

## Tim Peters



Tim Peters est un programmeur Python. On lui doit, entre autres choses, la rédaction du « Zen of Python », des règles de bonnes pratiques à suivre lorsqu'on programme dans ce langage [1]. Il a développé un algorithme originellement appelé « `listsort` », puis renommé « `Timsort` » pour trier les tableaux et listes. Tellement efficace qu'il est aussi utilisé actuellement en Java (`Arrays.sort`) !

Crédit photo : [2]

# UE09 – Algorithmique Diviser pour trier

Nicolas HENDRIKX & Simon LIÉNARDY

# Sommaire du thème 1

1. Un premier exercice
2. Définitions
3. Exemples d'algorithme
4. Caractéristiques (domaine, exactitude, efficacité)
5. La notion de complexité
6. Comparaison d'algorithmes polynomiaux
7. Approches de résolution

# Algorithmes de tris

- Tri croissant d'un tableau d'entiers de N éléments
  - Préconditions ?  3 min.
  - Postconditions ?
- Aisément généralisable au tri décroissant, à d'autres types élémentaires ou structurés, au tri de fichier.
- Voir notamment <http://lwh.free.fr/pages/algo/algorithmes.htm>

# Algorithmes de tris : spécifications

- Préconditions :
  - On a un tableau d'éléments
  - Ces éléments sont comparables
    - Types primitifs →  $>$ ,  $<$ ,  $\leq$ ,  $\geq$
    - Autres objets ?  
possèdent une fonction `compareTo` qui définit une **relation d'ordre**
- Postconditions
  - Le tableau passé en entrée a été **permuté** (c'est-à-dire ?)
  - Ses éléments sont ordonnés  
 $t[0] \leq t[1] \leq \dots \leq t[N-1]$  // quel ordre ici ?

# Objectif du jour

Nous allons étudier deux algorithmes :

1. quicksort
2. mergesort

Ces algorithmes sont tous les deux utilisés dans l'implémentation du tri en Java.

En mode **collaboratif** ! (voir consignes au slide suivant)

# Travail collaboratif – phase 1a

1. Déplacez-vous sur votre rangée de manière à être à côté de quelqu'un que **vous ne connaissez pas** ou qui n'est pas dans votre groupe de labo.
2. ⏳ 20 min Lisez les transparents (et ou la page Wiki) sur :
  - **Allées latérales** : sur le tri rapide (quicksort) :
    - Slides 10 à 27
    - [https://fr.wikipedia.org/wiki/Tri\\_rapide](https://fr.wikipedia.org/wiki/Tri_rapide)
    - <https://www.sortvisualizer.com/quicksort/>
  - **Bloc central** : sur le tri fusion (mergesort) :
    - Slides 28 à 37
    - [https://fr.wikipedia.org/wiki/Tri\\_fusion](https://fr.wikipedia.org/wiki/Tri_fusion)
    - <https://www.sortvisualizer.com/mergesort/>
3. Sur une feuille, réalisez un schéma (général) ou d'un tableau particulier mais différent de ceux donnés en exemple.

# Travail collaboratif – phase 1b

- Avec votre voisin : expliquez-vous mutuellement l'algorithme (vous avez étudié le même algo !), vérifiez si vous avez compris de la même manière, posez des questions pour être sûr d'avoir compris.
- Essayez de mettre en évidence dans l'algorithme les 3 phases d'un algorithme **diviser pour régner** [3] :
  1. **Diviser** le problème en un certains nombres de sous-problèmes qui sont des instances plus petites du même problème.
  2. **Régner** sur les sous-problèmes en les résolvant de manière récursive. si la taille d'un sous-problème est suffisamment réduite, on le résout directement.
  3. **Combiner** les solutions des sous-problèmes pour produire la solution du problème originel.
- ⏳ 15 min.

## Travail collaboratif – phase 2

- Formez des paires :
  - 1 étudiant ayant étudié le quicksort
  - 1 étudiant ayant étudié le mergesort
- Expliquez-vous l'un l'autre l'algorithme ⏳ 15 min.
  - Quicksort : expliquez en particulier à quoi sert le **pivot**.
  - Merge sort : expliquez l'intérêt de la **fusion** et pourquoi elle est efficace.
- Réfléchissez aux complexités temporelles et spatiales. ⏳ 10 min.
  - Réfléchissez à 2 : dans le quicksort, existe-t-il un **mauvais choix de pivot** (= si on le choisit, l'algo devient inefficace)?

# Sommaire

1. Quicksort
2. Mergesort

# Quicksort

- Solution du *tri rapide (quicksort)* :

- à chaque itération i,

- On désigne un élément pivot (par exemple l'élément du milieu du tableau)
    - On effectue des permutations pour que tout élément à gauche du pivot soit plus petit que lui et que tout élément à droite du pivot soit plus grand que lui
    - On réapplique l'algorithme aux deux sous-tableaux  $t[0, \text{indicePivot}-1]$  et  $t[\text{indPivot}+1, N-1]$

# Quicksort

- Vidéo comparant les performances du tri par sélection et du quicksort (Merci A. Comblin)
- [https://www.youtube.com/watch?v=cVMKXKoGu\\_Y](https://www.youtube.com/watch?v=cVMKXKoGu_Y)

# Quicksort

- Exemple: soit N = 5, et les nombres contenus dans le tableau = 11, 20, 16, 10, 2.

*1<sup>er</sup> appel: indicePivot = 2, pivot = 16*

0	1	<b>2</b>	3	4
11	20	<b>16</b>	10	2

# Quicksort

*On cherche, de gauche à droite, le 1<sup>er</sup> élément mal placé à gauche du pivot  
(mal placé = plus grand que le pivot)*

0	1	2	3	4
11	20	16	10	2

# Quicksort

*On cherche, de droite à gauche, le 1<sup>er</sup> élément mal placé à  
droite du pivot  
(mal placé = plus petit que le pivot)*

0	1	2	3	4
11	20	16	10	2

*On les permute*

0	1	2	3	4
11	2	16	10	20

# Quicksort

*On cherche, de gauche à droite, un autre élément mal placé à gauche du pivot*

*(mal placé = plus grand que le pivot)*

*Il n'y en a plus => on s'arrête sur le pivot*

0	1	<b>2</b>	3	4
11	2	<b>16</b>	10	20

# Quicksort

*On cherche, de droite à gauche, un autre élément mal placé à droite du pivot*

*(mal placé = plus petit que le pivot)*

0      1      **2**      3      4

11      2      **16**      10      20

*On les permute*

0      1      2      **3**      4

11      2      10      **16**      20

# Quicksort

L'élément pivot est à présent bien placé.

- On recommence avec le sous-tableau de gauche

*2<sup>ème</sup> appel: indicePivot = 1, pivot = 2*

0	1	2	3	4
11	2	10	16	20

# Quicksort

*On cherche, de gauche à droite, le 1<sup>er</sup> élément mal placé à gauche du pivot  
(mal placé = plus grand que le pivot)*



# Quicksort

On cherche, de droite à gauche, le 1<sup>er</sup> élément mal placé à droite du pivot

(mal placé = plus petit que le pivot). Aucun => pivot

0    **1**    2    3    4

11	2	10	16	20
----	---	----	----	----

On les permute

**0**    1    2    3    4

2	11	10	16	20
---	----	----	----	----

# Quicksort

L'élément pivot est à présent bien placé.

- On recommence avec le sous-sous-tableau de gauche: vide
- On continue avec le sous-sous tableau de droite

*4ème appel: indicePivot = 1, pivot = 11*

0	1	2	3	4
2	11	10	16	20

# Quicksort

*On cherche, de gauche à droite, le 1<sup>er</sup> élément mal placé à gauche du pivot*

*(mal placé = plus grand que le pivot)*

*Aucun élément à gauche => stop sur pivot*



# Quicksort

*On cherche, de droite à gauche, le 1<sup>er</sup> élément mal placé à droite du pivot*

*(mal placé = plus petit que le pivot)*

0	1	2	3	4
2	11	10	16	20

*On les permute*

0	1	2	3	4
2	10	11	16	20

# Quicksort

L'élément pivot est à présent bien placé.

0	1	2	3	4
2	10	11	16	20

- On recommence avec le sous-sous-tableau de gauche: 1 seul élément => trié
- On continue avec le sous-sous tableau de droite: vide

0	1	2	3	4
2	10	11	16	20

# Quicksort

On recommence avec le premier sous-tableau de droite: 1 seul élément => trié

0	1	2	3	4
2	10	11	16	20

# Quicksort

- Voir aussi sur [http://lwh.free.fr/pages/algo/tri/tri\\_rapide.htm](http://lwh.free.fr/pages/algo/tri/tri_rapide.htm)
- Merci à Loïc Dessart pour la vidéo suivante:  
<https://www.youtube.com/watch?v=aXXWXz5rF64>

## Quicksort – Pseudocode

```
quickSort(t[], indDeb, indFin) {  
    si indDeb < indFin  
    alors  
        indPivot ← appeler reordonner(t, indDeb, indFin)  
        appeler quickSort(t, indDeb, indPivot-1) ;  
        appeler quickSort (t, indPivot + 1, indFin) ;  
    fin_si  
}
```

# Quicksort - Pseudocode

```
reordonner(t[], i, j) {
    calculer indPivot ;
    pivot = t[indPivot] ;
    Tant que i < j exécuter
        trouver i tel que t[i] >= pivot ;
        trouver j tel que t[j] <= pivot ;
        si i < j, permuter t[i] et t[j] ;
            mettre à jour les valeurs de i, j, indPivot, pivot ;
    fin_tant
    retourner indPivot
}
```

# Sommaire

1. Quicksort

2. Mergesort

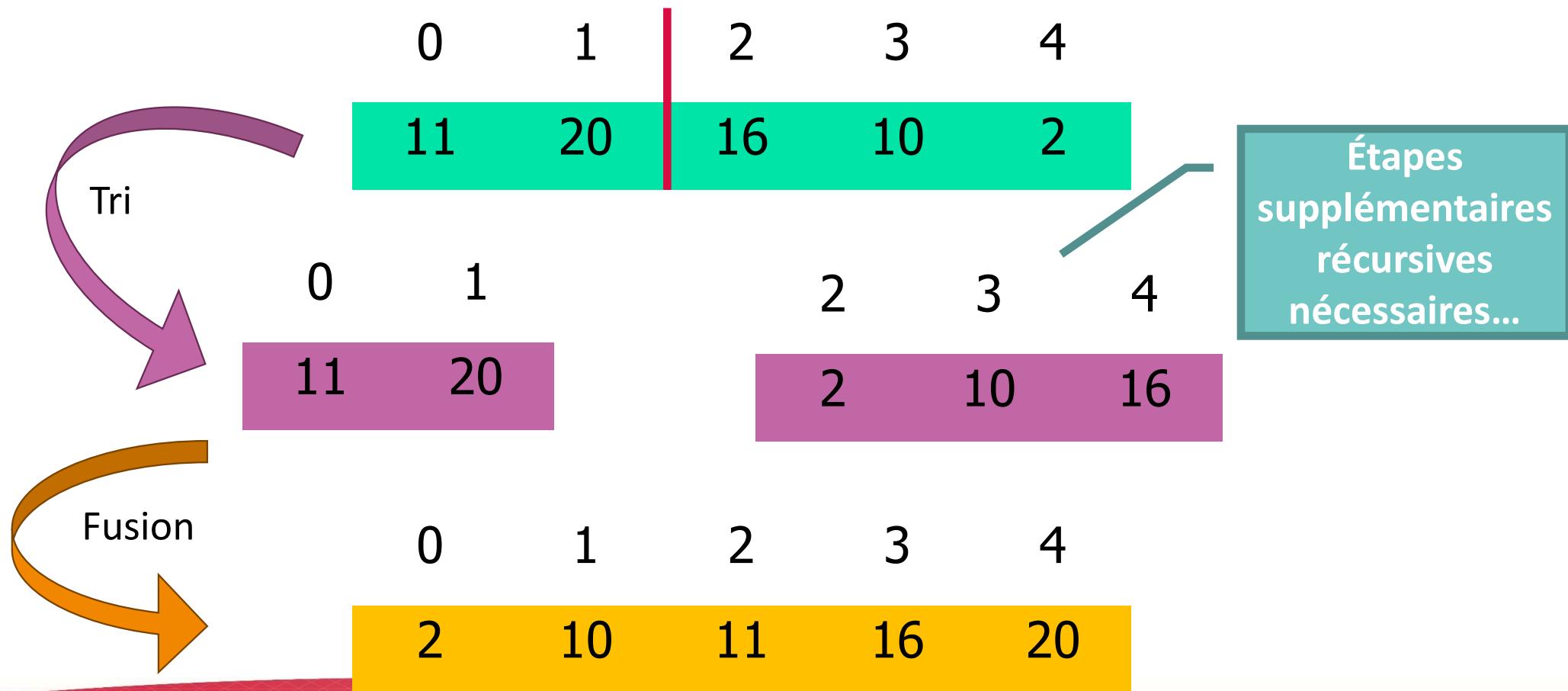
# Mergesort

Le tri par fusion repose sur l'idée que fusionner deux tableaux triés en un est **efficace**.

## Solution :

- Couper le tableau en deux sous-tableau
- Trier chacun des sous-tableau récursivement
- Fusionner les deux sous-tableau

# Mergesort - illustration générale



## Mergesort - pseudocode

```
// trie le tableau entre début (inclus) et fin (exclus)
mergesort(tab[], début, fin){
    si (fin - début) > 1 // cas de base
    alors
        milieu = début + (fin - début) / 2
        mergesort(tab, début, milieu)
        mergesort(tab, milieu, fin)
        fusion(tab, début, fin)
}
```

# Mergesort - pseudocode

```
fusion(tab, début, fin) {
    milieu = début + (fin - début) / 2
    copier tab[début..milieu - 1]
    dest = début
    src1 = milieu, src2 = 0
    tant que dest < fin:
        tab[dest] = min(tab[src1], copie[src2])
        incrémenter dest et src1 (ou src2)
    fin_tant
}
```

# Fusion

	dest	src1	
déb		mil	fin

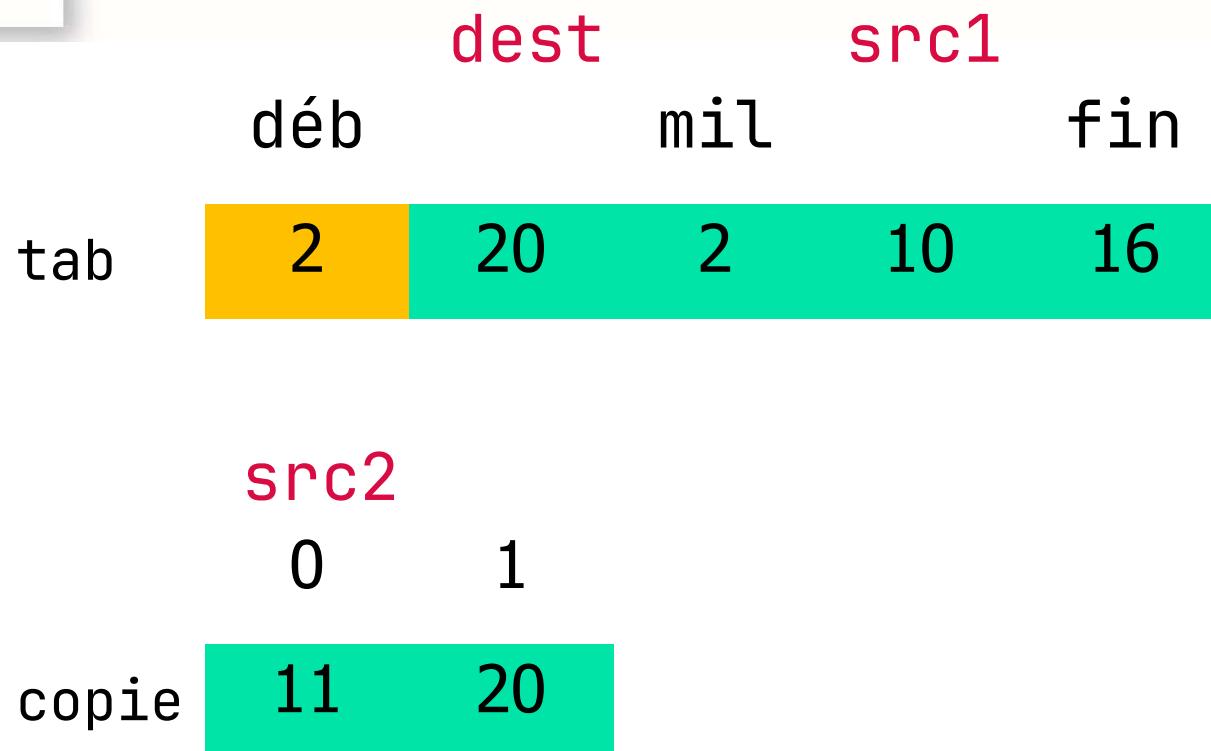
tab	11	20	2	10	16
-----	----	----	---	----	----

src2

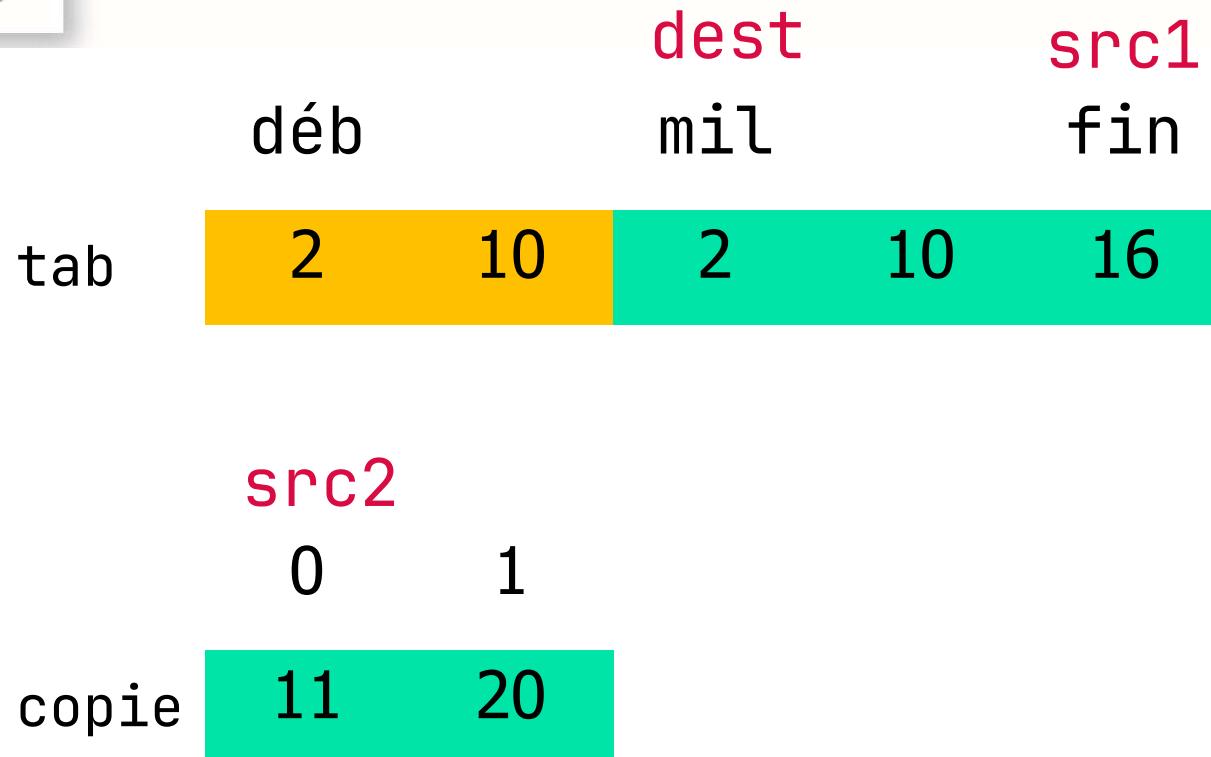
0      1

copie	11	20
-------	----	----

# Fusion



# Fusion



# Fusion

	dest			src1
	déb	mil		fin
tab	2	10	11	10 16
				src2
	0	1		
copie	11	20		

**Et cetera**

# Illustration

Dessiner la descente récursive pour le tableau

0	1	2	3	4
11	20	16	10	2

Illustrer ensuite la remontée des fusions successives.

# Mise en commun

- Debriefing : des questions ?
- Complexités ?

# Complexité

	Insertion	Sélection	Tas	Rapide	Fusion
Cas général	$O(N^2)$	$O(N^2)$	$O(N \log_2 N)$	$O(N \log_2 N)$	$O(N \log_2 N)$
Tableau trié	$O(N)$	$O(N^2)$	$O(N \log_2 N)$	$O(N \log_2 N)$	$O(N \log_2 N)$
Tableau trié en ordre inverse	$O(N^2)$	$O(N^2)$	$O(N \log_2 N)$	$O(N \log_2 N)$	$O(N \log_2 N)$
Autre cas particulier				$O(N^2)$	

D'où ?

# Sources

1. Tim Peters. (2004) *PEP 20 – The Zen of Python*. <https://peps.python.org/pep-0020/> (Consulté le 13/02/24)
2. Paul McLellan. (2021) *Supernaturally Fast Sorting*.  
[https://community.cadence.com/cadence\\_blogs\\_8/b/breakfast-bytes/posts/timsrt](https://community.cadence.com/cadence_blogs_8/b/breakfast-bytes/posts/timsrt) (consulté le 13/02/24)
3. Thomas Cormen, Charles Leiserson, Ronald Rivest & Clifford Stein. (2010) *Algorithmique*. Dunod