

Trier et rechercher

INFORMATIQUE COMMUNE - TP n° 4 - Olivier Reynet

À la fin de ce chapitre, je sais :

- ✍ coder un algorithme de tri simple et explicite
- ✍ évaluer le temps d'exécution d'un algorithme avec la bibliothèque `time`
- ✍ rechercher un élément dans un tableau séquentiellement ou par dichotomie itérative
- ✍ générer un graphique légendé avec la bibliothèque `matplotlib`

A Trier un tableau

- A1. On souhaite trier des listes Python, considérées ici comme des tableaux, avec des algorithmes différents (cf. algorithmes 1, 2 et 3). Chaque algorithme de tri est implémenté par une fonction Python. Le prototype de ces fonctions est `my_sort(t)`, où `t` est un paramètre formel qui représente le tableau à trier.

```
1  def my_sort(t):
2      # tri du tableau
3      # for i in range(len(t))
4      #     t[i] = ...
```

Cette fonction, une fois réalisée, trie le tableau `t` passé en paramètre mais ne renvoie rien (i.e. pas de `return`). Expliquer pourquoi.

Solution : Une liste Python est une variable muable. Les modifications apportées par l'algorithme sont donc réalisées directement sur la liste passée en paramètre. C'est pourquoi on n'a pas besoin d'effectuer un `return` pour récupérer le travail de la fonction.

- A2. Coder les algorithmes de tri par sélection, par insertion et par comptage en respectant le prototype défini à la question précédente ¹.
- A3. Tester ces algorithmes sur une **même** liste Python de longueur 20 et contenant de types `int` choisis aléatoirement entre 0 et 100.
- A4. Peut-t-on trier des listes de chaînes de caractères avec ces mêmes codes? Tester cette possibilité à l'aide de la liste `["Zorglub", "Spirou", "Fantasio", "Marsupilami", "Marsu", "Samovar", "Zantafio"]`. Analyser les résultats. Pourquoi est-ce possible? Pourquoi n'est-ce pas possible?
- La bibliothèque `matplotlib` permet de générer des graphiques à partir de données de type `list` qui constituent les abscisses et les ordonnées associées. La démarche à suivre est de :

1. On a le droit de collaborer, de se répartir les algorithmes et de s'échanger les codes s'ils sont corrects!

Algorithme 1 Tri par sélection

```

1: Fonction TRIER_SELECTION(t)
2:    $n \leftarrow \text{taille}(t)$ 
3:   pour  $i$  de 0 à  $n - 1$  répéter
4:      $\text{min\_index} \leftarrow i$                                 ▷ indice du prochain plus petit
5:     pour  $j$  de  $i + 1$  à  $n - 1$  répéter                    ▷ pour tous les éléments non triés
6:       si  $t[j] < t[\text{min\_index}]$  alors
7:          $\text{min\_index} \leftarrow j$                             ▷ c'est l'indice du plus petit non trié!
8:      $\text{échanger}(t[i], t[\text{min\_index}])$                     ▷ c'est le plus grand des triés!

```

Algorithme 2 Tri par insertion

```

1: Fonction TRIER_INSERTION(t)
2:    $n \leftarrow \text{taille}(t)$ 
3:   pour  $i$  de 1 à  $n - 1$  répéter
4:      $\text{à\_insérer} \leftarrow t[i]$ 
5:      $j \leftarrow i$ 
6:     tant que  $t[j - 1] > \text{à\_insérer}$  et  $j > 0$  répéter
7:        $t[j] \leftarrow t[j - 1]$                                 ▷ faire monter les éléments
8:        $j \leftarrow j - 1$ 
9:      $t[j] \leftarrow \text{à\_insérer}$                                 ▷ insertion de l'élément

```

Algorithme 3 Tri par comptage

```

1: Fonction TRIER_COMPTAGE( $t$ )
2:    $v_{\max} \leftarrow \text{maximum de } t$                         ▷  $v_{\max}$  est le plus grand entier à trier
3:    $n \leftarrow \text{taille}(t)$ 
4:    $\text{comptage} \leftarrow \text{un tableau de taille } v_{\max} + 1 \text{ initialisé avec des zéros}$ 
5:    $\text{résultat} \leftarrow \text{un tableau de taille } n$ 
6:   pour  $i$  de 0 à  $n - 1$  répéter                            ▷ dénombrer chaque valeur du tableau.
7:      $\text{key} \leftarrow t[i]$ 
8:      $\text{comptage}[\text{key}] \leftarrow \text{comptage}[\text{key}] + 1$         ▷ création d'un histogramme.
9:   pour  $i$  de 1 à  $v_{\max}$  répéter                            ▷ cumuler les décomptes.
10:     $\text{comptage}[i] \leftarrow \text{comptage}[i] + \text{comptage}[i - 1]$ 
11:   pour  $i$  de 0 à  $n - 1$  répéter                            ▷ créer le tableau trié à partir des cumuls.
12:     $\text{key} \leftarrow t[i]$ 
13:     $\text{résultat}[\text{comptage}[\text{key}] - 1] \leftarrow \text{key}$ 
14:     $\text{comptage}[\text{key}] \leftarrow \text{comptage}[\text{key}] - 1$ 
15:   renvoyer output

```

- importer la bibliothèque `from matplotlib import pyplot as plt`
- créer une figure `plt.figure()`
- tracer une courbe `plt.plot(x,y)` si `x` et `y` sont les listes des abscisses et des ordonnées associées. La bibliothèque trace les points $(x[i], y[i])$ sur le graphique.
- ajouter les éléments de légende et de titre,
- montrer la figure ainsi réalisée `plt.show()`.

La bibliothèque `time` permet notamment de mesurer le temps d'exécution d'un code. Un exemple de code utilisant ces deux bibliothèques est donné ci-dessous. Le graphique qui en résulte est montré sur la figure 1.

Code 1 – Exemple d'utilisation des bibliothèques `time` et `matplotlib`

```
1 import time
2 from matplotlib import pyplot as plt
3
4
5 def to_measure(d):
6     time.sleep(d) # Do nothing, wait for d seconds
7
8
9 # Simple use
10 tic = time.perf_counter()
11 to_measure(0.1)
12 toc = time.perf_counter()
13
14 print(f"Execution time : {toc - tic} seconds")
15
16 # Plotting results
17 timing = []
18 delay = [d / 1000 for d in range(1, 100, 5)]
19 for d in delay:
20     tic = time.perf_counter()
21     to_measure(d)
22     toc = time.perf_counter()
23     timing.append(toc - tic)
24
25 plt.figure()
26 plt.plot(delay, timing, color='cyan', label='fonction to_measure')
27 plt.xlabel('Delay', fontsize=18)
28 plt.ylabel("Execution time", fontsize=16)
29 plt.legend()
30 plt.show()
```

- A5. À l'aide de la bibliothèque `matplotlib`, tracer les temps d'exécution nécessaires au tri d'un même tableau d'entiers par les algorithmes implémentés. On pourra également les comparer à la fonction `sorted` de Python. Analyser les résultats. Essayer de qualifier les coûts des algorithmes en fonction de la taille du tableau d'entrée.



FIGURE 1 – Figure obtenue à partir des bibliothèques matplotlib et time et du code 1

Solution :**Code 2 – Trier des tableaux**

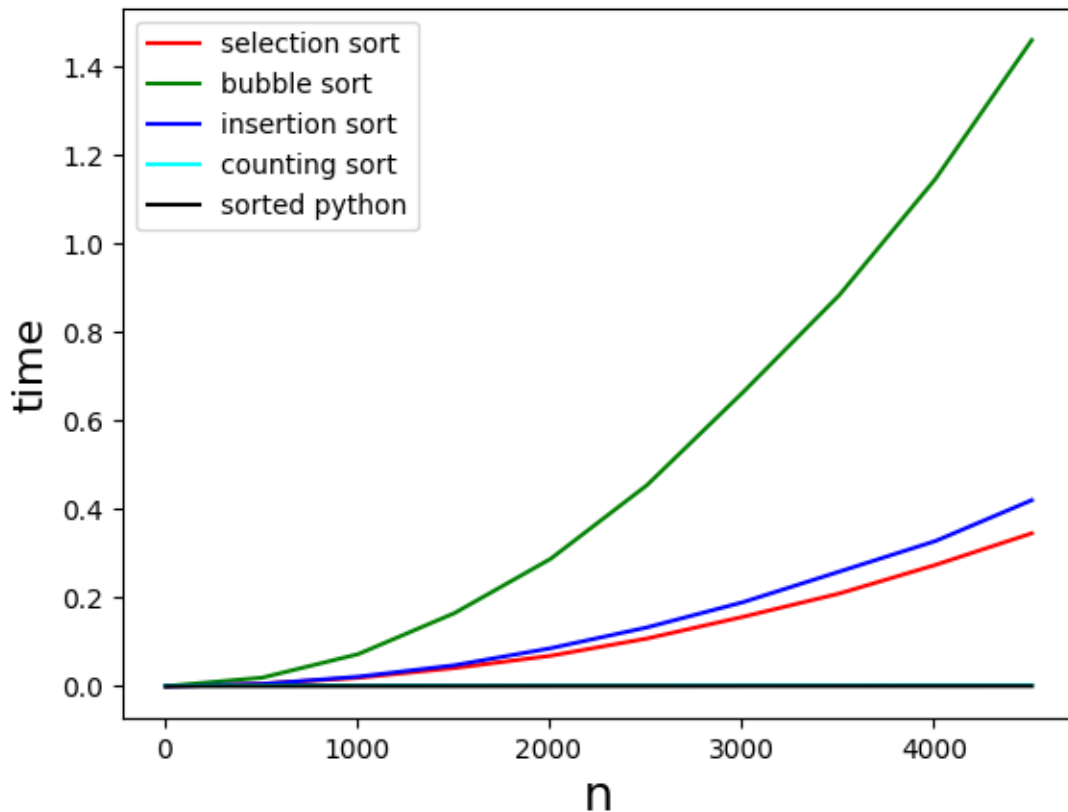
```
1 import time
2 from random import randint
3
4
5 def swap(t, i, j):
6     t[i], t[j] = t[j], t[i]
7
8
9 def selection_sort(t):
10     for i in range(len(t)):
11         min_index = i
12         for j in range(i + 1, len(t)):
13             if t[j] < t[min_index]:
14                 min_index = j
15         swap(t, i, min_index)
16         # print(t)
17
18
19 def insertion_sort(t):
20     for i in range(1, len(t)):
21         to_insert = t[i]
22         j = i
23         while t[j - 1] > to_insert and j > 0:
24             t[j] = t[j - 1]
25             j -= 1
26         t[j] = to_insert
27         # print(t)
28
29
30 def bubble_sort(t):
31     for i in range(len(t) - 1):
32         for j in range(len(t) - 1):
33             if t[j] > t[j + 1]:
34                 swap(t, j, j + 1)
35         # print(t)
36
37
38 def counting_sort(t):
39     v_max = max(t)
40     count = [0] * (v_max + 1)
41     for i in range(len(t)):
42         count[t[i]] += 1
43     for i in range(1, v_max + 1):
44         count[i] += count[i - 1]
45     output = [None for i in range(len(t))]
46     for i in range(len(t)):
47         output[count[t[i]] - 1] = t[i]
48         count[t[i]] -= 1
49     return output
50
51
52 def test_sort(t, method):
```

```

53     tmp = t[:]
54     if method in [selection_sort, bubble_sort, insertion_sort]:
55         method(tmp)
56     else:
57         tmp = method(tmp)
58     print(f"--> Method {method.__name__} :\n {t}\n {tmp}")
59
60
61 def sort_timing():
62     sizes = [i for i in range(10, 5000, 500)]
63     M = 5 * max(sizes)
64     results = []
65     for i in range(len(sizes)):
66         t = [randint(0, M) for _ in range(sizes[i])]
67         mem_t = t[:]
68         results.append([])
69         for method in [selection_sort, bubble_sort, insertion_sort,
70                        counting_sort, sorted]:
71             t = mem_t[:]
72             tic = time.perf_counter()
73             method(t)
74             toc = time.perf_counter()
75             results[i].append(toc - tic)
76         print(f"# {i} - {sizes[i]} -> {results}")
77
78     sel = [results[i][0] for i in range(len(sizes))]
79     bub = [results[i][1] for i in range(len(sizes))]
80     ins = [results[i][2] for i in range(len(sizes))]
81     cnt = [results[i][3] for i in range(len(sizes))]
82     p = [results[i][4] for i in range(len(sizes))]
83
84     from matplotlib import pyplot as plt
85
86     plt.figure()
87     plt.plot(sizes, sel, color='red', label='selection sort')
88     plt.plot(sizes, bub, color='green', label='bubble sort')
89     plt.plot(sizes, ins, color='blue', label='insertion sort')
90     plt.plot(sizes, cnt, color='cyan', label='counting sort')
91     plt.plot(sizes, p, color='black', label='sorted python')
92     plt.xlabel('n', fontsize=18)
93     plt.ylabel('time', fontsize=16)
94     plt.legend()
95     plt.show()
96
97 if __name__ == "__main__":
98     N = 20
99     M = 100
100
101     t = [randint(0, M) for _ in range(N)]
102
103     for method in [selection_sort, insertion_sort, bubble_sort, counting_sort]:
104         test_sort(t, method)
105
106     t = ["Zorglub", "Spirou", "Fantasio", "Marsupilami", "Marsu", "Samovar", "

```

```
Zantafio"]  
107 for method in [selection_sort, insertion_sort, bubble_sort]: # ,  
    counting_sort]:  
108     test_sort(t, method)  
109  
110 sort_timing()
```



B Tri bulle (Facultatif)

Le tri bulle est un tri par comparaisons et échanges, stable et en place. Il n'est pas efficace et sa vocation reste pédagogique.

Le principal avantage du tri bulle est qu'il est simple à expliquer. Son principe est le suivant : on prend chaque élément du tableau et on le fait monter d'une place dans le tableau si celui-ci est plus grand que son successeur en échangeant leurs places. Sinon, on fait prendre le successeur et on opère à l'identique.

Algorithme 4 Tri bulle

```

1: Fonction BUBBLE_SORT(t)
2:   n ← taille(t)
3:   pour i de 0 à n-2 répéter
4:     pour j de 0 à n-i-2 répéter
5:       si t[j] > t[j+1] alors
6:         échange(t[j], t[j+1])

```

▷ faire monter la bulle

C Recherche d'un élément dans un tableau

On considère une liste L contenant des éléments de type **int**. Cette liste est **triée** par ordre croissant de ses éléments. On veut savoir si un élément x est présent dans L et comparer les performances des approches séquentielles et la dichotomiques. On dispose des algorithmes 5, 6 et 7.

Algorithme 5 Recherche séquentielle d'un élément dans un tableau

```

1: Fonction RECHERCHE_SÉQUENTIELLE(t, elem)
2:   n ← taille(t)
3:   pour i de 0 à n - 1 répéter
4:     si t[i] = elem alors
5:       renvoyer i
6:   renvoyer l'élément n'a pas été trouvé

```

▷ élément trouvé, on renvoie sa position dans t

Algorithme 6 Recherche d'un élément par dichotomie dans un tableau trié

```

1: Fonction RECHERCHE_DICHOTOMIQUE(t, elem)
2:   n ← taille(t)
3:   g ← 0
4:   d ← n-1
5:   tant que g ≤ d répéter
6:     m ← (g+d)//2
7:     si t[m] < elem alors
8:       g ← m + 1
9:     sinon si t[m] > elem alors
10:      d ← m - 1
11:   sinon
12:     renvoyer m
13:   renvoyer l'élément n'a pas été trouvé

```

▷ ≤ cas où valeur au début, au milieu ou à la fin
 ▷ Division entière : un indice est un entier!
 ▷ l'élément devrait se trouver dans t[m+1, d]
 ▷ l'élément devrait se trouver dans t[g, m-1]
 ▷ l'élément a été trouvé

- C1. Coder l'algorithme de recherche séquentielle d'un élément dans un tableau. Lorsque l'élément n'est pas présent dans le tableau, la fonction Python renvoie None. Sinon, elle renvoie l'indice de l'élément trouvé dans le tableau. Vérifier que cet algorithme fonctionne sur un tableau d'entiers de 20 éléments rempli aléatoirement. Dans le pire des cas, quel est le coût d'une recherche séquentielle en fonction de la taille du tableau?
- C2. Coder l'algorithme 6. Lorsque l'élément n'est pas présent dans le tableau, la fonction Python renvoie None. Sinon, elle renvoie l'indice de l'élément trouvé dans le tableau. Vérifier que cet algorithme

Algorithme 7 Recherche d'un élément par dichotomie dans un tableau trié, renvoyer l'indice minimal en cas d'occurrences multiples.

```

1: Fonction RECHERCHE_DICHOTOMIQUE_INDICE_MIN(t, elem)
2:   n ← taille(t)
3:   g ← 0
4:   d ← n-1
5:   tant que g < d répéter                                ▷ attention au strictement inférieur!
6:     m ← (g+d)//2                                          ▷ Un indice de tableau est un entier!
7:     si t[m] < elem alors
8:       g ← m + 1                                          ▷ l'élément devrait se trouver dans t[m+1, d]
9:     sinon
10:      d ← m                                              ▷ l'élément devrait se trouver dans t[g, m]
11:   si t[g] = elem alors
12:     renvoyer g
13:   sinon
14:     renvoyer l'élément n'a pas été trouvé

```

fonctionne sur un tableau d'entiers de 20 éléments rempli aléatoirement et trié.

- C3. Coder l'algorithme 7. Vérifier que cet algorithme fonctionne sur un tableau d'entiers de 20 éléments rempli aléatoirement et trié et que l'indice renvoyé est bien l'indice minimal de la première occurrence de l'élément recherché.
- C4. On suppose que la longueur de la liste est une puissance de 2, c'est à dire $n = 2^p$ avec $p \geq 1$. Combien d'étapes l'algorithme 7 comporte-t-il? En déduire le nombre de comparaisons effectuées, dans le cas où l'élément est absent, en fonction de p puis de n , et comparer avec l'algorithme de recherche séquentielle.
- C5. Tracer le graphique des temps d'exécution des algorithmes précédent en fonction de n . Les tracés sont-ils cohérents avec les calculs des coûts effectués précédemment?
- C6. La recherche dichotomique fonctionne-t-elle sur les listes non triées? Donner un contre-exemple si ce n'est pas le cas.

Solution :

Code 3 – Recherche un élément dans un tableau

```

1 import time
2 from random import randint
3
4
5 def seq_search(t, elem):
6     for i in range(len(t)):
7         if t[i] == elem:
8             return i
9     return None
10
11
12 def dichotomic_search(t, elem):

```

```
13     g = 0
14     d = len(t) - 1
15     while g <= d:
16         m = (d + g) // 2
17         # print(g, m, d)
18         if t[m] < elem:
19             g = m + 1
20         elif t[m] > elem:
21             d = m - 1
22         else:
23             return m
24     return None
25
26
27 def dichotomic_search_left_most(t, elem):
28     g = 0
29     d = len(t) - 1
30     while g < d:
31         m = (d + g) // 2
32         # print(g, m, d)
33         if t[m] < elem:
34             g = m + 1
35         else:
36             d = m
37     if t[g] == elem:
38         return g
39     else:
40         return None
41
42
43 def search_timing():
44     sizes = [i for i in range(10, 100000, 500)]
45     M = 5 * max(sizes)
46     results = []
47     for i in range(len(sizes)):
48         t = sorted([randint(0, M) for _ in range(sizes[i])])
49         mem_t = t[:]
50         results.append([])
51         for method in [seq_search, dichotomic_search,
52                        dichotomic_search_left_most]:
53             t = mem_t[:]
54             tic = time.perf_counter()
55             method(t, M // 4)
56             toc = time.perf_counter()
57             results[i].append(toc - tic)
58         print(f"#{i} - {sizes[i]} -> {results}")
59
60     seq = [results[i][0] for i in range(len(sizes))]
61     dics = [results[i][1] for i in range(len(sizes))]
62     dicslm = [results[i][2] for i in range(len(sizes))]
63
64     from matplotlib import pyplot as plt
65
66     plt.figure()
67     plt.plot(sizes, seq, color='cyan', label='Sequential search')
```

```

67     plt.plot(sizes, dics, color='blue', label='Dichotomic search')
68     plt.plot(sizes, dicslm, color='black', label='Dichotomic search left most')
69     plt.xlabel('n', fontsize=18)
70     plt.ylabel('time', fontsize=16)
71     plt.legend()
72     plt.show()
73
74
75     if __name__ == "__main__":
76
77         t = [0, 1, 2, 4, 7, 7, 9, 13, 17, 65, 99, 99, 99, 99, 101, 111, 111, 111,
78             113]
79
80         print(t)
81         for value in t:
82             print(value, seq_search(t, value), dichotomic_search(t, value),
83                   dichotomic_search_left_most(t, value),
84                   dichotomic_search(t, value) == dichotomic_search_left_most(t,
85                                     value))
86
87         search_timing()

```

