

Fonction `sorted` Diviser pour régner

Christophe Viroulaud

Terminale - NSI

Algo 01

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

En tant que langage de haut niveau, Python offre des méthodes permettant d'effectuer efficacement certaines tâches courantes.

La méthode `sort` trie en place un tableau. .

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

Quel algorithme de tri est implémenté dans la méthode `sorted` ?

Sommaire

1. Rappel : des algorithmes de tris déjà connus

1.1 Tri par sélection

1.2 Tri par insertion

1.3 Comparaison des performances

2. Nouvelle approche

3. Performances du tri fusion

4. Timsort

Fonction `sorted` Diviser pour régner

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

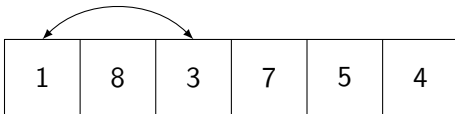
Mesure de la durée d'exécution

Complexité

Timsort

Tri par sélection (en place)

- 1 Pour chaque élément du tableau
- 2 Trouver le plus petit élément dans la partie non triée.
- 3 Échanger cet élément avec le premier de la partie non triée.



Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

Tri par sélection (en place)

- 1 Pour chaque élément du tableau
- 2 Trouver le plus petit élément dans la partie non triée.
- 3 Échanger cet élément avec le premier de la partie non triée.

Activité 1 :

1. Construire par compréhension un tableau de 10 entiers aléatoires compris entre 1 et 100.
2. Écrire la fonction `tri_selection(tab: list)`
→ `None` qui trie le tableau en place.

```
1 tab = [randint(1, 100) for _ in range(10)]
```

```
1 def tri_selection(tab: list) -> None:  
2     for i in range(len(tab)):  
3         # trouver le mini  
4         i_mini = i  
5         for j in range(i+1, len(tab)):  
6             if tab[j] < tab[i_mini]:  
7                 i_mini = j  
8         # échanger  
9         tab[i], tab[i_mini] = tab[i_mini], tab[i]
```

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

Sommaire

1. Rappel : des algorithmes de tris déjà connus

1.1 Tri par sélection

1.2 Tri par insertion

1.3 Comparaison des performances

2. Nouvelle approche

3. Performances du tri fusion

4. Timsort

Fonction **sorted**
Diviser pour régner

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité


Timsort

Tri par insertion (en place)


- 1 Pour chaque élément du tableau
- 2 Tant que l'élément précédent est inférieur
- 3 Permuter les deux éléments

1	3	5	2	7	4
---	---	---	---	---	---

1	3	2	5	7	4
---	---	---	---	---	---



1	2	3	5	7	4
---	---	---	---	---	---



Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

Tri par insertion (en place)

- 1 Pour chaque élément du tableau
- 2 Tant que l'élément précédent est inférieur
- 3 Permuter les deux éléments

Activité 2 :

1. Construire par compréhension un tableau de 10 entiers aléatoires compris entre 1 et 100.
2. Écrire la fonction `tri_insertion(tab: list)`
→ `None` qui trie le tableau en place.

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

```
1 def tri_insertion(tab: list) -> None:
2     for i in range(len(tab)-1):
3         j = i+1
4         # tant que le précédent est inférieur
5         while j > 0 and tab[j] < tab[j-1]:
6             # permuter
7             tab[j], tab[j-1] = tab[j-1], tab[j]
8             j -= 1
```

Sommaire

1. Rappel : des algorithmes de tris déjà connus

1.1 Tri par sélection

1.2 Tri par insertion

1.3 Comparaison des performances

2. Nouvelle approche

3. Performances du tri fusion

4. Timsort

Fonction **sorted**
Diviser pour régner

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

Activité 3 :

1. Construire par compréhension un tableau de 10000 entiers compris entre 1 et 10000.
2. Mesurer la durée d'exécution de la méthode `sort` et des deux fonctions précédentes.

```
1 tab = [randint(1, 10000) for _ in range  
        (10000)]  
2 deb = time()  
3 tri_selection(tab)  
4 fin = time()  
5 print(fin-deb)
```

Code 1 – tri par sélection

```
1 >>> sélection 4.557838678359985  
2 >>> insertion 3.959839105606079  
3 >>> sort 0.0019469261169433594
```

Code 2 – Résultats

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

Évolution du nombre d'itérations

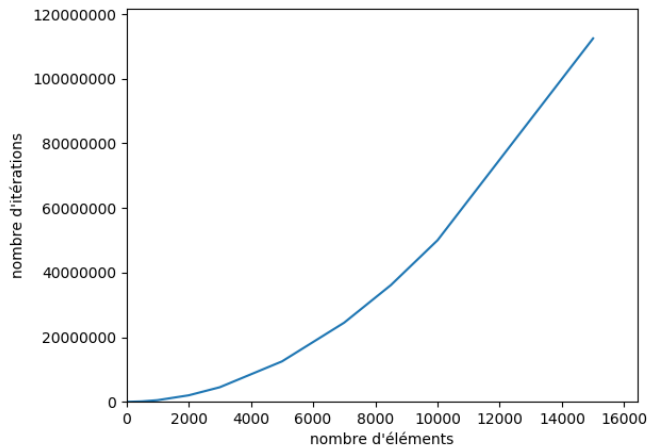


FIGURE 1 – Tri par sélection : complexité **quadratique**

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

1. Rappel : des algorithmes de tris déjà connus
2. **Nouvelle approche**
 - 2.1 Résoudre de petits problèmes
 - 2.2 ...pour solutionner un gros problème
 - 2.3 Diviser pour régner : un algorithme récursif
 - 2.4 Étape de la fusion
3. Performances du tri fusion
4. Timsort

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

...pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

Observation

Une liste qui contient 0 ou 1 élément est triée.

8 5

FIGURE 2 – Deux listes triées

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

Deux listes d'un élément chacune peuvent être fusionnées en une liste triée de deux éléments.

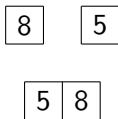


FIGURE 3 – Fusionner 2 listes de 1 élément

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

Principe

En résolvant des petits problèmes, nous pouvons remonter à des problèmes plus importants en appliquant le même principe.

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

1. Rappel : des algorithmes de tris déjà connus
2. **Nouvelle approche**
 - 2.1 Résoudre de petits problèmes
 - 2.2 **...pour solutionner un gros problème**
 - 2.3 Diviser pour régner : un algorithme récursif
 - 2.4 Étape de la fusion
3. Performances du tri fusion
4. Timsort

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

...pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

...pour solutionner un gros problème

Fonction `sorted`
Diviser pour régner

8	5	4	7	9	6	3
---	---	---	---	---	---	---

FIGURE 4 – Un gros problème : trier une liste

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

...pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

...pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

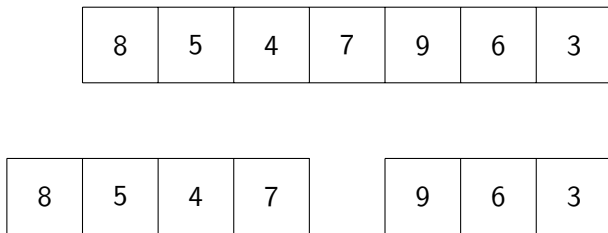


FIGURE 5 – Séparer la liste en deux listes plus petites

8	5	4	7	9	6	3
---	---	---	---	---	---	---

8	5	4	7
---	---	---	---

9	6	3
---	---	---

8	5
---	---

4	7
---	---

9	6
---	---

3

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

...pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

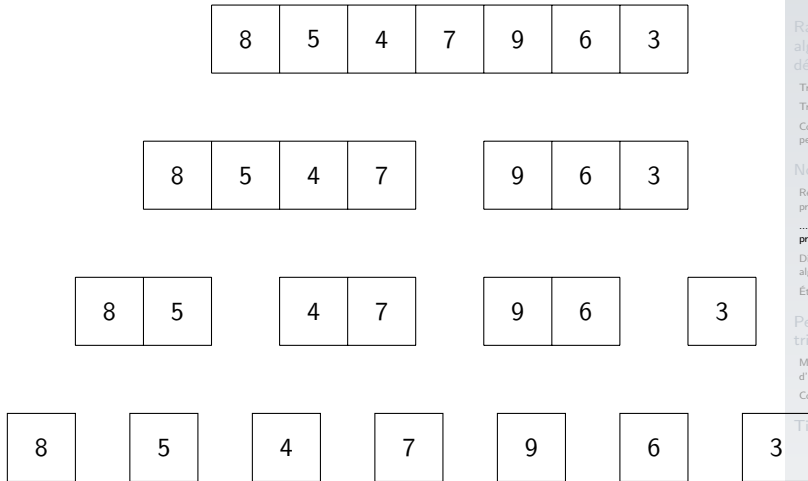


FIGURE 6 – Obtenir de petits problèmes

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

...pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Tri sort

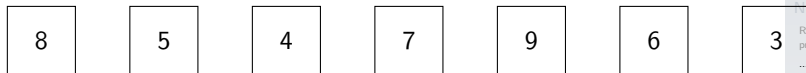


FIGURE 7 – Résoudre les petits problèmes

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

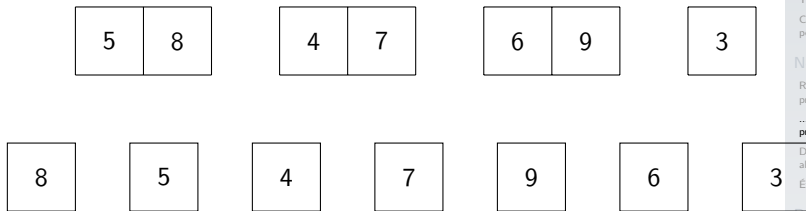


FIGURE 8 – Trier...

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

...pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

...pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

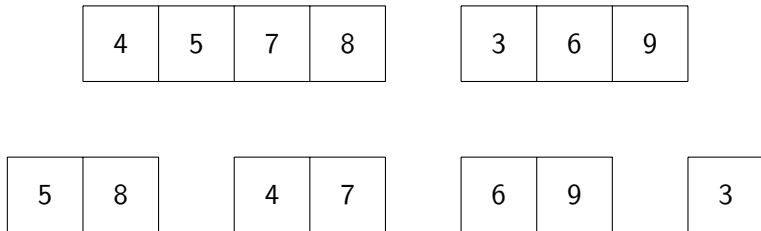


FIGURE 9 – ...et remonter

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

...pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

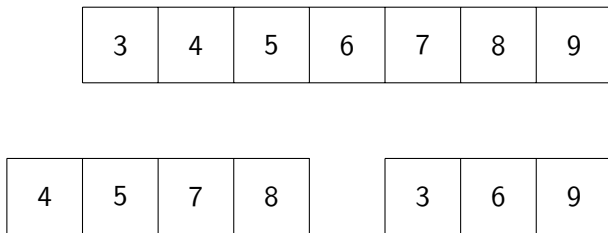


FIGURE 10 – Le tri se termine

1. Rappel : des algorithmes de tris déjà connus
2. **Nouvelle approche**
 - 2.1 Résoudre de petits problèmes
 - 2.2 ...pour solutionner un gros problème
 - 2.3 **Diviser pour régner : un algorithme récursif**
 - 2.4 Étape de la fusion
3. Performances du tri fusion
4. Timsort

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

...pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

Diviser pour régner : un algorithme récursif

- ▶ **cas limite** : la liste est de taille minimale
- ▶ **sinon**
 - ▶ on coupe la liste en 2,
 - ▶ **appel récursif** sur chaque liste
 - ▶ **fusionner** les listes lors de la remontée d'appel

Code 3 – Tri fusion

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

- ▶ **cas limite** : la liste est de taille minimale
- ▶ sinon
 - ▶ on coupe la liste en 2,
 - ▶ **appel récursif** sur chaque liste
 - ▶ **fusionner** les listes lors de la remontée d'appel

Code 4 – Tri fusion

Activité 4 : Soit la fonction `fusionner(tab: list, deb: int, fin: int) → None` qui trie les éléments de `tab` entre les indices `deb` et `fin`.
Écrire la fonction `tri_fusion(tab: list, deb: int, fin: int) → None` qui trie le tableau *en place*.

Rappel : des algorithmes de tri déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

```
1 def tri_fusion(tab: list, deb: int, fin: int) ->  
  None:  
2 if deb < fin:  
3     milieu = (deb+fin)//2  
4     tri_fusion(tab, deb, milieu)  
5     tri_fusion(tab, milieu+1, fin)  
6     fusionner(tab, deb, fin)
```

Code 5 – Tri fusion

Rappel : des
algorithmes de tris
déjà connus

Tri par sélection

Tri par insertion

Comparaison des
performances

Nouvelle approche

Résoudre de petits
problèmes

...pour solutionner un gros
problème

**Diviser pour régner : un
algorithme récursif**

Etape de la fusion

Performances du
tri fusion

Mesure de la durée
d'exécution

Complexité

Timsort

1. Rappel : des algorithmes de tris déjà connus
2. **Nouvelle approche**
 - 2.1 Résoudre de petits problèmes
 - 2.2 ...pour solutionner un gros problème
 - 2.3 Diviser pour régner : un algorithme récursif
 - 2.4 **Étape de la fusion**
3. Performances du tri fusion
4. Timsort

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

...pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

Étape de la fusion

Lors de la remontée d'appel, la *fusion* assemble deux tableaux triés en un seul.

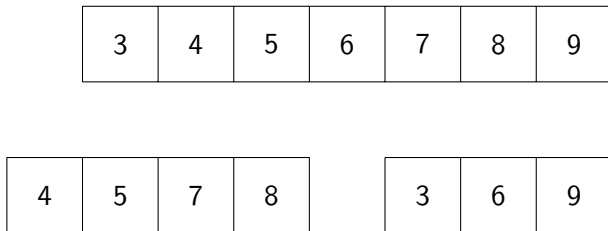


FIGURE 11 – Fusionner

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

En pratique, nous effectuons un tri *place*.

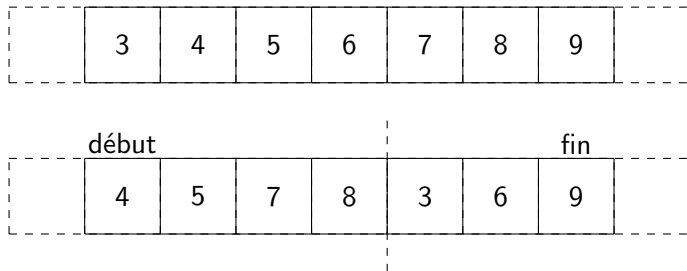


FIGURE 12 – Fusionner

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

Pour assembler les deux parties du tableau, il faut prendre le plus petit élément, jusqu'à vider un des deux blocs. Puis on complète avec les éléments restants.

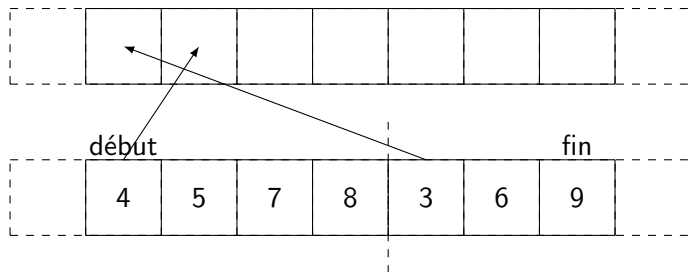


FIGURE 13 – Fusionner

Rappel : des algorithmes de tri déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

Puis on complète avec les éléments restants.

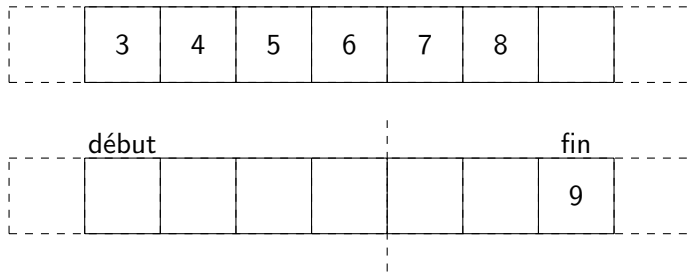


FIGURE 14 – Compléter

Remarque

En pratique nous utiliserons un tableau temporaire pour fusionner.

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

Activité 5 : Écrire la fonction `fusionner(tab: list, deb: int, fin: int) → None` qui assemble les éléments de `tab` entre les indices `deb` et `fin`. Les éléments seront d'abord stockés dans un tableau temporaire qui viendra ensuite écraser la partie de `tab`.

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

```
1 def fusionner(tab: list, deb: int, fin: int) ->  
  None:  
2     res = [0 for _ in range(fin-deb+1)]  
3     milieu = (deb+fin)//2  
4     i = deb  
5     j = milieu+1  
6     k = 0
```

Code 6 – initialiser

Rappel : des
algorithmes de tris
déjà connus

Tri par sélection

Tri par insertion

Comparaison des
performances

Nouvelle approche

Résoudre de petits
problèmes

...pour solutionner un gros
problème

Diviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusion

Mesure de la durée
d'exécution

Complexité

Timsort

```
1 while i <= milieu and j <= fin:
2     if tab[i] < tab[j]:
3         res[k] = tab[i]
4         i += 1
5     else:
6         res[k] = tab[j]
7         j += 1
8     k += 1
```

Code 7 – assembler

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort


```
1 for i1 in range(i, milieu+1):
2     res[k] = tab[i1]
3     k += 1
4 for j1 in range(j, fin+1):
5     res[k] = tab[j1]
6     k += 1
```

Code 8 – compléter

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

```
1 # remplacement tab par res
2 for k in range(fin-deb+1):
3     tab[deb+k] = res[k]
```

Code 9 – remplacer

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

1. Rappel : des algorithmes de tris déjà connus

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

2. Nouvelle approche

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

3. Performances du tri fusion

Diviser pour régner : un algorithme récursif

3.1 Mesure de la durée d'exécution

Étape de la fusion

3.2 Complexité

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

4. Timsort

Timsort

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

Activité 6 :

1. Construire par compréhension un tableau de 10000 entiers compris entre 1 et 10000.
2. Mesurer la durée d'exécution de la fonction `tri_fusion` et des deux fonctions précédentes.

```
1 tab = [randint(1, 10000) for _ in range  
      (10000)]  
2 deb = time()  
3 tri_fusion(tab2, 0, len(tab)-1)  
4 fin = time()  
5 print("fusion ", fin-deb)
```

```
1 >>> sélection 4.557838678359985  
2 >>> insertion 3.959839105606079  
3 >>> sort 0.0019469261169433594  
4 >>> fusion 0.1485743522644043
```

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

...pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

1. Rappel : des algorithmes de tris déjà connus

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

2. Nouvelle approche

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

3. Performances du tri fusion

Diviser pour régner : un algorithme récursif

3.1 Mesure de la durée d'exécution

Étape de la fusion

3.2 Complexité

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

4. Timsort

Timsort

Complexité du découpage

- ▶ À chaque appel de la fonction `tri_fusion` nous divisons la liste en deux.

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

- ▶ À chaque appel de la fonction `tri_fusion` nous divisons la liste en deux.
- ▶ Combien de fois faut-il couper la liste en deux pour obtenir des listes d'un élément ?

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

Mathématiquement nous cherchons a tel que :

$$\frac{n}{2^a} = 1$$

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

Le logarithme base 2 noté \log_2 se définit : $\log_2(2^x) = x$.

$$\frac{n}{2^a} = 1$$

$$\iff n = 2^a$$

$$\iff \log_2 n = \log_2(2^a)$$

$$\iff \log_2 n = a$$

À retenir

La complexité du découpage en sous-listes est **logarithmique**.

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

Complexité de la fusion

La fonction `fusionner` réalise n comparaisons pour assembler deux listes de taille $\frac{n}{2}$.

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

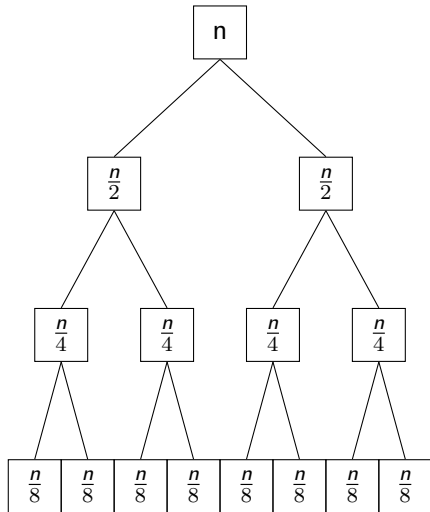
Étape de la fusion

Performances du tri fusion

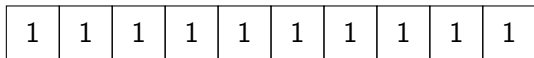
Mesure de la durée d'exécution

Complexité

Timsort



.....



$$n$$

$$2 \times \left(\frac{n}{2}\right) = n$$

$$2^2 \times \left(\frac{n}{2^2}\right) = n$$

$$2^3 \times \left(\frac{n}{2^3}\right) = n$$

$$2^{\log_2(n)} \times \left(\frac{n}{2^{\log_2(n)}}\right) = n$$

À retenir

Chaque niveau de fusion a un coup de n et il y a $\log_2(n)$ niveaux.

$$O(n \times \log_2(n))$$

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

Activité 7 : Sachant que la complexité du tri par sélection est quadratique, la comparer au tri fusion pour un tableau de 100, 1000, 10000, 100000 éléments.

$$\log_2(n) = \frac{\ln n}{\ln 2}$$

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

éléments	tri par sélection	tri fusion
100	10^4	664
1000	10^6	9966
10000	10^8	132877
100000	10^{10}	1660964

1. Rappel : des algorithmes de tris déjà connus
2. Nouvelle approche
3. Performances du tri fusion
4. Timsort

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort

La méthode native `sort` implémente l'algorithme **Timsort** mis au point par Tim Peters en 2002. C'est un algorithme hybride de plusieurs tris.

Activité 8 : Recherche :

- ▶ Donner les algorithmes de tris utilisés dans Timsort.
- ▶ Détailler dans quel cas est utilisé chacun des tris.
- ▶ En discutant de la complexité, expliquer pour quelle raison le tri par insertion est plus intéressant que le tri par sélection.

Rappel : des algorithmes de tris déjà connus

Tri par sélection

Tri par insertion

Comparaison des performances

Nouvelle approche

Résoudre de petits problèmes

... pour solutionner un gros problème

Diviser pour régner : un algorithme récursif

Étape de la fusion

Performances du tri fusion

Mesure de la durée d'exécution

Complexité

Timsort