

Fonction `sorted`
Diviser pour régner

Christophe Viroulaud

Terminale - NSI

Algo 01

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. .

tri en place ou nouveau tableau

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 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

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

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

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 sélection (en place)

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.
  
```

3	8	1	7	5	4
---	---	---	---	---	---

1	8	3	7	5	4
---	---	---	---	---	---

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.

3	8	1	7	5	4
---	---	---	---	---	---

1	8	3	7	5	4
---	---	---	---	---	---

Fonction `sorted`
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Fonction `sorted` Diviser pour régner

└─ Rappel : des algorithmes de tris déjà connus

└─ Tri par sélection

└─ Tri par sélection (en place)

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.

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.

Fonction `sorted`
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Fonction `sorted` Diviser pour régner

└ Rappel : des algorithmes de tris déjà connus

└ Tri par sélection

└ Correction

Correction

```

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

2
3
4
5
6
7
8
9

```

Correction

```

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]

```

Fonction `sorted`
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Fonction `sorted`Diviser pour régner

└─ Rappel : des algorithmes de tris déjà connus

└─ Tri par insertion

└─ Sommaire

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

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 régnerRappel : des
algorithmes de tris
déjà connus

Tri par sélection

Tri par insertionComparaison des
performances

Nouvelle approche

Résoudre de petits
problèmes...pour solutionner un gros
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

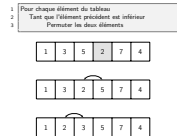
Fonction `sorted` Diviser pour régner

- Rappel : des algorithmes de tris déjà connus

- Tri par insertion

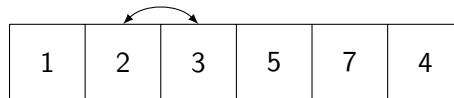
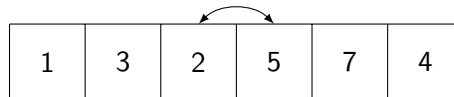
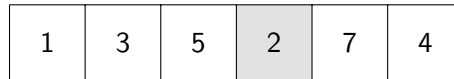
- Tri par insertion (en place)

Tri par insertion (en place)



Tri par insertion (en place)

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

Fonction `sorted`
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Fonction `sorted` Diviser pour régner

└ Rappel : des algorithmes de tris déjà connus

└ Tri par insertion

└ Tri par insertion (en place)

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.

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.

Fonction `sorted`
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Fonction `sorted` Diviser pour régner

└─ Rappel : des algorithmes de tris déjà connus

└─ Tri par insertion

└─ Correction

Correction

```

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

```

Correction

```

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

```

Fonction `sorted`
Diviser pour régnerRappel : des
algorithmes de tris
déjà connus

Tri par sélection

Tri par insertionComparaison des
performances

Nouvelle approche

Résoudre de petits
problèmes...pour solutionner un gros
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Fonction `sorted`Diviser pour régner

└─ Rappel : des algorithmes de tris déjà connus

└─ Comparaison des performances

└─ Sommaire

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

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

Fonction `sorted` Diviser pour régner

└─ Rappel : des algorithmes de tris déjà connus

└─ Comparaison des performances

└─ Comparaison des performances

créer un nouveau tableau pour chaque fonction !!

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.

Comparaison des performances

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.

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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Fonction `sorted` Diviser pour régner

└─ Rappel : des algorithmes de tris déjà connus

└─ Comparaison des performances

└─ Correction

Correction

```

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

Correction

```

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

Fonction `sorted`
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Fonction `sorted` Diviser pour régner

- Rappel : des algorithmes de tris déjà connus

- Comparaison des performances

- Évolution du nombre d'itérations

15000 éléments → 100 millions d'itérations

Évolution du nombre d'itérations

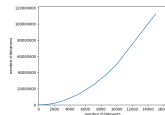
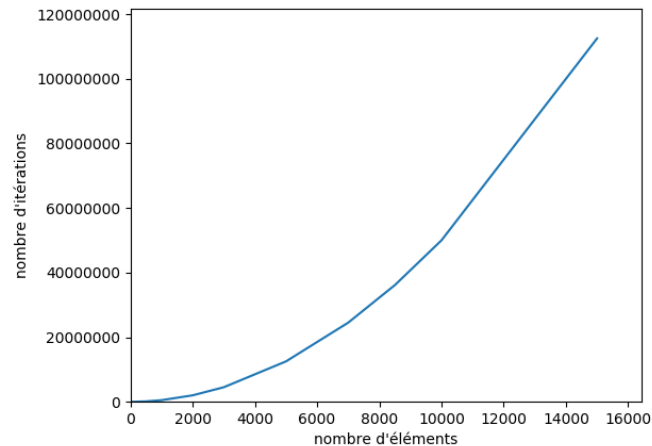


FIGURE 1 – Tri par sélection : complexité quadratique

Évolution du nombre d'itérations

FIGURE 1 – Tri par sélection : complexité **quadratique**Fonction `sorted`
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Sommaire

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

Fonction `sorted` Diviser pour régner

└ Nouvelle approche

└ Résoudre de petits problèmes

└ Résoudre de petits problèmes

La propriété triviale suivante va nous permettre de construire une nouvelle méthode de tri :

Résoudre de petits problèmes

Observation

Une liste qui contient 0 ou 1 élément est triée.

8 5

FIGURE 2 – Deux listes triées

Résoudre de petits problèmes

Observation

Une liste qui contient 0 ou 1 élément est triée.

8

5

FIGURE 2 – Deux listes triées

Fonction `sorted`
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Fonction `sorted` Diviser pour régner

└ Nouvelle approche

└ Résoudre de petits problèmes

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



FIGURE 3 – Fusionner 2 listes de 1 élément

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

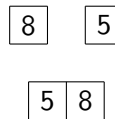


FIGURE 3 – Fusionner 2 listes de 1 élément

Fonction `sorted`
Diviser pour régner

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

Fonction `sorted` Diviser pour régner

└ Nouvelle approche

└ Résoudre de petits problèmes

Principe

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

Principe

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

Fonction `sorted` Diviser pour régner

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

Fonction `sorted`Diviser pour régner

└ Nouvelle approche

└ └ ...pour solutionner un gros problème

└ └ └ Sommaire

Sommaire

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 Étapes de la fusion
3. Performances du tri fusion
4. Timsort

Sommaire

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

Fonction `sorted`
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Fonction `sorted` Diviser pour régner

└ Nouvelle approche

└ ...pour solutionner un gros problème

└ ...pour solutionner un gros problème

essayons de nous ramener à de petits problèmes

...pour solutionner un gros problème

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

FIGURE 4 – Un gros problème : trier une liste

...pour solutionner un gros problème

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

FIGURE 4 – Un gros problème : trier une liste

Fonction `sorted`
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Fonction `sorted` Diviser pour régner

└ Nouvelle approche

└ ...pour solutionner un gros problème



Pour se ramener à un problème plus petit, séparons la liste en deux listes

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

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

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

FIGURE 5 – Séparer la liste en deux listes plus petites

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

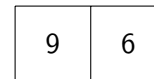
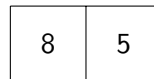
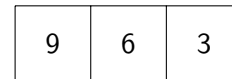
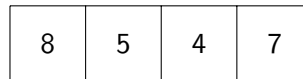
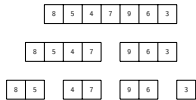
Complexité

Timsort

Fonction `sorted` Diviser pour régner

└ Nouvelle approche

└ ...pour solutionner un gros problème

Fonction `sorted`
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Fonction `sorted` Diviser pour régner

└ Nouvelle approche

└ ...pour solutionner un gros problème

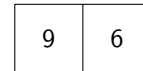
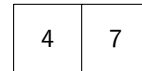
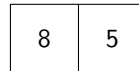
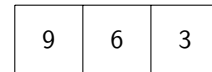
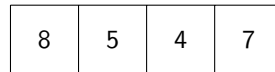
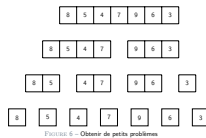


FIGURE 6 – Obtenir de petits problèmes

Fonction `sorted`
Diviser pour régner

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

Fonction `sorted`Diviser pour régner

- └ Nouvelle approche
- └ ...pour solutionner un gros problème



FIGURE 7 – Résoudre les petits problèmes

Fonction `sorted` Diviser pour régner

└ Nouvelle approche

└ ...pour solutionner un gros problème



FIGURE 8 – Trier...

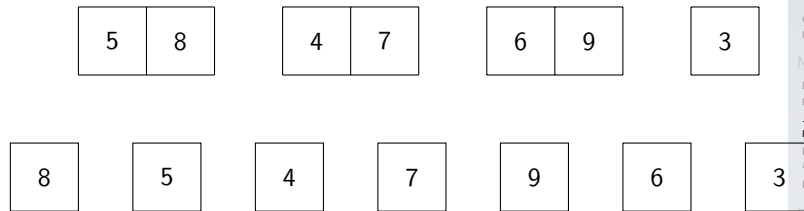


FIGURE 8 – Trier...

Fonction `sorted`
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

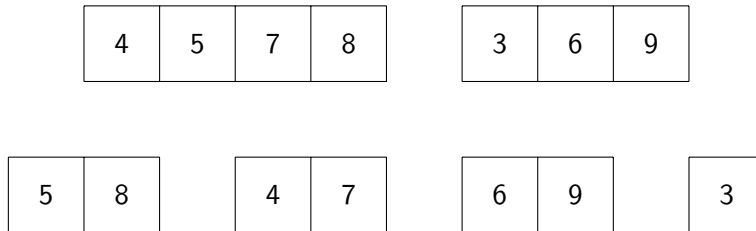
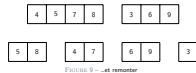
Complexité

Timsort

Fonction `sorted` Diviser pour régner

└ Nouvelle approche

└ ...pour solutionner un gros problème

Fonction `sorted`
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Fonction `sorted` Diviser pour régner

└ Nouvelle approche

└ ...pour solutionner un gros problème

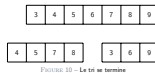


FIGURE 10 – Le tri se termine

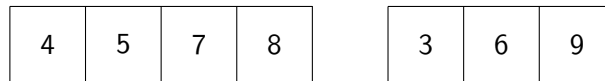
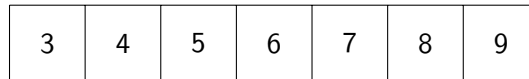


FIGURE 10 – Le tri se termine

Fonction `sorted`
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Fonction `sorted`Diviser pour régner

└ Nouvelle approche

└ Diviser pour régner : un algorithme récursif

└ Sommaire

Sommaire

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

Sommaire

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

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

Fonction `sorted` Diviser pour régner

└ Nouvelle approche

└ Diviser pour régner : un algorithme récursif

└ Diviser pour régner : un algorithme récursif

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

```

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

Fonction `sorted`
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Fonction `sorted` Diviser pour régner

└ Nouvelle approche

└ 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 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.

```

- **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*.

Fonction `sorted`
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Fonction sortedDiviser pour régner

└ Nouvelle approche

└ Diviser pour régner : un algorithme récursif

└ Correction

Correction

```

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

Correction

```

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

Fonction sorted
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Fonction `sorted`Diviser pour régner

Nouvelle approche

Étape de la fusion

Sommaire

Sommaire

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

Sommaire

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

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

Fonction `sorted` Diviser pour régner

└ Nouvelle approche

└ Étape de la fusion

└ Étape de la fusion

Étape de la fusion

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

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

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

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

FIGURE 11 – Fusionner

Étape de la fusion

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

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

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

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

FIGURE 11 – Fusionner

Fonction `sorted`
Diviser pour régner

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

Fonction `sorted` Diviser pour régner

└ Nouvelle approche

└ Étape de la fusion

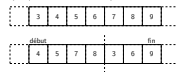
En pratique, nous effectuons un tri *place*.

FIGURE 12 – Fusionner

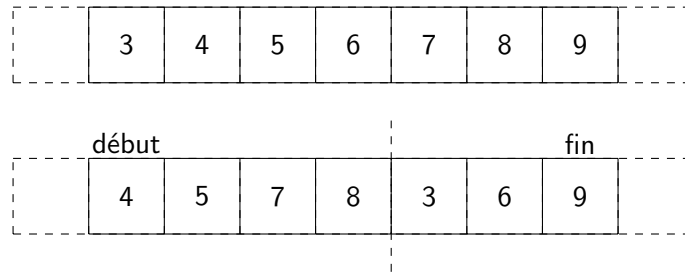
En pratique, nous effectuons un tri *place*.

FIGURE 12 – Fusionner

Fonction `sorted`
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Fonction `sortedDiviser pour régner`

Nouvelle approche

Étape de la fusion

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.

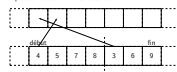


FIGURE 13 – Fusionner

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.

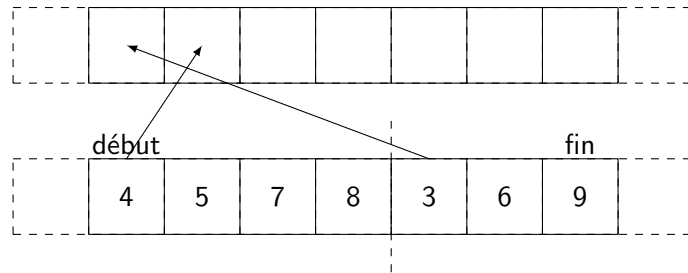


FIGURE 13 – Fusionner

Fonction `sorted`
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

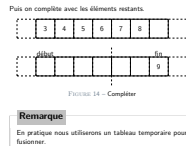
Complexité

Timsort

Fonction `sorted` Diviser pour régner

Nouvelle approche

Étape de la fusion



Puis on complète avec les éléments restants.

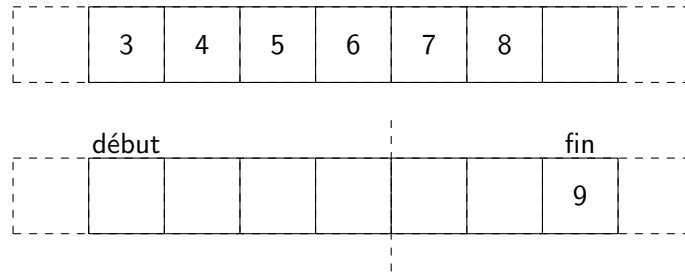


FIGURE 14 – Compléter

Remarque

En pratique nous utiliserons un tableau temporaire pour fusionner.

Fonction `sorted`
Diviser pour régner

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

Fonction `sorted` Diviser pour régner

└ Nouvelle approche

└ Étape de la fusion

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`.

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`.

Fonction `sorted`
Diviser pour régner

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

Fonction `sorted` Diviser pour régner

└ Nouvelle approche

└ Étape de la fusion

└ Correction

Correction

```

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 5 – initialiser

Correction

```

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

Fonction `sorted`
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Fonction `sorted` Diviser pour régner

└ Nouvelle approche

└ Étape de la fusion

└ Correction

Correction

```

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

Correction

```

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

Fonction `sorted`
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Fonction `sorted` Diviser pour régner

└ Nouvelle approche

└ Étape de la fusion

└ Correction

Correction

```

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

Correction

```

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

Fonction `sorted`
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Fonction `sortedDiviser pour régner`

└ Nouvelle approche

└ Étape de la fusion

└ Correction

Correction

```

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

```

Code 9 – remplacer

Correction

```

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

```

Code 9 – remplacer

Fonction `sorted`
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Sommaire

1. Rappel : des algorithmes de tris déjà connus

2. Nouvelle approche

3. Performances du tri fusion

3.1 Mesure de la durée d'exécution

3.2 Complexité

4. Timsort

Fonction `sorted` Diviser pour régner

- └ Performances du tri fusion
 - └ Mesure de la durée d'exécution
 - └ Mesure de la durée d'exécution

Mesure de la durée d'exécution

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.

Mesure de la durée d'exécution

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.

Fonction `sorted`
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Fonction sortedDiviser pour régner

- Performances du tri fusion
 - Mesure de la durée d'exécution
 - Correction

Correction

```
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
```

Correction

```
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
```

Fonction sortedDiviser pour régner

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

Sommaire

1. Rappel : des algorithmes de tris déjà connus

2. Nouvelle approche

3. Performances du tri fusion

3.1 Mesure de la durée d'exécution

3.2 Complexité

4. Timsort

Fonction `sorted` Diviser pour régner

└─ Performances du tri fusion

└─ Complexité

└─ Complexité du découpage

Complexité du découpage

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

Complexité du découpage

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

Fonction `sorted`
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Fonction `sorted` Diviser pour régner

└─ Performances du tri fusion

└─ Complexité

└─ Complexité du découpage

Complexité du découpage

- 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 ?

Complexité du découpage

- 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 ?

Fonction `sorted`
Diviser pour régner

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

Fonction `sorted` Diviser pour régner

└─ Performances du tri fusion

└─ Complexité

└─ Complexité du découpage

Complexité du découpage

- 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 ?
-

Complexité du découpage

- 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 ?
-

Fonction `sorted`
Diviser pour régner

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

Mathématiquement nous cherchons a tel que :

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

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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution**Complexité**

Timsort

Fonction `sorted` Diviser pour régner

└ Performances du tri fusion

└ Complexité

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

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

$$\Leftrightarrow n = 2^a$$

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

$$\Leftrightarrow \log_2 n = a$$

À retenirLa complexité du découpage en sous-listes est **logarithmique**.Le logarithme base 2 noté \log_2 se définit : $\log_2(2^x) = x$.

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

$$\Leftrightarrow n = 2^a$$

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

$$\Leftrightarrow \log_2 n = a$$

À retenirLa complexité du découpage en sous-listes est **logarithmique**.Fonction `sorted`
Diviser pour régner

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

Fonction `sorted` Diviser pour régner

└─ Performances du tri fusion

└─ Complexité

└─ Complexité de la fusion

qq soit le niveau, pour fusionner on se contente de parcourir une fois un tableau

Complexité de la fusion

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

Complexité de la fusion

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

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

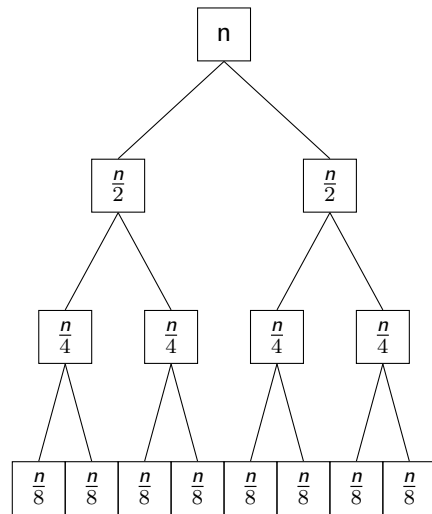
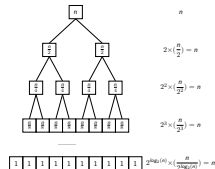
Complexité

Timsort

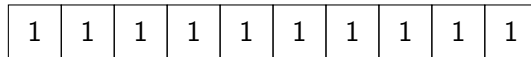
Fonction sortedDiviser pour régner

└ Performances du tri fusion

└ Complexité



.....



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

Fonction sorted
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution

Complexité

Timsort

Fonction `sorted` Diviser pour régner

└ Performances du tri fusion

└ Complexité

└ Complexité du tri fusion

Complexité du tri fusion

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

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

Complexité du tri fusion

À retenir

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

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

Fonction `sorted`
Diviser pour régnerRappel : 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
problèmeDiviser pour régner : un
algorithme récursif

Étape de la fusion

Performances du
tri fusionMesure de la durée
d'exécution**Complexité**

Timsort

Fonction `sorted` Diviser pour régner

└─ Performances du tri fusion

└─ Complexité

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.

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

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.

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

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

- Fonction `sorted`Diviser pour régner
 - Performances du tri fusion
 - Complexité
 - Correction

Correction

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

Correction

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

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

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

Sommaire

1. Rappel : des algorithmes de tri déjà connus

2. Nouvelle approche

3. Performances du tri fusion

4. Timsort

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

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.

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.