Module EA4 – Éléments d'Algorithmique II

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

Université Paris Diderot L2 Informatique Année universitaire 2014-2015

Interrogation $n^{\circ} 1$:

en TD la semaine la première semaine de mars

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(L)

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(L)

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
```

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

Exemple:

3 5 1 7 4 6 2















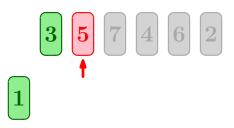


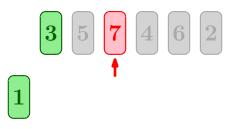


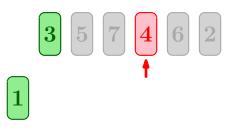


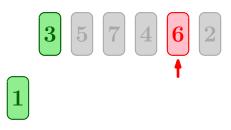






















Exemple:



 $oxed{2}$

Exemple:

5 7 4 6

 $egin{bmatrix} m{1} \end{bmatrix} m{2} m{3}$

Exemple:



 $egin{bmatrix} 1 \ \end{bmatrix} egin{bmatrix} 2 \ \end{bmatrix} egin{bmatrix} 3 \ \end{bmatrix}$

Exemple:

5 [7]

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

Exemple:



 $egin{pmatrix} oldsymbol{1} egin{pmatrix} oldsymbol{2} oldsymbol{3} egin{pmatrix} oldsymbol{4} \end{pmatrix} egin{pmatrix} oldsymbol{4} \end{pmatrix}$

Exemple:



 $\begin{bmatrix} 1 \end{bmatrix} \begin{bmatrix} 2 \end{bmatrix} \begin{bmatrix} 3 \end{bmatrix} \begin{bmatrix} 4 \end{bmatrix}$

Exemple:



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

Exemple:



 $egin{bmatrix} 1 \ 2 \ 3 \ 4 \ 5 \ \end{bmatrix}$

Exemple:

7

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

Exemple:

1234567

Tri par sélection

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 != []) :
    m = minimum(L)
    L.remove(m)
    res.append(m)
  return res
```

Tri par sélection

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
```

TRIER UNE LISTE

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
```

Exemple:

3 5 1 7 4 6 2

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

Exemple:

 $oxed{3}$

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

Exemple:

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

 $oxed{3}$

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

Exemple:



3 5

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

Exemple:

 $oxed{3} oxed{5}$

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

Exemple:

7 4 6 2



```
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
```

Exemple:

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



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

Exemple:

4 6 2

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

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

Exemple:

6 2

```
\begin{array}{|c|c|c|c|c|c|}\hline 1 & 3 & \color{red} \color{red} \color{red} \color{blue} \color{blu
```

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

Exemple:

6

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

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

Exemple:

2

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

Exemple:

2

1 3 4 5 6 7

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

Exemple:

 1
 2
 3
 4
 5
 6
 7

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

Exemple:

1234567

```
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, 0, -1) : #parcours de droite à gauche
    if T[j-1] > T[j] :
        T[j-1], T[j] = T[j], T[j-1]
    else : break
  return T
```



```
def tri_insertion(T) :
   for i in range(1, len(T)) :
    for j in range(i, 0, -1) : #parcours de droite à gauche
      if T[j-1] > T[j] :
        T[j-1], T[j] = T[j], T[j-1]
      else : break
   return T
```

TRI PAR INSERTION DANS UN TABLEAU



```
def tri_insertion(T) :
   for i in range(1, len(T)) :
    for j in range(i, 0, -1) : #parcours de droite à gauche
      if T[j-1] > T[j] :
        T[j-1], T[j] = T[j], T[j-1]
      else : break
   return T
```



```
def tri_insertion(T) :
   for i in range(1, len(T)) :
    for j in range(i, 0, -1) : #parcours de droite à gauche
      if T[j-1] > T[j] :
        T[j-1], T[j] = T[j], T[j-1]
      else : break
   return T
```



```
def tri_insertion(T) :
   for i in range(1, len(T)) :
    for j in range(i, 0, -1) : #parcours de droite à gauche
      if T[j-1] > T[j] :
        T[j-1], T[j] = T[j], T[j-1]
      else : break
   return T
```

TRI PAR INSERTION DANS UN TABLEAU



```
def tri_insertion(T) :
   for i in range(1, len(T)) :
    for j in range(i, 0, -1) : #parcours de droite à gauche
      if T[j-1] > T[j] :
        T[j-1], T[j] = T[j], T[j-1]
      else : break
   return T
```



```
def tri_insertion(T) :
   for i in range(1, len(T)) :
    for j in range(i, 0, -1) : #parcours de droite à gauche
      if T[j-1] > T[j] :
        T[j-1], T[j] = T[j], T[j-1]
      else : break
   return T
```

TRI PAR INSERTION DANS UN TABLEAU



```
def tri_insertion(T) :
   for i in range(1, len(T)) :
    for j in range(i, 0, -1) : #parcours de droite à gauche
      if T[j-1] > T[j] :
        T[j-1], T[j] = T[j], T[j-1]
      else : break
   return T
```



```
def tri_insertion(T) :
  for i in range(1, len(T)) :
    for j in range(i, 0, -1) : #parcours de droite à gauche
    if T[j-1] > T[j] :
        T[j-1], T[j] = T[j], T[j-1]
    else : break
  return T
```

TRI PAR INSERTION DANS UN TABLEAU



```
def tri_insertion(T) :
  for i in range(1, len(T)) :
    for j in range(i, 0, -1) : #parcours de droite à gauche
    if T[j-1] > T[j] :
        T[j-1], T[j] = T[j], T[j-1]
    else : break
  return T
```

TRI PAR INSERTION DANS UN TABLEAU



```
def tri_insertion(T) :
  for i in range(1, len(T)) :
    for j in range(i, 0, -1) : #parcours de droite à gauche
    if T[j-1] > T[j] :
        T[j-1], T[j] = T[j], T[j-1]
    else : break
  return T
```



```
def tri_insertion(T) :
  for i in range(1, len(T)) :
    for j in range(i, 0, -1) : #parcours de droite à gauche
    if T[j-1] > T[j] :
        T[j-1], T[j] = T[j], T[j-1]
    else : break
  return T
```



```
def tri_insertion(T) :
  for i in range(1, len(T)) :
    for j in range(i, 0, -1) : #parcours de droite à gauche
    if T[j-1] > T[j] :
        T[j-1], T[j] = T[j], T[j-1]
    else : break
  return T
```



```
def tri_insertion(T) :
  for i in range(1, len(T)) :
    for j in range(i, 0, -1) : #parcours de droite à gauche
    if T[j-1] > T[j] :
        T[j-1], T[j] = T[j], T[j-1]
    else : break
  return T
```

TRI PAR INSERTION DANS UN TABLEAU



```
def tri_insertion(T) :
   for i in range(1, len(T)) :
    for j in range(i, 0, -1) : #parcours de droite à gauche
      if T[j-1] > T[j] :
        T[j-1], T[j] = T[j], T[j-1]
      else : break
   return T
```



```
def tri_insertion(T) :
  for i in range(1, len(T)) :
    for j in range(i, 0, -1) : #parcours de droite à gauche
    if T[j-1] > T[j] :
        T[j-1], T[j] = T[j], T[j-1]
    else : break
  return T
```



```
def tri_insertion(T) :
  for i in range(1, len(T)) :
    for j in range(i, 0, -1) : #parcours de droite à gauche
    if T[j-1] > T[j] :
        T[j-1], T[j] = T[j], T[j-1]
    else : break
  return T
```



```
def tri_insertion(T) :
  for i in range(1, len(T)) :
    for j in range(i, 0, -1) : #parcours de droite à gauche
    if T[j-1] > T[j] :
        T[j-1], T[j] = T[j], T[j-1]
    else : break
  return T
```



```
def tri_insertion(T) :
  for i in range(1, len(T)) :
    for j in range(i, 0, -1) : #parcours de droite à gauche
    if T[j-1] > T[j] :
        T[j-1], T[j] = T[j], T[j-1]
    else : break
  return T
```

COMPLEXITÉ

Tri par sélection $\Theta(n^2)$ comparaisons dans tous les cas

Tri par insertion

 $\Theta(n^2)$ comparaisons au pire

Questions

- peut-on être plus précis pour le tri par insertion?
- peut-on faire mieux que $\Theta(n^2)$ au pire?