1. Structure de données I

1.1 Structures de données - Rappels de première

L'an dernier vous avez découvert 3 très belles structures de données qui vous ont permis de faire votre premier voyage au pays de Python : le tuple, le tableau et le dictionnaire. Chacune de ces structures ont des qualités ou défauts différents qui font qu'en fonction du problème à coder le programmeur privilégiera l'une ou l'autre de ces structures. Il existe bien d'autres structures qui seront étudiées cette année au chapitre 3, les structures de liste, de pile et de file. Peut-être aurons-nous aussi la chance de voir le temps d'un exercice la structure ensemble.

1.1.1 Tuple, tableau, dictionnaire

- le tuple :

```
>>> hedy_tuple = ("Hedy", "Lamarr", 9, 11, 1914)
```

Les valeurs stockées peuvent être de type différents (ici nous avons des chaînes de caractères, *String*, et des entiers, *integer*).
Le tuple est entre parenthèses [1], les valeurs étant séparées par des virgules.
Les valeurs ne peuvent être modifiées, la taille du tuple non plus.

- le tableau :

```
>>> hedy_tab = [9, 11, 1914]
```

Les valeurs stockées doivent être du même type [2] (ici, il n'y a que des entiers, *integer*).
Le tableau est entre crochets, les valeurs étant séparées par des virgules.
Les valeurs sont modifiables, mais pas la taille du tableau [3].

1. Techniquement, les parenthèses ne sont pas obligatoires, mais très fortement conseillées au moment de la création du **tuple**. Par contre le **tuple** est toujours affiché dans la console avec ses parenthèses.
2. En réalité, la structure **tableau** n'est pas implémenté en Python. A la place, il y a la structure **liste** qui permet de stocker des valeurs de types différents.
3. Sauf qu'à nouveau, nous n'avons pas de **tableau** mais des **listes** dont nous pouvons modifier la taille sans aucun souci avec par exemple la méthode `append()`.

- le dictionnaire :

```
>>> hedy_dico = {"prénom" : "Hedy",
    "nom" : "Lamarr",
    "jour" : 9,
    "mois" : 11,
    "année" : 1914}
```

Les valeurs stockées sont des couples clef-valeur, séparés par 2 points. Les clefs et les valeurs pouvant être de n'importe quel type.
Un dictionnaire est entre accolades, les couples clef-valeur étant séparés par une virgule.
Les clefs doivent être uniques.
Les valeurs peuvent être modifiables, la taille du dictionnaire également.

R La structure **tableau** est bien pratique mais elle peut aussi s'avérer insuffisante dans bien des cas. En effet, dans le tableau `hedy_tab = [9 , 11 , 1914]` on peut oublier si on est dans un système français jour/mois/année ou dans un système anglo-saxons month/day/year. La structure **dictionnaire**, bien qu'un peu plus lourde, permet de lever toute ambiguïté.

1.1.2 Longueur de la structure

On peut obtenir la longueur d'une structure tuple, tableau ou dictionnaire avec la fonction `len()`.

Pour un tuple,
on obtient le nombre n **d'éléments** présents dans le tuple.

```
>>> len(hedy_tuple)
5
```

Pour un tableau,
on obtient le nombre n **d'éléments** présents dans le tableau.

```
>>> len(hedy_tab)
3
```

Pour un dictionnaire,
on obtient le nombre n de couples **clef-valeur** présents dans le dictionnaire.

```
>>> len(hedy_dico)
5
```

1.1.3 Accéder, modifier, rajouter, supprimer une valeur

1.1.3.1 Accéder à une valeur

Avec un tuple,
il suffit de donner l'**index** de la case contenant la **valeur** attendue.
Attention, on commence toujours en comptant à partir de 0 jusqu'à $n-1$, n étant le nombre d'éléments du tuple.

```
>>> hedy_tuple[4]
1914
```

Avec un tableau,
comme pour le tuple, il suffit de donner l'**index** de la case contenant la **valeur** attendue.

```
>>> hedy_tab[2]
1914
```

Avec un dictionnaire,
il suffit de donner la **clef** associée la **valeur** attendue.

```
>>> hedy_dico["année"]
1914
```

1.1.3.2 Modifier une valeur

Avec un tuple,
Impossible ! ! !

Avec un tableau,
il suffit de donner l'**index** de la case contenant la **valeur** à modifier et la nouvelle valeur.

```
>>> hedy_tab[2] = 2023
```

Avec un dictionnaire,
il suffit de donner la **clef** associée à la la **valeur** à modifier et la nouvelle valeur.

```
>>> hedy_dico["année"] = 2023
```

1.1.3.3 Rajouter une valeur

Avec un tuple,
Impossible ! ! !

Avec un tableau,
Impossible ! ! !

Avec un dictionnaire,
il suffit de donner un couple **clef-valeur** comme lorsque l'on veut modifier une valeur.

```
>>> hedy_dico["star"] = True
```

1.1.3.4 Supprimer une valeur

Avec un tuple,
Impossible ! ! !

Avec un tableau,
Impossible ! ! !

Avec un dictionnaire,
il suffit d'utiliser la fonction `del()` [4] et de préciser la **clef** du couple que l'on souhaite supprimer.

```
>>> del(hedy_dico["star"])
```

4. `del` pour delete.

1.1.4 Accéder à toutes les informations

On utilisera une boucle **POUR**.

Pour un tuple,
l'itérateur peut-être

- l'**index** de la case

```
for i in range(len(hedy_tuple)):
    print(hedy_tuple[i])
```

- la **valeur** de la case

```
for info in range(len(hedy_tuple)):
    print(info)
```

Pour un tableau,
l'itérateur peut-être

- l'**index** de la case

```
for i in range(len(hedy_tab)):
    print(hedy_tab[i])
```

- la **valeur** de la case

```
for info in range(len(hedy_tab)):
    print(info)
```

Pour un dictionnaire,
L'itérateur peut être

- la **clef** en utilisant la méthode `keys()`.

```
for clef in hedy_dico.keys() :
    print(clef)
```

- la **valeur** en utilisant la méthode `values()`.

```
for val in hedy_dico.values() :
    print(val)
```

- un tuple **clef**, **valeur** en utilisant la méthode `items()`.

```
for clef, val in hedy_dico.items() :
    print(clef, ":", val)
```

1.1.5 Tester l'appartenance

On utilisera le mot clef `in` et on obtiendra un résultat sous la forme d'un booléen.

Pour un tuple,

```
>>> "Lamarr" in hedy_tuple
True
```

Pour un tableau,

```
>>> 1984 in hedy_tab
False
```

Pour un dictionnaire,
On peut tester l'appartenance

- d'une **clef** en utilisant la méthode `keys()`.

```
>>> "année" in hedy_dico.keys()
True
```

- d'une **valeur** en utilisant la méthode `values()`.

```
>>> "lamare" in hedy_dico.values()
    :
False
```

- un tuple **clef**, **valeur** en utilisant la méthode `items()`.

```
>>> ("nom","Lamarr") in hedy_dico.
    items() :
True
```

1.1.6 Construction par compréhension

Dans les exemples nous allons obtenir tous les carrés parfaits entre 0^2 et 100^2 non multiple de 2 ou de 7.

Pour un tuple,
impossible ! ! !

Pour un tableau,

```
car_tab = [i**2 for i in range
    (101)
        if (i%2 !=0) and (i%7 !=
    0)]
```

Pour un dictionnaire,
on rajoutera le nombre `i` comme **clef** associée à la **valeur** `i**2`

```
car_dico = {
        i:i**2 for i in range(101)
        if (i%2 !=0) and (i%7 !=
    0)}
```

1.2 Les algorithmes de base vue en première

Ces algorithmes sont à connaître par cœur. En effet, ils sont à la base de tous les raisonnements que l'on peut avoir en informatique. Et si vous n'êtes pas encore convaincu sachez que très souvent le premier exercice de l'épreuve pratique [5] est l'écriture d'un programme fortement inspiré d'un de ces algorithmes.

1.2.1 Somme des éléments d'un tableau

```
def somme(tab) :
    """
    Entrée :
    tab est le tableau de valeurs sommables
    Sortie :
    reponse est la somme de toutes les valeurs du tableau
    """
    reponse  = 0
    for i in range(1,len(tab)) :
        reponse = reponse + tab[i]
    return reponse
```

La complexité de cet algorithme est **linéaire**, autrement dit, en utilisant la notation de Landau [6], la complexité est en $O(n)$.

1.2.2 Recherche du minimum d'un tableau

```
def mini(tab) :
    """
    Entrée :
    tab est le tableau de valeurs comparables
    Sortie :
    reponse est la valeur minimale
    """
    reponse = tab[0]
    for i in range(1,len(tab)) :
        if tab[i] < reponse :
            reponse = tab[i]
    return reponse
```

La complexité de cet algorithme est également **linéaire**, que l'on note $O(n)$.

5. Soit 4 points de la note finale pour le Bac !
On peut avoir accès à la banque des sujets sur le lien officiel suivant :
https://eduscol.education.fr/2661/banque-des-epreuves-pratiques-de-specialite-nsi.

6. Edmund Georg Hermann Landau (Berlin, 14 février 1877 - Berlin, 19 février 1938) est un mathématicien allemand, auteur de 253 publications mathématiques, en grande partie sur la théorie des nombres.

1.2.3 Recherche d'une valeur dans un tableau

```
def recherche(tab, cible) :
    """
    Entrées :
    tab est un tableau de valeurs comparables contenant peut-être la cible
    cible est la valeur recherché dans le tableau
    Sortie :
    position est la position de la cible recherchée dans le tableau
    position prend la valeur -1 si la cible est absente du tableau
    """
    position = -1
    i = 0
    while i<len(tab) and position == -1 :
        if tab[i] == cible :
            position = i
        i = i + 1
    return position
```

La complexité de cet algorithme est à nouveau **linéaire**, que l'on note $O(n)$.

1.2.4 Exercices

Les 2 premiers programmes que vous allez écrire dans ses exercices sont des variantes des algorithmes précédents. Il faut aussi les connaître.

exercice 1.1 — Maximum.

1. Écrire en Python une fonction `maxi` qui prend en argument un tableau de valeurs comparables et qui renvoie la plus grande valeur et sa position.
2. Reprendre votre programme sachant que le tableau ne contient que des notes entre 0 et 20.

■

exercice 1.2 — Occurrence. Écrire en Python une fonction `nb_occurence` qui prend en argument un tableau et une valeur et qui renvoie le nombre d'occurrence de cette valeur dans le tableau. ■

exercice 1.3 — Paire Impaire. Écrire en Python une fonction qui prend en argument un tuple composée uniquement d'entiers et qui renvoie deux listes : la première contient tous les nombres paris et la seconde tous les nombres impairs du tuple.

1. Dans un premier temps, on pourra utiliser deux listes vides auxquelles on rajoutera les nombres pairs ou impairs à l'aide de la méthode `append()`.
2. Dans un deuxième temps, on utilisera uniquement la méthode de la compréhension.

■

exercice 1.4 — Statistique. On dispose d'un tableau contenant des notes au Bac Blanc de tous les élèves de NSI du lycée *John Conway*. Écrire en Python une fonction qui prend en argument ce type de tableau et qui renvoie un dictionnaire contenant la note minimale, la moyenne et la note maximale. ■

exercice 1.5 — Fonction `mystere`. Soit la fonction `mystere` suivante, prenant en argument deux tableaux d'entiers :

```
def mystere(tab1, tab2):
    liste = []
    i, j = 0, 0
    while i < len(tab1) and j < len(tab2) :
        if tab1[i] < tab2[j] :
            liste.append(tab1[i])
            i = i + 1
        else :
            liste.append(tab2[j])
            j = j + 1
    return liste
```

1. Quel est le but de cette fonction `mystere` ?
2. Quelle la spécification pour que cette fonction puisse bien jouer son rôle ?

■

exercice 1.6 — Le grand retour de l'ISS.
L'API http://api.open-notify.org/iss-now.json permet d'obtenir la position actuelle de l'ISS sous un format JSON[a]. Ainsi le programme suivant récupère ces données sous la forme d'un dictionnaire :

```
import requests
url = 'http://api.open-notify.org/iss-now.json'
# ligne 6 pour contourner le proxy du lycée
# à supprimer à la maison ! (modifier aussi la ligne 7)
proxy_lycee = {'http': 'http://172.30.137.29:3128'}
lecture = requests.get(url, proxies=proxy_lycee)
ouEstISS = lecture.json()
```

1. Regarder à la structure de l'objet `ouEstISS` puis écrire une fonction qui prend en argument l'adresse url de l'API et qui renvoie, sous la forme d'un tuple les coordonnées terrestres de l'ISS.
2. L'API http://api.open-notify.org/astros.json permet d'obtenir le nom de toutes les personnes actuellement dans l'espace. Écrire une fonction qui prend en argument l'adresse url de l'API et qui renvoie le nombre de personnes dans l'ISS.

■

a. JavaScript Object Notation, JSON, est un format de données textuelles dérivé de la notation des objets du langage JavaScript. Un document JSON comprend deux types d'éléments structurels : des couples clé-valeur, des listes ordonnées de valeurs.

exercice 1.7 — Méta-données `EXIF`. J'ai pris une photo de mon chat *Chouchou*, mais je ne me rappelle plus où exactement. Le programme suivant permet de récupérer les méta-données EXIF.

```
from PIL import Image                # pour travailler sur des images numériques
photo = Image.open("chouchou0.jpg")      # pour charger la photo
info = photo._getexif()                  # récupération des métadonnées EXIF
```

Écrire une fonction qui prend en argument le nom d'une photo avec son extension sous la forme d'une chaîne de caractère et qui renvoie les coordonnées terrestres de la prise de vue sous la forme d'un tuple. La latitude et la longitude seront données sous forme décimale. ■

1.2.5 Trier les valeurs d'un tableau par la méthode du tri par insertion

1.2.5.1 Le principe

C'est un des tris les plus simples.
On parcourt le tableau de la gauche vers la droite et en maintenant une partie déjà triée sur la gauche :

déjà trié									pas encore trié			
$T[0]$	$T[1]$	$T[2]$	...	$T[j-1]$	$T[j]$	...	$T[i-2]$	$T[i-1]$	$T[i]$	$T[i+1]$	...	$T[n-1]$

On va alors prendre la valeur non encore triée la plus à gauche, $T[i]$. On va la comparer à la première valeur triée juste en-dessous, $T[i-1]$.

déjà trié									pas encore trié			
$T[0]$	$T[1]$	$T[2]$	...	$T[j-1]$	$T[j]$	...	$T[i-2]$	$T[i-1]$	$T[i]$	$T[i+1]$	...	$T[n-1]$

Deux solutions :

- Soit $T[i-1] \leq T[i]$, alors $T[i]$ est à sa bonne place et on ne fait rien :

déjà trié										pas encore trié		
$T[0]$	$T[1]$	$T[2]$	...	$T[j-1]$	$T[j]$	...	$T[i-2]$	$T[i-1]$	$T[i]$	$T[i+1]$	...	$T[n-1]$

- Soit $T[i-1] > T[i]$, alors, on redescend la valeur de $T[i]$ d'une case :

déjà trié										pas encore trié		
$T[0]$	$T[1]$	$T[2]$	...	$T[j-1]$	$T[j]$	...	$T[i-2]$	$T[i]$	$T[i-1]$	$T[i+1]$	...	$T[n-1]$

puis on recommence en comparant les valeurs $T[i-2]$ et $T[i]$:

déjà trié										pas encore trié		
$T[0]$	$T[1]$	$T[2]$	...	$T[j-1]$	$T[j]$	...	$T[i-2]$	$T[i]$	$T[i-1]$	$T[i+1]$	...	$T[n-1]$

...

en poursuivant ainsi, de proche en proche, on trouvera une valeur $T[j-1] \leq T[i]$ et $T[i]$ obtiendra sa place dans la partie du tableau triée.

déjà trié										pas encore trié		
$T[0]$	$T[1]$	$T[2]$	...	$T[j-1]$	$T[i]$	$T[j]$	...	$T[i-2]$	$T[i-1]$	$T[i+1]$	...	$T[n-1]$

1.2.5.2 L'algorithme

```
for i in range(1,n) :
    # on prend un nouvel indice
    j = i
    while (j > 0) and (T[j-1] > T[j]) :
        # tant qu'à gauche, il y a un élément plus grand
        # on les échange
        T[j], T[j-1] = T[j-1], T[j]
        j = j - 1
```

C'est un algorithme de **complexité est quadratique** et on la note $O(n^2)$.

1.2.6 Trier les valeurs d'un tableau par la méthode du tri par sélection

1.2.6.1 Le principe

C'est a aussi un des tris les plus simples.
Le tri par sélection consiste à sélectionner la valeur que l'on souhaite ranger au lieu de prendre la première venue. On maintient toujours une partie déjà triée sur la gauche et, parmi les valeurs non encore triées du tableau, on va sélectionner celle qui est la plus petite. On placera alors cette valeur comme la plus grande des valeurs triées :

déjà trié					pas encore trié							
$T[0]$	$T[1]$	$T[2]$	...	$T[i-1]$	$T[i]$	$T[i+1]$	...	$T[j-1]$	$T[j]$	$T[j+1]$	...	$T[n-1]$

On regarde toutes les valeurs non encore triées et on cherche la plus petite.

déjà trié					pas encore trié							
$T[0]$	$T[1]$	$T[2]$	...	$T[i-1]$	$T[i]$	$T[i+1]$	...	$T[j-1]$	$T[j]$	$T[j+1]$	...	$T[n-1]$

Deux solutions :

- Soit $T[i]$ est la valeur la plus petite des valeurs non triées, alors $T[i]$ est à sa bonne place et on ne fait rien :

déjà trié						pas encore trié						
$T[0]$	$T[1]$	$T[2]$	...	$T[i-1]$	$T[i]$	$T[i+1]$	...	$T[j-1]$	$T[j]$	$T[j+1]$	...	$T[n-1]$

- Soit $T[j]$ est la valeur la plus petite :

déjà trié					pas encore trié							
$T[0]$	$T[1]$	$T[2]$	...	$T[i-1]$	$T[i]$	$T[i+1]$	...	$T[j-1]$	$T[j]$	$T[j+1]$	...	$T[n-1]$

On échange $T[j]$ avec la première valeur non triée $T[i]$:

déjà trié					pas encore trié							
$T[0]$	$T[1]$	$T[2]$	...	$T[i-1]$	$T[j]$	$T[i+1]$	...	$T[j-1]$	$T[i]$	$T[j+1]$	...	$T[n-1]$

La valeur $T[j]$ est alors à sa place :

déjà trié						pas encore trié						
$T[0]$	$T[1]$	$T[2]$	...	$T[i-1]$	$T[j]$	$T[i+1]$	...	$T[j-1]$	$T[i]$	$T[j+1]$	...	$T[n-1]$

On poursuit ensuite avec les valeurs non encore triées.

1.2.6.2 L'algorithme

L'algorithme doit faire 2 choses. Chercher le minimum parmi les valeurs non triées puis l'échanger avec la première valeur non encore triée.

```
for i in range(0,n) :
    indice_min = i
    # on cherche le minimum à partir de T[i]
    for j in range(i+1,n) :
        if T[indice_min] > T[j] :
            indice_min = j
            # on échange
                T[indice_min], T[i] = T[i], T[indice_min]
```

On dit à nouveau que la **complexité est quadratique**, notée $O(n^2)$.

R Actuellement les meilleurs algorithmes de tri sont de complexité $O(nlog(n))$.

1.2.7 Exercices

exercice 1.8 On utilise un algorithme de tri sur la liste `[8, 13, 12, 7, 15, 5]`. Après quelques étapes, cet algorithme est interrompu. La liste alors la valeur `[7, 8, 12, 13, 15, 5]`.
Peut-on dire quel est l'algorithme de tri qui a été utilisé ? Si oui, lequel.
Exercice inspiré par « Numérique et Sciences Informatiques - Première » par Serge Bays, editeur Ellipses. ■

exercice 1.9 Ranger à la main, par ordre croissant le tableau suivant à l'aide de l'algorithme de tri par insertion :

66	64	10	99	55	7	63	98	41	1	59	89	96	47	12	0	46	5

■

exercice 1.10 Faire de même à l'aide de l'algorithme de tri par sélection. ■

exercice 1.11 — Le tri à bulles. L'algorithme du tri à bulles consiste à parcourir une liste plusieurs fois jusqu'à ce qu'elle soit triée, en comparant à chaque parcours les éléments consécutifs et en procédant à leur échange s'ils sont mal triés.

1. Programmer une fonction qui trie un tableau de valeurs comparables par la méthode du tri à bulles.
2. Quel est le pire cas ? En déduire la complexité de cette méthode.

■

exercice 1.12 — les voisins. On dispose d'une liste `points` contenant des couples abscisse/ordonnée sous la forme de tuple. Programmer une fonction `voisins` prenant en argument l'entier `k`, la liste de coordonnées `points` et le tuple `ici` contenant des coordonnées. Cette fonction doit renvoyer, dans l'ordre croissant, les k points les plus proches du point `ici`.

1. Préciser les spécifications de la fonction `voisins`.
2. Programmer cette fonction.

■

1.2.8 Algorithme de recherche dichotomique

1.2.8.1 Le principe

Pour rechercher une valeur dans un tableau par l'algorithme de recherche dichotomique **il faut au préalable avoir un tableau de valeurs classées** dans un ordre croissant ou décroissant. Dans la suite nous choisirons un tableau rangé dans l'ordre croissant. Le principe consiste à couper le tableau en 2 avec la valeur médiane donnée par `(mini + MAXI)//2`.
Si la valeur recherchée est la médiane nous avons terminé, sinon, nous devons prendre une moitié du tableau restant. Cette moitié de tableau est soit les valeurs allant de `T[mini]` à `T[Med]` soit les valeurs allant de `T[Med]` à `T[MAXI]`. Or comme la valeur recherchée n'est pas la valeur médiane, on peut très légèrement améliorer notre algorithme en supprimant la valeur médiane. Autrement dit, on ne garde que les valeurs allant de `T[mini]` à `T[Med-1]` soit les valeurs allant de `T[Med+1]` à `T[MAXI]`.
Dans l'exemple suivant nous recherchons dans le tableau `T = [1, 3, 4, 6, 7, 8, 10, 13, 14]` et la valeur 6.

Indice	mini=0				Med = (0+8)//2 = 4				MAXI = 8
Tableau `T`	1	3	4	6	7	8	10	13	14

Comme le nombre 6 est plus petit que `T[Med]=T[4]=7`, on ne va garder que la partie du tableau entre les indices `mini=0` et `Med-1=3`. On rejette la partie supérieure du tableau (en rouge dans la suite).

Indice	mini=0	Med = (0+3)//2 = 1		MAXI = 3					
Tableau T	1	3	4	6	7	8	10	13	14

Comme le nombre 6 est plus grand que `T[Med]=T[1]=3`, on ne va garder que la partie du tableau entre les indices `Med+1=2` et `MAXI=3`. On rejette la partie inférieure du tableau (en rouge dans la suite).

Indice			mini =2 Med = (2+3)//2 = 2	MAXI = 3					
Tableau T	1	3	4	6	7	8	10	13	14

Comme le nombre 6 est plus grand que `T[Med]=T[2]=4`, on ne va garder que la partie du tableau entre les indices `Med+1=3` et `MAXI=3`. On rejette la partie inférieure du tableau (en rouge dans la suite).

Indice				mini =3 Med = (3+3)//2 = 3 MAXI = 3					
Tableau T	1	3	4	6	7	8	10	13	14

Comme le nombre 6 est exactement le nombre `T[Med]=T[3]=6`, nous avons terminé notre travail. **Nous pouvons donc conclure que le nombre 6 est bien présent dans le tableau et qu'il se situe à la case indexé 3.**

1.2.8.2 L'algorithme

L'algorithme va couper en 2 le tableau tant que l'indice `mini` sera inférieur à l'indice MAXI. Ensuite il vérifie si la cible est ou non atteinte.

```
mini = 0
med = 0
maxi = n-1
while mini < maxi:
        med = (mini + maxi)//2
        if T[med] < cible :
                # cible est dans la partie supérieure du tableau
                mini = med +1
        elif T[med] > cible :
                # cible est dans la partie inférieure du tableau
                maxi = med -1
        else :
                maxi = med
                mini = med
if T[mini] == cible :
    indice = mini
else:
    indice = -1
```

La complexité est **logarithmique**, on le note $O(log(n))$.
Pour un entier n strictement positif est, $\log_2(n)$ est le nombre des bits de son écriture binaire diminué d'une unité. Par exemple, pour le nombre 666 , il s'écrit en binaire 1010011010_2 donc $\log_2(666) = 10-1 = 9$.

Exercice

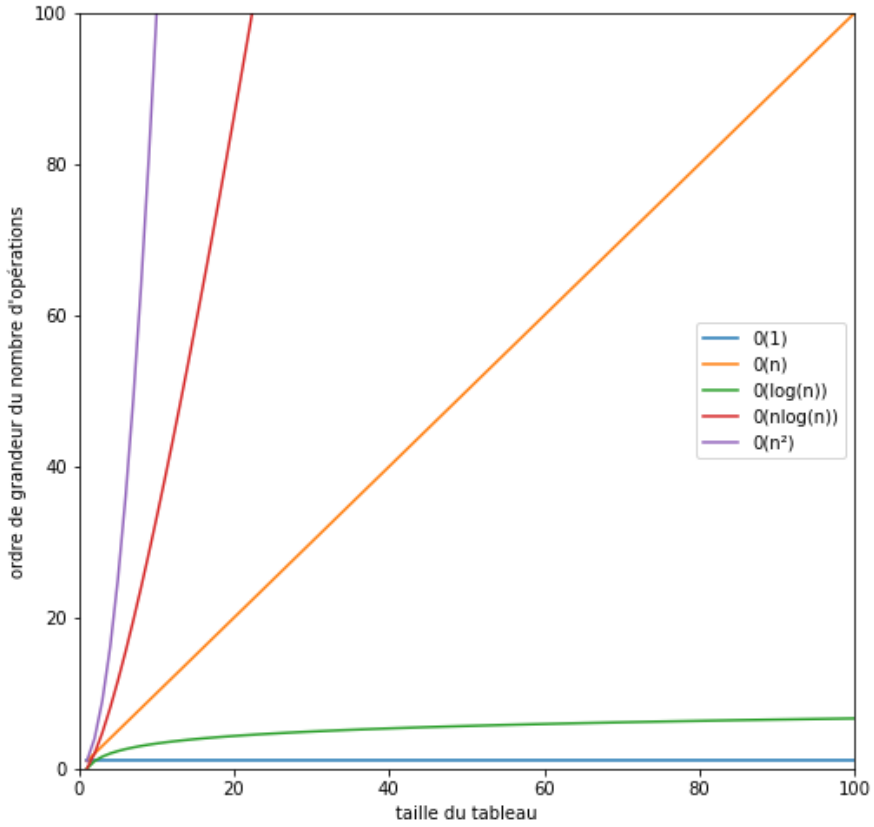
exercice 1.13 Faire tourner « à la main » cet algorithme pour `T=[1, 5, 24, 28, 29, 31, 33, 34, 52]` pour trouver la cible 34.
Refaire tourner avec la cible 35. ■

1.3 Retour sur les complexités

Voici les différentes complexités que vous avez vu l'an dernier :

- la complexité en temps constant, noté $O(1)$.
  Quelle que soit la taille du tableau, la réponse se fait avec un même nombre constant d'opérations.
- la complexité linéaire, noté $O(log(n))$.
  Si le tableau est de taille $n = 10$, comme $10 = 1010_2$, il faut de l'ordre de $4-1 = 3$ opérations pour effectuer l'instruction.
  Si le tableau est de taille $n = 100$, comme $10 = 1100100_2$, il faut de l'ordre de $7-1 = 6$ opérations pour effectuer l'instruction.
  Si le tableau est de taille $n = 1\,000$, comme $1\,000 = 1111101000_2$, il faut de l'ordre de $10-1 = 9$ opérations pour effectuer l'instruction.
  ...
- la complexité linéaire, noté $O(n)$.
  Si le tableau est de taille $n = 10$, il faut des dizaines d'opérations pour effectuer l'instruction.
  Si le tableau est de taille $n = 100$, il faut des centaines d'opérations pour effectuer l'instruction.
  Si le tableau est de taille $n = 1\,000$, il faut des milliers d'opérations pour effectuer l'instruction.
  ...
- la complexité linéaire, noté $O(nlog(n))$.
  Si le tableau est de taille $n = 10$, comme $log(10) = 3$, $nlog(n) = 30$, il faut des dizaines d'opérations (voire quelques centaines) pour effectuer l'instruction.
  Si le tableau est de taille $n = 100$, comme $log(100) = 6$, $nlog(n) = 600$, il faut des centaines d'opérations (voire quelques milliers) pour effectuer l'instruction.
  Si le tableau est de taille $n = 1\,000$, comme $log(1\,000) = 9$, $nlog(n) = 9\,000$, il faut des dix-milliers d'opérations pour effectuer l'instruction.
  ...
- la complexité quadratique, noté $O(n^2)$.
  Si le tableau est de taille $n = 10$, comme $10^2 = 100$, il faut des centaines d'opérations pour effectuer l'instruction.
  Si le tableau est de taille $n = 100$, comme $100^2 = 10\,000$, il faut des dix-milliers d'opérations pour effectuer l'instruction.
  Si le tableau est de taille $n = 1\,000$, comme $1\,000^2 = 1\,000\,000$, il faut des millions d'opérations pour effectuer l'instruction.
  ...

On peut visualiser ces différentes complexités dans le graphique ci-dessous.



Exercice

exercice 1.14 Le lien suivant https://wiki.python.org/moin/TimeComplexity permet d'obtenir la complexité des opérations sur les structures de base en Python.

1. Quelle est la complexité pour effectuer la copie d'une liste et pour rechercher un minimum ? Obtient-on des résultats correspondants à ce qui est attendu ?
2. Quelle est la complexité pour insérer un élément ? Comment expliquer ce résultat ?
3. Quelle est la complexité pour obtenir la longueur d'une liste ? Comment expliquer ce résultat ?