Algo 7 – Recherche dichotomique

I – Trier un tableau avec Python

1.1 Création de tableaux en langage Python

On peut **créer le tableau `t`** ci-dessous de deux façons :

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

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

1.2 Trier un tableau avec les solutions natives de Python

On peut **trier un tableau** :

→ en utilisant des fonctions personnelles : (voir les activités Tris de la partie ALGORITHMIQUE)

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

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

>>> t
>>> [13, 55, 54, 57, 12, 80, 99, 16, 72, 30]
```

→ en utilisant la méthode (des tableaux) native **`sort`** qui tri directement sur **place `t`** par effet de bord.

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

Attention : surtout pas _____

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

1.3 Trier les strings

On peut **trier un tableau** de strings avec l'ordre lexicographique (ordre alphabétique comme dans les dictionnaires).

Les 127 premières valeurs sont issues du code _____ : par exemple A ↔ 65, B ↔ 66, mais a ↔ 97

L'ensemble des valeurs, de 1 à plus de 4 milliards, forme le code _____.

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

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

Cours : Que veut dire trier des strings entre eux ? _____

II – Principe de la recherche dichotomique : un TANT QUE et une comparaison

III – Algorithme et coût de la recherche dichotomique

But : fournir l'indice de la première occurrence de x dans un tableau t. La valeur -1 indique l'absence de x dans t.

Description des entrées / sortie

- **ENTREES** :
  - **x** : un élément voulu qu'on tente de rechercher dans le tableau
  - **t** : un tableau trié contenant un ensemble d'éléments (PRECONDITON : trié en ordre croissant).
  - (entrée optionnelle selon le langage d'implémentation) **longueur** : le nombre d'éléments dans t.
- **SORTIE** : un entier correspondant à un indice i du tableau tel que t[i] contient bien x, ou la réponse vide (-1).

Algorithme

Dans l'algorithme ci-dessous, l'opérateur // désigne une division euclidienne.

```
r ← -1
g ← 0
d ← longueur - 1
TANT QUE (g <= d) ET (r vaut -1)
    c ← (g + d) // 2
    SI t[c] < x
        g ← (c + 1)        on modifie g pour "supprimer" la partie gauche
    SINON SI t[c] > x
        d ← (c – 1)        on modifie d pour "supprimer" la partie droite
    SINON
        r ← c              x est l'élément de t[c] : on sort du TANT QUE en modifiant r
    Fin du Si
Fin Tant que
Renvoyer r
```

La condition de poursuite nécessite deux conditions valides, il existe donc deux façons d'arrêter le TANT QUE :

Lien entre logarithme 2 d'un entier et nombre de bits de cet entier

Si on peut écrire un nombre n sous la forme $n = 2^x$, on peut alors écrire

Que vaut log2(1) ? _____

Que vaut log2(2) ? _____

Que vaut log2(4) ? _____

Que vaut log2(8) ? _____

Que vaut log2(16) ? _____

Que vaut log2(32) ? _____

Pour les nombres non entiers, on obtient ceci puisque la fonction log est croissante :

```
>>> import math
>>> math.log2(17)
4.087462841250339
```

Nombre d'étapes de 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 : Classer les coûts algorithmiques du plus performant au moins performant :
linéaire – constant – exponentiel – quadratique – logarithmique

IV – Implémentation Python

```
def recherche_dicho(t, x):
    '''Fonction qui cherche l'élément x dans le tableau trié de façon dichotomique

    :: param t(list)      :: un tableau trié d'éléments
    :: param x (variable) :: l'élément à chercher dans le tableau
    :: return (int)       :: l'indice de l'élément ou -1
    .. Précondition 1 :: le tableau ne doit pas être vide
    .. Précondition 2 :: le tableau doit être TRIE de façon croissante
    .. Précondition 3 :: x doit être du même type que les éléments du tableau

    '''
    r = -1
    g = 0
    d = len(t) - 1
    while g <= d and r == -1:
        c = (g + d) // 2
        if t[c] < x:
            g = c + 1
        elif t[c] > x:
            d = c - 1
        else:
            r = c
    return r
```

IV – Preuve de terminaison

La preuve de terminaison d'un algorithme permet d'affirmer qu'il s'arrête de façon certaine sur toutes les entrées valides qu'on lui fournit (*c'est à dire les entrées respectant les préconditions*).

Pour faire la preuve de terminaison, il faut montrer que les deux propositions suivantes sont vraies :

- Proposition A : les boucles s'expriment sous la forme TANT QUE VARIANT > 0
- Proposition B : le VARIANT est une suite u_n strictement décroissante d'entiers.

(A et B) implique alors que l'algorithme se termine.

Exo de cours ° Montrer que `TANT QUE g <= d` revient à écrire `TANT QUE VARIANT > 0`. On pourra même montrer que cela revient en réalité à écrire `TANT QUE longueur_restante > 0`. CONCLURE sur la terminaison de cet algorithme.

CONCLUSION : Vous devez être capable d'implémenter une fonction réalisant cette recherche, à partir de … rien, sauf vos connaissances.