02 - Le projet tri rapide d'une liste

Lionel Bastard & Jean-Christophe Toussaint

12 novembre 2025

1 Introduction

On génère une liste aléatoire A de $N = 1000$ entiers que l'on désire trier par ordre croissant.

1. Utiliser la fonction numpy `np.random.randint` pour générer les nombres. Que remarquez-vous quand vous exécutez deux fois cette fonction ?
2. Faire la même chose en la précédant de l'instruction `np.random.seed(12345)` qui initialise le générateur de nombres aléatoires avec une graine. Même question que précédemment.

2 Tri par bulle

1. Dévélopper une fonction `Tri_bulle(A)` permettant de trier le tableau de manière croissante. La méthode la plus simple dite de "tri par bulle" consiste à permuter si nécessaire, deux éléments successifs du tableau et à recommencer dès le début du tableau tant qu'il n'est pas trié.

```
1: function TRI_BULLE(T)
2:     n ← LEN(T)
3:     change ← True
4:     while change == True do
5:         change ← False
6:         for i ← 0 to n − 1 do
7:             j ← i + 1
8:             if T[i] > T[j] then
9:                 change ← True
10:                T ← ECHANGER(T, i, j)
       ▷ T[i] ↔ T[j]
11:            end if
12:        end for
13:    end while
14:    return T
15: end function
```

   Noter que le symbole ▷ désigne le début d'un commentaire.

2. On désire mesurer le temps CPU nécessaire pour faire le tri. Utiliser la portion de code suivant :

```python
from time import process_time

# Start the stopwatch / counter
t1_start = process_time()

A=tri_bulle(A)

# Stop the stopwatch / counter
t1_stop = process_time()
print("Elapsed time (s):", t1_stop-t1_start)
```

Etudier l'évolution du temps de calcul en fonction de $N = 2^n$ pour $n \in \mathbb{N}$. Tracer le temps de calcul en fonction de N en représentation loglog. Pour cette étude de performance, on initialisera le générateur avec la même graine pour générer toujours le même tableau pour une taille donnée.
L'autre possibilité serait de faire une étude statistique sur un grand nombre d'échantillons de tableaux de même taille.

3. Hoare a inventé en 1962, une méthode de type "diviser pour régner" permettant de trier un tableau en $N \ln N$ opérations. Son algorithme s'écrit :

```
1: function TRI_RAPIDE(T, g, d)
2:     if g < d then
3:         ip, T ← PARTITION(T, g, d)
4:         T ← TRI_RAPIDE(T, g, ip)
5:         T ← TRI_RAPIDE(T, ip+1, d)
6:     end if
7:     return T
8: end function
9:
10: function PARTITION(T, g, d)
11:     pivot ← T[g]
12:     i ← g − 1
13:     j ← d + 1
14:     while true do
15:         repeat
16:             j ← j − 1
17:         until T[j] <= pivot
18:         repeat
19:             i ← i + 1
20:         until T[i] >= pivot
21:         if i < j then
22:             T ← ECHANGER(T, i, j)    ▷ T[i] ↔ T[j]
23:         else
24:             return j, T
25:         end if
26:     end while
27: end function
```

Noter que le symbole ▷ désigne le début d'un commentaire.

4. La boucle conditionnelle `repeat ... until (condition)` n'existe pas en Python, on vous demande de proposer une implémentation possible. On prend l'exemple d'une boucle incrémentale gérée par une variable i variant de 0 à 9 avant l'incrémentation.

1: $i \leftarrow 0$
2: **repeat** ▷ afficher i
3: $\quad i \leftarrow i + 1$
4: **until** $i == 10$

Cet algorithme peut être traduit en python par :

```
i=0
while True:
  print(i)
  i=i+1
  if i==10:
    break
```

Tester cette implémentation dans un programme à part.

5. Implémenter l'algorithme suivant sous la forme d'une fonction Python. On rappelle que l'échange des valeurs de deux variables $a \leftrightarrow b$ en python s'écrit : $a,\ b = b,\ a$. Cette instruction fonctionne aussi pour 2 éléments (i, j) d'un tableau numpy : $T[i], T[j] = T[j], T[i]$. Une version plus optimisée s'écrit : $T[[i, j]] = T[[j, i]]$
6. Son appel tri_rapide(A, 0, len(A)-1) trie le tableau A in-place.
7. Etudier et tracer l'évolution du temps de calcul en fonction de N. Comparez avec la méthode précédente.