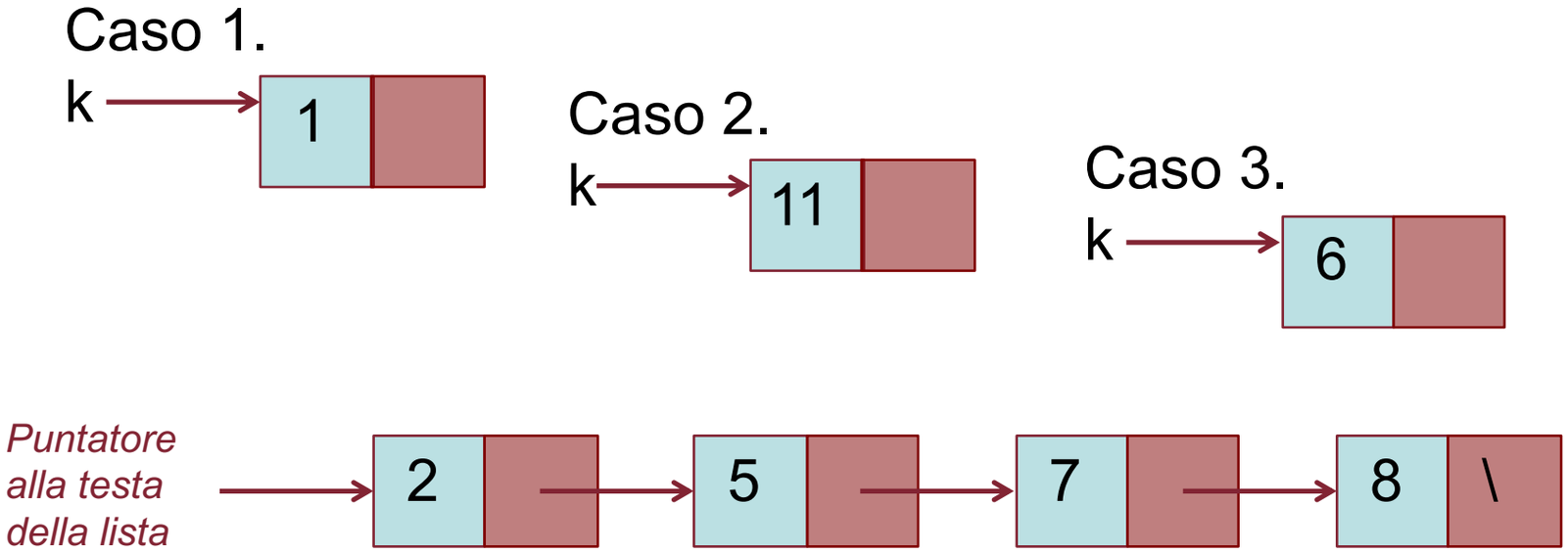
Esercizio svolto:

Esercizio. Data in input una lista ordinata di interi tramite il puntatore al primo elemento, ed un elemento da inserire, aggiungere tale elemento alla lista in modo da rispettare l'ordinamento.

Soluzione.



Esercizio svolto:

```
fun inserisci_inOrd(p: punt. a testa,
                    k: punt. ad elem. da inserire):
   if p == NULL:
         return k
   if k->key ≤ p->key:
         #l'elemento va al primo posto
         k->next = p
         return k
   p1 = p
   p2 = p1->next
   while p2≠NULL AND p2->key ≤ k->key:
         p1 =p2
         p2 =p1->next
   # k va posizionato tra p1 e p2
   p1->next = k
   k->next = p2
   return p
```

Le liste sono strutture dati inerentemente ricorsive. Pertanto, tutti gli algoritmi proposti possono essere implementati sia in versione iterativa che ricorsiva.
Spesso la versione ricorsiva è più compatta, consentendo di non dover distinguere i vari casi (inserimento in testa, in coda o in un punto intermedio della lista).
La funzione di inserimento in una lista ordinata ne è un esempio:

```
fun inserisci_inOrd_Ric(p: punt. alla testa,
                        k: punt. ad elem. da inserire):
    if p == NULL OR k->key ≤ p->key:
       # k deve diventare  la testa della lista
       k->next ← p
       return k
    p->next = inserisci_inOrd_Ric(p->next,k)
    return p
```

Esercizio svolto:

```
fun inserisci_inOrd(p: punt. a testa,
                    k: punt. ad elem. da inserire):
   if p = NULL:
         return k
   if k->key ≤ p->key:
         #l'elemento va al primo posto
         k->next = p
         return k
   p1 = p
   p2 = p1->next
   while p2≠NULL AND p2->key ≤ k->key:
         p1 =p2
         p2 =p1->next
   # k va posizionato tra p1 e p2
   p1->next = k
   k->next = p2
   return p
```