4 – Recherche dichotomique

I – Trier un tableau avec Python

On peut **créer un tableau** :

→ par **extension** :
```
tab = [0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
```

→ par **compréhension** :
```
tab = [ 5xv for v in range(10)]
```

→ par extension successive :
```
tab = []
for v in range(10):
    tab.append(5xv)
```

Cours : Qu'est-ce qu'une précondition ?

Cours : Quel est le coût d'une recherche linéaire ? Pouquoi ?

On peut **trier un tableau** :

→ en utilisant des fonctions personnelles (voir les activités Tris)

→ en utilisant la fonction native **`sorted`** qui renvoie une **copie triée de tab_initial**

```
>>> tab_initial = [13, 55, 54, 57, 12, 80, 16, 72, 30, 99]
>>> tab_trie = sorted(tab_initial)
>>> tab_trie
[12, 13, 16, 30, 54, 55, 57, 72, 80, 99]
>>> tab_initial
[13, 55, 54, 57, 12, 80, 16, 72, 30, 99]
```

→ en utilisant la méthode (des tableaux) native **`sort`** qui modifie et tri sur **place tab**_initial par effet de bord.

```
>>> tab_initial = [13, 55, 54, 57, 12, 80, 16, 72, 30, 99]
>>> tab_initial.sort()
>>> tab_initial
[12, 13, 16, 30, 54, 55, 57, 72, 80, 99]
```

Après q3 : Cours : Comment connaître la valeur UNICODE du caractère 'Z' ?

Après q3 :Cours : Comment connaître le caractère dont la valeur unique est 9822 ?

Après q3 : Cours : Que veut dire trier des strings entre eux ?

II – Principe de la recherche dichotomique

Rappel sur le lien entre logarithme 2 d'un entier et nombre de bits de cet entier

Que vaut log2(2) ?

Que vaut log2(4) ?

Que vaut log2(8) ?

Que vaut log2(16) ?

Quelle est la formule pour trouver le nombre de bits b d'un entier n ?

b =

Cours : Classer les coûts algorithmiques du plus performant au moins performant :
linéaire – constant – exponentiel – quadratique – logarithmique

Cours - Nombre d'étapes maximales avec la recherche dichotomique

→ Combien faut-il d'étapes dans le pire des cas pour trouver un nombre dans un tableau de n éléments cette fois ?

→ Quel est le coût de la recherche dichotomique ? Pourquoi ?

Cours – Explication globale du principe de la recherche dichotomique

→ On place au départ l'indice gauche **g à 0** et l'indice droite **d à 0**

→ On calcule l'indice central avec **`c = (g+d) // 2`**

→ **TANT QUE il reste des cases à explorer**

→ si **`t[c] == x`** → on renvoie c

→ si **`t[c] > x`** → on peut supprimer de la recherche tout ce qui est à droite : **`d =`**

→ si **`t[c] < x`** → on peut supprimer de la recherche tout ce qui est à gauche : **`g =`**

III – Preuve de terminaison

Cours - ALGORITHME RECHERCHE DICHOTOMIQUE

```
g ← 0
d ← longueur - 1
TANT QUE g <= d
      c ← (g + d) // 2
      SI tableau[c] < x
            g ← (c + 1)
      SINON SI tableau[c] > x
            d ← (c - 1)
      SINON
            Renvoyer c
      Fin du Si
Fin Tant que
Renvoyer VIDE
```

Cours - PREUVE DE TERMINAISON

Pour prouver la terminaison, il faut

06-feuille° Montrer que `TANT QUE g <= d` revient à écrire `TANT QUE longueur > 0`

Combien perd-t-on d'éléments si on a (k-1) +**1** +k élements ? (le **1** représente le pilier central)

Combien perd-t-on d'éléments si on a k +**1** + k élements ? (le **1** représente le pilier central)

07-feuille° En reprenant l'essentiel de la démonstration ci-dessus liée au variant **longueur$_N$** et le fait qu'on puisse écrire la condition de la boucle non bornée sous la forme **TANT QUE longueur$_N$ > 0,** que pouvez-vous en conclure sur la terminaison de cet algorithme.

A faire et à savoir refaire : réaliser seul la fonction implémentant la recherche dichotomique :

```
def recherche_dicho(tableau, x):
      '''Fonction qui cherche l'élément x dans le tableau trié de façon dichotomique'''
```