

# Complexité

INFORMATIQUE COMMUNE - TP n° 2.2 - Olivier Reynet

## À la fin de ce chapitre, je sais :

- ☞ calculer la complexité d'un algorithme simple
- ☞ donner les complexités dans le pire des cas des tris comparatifs génériques
- ☞ expliquer la différence entre le tri rapide et le tri fusion

## A Complexité d'algorithmes simples

A1. Calculer la complexité de l'algorithme 1. Y-a-il un pire et un meilleur des cas?

---

### Algorithme 1 Produit scalaire

---

```
1: Fonction PRODUIT_SCALAIRE( $x, y$ )                                ▷  $x$  et  $y$  sont des vecteurs à  $n$  éléments
2:    $s \leftarrow 0$ 
3:   pour  $i = 0$  à  $n - 1$  répéter
4:      $s \leftarrow s + x_i y_i$                                        ▷ coût?
5:   renvoyer  $s$ 
```

---

A2. Calculer la complexité de l'algorithme 2 dans le meilleur et dans le pire des cas.

---

### Algorithme 2 Palindrome

---

```
1: Fonction PALINDROME( $w$ )
2:    $n \leftarrow$  la taille de la chaîne de caractères  $w$ 
3:    $i \leftarrow 0$ 
4:    $j \leftarrow n - 1$ 
5:   tant que  $i < j$  répéter
6:     si  $w[i] = w[j]$  alors
7:        $i \leftarrow i + 1$ 
8:        $j \leftarrow j - 1$ 
9:     sinon
10:      renvoyer Faux
11:   renvoyer Vrai
```

---

A3. Calculer la complexité de l'algorithme 3. Y-a-t-il un pire et un meilleur des cas? On fait l'hypothèse que la fonction PUISSANCE( $a, b$ ) est de complexité constante  $O(1)$ . Cette hypothèse vous semble-t-elle raisonnable?

**Algorithme 3** Évaluation simple d'un polynôme

---

```

1: Fonction EVAL_POLYNÔME(p, v)
2:    $d \leftarrow$  degré de p
3:    $r \leftarrow p[0]$ 
4:   pour  $i = 1$  à  $d$  répéter
5:      $r \leftarrow r + p[i] \times \text{PUISSANCE}(v, i)$ 
6:   renvoyer r

```

---

A4. Utiliser la méthode de Horner (cf. algorithme 4) pour écrire un autre algorithme pour évaluer un polynôme. Quelle complexité pouvez-vous obtenir? Est-ce plus rapide?

**Algorithme 4** Évaluation d'un polynôme par la méthode d'Horner

---

```

1: Fonction HORNER(p, d, v)
2:    $r \leftarrow p[d]$ 
3:   pour  $i = d - 1$  à 0 répéter
4:      $r \leftarrow r \times v$ 
5:      $r \leftarrow r + p[i]$ 
6:   renvoyer r

```

---

**B Tri fusion****Algorithme 5** Tri fusion

---

```

1: Fonction TRI_FUSION(t)
2:    $n \leftarrow$  taille de t
3:   si  $n < 2$  alors
4:     renvoyer t
5:   sinon
6:      $t_1, t_2 \leftarrow \text{DÉCOUPER\_EN\_DEUX}(t)$ 
7:     renvoyer FUSION(TRI_FUSION( $t_1$ ), TRI_FUSION( $t_2$ ))

```

---

B1. Programmer le tri fusion en Python.

B2. Quelle est la complexité de cet algorithme? Y-a-t-il un pire et un meilleur cas?

B3. Vérifier, en mesurant le temps d'exécution, la justesse de votre calcul précédent.

**C Tri rapide**

C1. Programmer le tri rapide en Python.

C2. Quelle est la complexité de cet algorithme? Y-a-t-il un pire et un meilleur cas?

C3. Vérifier, en mesurant le temps d'exécution, la justesse de vos calculs précédents.

---

**Algorithme 6** Découper en deux

---

```
1: Fonction DÉCOUPER_EN_DEUX(t)
2:    $n \leftarrow$  taille de t
3:    $t_1, t_2 \leftarrow$  deux listes vides
4:   pour  $i = 0$  à  $n//2 - 1$  répéter
5:     AJOUTER( $t_1$ , t[i])
6:   pour  $j = n//2$  à  $n - 1$  répéter
7:     AJOUTER( $t_2$ , t[j])
8:   renvoyer  $t_1, t_2$ 
```

---

---

**Algorithme 7** Fusion de deux sous-tableaux triés

---

```
1: Fonction FUSION( $t_1, t_2$ )
2:    $n_1 \leftarrow$  taille de  $t_1$ 
3:    $n_2 \leftarrow$  taille de  $t_2$ 
4:    $n \leftarrow n_1 + n_2$ 
5:   t  $\leftarrow$  une liste vide
6:    $i_1 \leftarrow 0$ 
7:    $i_2 \leftarrow 0$ 
8:   pour k de 0 à n - 1 répéter
9:     si  $i_1 \geq n_1$  alors
10:      AJOUTER(t,  $t_2[i_2]$ )
11:       $i_2 \leftarrow i_2 + 1$ 
12:     sinon si  $i_2 \geq n_2$  alors
13:      AJOUTER(t,  $t_1[i_1]$ )
14:       $i_1 \leftarrow i_1 + 1$ 
15:     sinon si  $t_1[i_1] \leq t_2[i_2]$  alors
16:      AJOUTER(t,  $t_1[i_1]$ )
17:       $i_1 \leftarrow i_1 + 1$ 
18:     sinon
19:      AJOUTER(t,  $t_2[i_2]$ )
20:       $i_2 \leftarrow i_2 + 1$ 
21:   renvoyer t
```

---

---

**Algorithme 8** Tri rapide

---

```
1: Fonction TRI_RAPIDE(t)
2:    $n \leftarrow$  taille de t
3:   si  $n < 2$  alors
4:     renvoyer t
5:   sinon
6:      $t_1, \text{pivot}, t_2 \leftarrow$  PARTITION(t)
7:     renvoyer CONCATÉNER(TRI_RAPIDE( $t_1$ ), pivot, TRI_RAPIDE( $t_2$ ))
```

---

---

**Algorithme 9** Partition en deux sous-tableaux

---

```
1: Fonction PARTITION(t)
2:    $n \leftarrow$  taille de t
3:   pivot  $\leftarrow$  0
4:   i_pivot  $\leftarrow$  un nombre au hasard entre 0 et  $n - 1$  inclus
5:    $t_1, t_2$  deux listes vides
6:   pour  $k = 0$  à  $n$  répéter
7:     si  $k = i\_pivot$  alors
8:       pivot  $\leftarrow$  t[k]
9:     sinon si  $t[k] \leq t[i\_pivot]$  alors
10:      AJOUTER( $t_1$ , t[k])
11:    sinon
12:      AJOUTER( $t_2$ , t[k])
13:  renvoyer  $t_1$ , pivot,  $t_2$ 
```

---

**D Versions en place des tris**

---

**Algorithme 10** Tri fusion

---

```
1: Fonction TRI_FUSION(t, g, d)
2:   si  $g < d$  alors
3:      $m \leftarrow (g+d)/2$  ▷ on découpe au milieu
4:     TRI_FUSION(t, g, m)
5:     TRI_FUSION(t, m+1, d)
6:     FUSION(t, g, m, d) ▷ Sinon on n'arrête!
```

---

---

**Algorithme 11** Fusion de sous-tableaux triés

---

```
1: Fonction FUSION(t, g, m, d)
2:   ng ← m - g + 1
3:   nd ← d - m
4:   G, D ← deux tableaux de taille ng et nd
5:   pour k de 0 à ng répéter
6:     G[k] ← t[g + k]
7:   pour k de 0 à nd répéter
8:     D[k] ← t[m + 1 + k]
9:   i ← 0
10:  j ← 0
11:  k ← g
12:  tant que i < ng et j < nd répéter
13:    si G[i] ≤ D[j] alors
14:      t[k] ← G[i]
15:      i ← i + 1
16:    sinon
17:      t[k] ← D[j]
18:      j ← j + 1
19:    k ← k + 1
20:  tant que i < ng répéter
21:    t[k] ← G[i]
22:    i ← i + 1
23:    k ← k + 1
24:  tant que j < nd répéter
25:    t[k] ← D[j]
26:    j ← j + 1
27:    k ← k + 1
```

---

---

**Algorithme 12** Tri rapide

---

```
1: Fonction TRI_RAPIDE(t, p, d)
2:   si p < d alors
3:     i_pivot ← PARTITION(t, p, d)
4:     TRI_RAPIDE(t, p, i_pivot - 1)
5:     TRI_RAPIDE(t, i_pivot + 1, d)
```

▷ Sinon on n'arrête!

---

---

**Algorithme 13** Partition en deux sous-tableaux

---

```
1: Fonction PARTITION(t, p, d)
2:   i_pivot ← un nombre au hasard entre p et d inclus
3:   pivot ← t[i_pivot]
4:   ÉCHANGER(t,d,i_pivot)                                ▷ On met le pivot à la fin du tableau
5:   i = p - 1                                             ▷ i va pointer sur le dernier élément du premier tableau
6:   pour j = p à d -1 répéter
7:     si t[j] ≤ pivot alors
8:       i ← i + 1
9:       ÉCHANGER(t,i,j)                                ▷ On échange les places de t[i] et t[j]
10:  t[d] ← t[i+1]                                         ▷ t[i+1] appartient au tableau de droite
11:  t[i+1] ← pivot                                       ▷ Le pivot est entre les deux tableaux
12:  renvoyer i + 1                                     ▷ La place du pivot!
```

---