I21 - Algorithmique élémentaire
TP 3 - Analyse en pire, meilleur et cas moyen.

Année universitaire 2022/23

On rappelle le code source permettant de tracer une courbe avec `mathplotlib`:

```
import matplotlib.pyplot as plt

# On cree la liste des abscisses des point que l'on veut tracer
abscisse = []
for i in range(10):
  abscisse += [i]

# On cree la liste des ordonnees des points que l'on veut tracer:
carre = []
for i in range(len(abscisse)):
  carre += [abscisse[i]**2]

cube = []
for i in range(len(abscisse)):
  cube += [abscisse[i]**3]

# plot() trace les courbes correspondantes en leur associant un nom
# et show() les affiche a l'ecran.
# ATTENTION au caractere , avant le =
courbe_carre, =  plt.plot(abscisse, carre, label='x^2')
courbe_cube, =  plt.plot(abscisse, cube, label='x^3')
plt.legend(handles=[courbe_carre, courbe_cube])
plt.show()
```

Recherche séquentielle

On considère le problème suivant:

Problème	:	Recherche
Entrée	:	`T` un tableau d'entiers et `x` un entier
Sortie	:	`Vrai` si `x` est dans `T`, `Faux` sinon

1. Écrire une fonction `recherche(x,T)` qui renvoie `True` si `x` apparait dans le tableau `T` et `False` sinon. La fonction devra de plus renvoyer le nombre d'instructions qu'elle effectue.
2. Écrire une fonction `recherche_instance(n)` qui renvoie une instance aléatoire du problème de recherche pour un tableau de taille `n`.
3. Écrire deux fonctions `recherche_meilleur_cas(n)` et `recherche_pire_cas(n)` qui renvoient respectivement une instance aléatoire favorable et défavorable du problème de recherche pour un tableau de taille `n`.
4. Étudier la complexité de la fonction `recherche(x,T)` en terme de nombre de nombre d'instructions et tracer trois courbes correspondant au pire cas, meilleur cas et cas moyen.

Tris Standards

On considère le problème suivant:

Problème	:	Tri
Entrée	:	`T` un tableau d'entiers
Sortie	:	`T` trié dans l'ordre croissant

1. Écrire trois fonctions de tris `selection(T)`, `bulles(T)` et `insertion(T)`. Chaque fonction devra de plus renvoyer le nombre d'instructions qu'elle effectue.
2. Écrire une fonction `tri_instance(n)` qui renvoie une instance aléatoire du problème de tri pour un tableau de taille `n`.
3. Pour chaque algo de tri, écrire deux fonctions `*_meilleur_cas(n)` et `*_pire_cas(n)` qui renvoient respectivement une instance de taille `n` aléatoire favorable et défavorable pour l'algorithme en question.
4. Étudier la complexité des trois algorithmes en terme de nombre de nombre d'instructions et tracer pour chacun trois courbes correspondant au pire cas, meilleur cas et cas moyen.