Module EA4 – Éléments d'Algorithmique

Dominique Poulalhon
dominique.poulalhon@liafa.univ-paris-diderot.fr

Université Paris Diderot
L2 Informatique, Math-Info et EIDD
Année universitaire 2013-2014

Première interrogation lundi 17 février

durée : 1h15

- Amphi 10 E : groupes Info 1, Info 2 et Info 3 (A-K)
  début à 11h30 précises
- Amphi 12 E : groupes Info 3 (L-Z), Info 4 et Math-Info
  début à 11h45

Supprimer les doublons dans une liste

`sans_doublons(L)`

Étant donné une liste `L`, construire une liste contenant une et une seule occurrence de chaque élément apparaissant dans `L`

Supprimer les doublons dans une liste

sans_doublons(L)

Étant donné une liste L, construire une liste contenant une et une seule occurrence de chaque élément apparaissant dans L

```
def sans_doublons(L) :
  res = []
  for elt in L :
    if not recherche(elt, res) : res += [elt]
  return res
```

Supprimer les doublons d'un tableau *trié*

`sans_doublons(T)`

Étant donné un tableau `T` *trié*, construire un tableau contenant une et une seule occurrence de chaque élément apparaissant dans `T`

Supprimer les doublons d'un tableau *trié*

sans_doublons(T)

Étant donné un tableau T *trié*, construire un tableau contenant une et une seule occurrence de chaque élément apparaissant dans T

```
def sans_doublons(T) :
  tmp, res = None, []
  for elt in T :
    if elt != tmp : tmp, res = elt, res + [elt]
  return res
```

Trier une liste

`tri(L)`

Étant donné une liste `L` d'éléments comparables, construire la liste des éléments de `L` classés en ordre croissant

Trier une liste

`tri(L)`

Étant donné une liste `L` d'éléments comparables, construire la liste des éléments de `L` classés en ordre croissant

`tri_en_place(L)`

Étant donné une liste `L` d'éléments comparables, réordonner les éléments de `L` en ordre croissant

tri(L)

Étant donné une liste `L` d'éléments comparables, construire la liste des éléments de `L` classés en ordre croissant

```
def tri_selection(L) :
  res = []
  while(L != []) :
    res += [supprime_minimum(L)]
  return res
```

tri_en_place(L)

Étant donné une liste L d'éléments comparables, réordonner les éléments de L en ordre croissant

```
def tri_selection(T) :
  for i in range(len(T)) :
    min = indice_minimum(T, i)
      # indice du plus petit élément de T[i:]
    T[i], T[min] = T[min], T[i]
  return T
```

tri(L)

Étant donné une liste L d'éléments comparables, construire la liste des éléments de L classés en ordre croissant

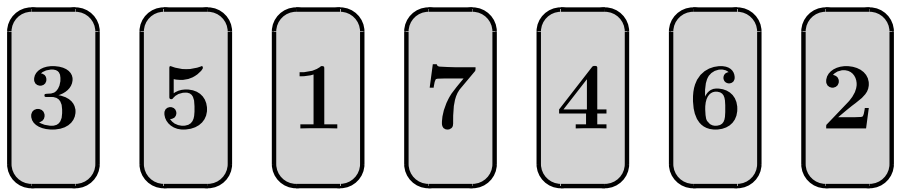
Taille de l'entrée

= longueur de la liste

Opérations élémentaires prises en compte

- comparaisons entre éléments de la liste
- échanges d'éléments de la liste

Exemple :




```
def tri_insertion(L) :
  res = []
  for elt in L : insertion_triee(elt, res)
  return res
```

Tri par insertion

Exemple :


```
def tri_insertion(L) :
  res = []
  for elt in L : insertion_triee(elt, res)
  return res
```

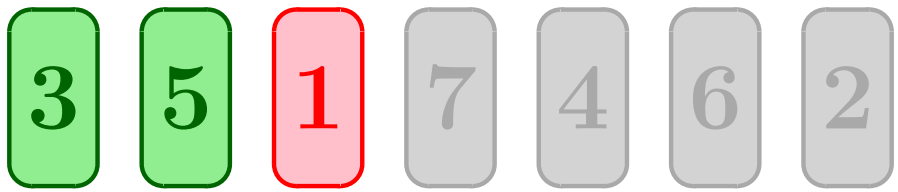
Tri par insertion

Exemple :


```
def tri_insertion(L) :
  res = []
  for elt in L : insertion_triee(elt, res)
  return res
```

Tri par insertion

Exemple :




```
def tri_insertion(L) :
  res = []
  for elt in L : insertion_triee(elt, res)
  return res
```

Tri par insertion

Exemple :


```
def tri_insertion(L) :
  res = []
  for elt in L : insertion_triee(elt, res)
  return res
```

Tri par insertion

Exemple :


```
def tri_insertion(L) :
  res = []
  for elt in L : insertion_triee(elt, res)
  return res
```

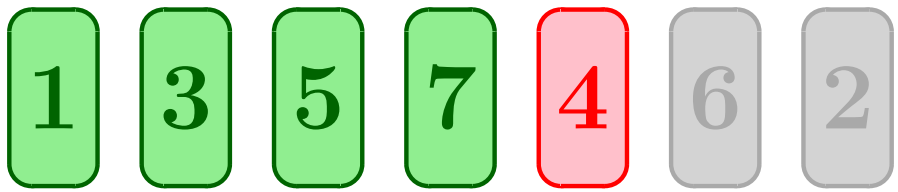
Tri par insertion

Exemple :


```
def tri_insertion(L) :
  res = []
  for elt in L : insertion_triee(elt, res)
  return res
```

Tri par insertion

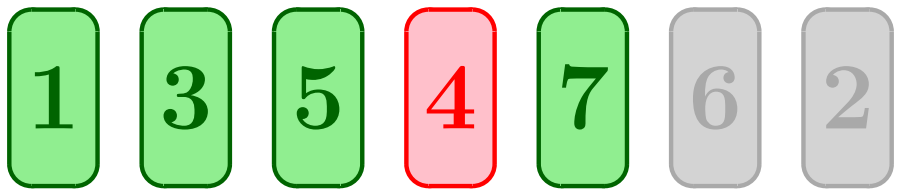
Exemple :




```
def tri_insertion(L) :
  res = []
  for elt in L : insertion_triee(elt, res)
  return res
```

Tri par insertion

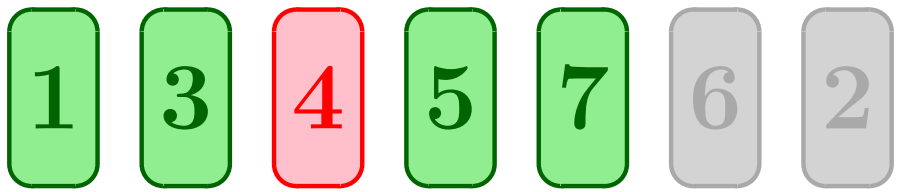
Exemple :




```
def tri_insertion(L) :
  res = []
  for elt in L : insertion_triee(elt, res)
  return res
```

Tri par insertion

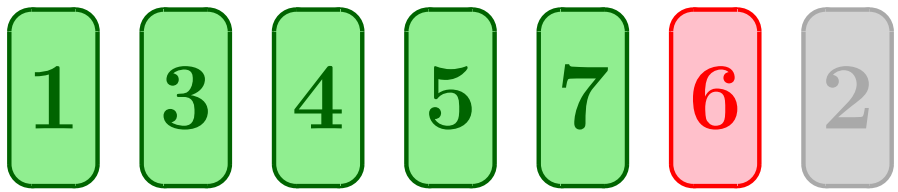
Exemple :




```
def tri_insertion(L) :
  res = []
  for elt in L : insertion_triee(elt, res)
  return res
```

Tri par insertion

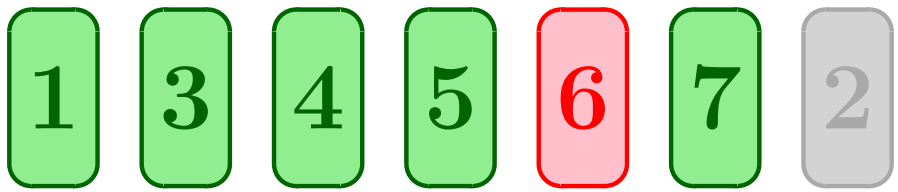
Exemple :




```
def tri_insertion(L) :
  res = []
  for elt in L : insertion_triee(elt, res)
  return res
```

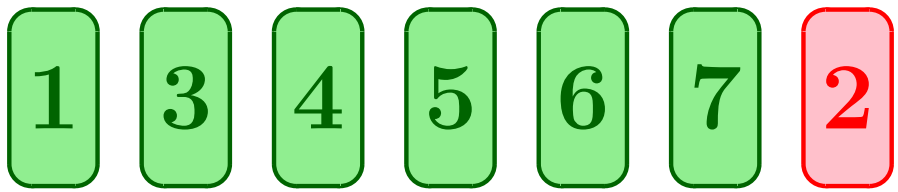
Tri par insertion

Exemple :




```
def tri_insertion(L) :
  res = []
  for elt in L : insertion_triee(elt, res)
  return res
```

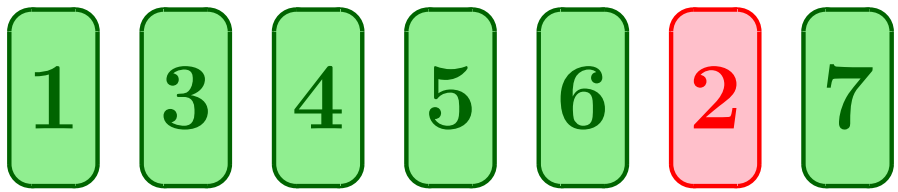
Exemple :




```
def tri_insertion(L) :
  res = []
  for elt in L : insertion_triee(elt, res)
  return res
```

Tri par insertion

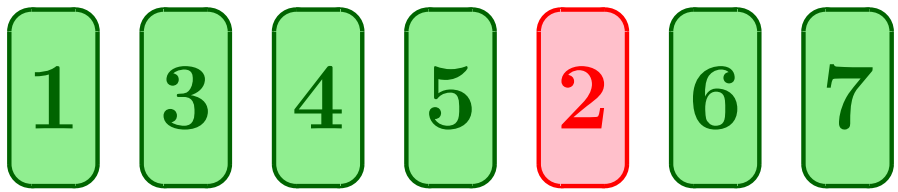
Exemple :




```
def tri_insertion(L) :
  res = []
  for elt in L : insertion_triee(elt, res)
  return res
```

Tri par insertion

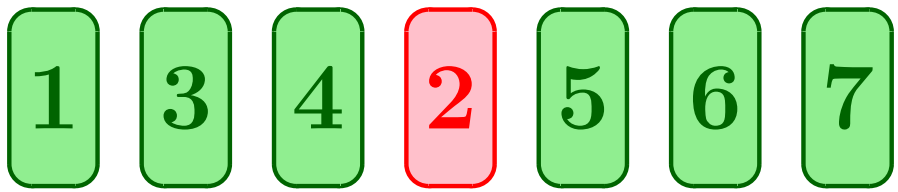
Exemple :




```
def tri_insertion(L) :
  res = []
  for elt in L : insertion_triee(elt, res)
  return res
```

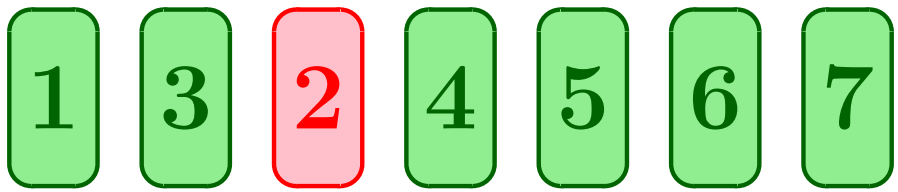
Tri par insertion

Exemple :




```
def tri_insertion(L) :
  res = []
  for elt in L : insertion_triee(elt, res)
  return res
```

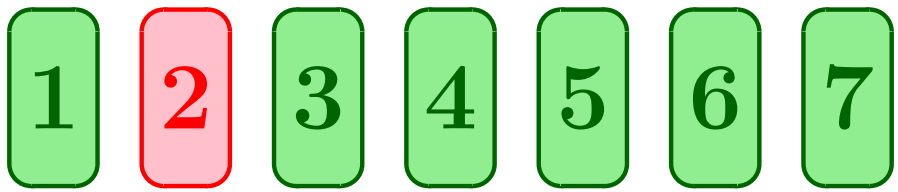
Exemple :




```
def tri_insertion(L) :
  res = []
  for elt in L : insertion_triee(elt, res)
  return res
```

Tri par insertion

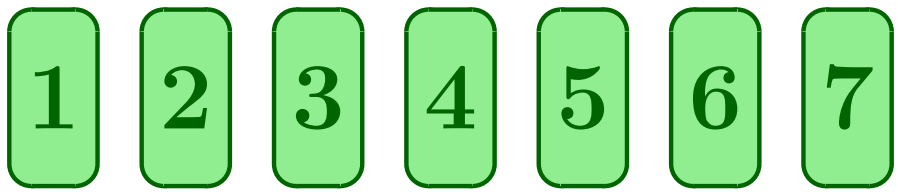
Exemple :




```
def tri_insertion(L) :
  res = []
  for elt in L : insertion_triee(elt, res)
  return res
```

Tri par insertion

Exemple :




```
def tri_insertion(L) :
  res = []
  for elt in L : insertion_triee(elt, res)
  return res
```

Tri par insertion

```
def tri_insertion(L) :
  res = []
  for elt in L : insertion_triee(elt, res)
  return res

def insertion_triee(x, L) :
  for elt in L :
    if x < elt : break
  ## insertion de x avant elt dans L
  return res
```

```
def insertion_triee(x, L) :
  for elt in L :
    if x < elt : break
  ## insertion de x avant elt dans L
  return res
```

Cas d'une liste chaînée

insertion par modification du chaînage

Cas d'un tableau

insertion par déplacements multiples

```
def insertion_triee(x, L) :
  for elt in L :
    if x < elt : break
  ## insertion de x avant elt dans L
  return res
```

Cas d'une liste chaînée

insertion par modification du chaînage $\Longrightarrow$ coût constant

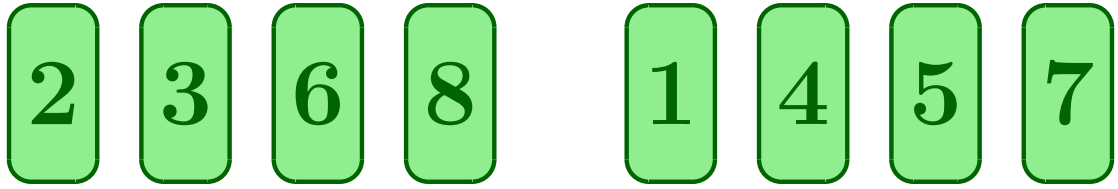
Cas d'un tableau

insertion par déplacements multiples $\Longrightarrow$ coût linéaire

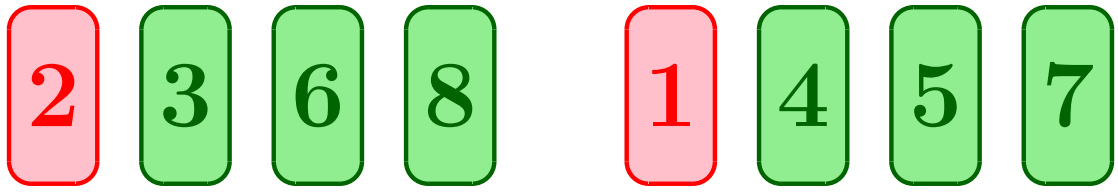
Tri par insertion dans un tableau

```
def tri_insertion(T) :
  for i in range(1, len(T)) :
    for j in range(i) :
      if T[i-j-1] > T[i-j] :
        T[i-j-1], T[i-j] = T[i-j], T[i-j-1]
      else : break
  return T
```

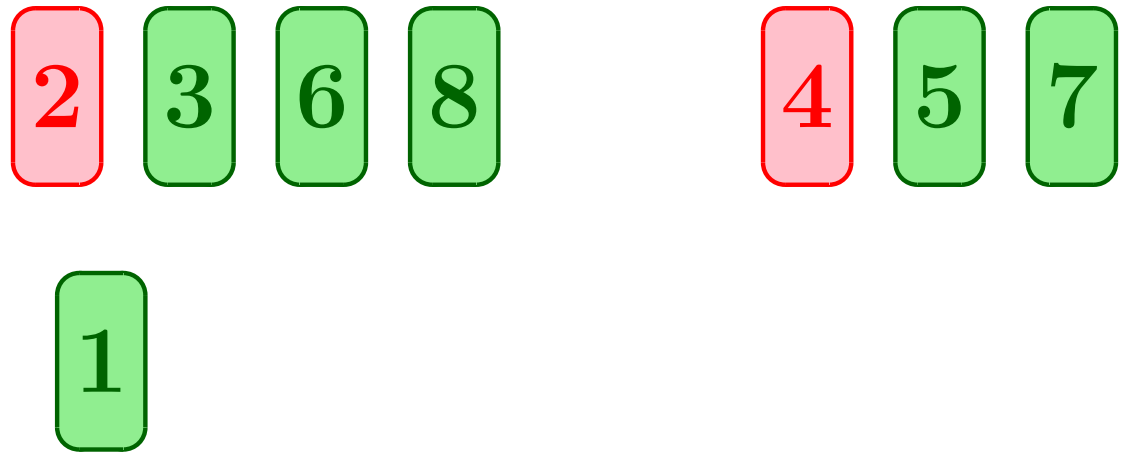
Une étape : la fusion de listes triées



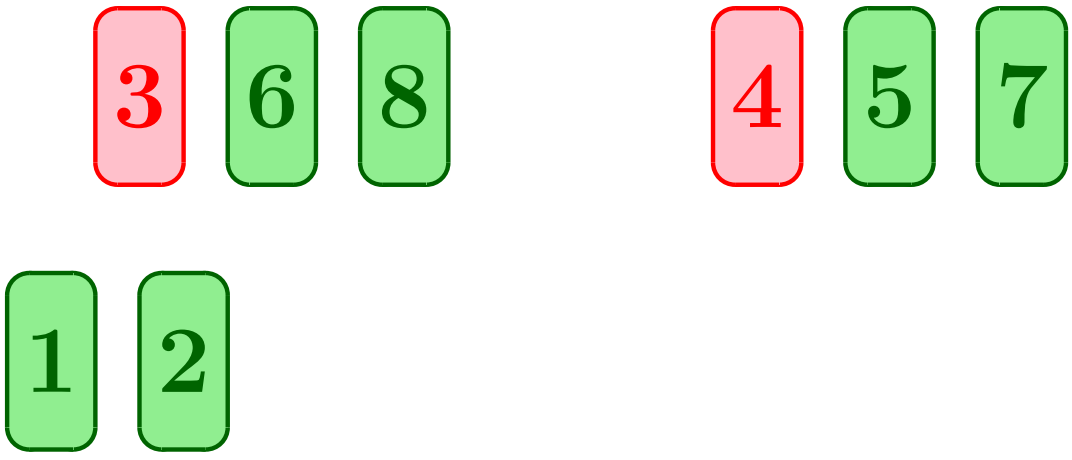
Une étape : la fusion de listes triées



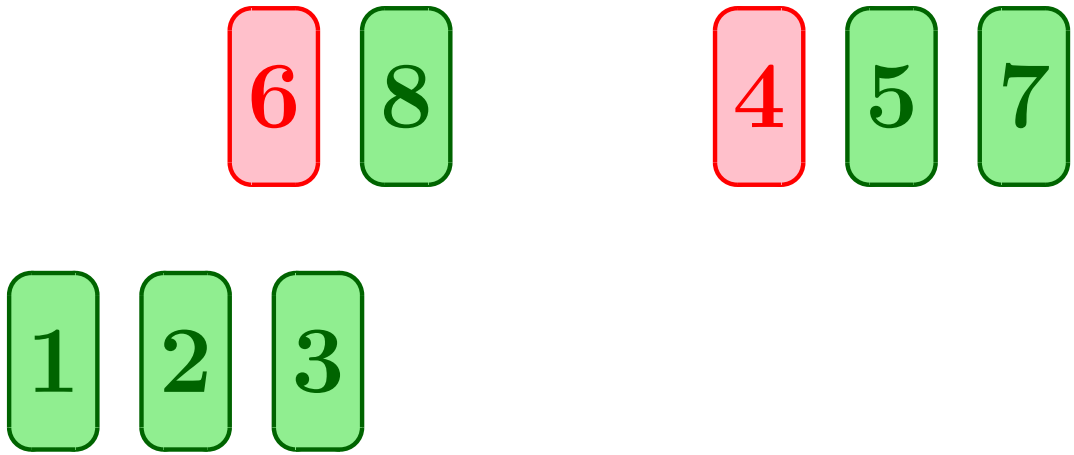
Une étape : la fusion de listes triées



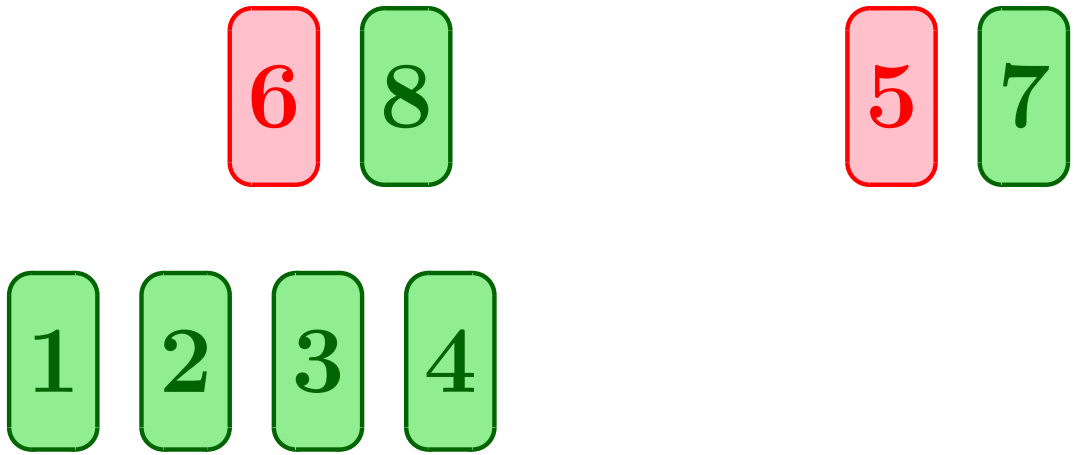
Une étape : la fusion de listes triées



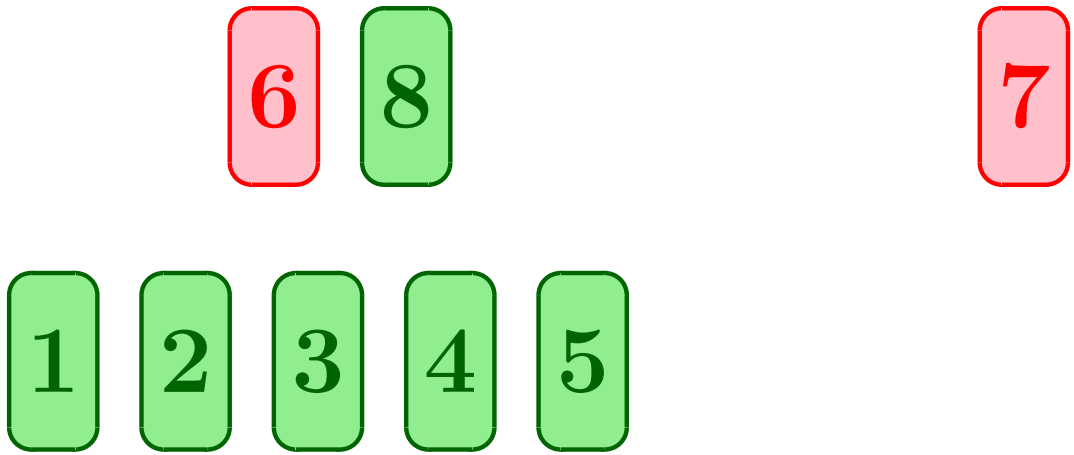
Une étape : la fusion de listes triées



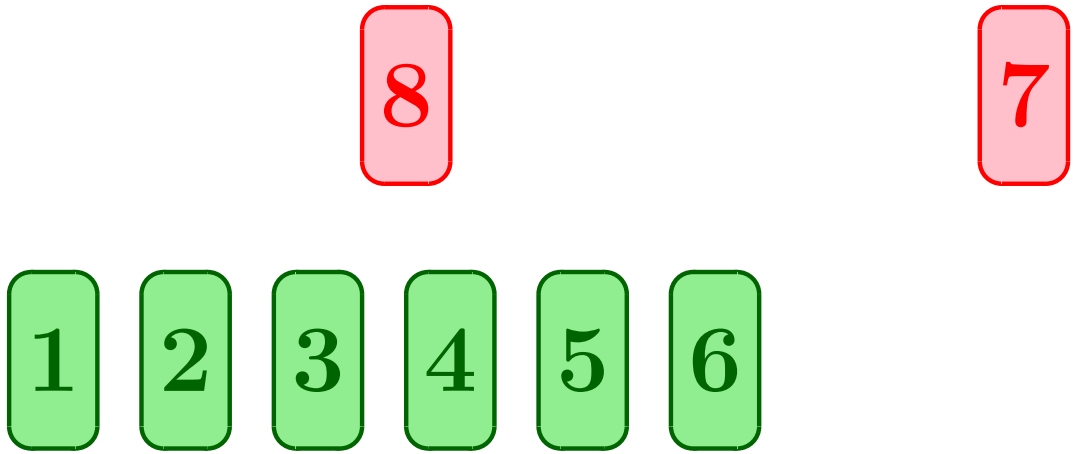
Une étape : la fusion de listes triées



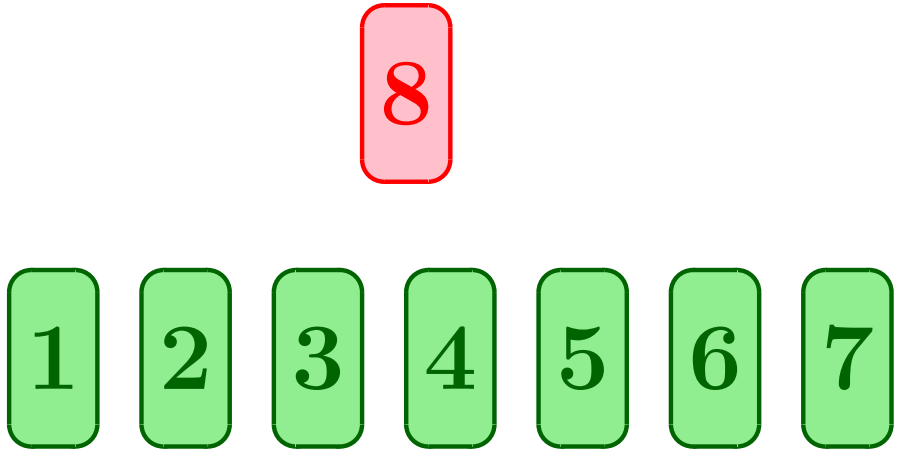
Une étape : la fusion de listes triées



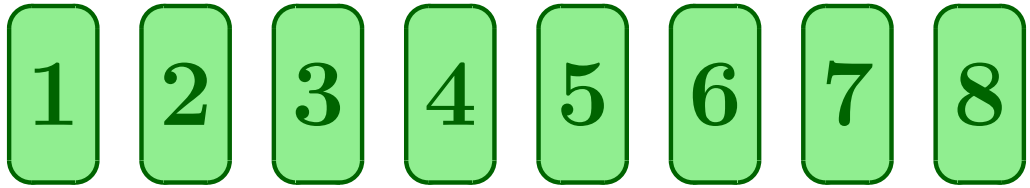
Une étape : la fusion de listes triées



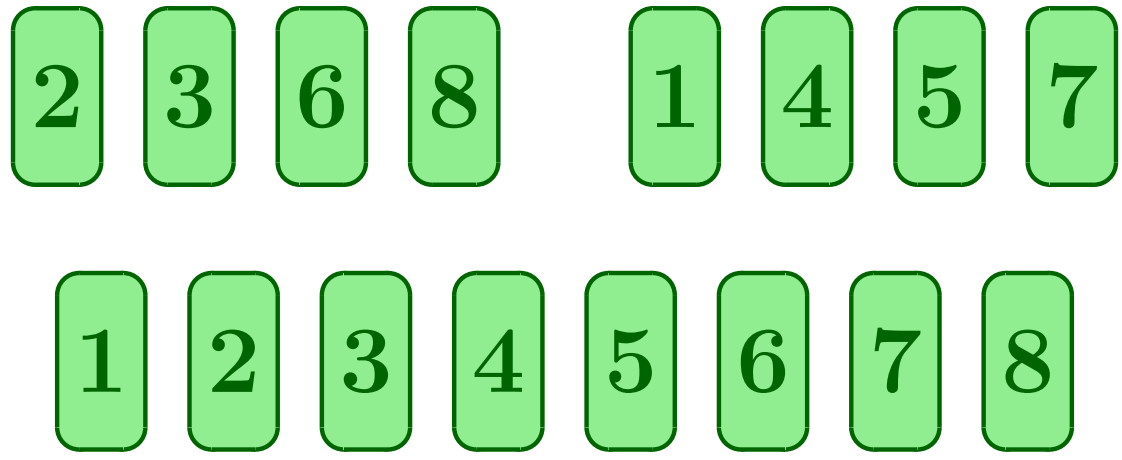
Une étape : la fusion de listes triées



Une étape : la fusion de listes triées



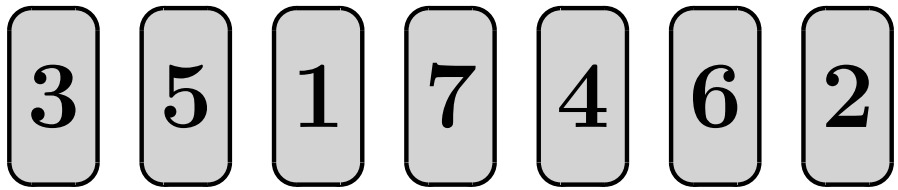
Une étape : la fusion de listes triées



Une étape : la fusion de listes triées

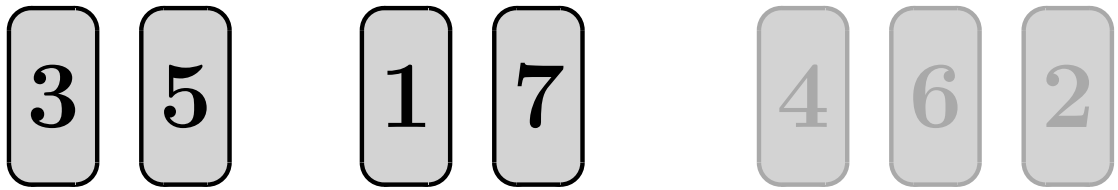
```
def fusion(L1, L2) :
  if len(L1) == 0 : return L2
  elif len(L2) == 0 : return L1
  elif L1[0] < L2[0] : return [L1[0]] + fusion(L1[1:], L2)
  else : return [L2[0]] + fusion(L1, L2[1:])
```

Exemple :



Exemple :

Exemple :

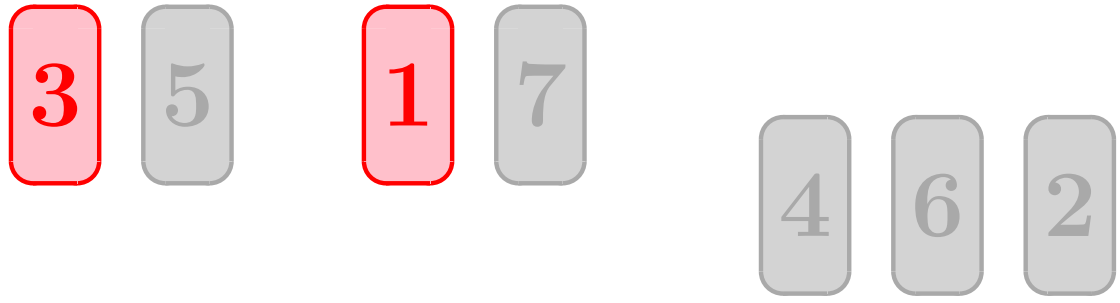


Exemple :

Exemple :

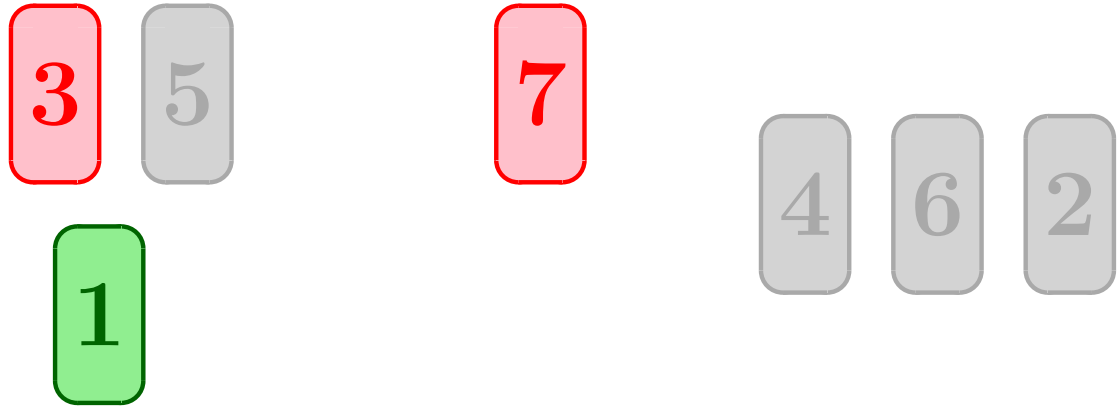
Exemple :

Exemple :



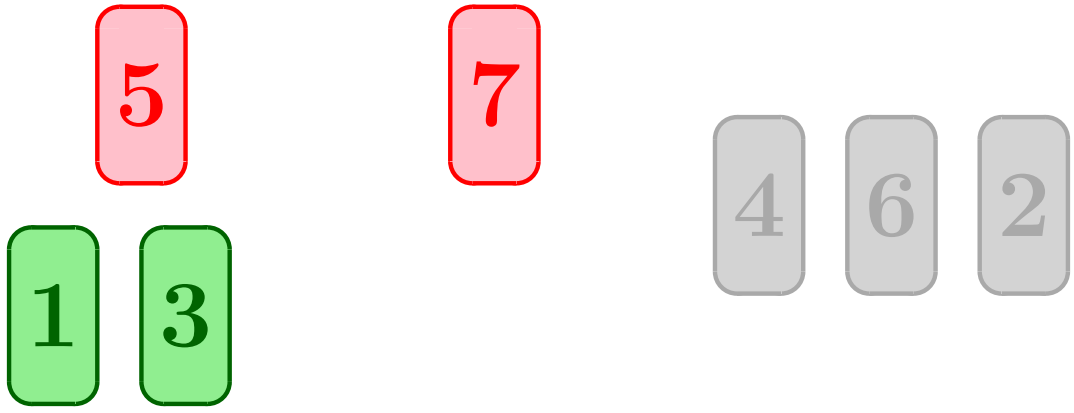
Tri par fusion

Exemple :

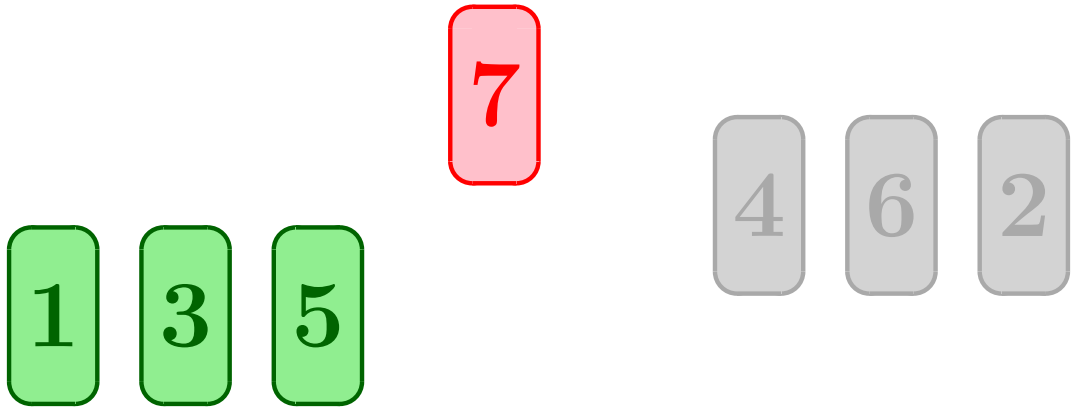


Tri par fusion

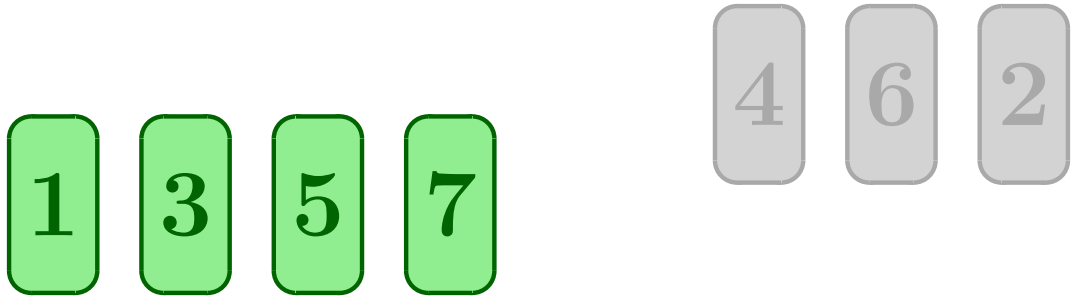
Exemple :



Exemple :



Exemple :



Exemple :

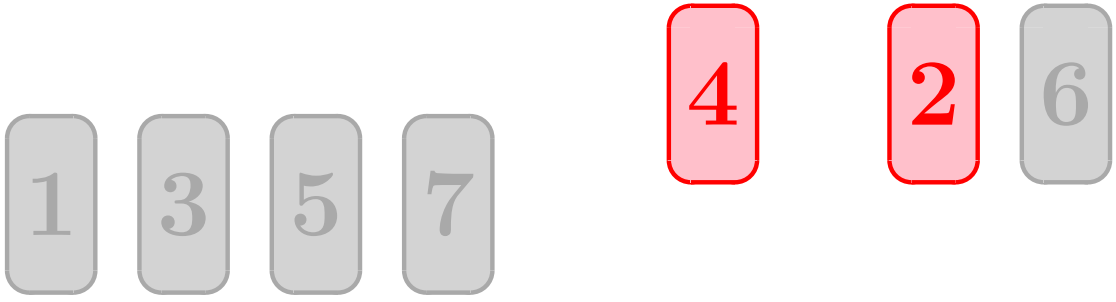
Exemple :

Exemple :

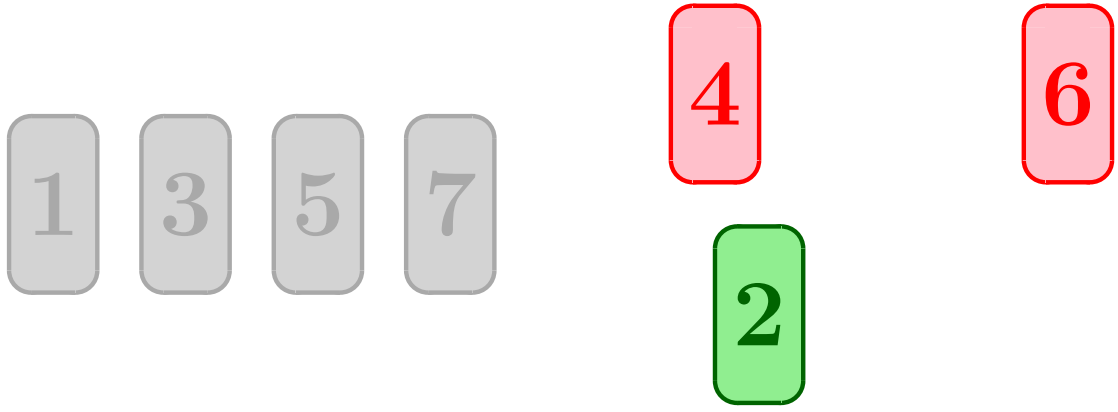
Exemple :

Exemple :

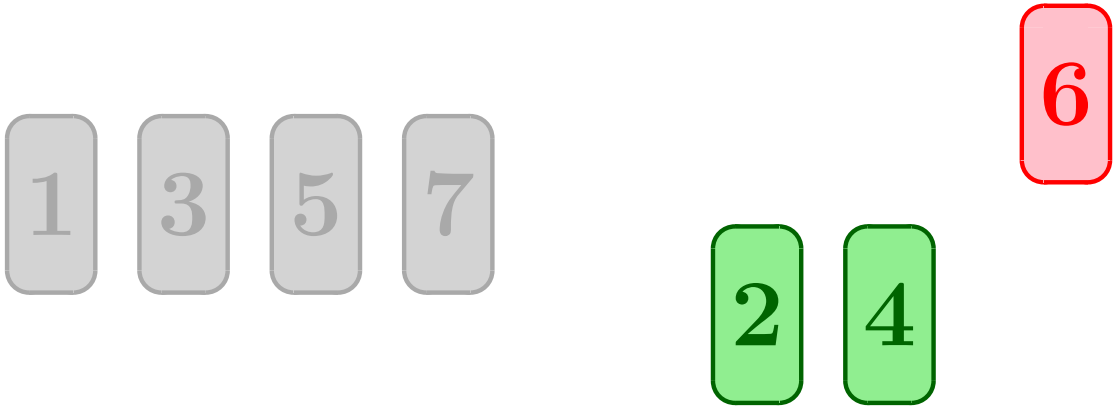
Exemple :



Exemple :



Exemple :

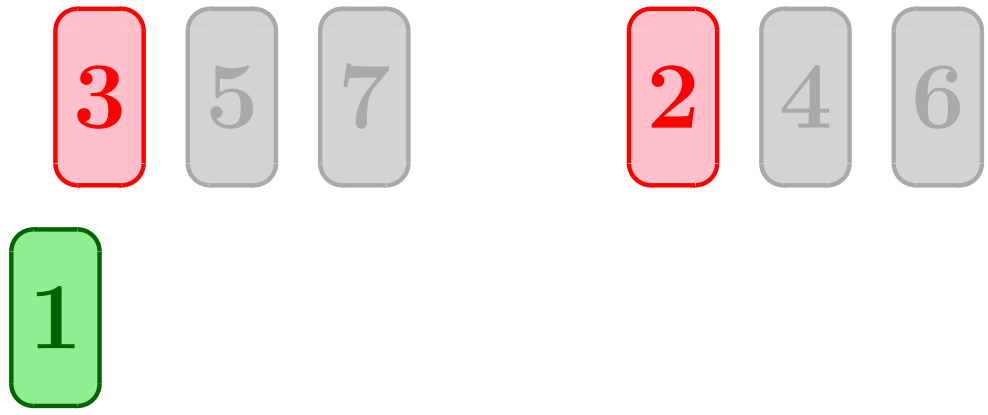


Exemple :

Exemple :

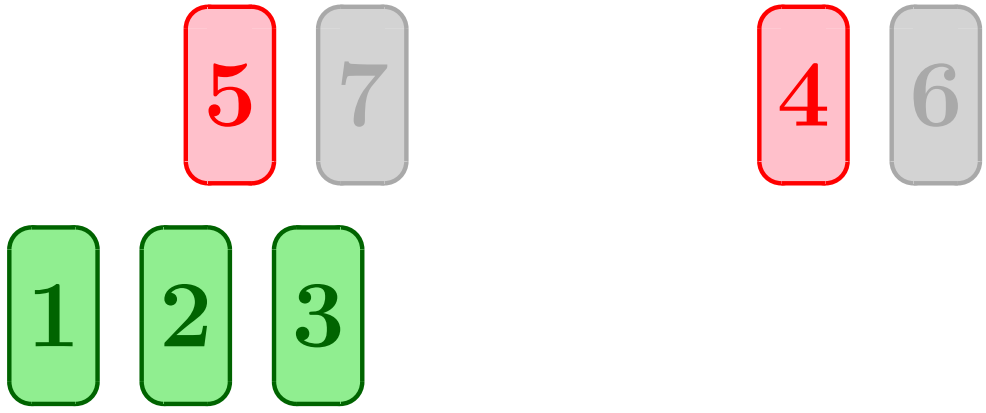
Exemple :

Exemple :

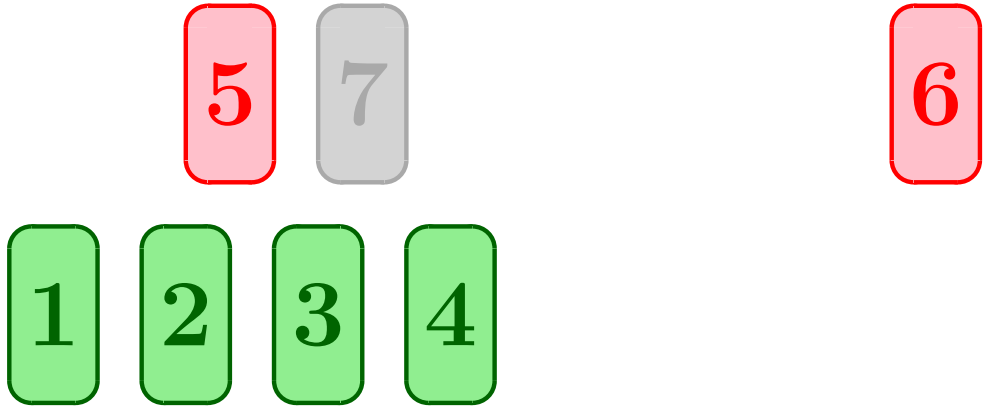


Exemple :

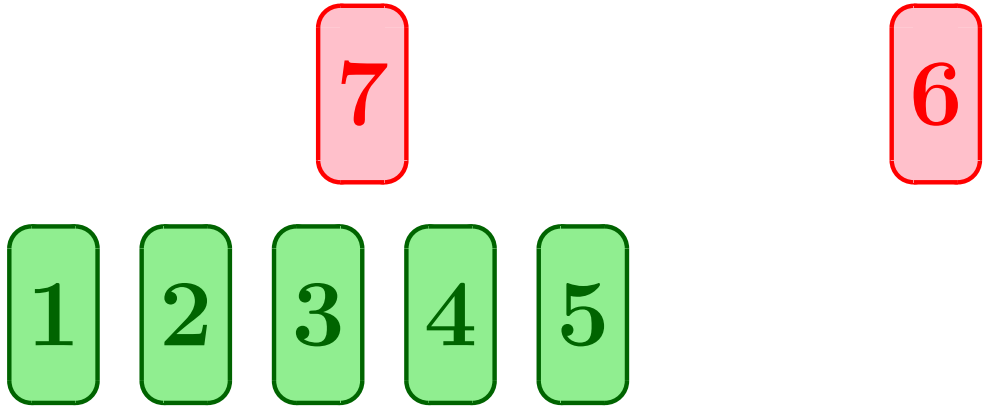
Exemple :



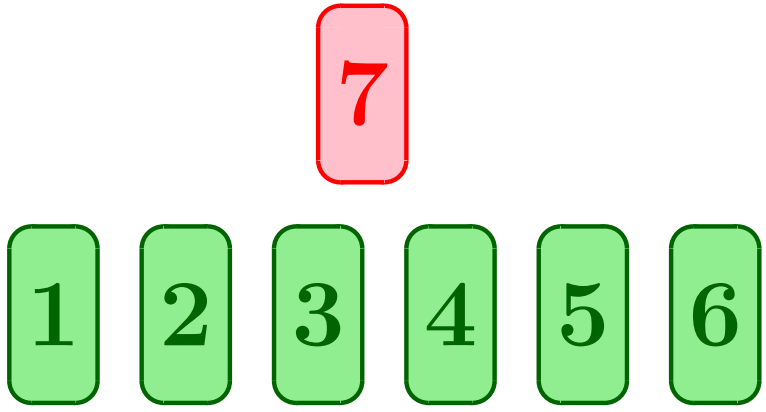
Exemple :



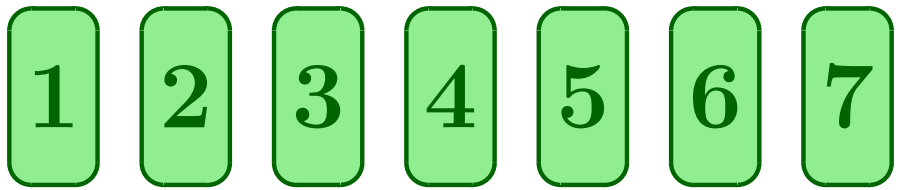
Exemple :



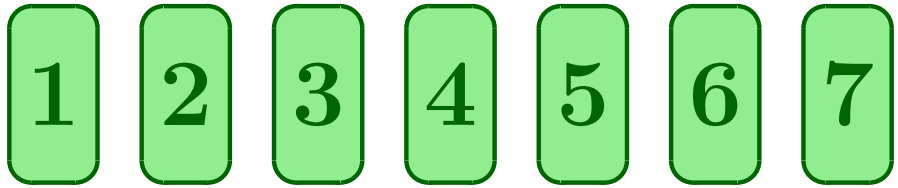
Exemple :



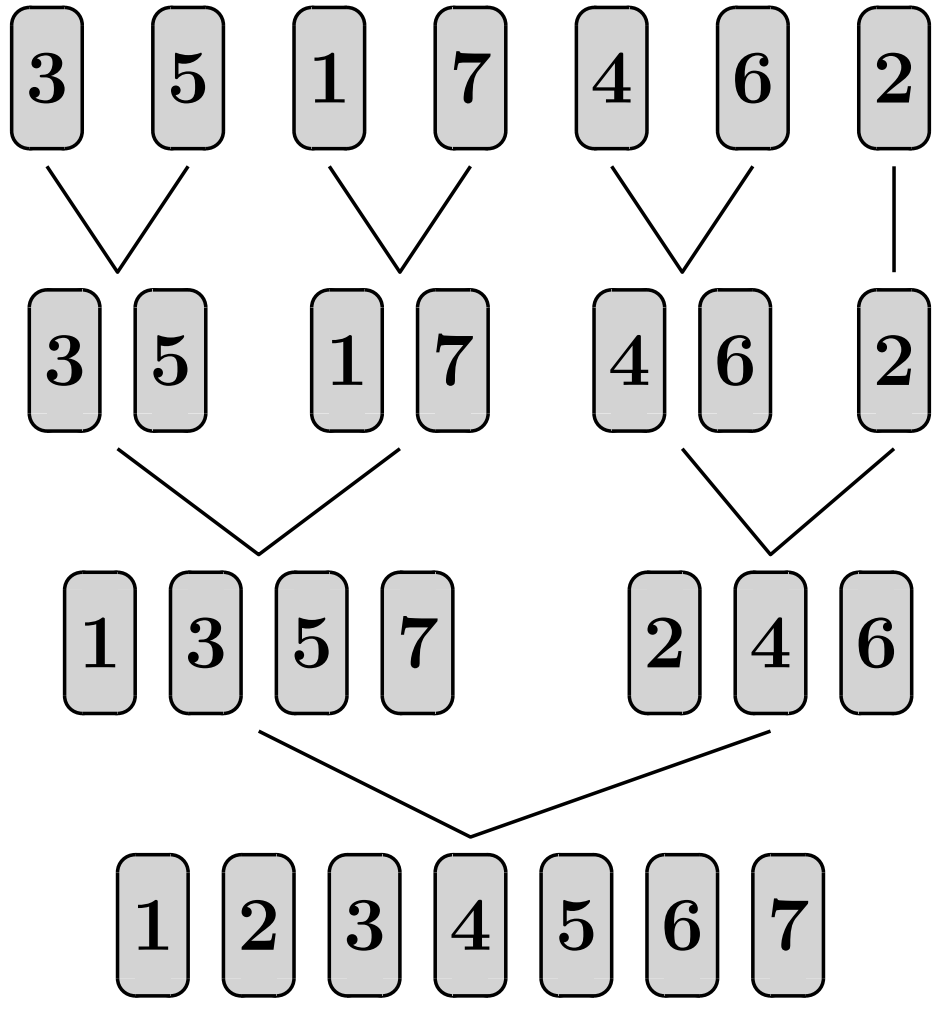
Exemple :



Exemple :



Exemple :



Tri par fusion

```
def tri_fusion(T, debut, fin) :
  if fin - debut < 2 : return T[debut:fin]
  else :
    milieu = (debut + fin)//2
    gauche = tri_fusion(T, debut, milieu)
    droite = tri_fusion(T, milieu, fin)
    return fusion(gauche, droite)
```