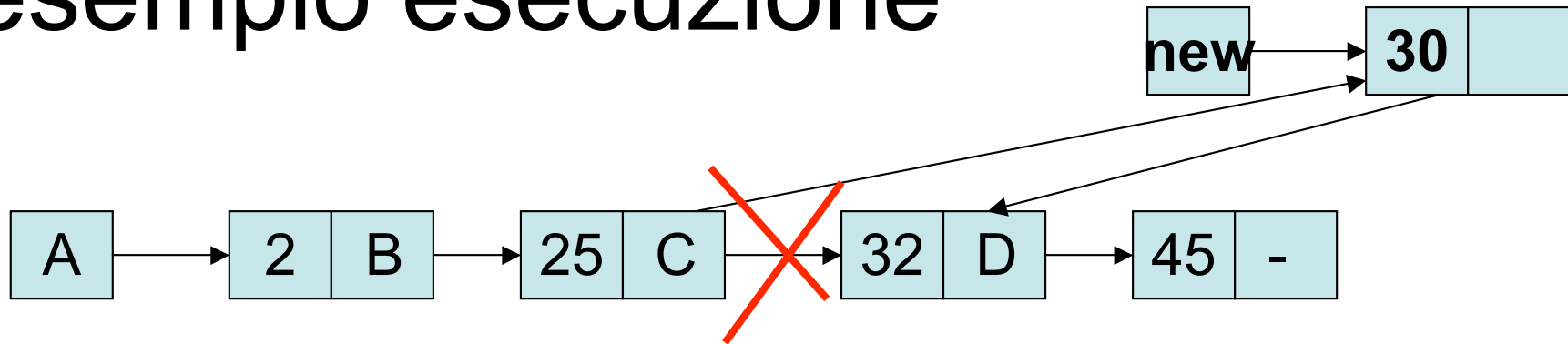
Ricorsione su liste concatenate:

l'esempio dell'inserimento di un elemento in una lista ordinata

versione1

```
ListaPtr insListOrd (ListaPtr L, int val)
{/*versione ricorsiva dell'inserimento in una lista
ordinata
prec: La lista deve essere ordinata in ordine crescente*/
postc: inserisce val nella prima posizione consistente con
l'ordine */
ListaPtr new;
if (L == NULL || (L)->elem >= val)
    {new = malloc(sizeof(Lista));
    assert(new);
    new->elem = val;
    new->next = L;
    return new;}
else L -> next = insListOrd(L->next, val);
return L;}
```

esempio esecuzione



```
insListOrd (A, 30)
/*L=A, L->elem < val L->next = B*/
  A -> next = insListOrd (B, 30)
/*L = B, L->elem < val, L->next = C*/
  B -> next = insListOrd (C, 30)
/*L=C, L->elem > val*/
```

Qui A->next = B

Qui B->next = new

**Restituisce new,
con new->next = C,**

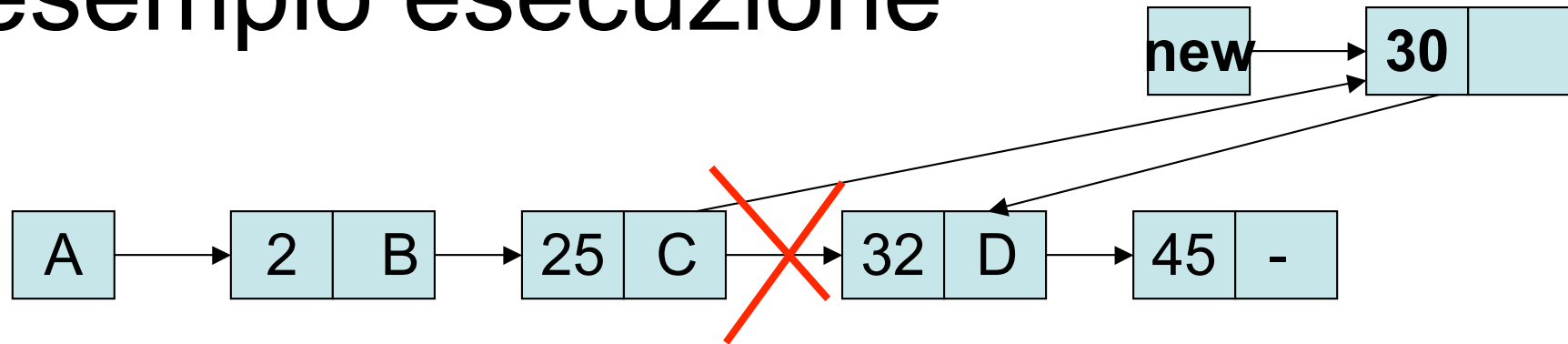
```
if (L == NULL || (L)->elem >= val)
{new = malloc(sizeof(Lista));
  assert(new);
  new->elem = val;
  new->next = L;
  return new;}
else L -> next = insListOrd(L->next, val);
return L;}
```

Versione iterativa dell'inserimento di un elemento in una lista ordinata

versione1

```
ListaPtr insListOrdIt (ListaPtr L, int val)
{ /* versione iterativa dell'inserimento in una lista ordinata
prec: La lista deve essere ordinata in ordine crescente
postc: inserisce val nella prima posizione consistente con
l'ordine */
ListaPtr temp = L, prec = NULL, new;
new = malloc(sizeof(Lista));
assert(new);
new->elem = val;
while (temp != NULL && temp -> elem < val)
{ prec = temp; temp = temp -> next; }
if (prec != NULL)
{ prec->next = new; new -> next = temp; }
else { new -> next = L; L = new; }
return L; }
```

esempio esecuzione



```
insListOrdIt (A, 30)
```

```
/*temp=L=A,prec = NULL,temp->elem=2 < val temp->next = B*/
```

```
prec = A , temp = B
```

```
/*temp = B,prec = A, B->elem = 25< val, temp->next = C*/
```

```
prec = B , temp = C
```

```
/*temp=C,prec = B, C->elem = 32 > val*/
```

```
while (temp != NULL && temp -> elem < val)
{prec = temp;temp = temp ->next; }
if (prec != NULL)
{prec->next = new; new -> next = temp;}
else { new -> next = L; L = new; }
return L;}
```

**Uscita ciclo prec !=NULL
prec -> next (che è C)
diventa new
e new ->next = temp
(che è C)**

Confronto ricorsione iterazione

- Tempo:
 - Ricorsione: si esegue, oltre ai confronti necessari per trovare il punto di inserimento, un numero lineare nel numero degli elementi (caso peggiore) di assegnamenti (al rientro dalle chiamate)
 - Iterazione: si esegue un numero costante di assegnamenti (lo stesso numero di confronti del caso ricorsivo)
- Spazio:
 - Ricorsione: si occupa uno spazio di memoria per le chiamate lineare nel numero degli elementi (caso peggiore)
 - Iterazione: si occupa uno spazio di memoria costante

Ricorsione su liste concatenate:

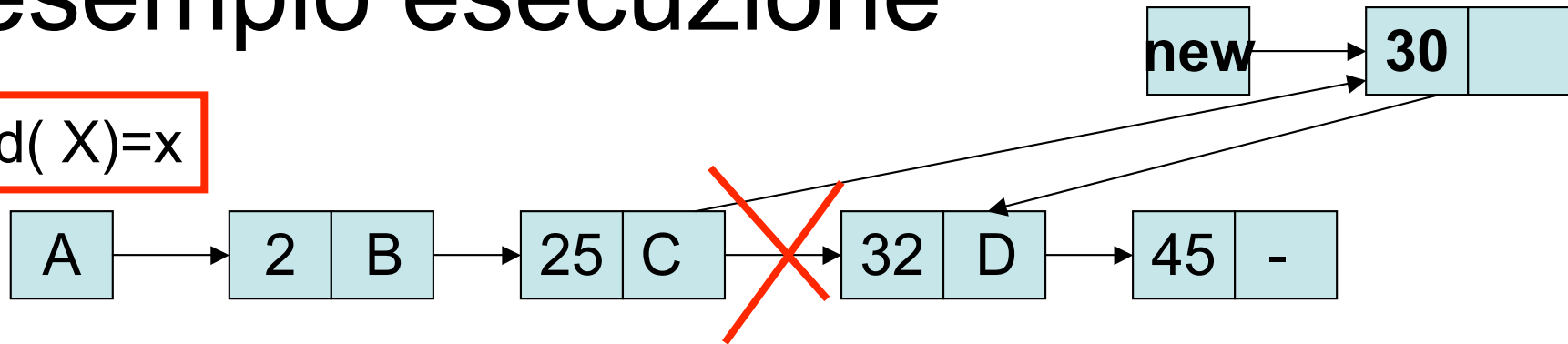
l'esempio dell'inserimento di un elemento in una lista ordinata

versione2

```
void insListOrd2 (ListaPtr *L, int val)
{ /* versione ricorsiva dell'inserimento in una lista ordinata
  prec: La lista deve essere ordinata in ordine crescente
  postc: inserisce val nella prima posizione consistente con
        l'ordine */
  ListaPtr new;
  if (*L == NULL || (*L)->elem >= val)
    { new = malloc(sizeof(Lista));
      assert(new);
      new->elem = val;
      new->next = *L;
      *L = new; }
  else insListOrd2(&(*L)->next, val); }
```

esempio esecuzione

Ind(X)=x



```
insListOrd2 (a, 30)
```

```
/* L=a, *L = A, A->elem=2 < val */
```

```
insListOrd2 (b, 30)
```

```
/* L=b, *L = B, B->elem = 25 < val*/
```

```
insListOrd2 (c, 30)
```

```
/* L=c, *L = C, C->elem > val*/
```

```
if (*L == NULL || (*L)->elem >= val)
{new = malloc(sizeof(Lista));
  assert(new);
  new->elem = val;
  new->next = *L;
  *L = new;}
else insListOrd2(&(*L)->next, val);}
```

**new->next = C,
C = new**

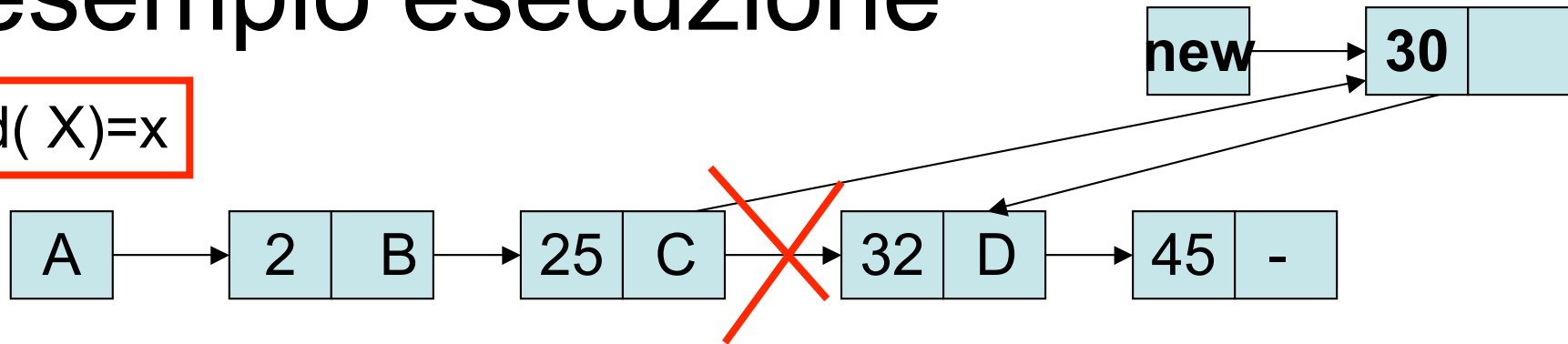
Versione iterativa dell'inserimento di un elemento in una lista ordinata

versione2

```
void insListOrdIt2 (ListaPtr *L, int val)
{/*versione iterativa dell'inserimento in una lista
  ordinata
prec: La lista deve essere ordinata in ordine crescente
postc: inserisce val nella prima posizione consistente
  con l'ordine */
ListaPtr prec = NULL, temp = *L, new;
new = malloc(sizeof(Lista));
assert(new);
new->elem = val;
while (temp != NULL && temp->elem < val)
{prec = temp; temp = temp->next;}
if (prec != NULL)
{prec->next = new; new -> next = temp;}
else { new -> next = *L; *L = new; }
}
```

esempio esecuzione

Ind(X)=x



```
insListOrdIt2 (a, 30)
```

```
/*temp=*L=A, prec = NULL, temp->elem=2 < val temp->next = B*/
```

```
prec = A , temp = B
```

```
/*temp = B, prec = A, B->elem = 25< val, temp->next = C*/
```

```
prec = B , temp = C
```

```
/*temp=C,prec = B, C->elem = 32 > val*/
```

```
temp = *L
...
while (temp != NULL && temp->elem <val)
{prec = temp;temp = temp->next;}
if (prec != NULL)
{prec->next = new; new -> next = temp;}
else { new -> next = *L; *L = new; }
}
```

**Uscita ciclo prec !=NULL
B -> next (= C) diventa new
e new ->next = temp (=C)**

Confronto ricorsione iterazione, seconda versione

- **Tempo:**
 - Sono sostanzialmente equivalenti
- **Spazio:**
 - Ricorsione: si occupa uno spazio di memoria per le chiamate lineare nel numero degli elementi (caso peggiore)
 - Iterazione: si occupa uno spazio di memoria costante
- **Ma la funzione è ricorsiva in coda! Quindi può essere trasformata nell'iterativa equivalente a tempo di compilazione, risparmiando al programmatore la gestione dei puntatori di scorrimento della lista.**