Exercice 1

Chronométrage de la recherche séquentielle

On cherche à évaluer le temps mis par un programme de recherche séquentielle pour trouver un mot quelconque, dans un dictionnaire donnée.

1.1 Remettre les éléments de script dans l'ordre.

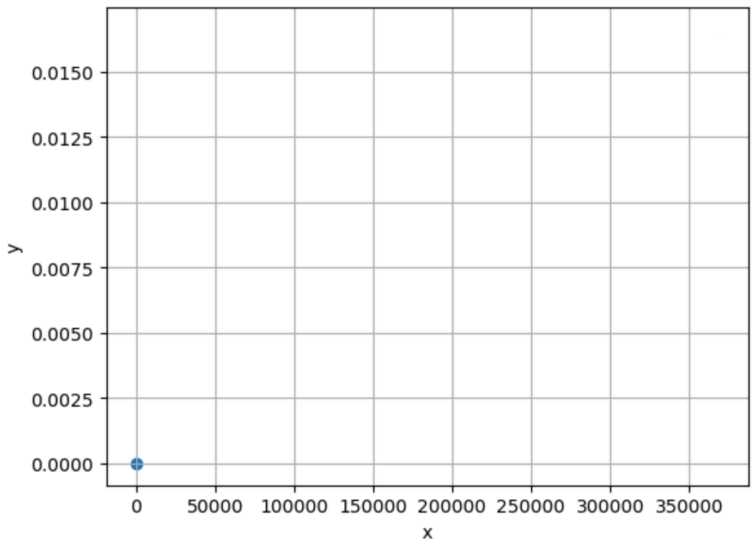
Préciser quels éléments font partie du programme principal. Et quels éléments font partie d'un bloc de code associé à une boucle.

- definition d'une fonction `recherche_seq` qui cherche un mot `X` dans un dictionnaire `mots`
- import des librairies numpy, time, random
- création d'une liste vide T pour stocker les valeurs de temps mesurées par chronométrage de la fonction
- création d'une liste vide L pour stocker les couples `[len(mots), temps moyen r]`
- mesure de t1
- mesure de t0
- tirage aléatoire d'un mot `X` dans `mots`
- appel de la fonction `recherche_seq(X,mots)` pour rechercher un mot `X` dans une liste `mots`
- boucle `for` pour répéter les opérations 100 fois
- ajouter t1 - t0 à la liste T
- ajouter `[len(mots), temps moyen r]` dans L
- ouvrir le fichier "gutenberg.txt" et charger les mots dans la liste `mots`
- calculer la moyenne des valeurs de T et les stocker dans une variable simple r.

1.2 Tracé d'un nuage de points

A partir de la liste `L`, on place les valeurs sur un même graphique : La taille du dictionnaire en abscisses, et le temps mesuré en ordonnée. Placer ces valeurs sur le graphique ci-dessous.

dictionnaire	taille	temps mesuré
aucun	0	0
liste_francais.txt	21 740	1.065e-3
pli07.txt	78 855	3.340e-3
gutenberg.txt	336 530	1.6059e-2
ods4.txt	369 085	1.6616e-2



1.3 Questions

1. Pourquoi faut-il répéter plusieurs fois la recherche dans une même liste, mais avec des mots `X` différents ?
2. Pourquoi les valeurs mesurées sont-elles croissantes avec la taille du dictionnaire ?
3. Les valeurs du temps mesuré, varient-elles de manière liénaire avec la taille du dictionnaire ? Pourquoi ?

1.4 Recherche dichotomique

On rappelle l'algorithme de la recherche dichotomique

```
def recherche_dicho(X,L):
    """recherche dans une liste L une valeur X
    Params:
    ------
    L: list, valeurs triees dans le sens croissant
    X : int ou str, valeur a trouver
    Return :
    --------
    milieu (indice dans la liste) si X est présent dans la liste
    -1 sinon"""
    # on initialise les indices début et fin aux extrémités de la liste
    gauche = 0
    droite = len(L)
    trouve = False

    while gauche <= droite and not trouve:
        # On se place au milieu de la liste
        milieu = (gauche + droite) // 2 # il, s'agit d'une division
   entière
        if L[milieu] == X:
            trouve = True
        elif L[milieu] < X:
            gauche = milieu + 1
        else:
            droite = milieu - 1
```

```
25      if not trouve : return -1
26      return milieu
```

a. Dans la fonction recherche_dicho, on réalise une comparaison entre X et L[milieu]. Que donnent les comparaisons suivantes, selon s'il s'agit d'un ordre numérique ou d'un ordre lexicographique ?

- 98 < 110
- 'AMEUTERAIT' > 'AMEUTERAS'

b. A partir du script de la fonction. Adapter celui-ci pour rechercher le zero d'une fonction lorsque l'on donne les bornes `a` et `b` pour `x` qui encadrent la valeur f(x)=0. f(a) et f(b) sont alors de signes opposés.

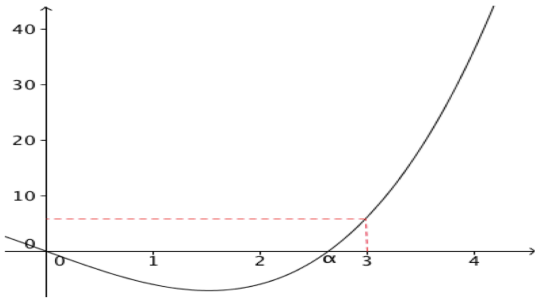


FIGURE 1 – f(x) = x**3 - 7x*

Besoin d'aide ?: : https ://fr.wikipedia.org/wiki/Méthode_de_dichotomie

c. Citer les principales différences entre les 2 scripts de recherche dichotomique

Exercice 2

Comparaison des fonctions de n

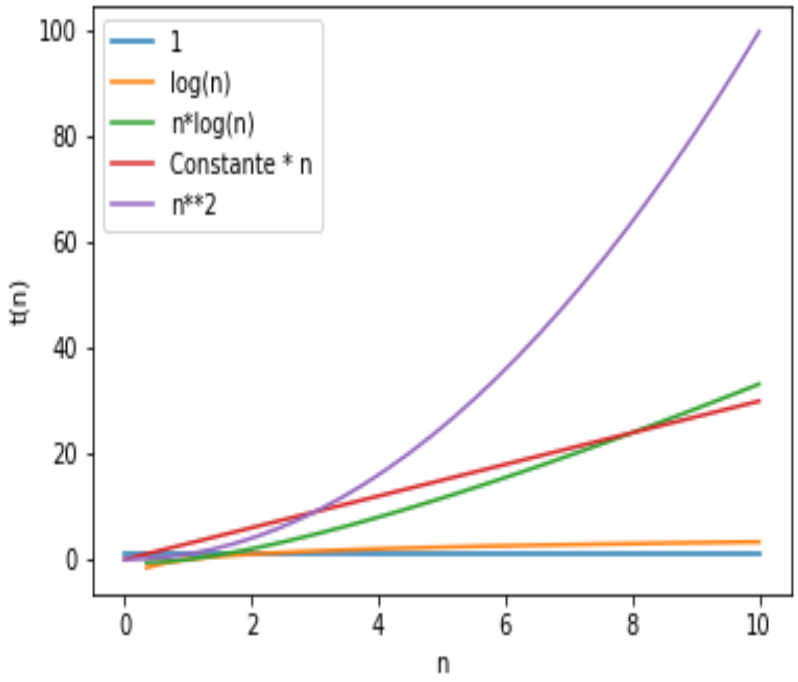


Figure 2 – n**2 domine les autres fonctions

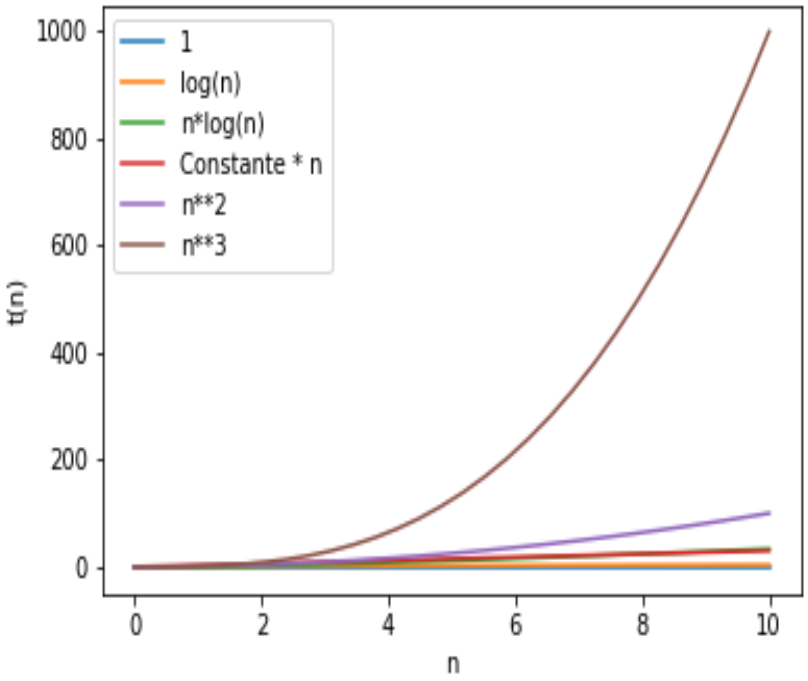


Figure 3 – n**3 domine les autres fonctions

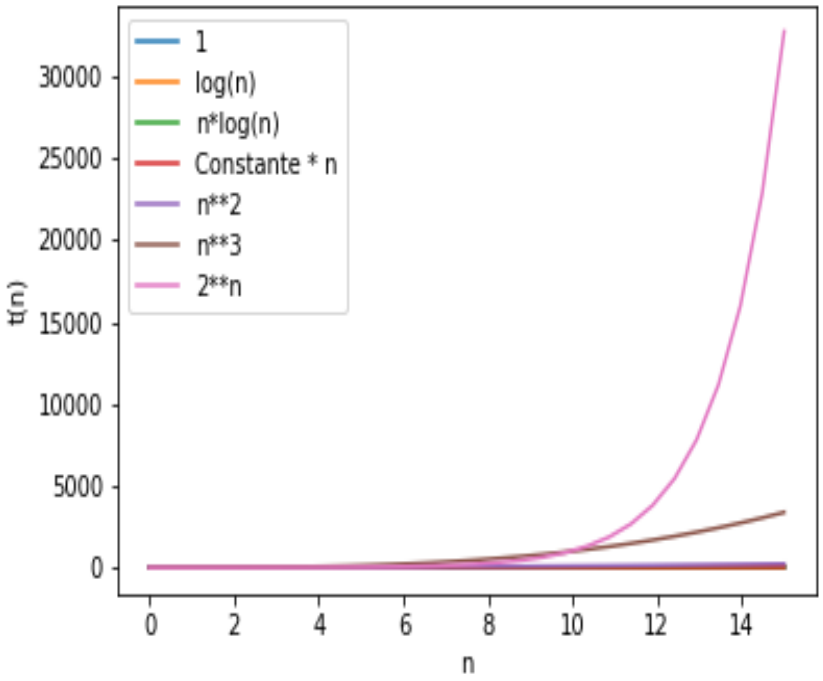


FIGURE 4 – 2**n domine les autres fonctions

a. Classer les fonctions par leur croissance : log, polynomial quadratique, n*log, linéaire, polynomial cubique, constante.

b. Que signifie la phrase : *La courbe n^2 est la plus divergente.*

c. L'écart de durée entre un algorithme de complexité linéaire, et un algorithme de complexité logarithmique : va-t-il augmenter ou diminuer lorsque la taille du paramètre n va augmenter ?

d. L'instruction `%%timeit` est une *magic function* en Python qui fournit une façon simple de mesurer le temps d'exécution de fragments de code Python. On mesure le temps mis par chacune des 2 fonctions de recherche pour trouver un mot, pris au hasard dans une même liste de mots.

```
%%timeit
recherche_mot(X,mots)
# affiche
1.88 ms ± 46.7 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each
    )
%%timeit
recherche_dicho(X,mots)
# affiche
2.48 µs ± 45.9 ns per loop (mean ± std. dev. of 7 runs, 100000 loops
    each)
```

- Calculer le rapport approximatif $\frac{seq}{dicho}$ pour le temps de recherche moyen mesuré pour ces 2 fonctions. (en ordre de grandeur)
- Ces valeurs vous semblent-elles cohérentes au vu des courbes ci-dessus ? Expliquez.