# 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

#### Contrôle continu

# 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 Ttrié, 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  $tri\acute{e}$ , 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
```

# tri(L)

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

# 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:

 $\boxed{3\ 5\ 1\ 7\ 4\ 6\ 2}$ 

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

```
3 5 1 7 4 6 2
```

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

```
3 5 1 7 4 6 2
```

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

```
3 5 1 7 4 6 2
```

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

```
3 1 5 7 4 6 2
```

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

```
1 3 5 7 4 6 2
```

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

```
1 3 5 7 4 6 2
```

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

```
1 3 5 7 4 6 2
```

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

```
1 3 5 4 7 6 2
```

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

```
1 3 4 5 7 6 2
```

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

```
1345762
```

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

```
1345672
```

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

```
1 3 4 5 6 7 2
```

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

```
1 3 4 5 6 2 7
```

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

```
1 3 4 5 2 6 7
```

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

```
1 3 4 2 5 6 7
```

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

```
1 3 2 4 5 6 7
```

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

```
1 2 3 4 5 6 7
```

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

```
1 2 3 4 5 6 7
```

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

```
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</pre>
```

## 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</pre>
```

## Cas d'une liste chaînée

insertion par modification du chaînage

 $\implies$  coût constant

## Cas d'un tableau

insertion par déplacements multiples

 $\implies$  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
```

#### TRI PAR FUSION

Une étape : la fusion de listes triées

2 3 6 8 1 4 5 7

#### TRI PAR FUSION

Une étape : la fusion de listes triées

**2 3 6 8 1 4 5 7** 

Une étape : la fusion de listes triées

3

 $oxed{1}$ 

Une étape : la fusion de listes triées

3

6

8

 $oldsymbol{4}$ 

5

7

 $oxed{1}oxed{2}$ 

Une étape : la fusion de listes triées

6

 $oldsymbol{4} oldsymbol{5}$ 



 $oxed{1}$ 

Une étape : la fusion de listes triées

 $\mathbf{6}$   $\mathbf{8}$ 

5 7

 $oxed{1} oxed{2} oxed{3} oxed{4}$ 

Une étape : la fusion de listes triées

 $\mathbf{6}$   $\mathbf{8}$ 

7

 $egin{array}{c} 1 \ 2 \ 3 \ 4 \ 5 \end{array}$ 

Une étape : la fusion de listes triées

8

7

 $oxed{1} oxed{2} oxed{3} oxed{4} oxed{5} oxed{6}$ 

Une étape : la fusion de listes triées

8

 $\boxed{1}\ \boxed{2}\ \boxed{3}\ \boxed{4}\ \boxed{5}\ \boxed{6}\ \boxed{7}$ 

Une étape : la fusion de listes triées

1 2 3 4 5 6 7 8

Une étape : la fusion de listes triées

2 3 6 8 1 4 5 7

 $\boxed{1}\boxed{2}\boxed{3}\boxed{4}\boxed{5}\boxed{6}\boxed{7}\boxed{8}$ 

```
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:])</pre>
```

## Exemple:

3 5 1 7 4 6 2

## Exemple:

3 5 1 7 4 6 2









Exemple:

3

5

1

 $\mathbf{4}$ 

3











# Exemple:

7

 $oxed{1}oxed{3}oxed{5}$ 







## Exemple:

1 3 5 7 4 6 2

# Exemple:

 1
 3
 5
 7
 4
 6
 2

# Exemple:

 1
 3
 5
 7
 4
 6
 2

# Exemple:

 1
 3
 5
 7
 4
 2
 6

# Exemple:

1

3

5 7

 $|\mathbf{4}|$ 

 $\mathbf{2}$ 

6

# Exemple:







2

# Exemple:

 $\boxed{1}\boxed{3}\boxed{5}\boxed{7}$ 

 $\mathbf{6}$ 

 $oldsymbol{2}igg|igg(oldsymbol{4}$ 

# Exemple:

1 3 5 7

 $oxed{2}oxed{4}oxed{6}$ 

## Exemple:

1 3 5 7 2 4 6

Exemple:

 $oldsymbol{1}$   $oldsymbol{3}$   $oldsymbol{5}$   $oldsymbol{7}$ 

2 4

6

# Exemple:

**3** 5 7

 $\mathbf{2}\Big]\Big[4$ 

6

1







# Exemple:





 $oxed{1} oxed{2} oxed{3}$ 

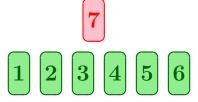
# Exemple:





 $oxed{1}$ 

# Exemple:



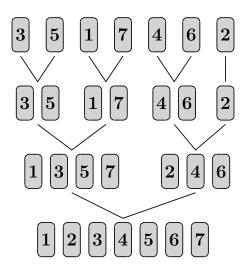
## Exemple:

1 2 3 4 5 6 7

#### Tri par fusion

## Exemple:

1 2 3 4 5 6 7



```
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)</pre>
```