# Tris, 2

## Gagner en efficacité

Lycée Pierre Corneille – MP

2016-2017

# Diviser pour régner (*divide and conquer*)

**Patron de conception** (*design pattern*)

# Diviser pour régner (*divide and conquer*)

**Patron de conception** (*design pattern*)

Bonne pratique issue de l'expérience

# Diviser pour régner (*divide and conquer*)

**Patron de conception** (*design pattern*)

Bonne pratique issue de l'expérience

Formulation abstraite

# Diviser pour régner (*divide and conquer*)

**Patron de conception** (*design pattern*)

Bonne pratique issue de l'expérience

Formulation abstraite

Principe général de résolution de nombreux problèmes concrets

# Diviser pour régner (*divide and conquer*)

1. **Divide**

# Diviser pour régner (*divide and conquer*)

1. **Divide**
   - Données de petite taille : traitement direct

# Diviser pour régner (*divide and conquer*)

1. **Divide**
   - Données de petite taille : traitement direct
   - Au-dessus d'un *seuil* : partition des données en sous-ensembles

# Diviser pour régner (*divide and conquer*)

1. **Divide**
   - Données de petite taille : traitement direct
   - Au-dessus d'un *seuil* : partition des données en sous-ensembles
2. **Recur**

   Traitement récursif des sous-ensembles

# Diviser pour régner (*divide and conquer*)

1. **Divide**
   - Données de petite taille : traitement direct
   - Au-dessus d'un *seuil* : partition des données en sous-ensembles
2. **Recur**
   Traitement récursif des sous-ensembles
3. **Conquer**
   Achever le traitement en rassemblant les sous-ensembles traités

# Tri par fusion (*merge sort*)

Comment trier un tableau $T$ ?

# Tri par fusion (*merge sort*)

Comment trier un tableau $T$ ?

1. **Divide**

# Tri par fusion (*merge sort*)

Comment trier un tableau $T$ ?

1. **Divide**
   - Tableau vide, singleton : déjà triés !

# Tri par fusion (*merge sort*)

Comment trier un tableau $T$ ?

1. **Divide**
   - Tableau vide, singleton : déjà triés !
   - Paire : triée en une comparaison

# Tri par fusion (*merge sort*)

Comment trier un tableau $T$ ?

1. **Divide**
   - Tableau vide, singleton : déjà triés !
   - Paire : triée en une comparaison
   - Tableau de longueur $n \geqslant 3$ :

# Tri par fusion (*merge sort*)

Comment trier un tableau $T$ ?

1. **Divide**
   - Tableau vide, singleton : déjà triés !
   - Paire : triée en une comparaison
   - Tableau de longueur $n \geqslant 3$ : diviser $T$ en deux sous-tableaux de taille moitié

$$T_1 = (T_k)_{0 \leqslant k < p} \qquad T_2 = (T_k)_{p \leqslant k < n}$$

# Tri par fusion (*merge sort*)

Comment trier un tableau $T$ ?

1. **Divide**
   - Tableau vide, singleton : déjà triés !
   - Paire : triée en une comparaison
   - Tableau de longueur $n \geqslant 3$ : diviser $T$ en deux sous-tableaux de taille moitié

$$T_1 = (T_k)_{0 \leqslant k < p} \qquad T_2 = (T_k)_{p \leqslant k < n}$$

2. **Recur**
   Trier $T_1$ et $T_2$

# Tri par fusion (*merge sort*)

Comment trier un tableau $T$ ?

1. **Divide**
   - Tableau vide, singleton : déjà triés !
   - Paire : triée en une comparaison
   - Tableau de longueur $n \geqslant 3$ : diviser $T$ en deux sous-tableaux de taille moitié

$$T_1 = (T_k)_{0 \leqslant k < p} \qquad T_2 = (T_k)_{p \leqslant k < n}$$

2. **Recur**
   Trier $T_1$ et $T_2$
3. **Conquer**
   Réunir les sous-tableaux triés $T_1$ et $T_2$ en un seul tableau trié : **fusion**.

# Fusion de deux tableaux ordonnés

```
def fusion(T, i, j, k):

  return Aux
```

**Données**

Les tranches `T[i:j]` et `T[j:k]` sont triées par ordre croissant.

# Fusion de deux tableaux ordonnés

```
def fusion(T, i, j, k):

  return Aux
```

**Données**

Les tranches `T[i:j]` et `T[j:k]` sont triées par ordre croissant.

**Résultat**

La liste `Aux` contient les valeurs réunies, triées par ordre croissant.

# Fusion de deux tableaux ordonnés

Tant qu'aucune des deux tranches n'est vidée :

```
a, b, Aux = i, j, []
while (a<j) and (b<k):
  if (T[a]<T[b]):
    Aux.append(T[a])
    a += 1
  else:
    Aux.append(T[b])
    b += 1
```

# Fusion de deux tableaux ordonnés

Si la tranche `T[i:j]` est vide :

```
if (a==j):
  Aux += T[b:k]
```

# Fusion de deux tableaux ordonnés

Si la tranche `T[i:j]` est vide :

```
if (a==j):
  Aux += T[b:k]
```

Si la tranche `T[j:k]` est vide :

```
else:
  Aux += T[a:j]
```

# Fusion de deux tableaux ordonnés

Exemple :

| $(T_k)_{0\leqslant k<5}$ | | | | | $(T_k)_{5\leqslant k<10}$ | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 5 | 9 | 12 | 3 | 4 | 5 | 7 | 11 |

# Fusion de deux tableaux ordonnés

Exemple :

| | $(T_k)_{0\leqslant k<5}$ | | | | | $(T_k)_{5\leqslant k<10}$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 5 | 9 | 12 | 3 | 4 | 5 | 7 | 11 |
| 1 | * | * | * | * | * | * | * | * | * | * |
| 1 | 2 | * | * | * | * | * | * | * | * | * |
| 1 | 2 | 3 | * | * | * | * | * | * | * | * |
| 1 | 2 | 3 | 4 | * | * | * | * | * | * | * |
| 1 | 2 | 3 | 4 | 5 | * | * | * | * | * | * |
| 1 | 2 | 3 | 4 | 5 | 5 | * | * | * | * | * |
| 1 | 2 | 3 | 4 | 5 | 5 | 7 | * | * | * | * |
| 1 | 2 | 3 | 4 | 5 | 5 | 7 | 9 | * | * | * |
| 1 | 2 | 3 | 4 | 5 | 5 | 7 | 9 | 11 | * | * |
| 1 | 2 | 3 | 4 | 5 | 5 | 7 | 9 | 11 | 12 | * |

# Récursion

Tri de la tranche `T[i:k]`

```
def tri_rec(T, i, k):
  # k = i : tranche vide
  # k = i + 1 : singleton
  if k==i+2:          # Tri d'une paire
    if T[i]>T[i+1]:
      T[i:k] = [T[i+1], T[i]]
  if (k>i+2):         # Cas général
    j = (i+k)//2
    tri_rec(T, i, j)
    tri_rec(T, j, k)
    T[i:k] = fusion(t, i, j, k)
```

# Récursion

Tri de la liste `T`

```
def tri_fusion(T):
  tri_rec(T, 0, len(T))
```

# Complexité

Tri par fusion d'un tableau de longueur $n$

# Complexité

Tri par fusion d'un tableau de longueur $n$

- **Complexité temporelle**
  $\mathcal{O}(n \lg n)$ comparaisons

# Complexité

Tri par fusion d'un tableau de longueur $n$

- **Complexité temporelle**
  $\mathcal{O}(n \lg n)$ comparaisons
- **Complexité spatiale**

# Complexité

Tri par fusion d'un tableau de longueur $n$

- **Complexité temporelle**
  $\mathcal{O}(n \lg n)$ comparaisons
- **Complexité spatiale**
  - Création de tableaux auxiliaires : $\mathcal{O}(n)$

# Complexité

Tri par fusion d'un tableau de longueur $n$

- **Complexité temporelle**
  $\mathcal{O}(n \lg n)$ comparaisons
- **Complexité spatiale**
  - Création de tableaux auxiliaires : $\mathcal{O}(n)$
  - Pile d'exécution de la fonction récursive : $\mathcal{O}(n)$

# Tri rapide (*quicksort*)

Comment trier un tableau $T$ ?

# Tri rapide (*quicksort*)

Comment trier un tableau $T$ ?

1. **Divide**

# Tri rapide (*quicksort*)

Comment trier un tableau $T$ ?

1. **Divide**
   - Tableau vide, singleton : déjà triés !

# Tri rapide (*quicksort*)

Comment trier un tableau $T$ ?

1. **Divide**
   - Tableau vide, singleton : déjà triés !
   - Tableau de longueur $n \geqslant 2$ :

# Tri rapide (*quicksort*)

Comment trier un tableau $T$ ?

1. **Divide**
   - Tableau vide, singleton : déjà triés !
   - Tableau de longueur $n \geqslant 2$ : diviser $T$ en trois sous-tableaux en fonction d'une **valeur pivot** $T_i$

# Tri rapide (*quicksort*)

Comment trier un tableau $T$ ?

1. **Divide**
   - Tableau vide, singleton : déjà triés !
   - Tableau de longueur $n \geqslant 2$ : diviser $T$ en trois sous-tableaux en fonction d'une **valeur pivot** $T_i$
2. **Recur**
   Trier $T_-$ et $T_+$

# Tri rapide (*quicksort*)

Comment trier un tableau $T$ ?

1. **Divide**
   - Tableau vide, singleton : déjà triés !
   - Tableau de longueur $n \geqslant 2$ : diviser $T$ en trois sous-tableaux en fonction d'une **valeur pivot** $T_i$
2. **Recur**
   Trier $T_-$ et $T_+$
3. **Conquer**
   Concaténer les listes triées $T_-$, $T_=$ et $T_+$

# Tri rapide

```
def trier(T):
  if (len(T)<2):
    # le tableau est déjà trié
    return T
  else:
    # Choix aléatoire du pivot
    k = random.randint(0, len(T))
    pivot = T[k]
```

# Tri rapide

```
# Divide : ventilation en trois sous-listes
I, E, S = [], [], []
for x in T:
  if (x<pivot):
    I.append(x)
  elif (x>pivot):
    S.append(x)
  else:
    E.append(x)
# Recur : tri des sous-listes
I = trier(I)
S = trier(S)
# Conquer : concaténation des sous-listes triées
return I+E+S
```

# Complexité temporelle

Tri rapide d'un tableau de longueur $n$

# Complexité temporelle

Tri rapide d'un tableau de longueur $n$

- Dans le pire des cas : $\mathcal{O}(n^2)$

# Complexité temporelle

Tri rapide d'un tableau de longueur $n$

- Dans le pire des cas : $\mathcal{O}(n^2)$
- En moyenne : $\mathcal{O}(n \lg n)$

# Complexité temporelle

Tri rapide d'un tableau de longueur $n$

- Dans le pire des cas : $\mathcal{O}(n^2)$
- En moyenne : $\mathcal{O}(n \lg n)$
- Un choix hasardeux du pivot vaut beaucoup mieux qu'un mauvais choix !