1 En autonomie

Les questions suivantes sont le minimum à savoir après le cours, à réaliser en autonomie.

1. Dresser le tableau de la mémoire lors de l'exécution de `recherche_seq(5, [3, 1, 4, 1, 5, 9])` (on utilisera la comparaison naturelle des entiers)
2. Écrire la trace de `recherche_seq_rec(5, [3, 1, 4, 1, 5, 9])`
3. Dans quel cas (état de la liste) l'algo de recherche séquentielle effectue-t-il le plus de comparaisons ? Le moins ? Pourquoi ?
4. Dans le cas d'une recherche infructueuse, quel est l'algo de recherche vu en cours en général le moins coûteux ? Pourquoi ?
5. Pour quelle fonction de comparaison a-t-on `est_triee([9, 5, 4, 3, 1, 1], comp)` qui vaut `True` ?
6. Et, pour laquelle a-t-on `est_triee([0, 1, 2, 4, 8, 5, 6, 7], comp)` qui vaut `True` ?

2 Recherches séquentielles renvoyant un indice

Nous avons vu en cours des algorithmes de recherche renvoyant une valeur booléenne. Vous allez dans cet exercice réaliser des fonctions donnant l'indice de l'élément trouvé. Le type des fonctions à réaliser est donc

`list`[`A`], `int`, `int`, `A`, `Callable`[[`A`, `A`], `int`] → `int`

où `A` est un type quelconque et `Callable[[A, A], int]` désigne le type de la fonction de comparaison (sa signature est `A`, `A` → `int`)

1. Avant de vous lancer dans ces implantations, précisez la spécification de ces fonctions de recherche :
    1. Si l'élément recherché est présent plusieurs fois dans la liste, quel indice renvoyer ?
    2. Idem si cet élément n'est pas dans la liste ?
2. Réalisez une implantation de la recherche séquentielle dans une liste non triée, puis dans une liste triée, qui donne le plus petit indice d'un élément présent dans la liste.
3. Idem pour le plus grand indice.

3 Point trop n'en faut!

Voici une implantation en Python erronée de la recherche laborieuse vue en cours.

```
def recherche_laborieuse_erronee(li: list[int], a: int,b: int ,x: int) -> bool:
    trouve=False
    for i in range(a,b):
        if li[i]==x:
            trouve=True
        else:
            trouve=False
    return trouve
```

1. Pour chacun des deux appels suivants à cette fonction de recherche, présentez sous forme d'un tableau les valeurs successives que prennent les variables `i` et `trouve` , ainsi que la valeur renvoyée par la fonction.
    1. `recherche_laborieuse_erronee([3, 1, 2, 4, 1], 0, 5, 1)`
    2. `recherche_laborieuse_erronee([3, 1, 2, 4, 5], 0, 5, 1)`
2. Quelle est l'erreur de programmation manifestement commise ? Proposez une réparation de ce code.
3. En fonction de n la longueur de la liste, combien de comparaisons sont effectuées lors d'une recherche laborieuse erronée ?
4. Proposez un nouveau corps de fonction, qui calcule exactement la même chose, donc produit des résultats erronés, mais bien plus efficacement que `recherche_laborieuse_erronee`

4 Recherche séquentielle triée

Le fait de savoir qu'une liste est triée peut nous permettre d'améliorer la recherche séquentielle.

Considérons par exemple la liste d'entiers `liste=[1, 1, 2, 3, 5, 6, 9]`. Cette liste est triée pour la comparaison usuelle des entiers. Supposons que l'on recherche l'entier `4` dans cette liste. Cette recherche est évidemment infructueuse.

1. Dresser le tableau donnant pour i allant de 0 à 6, les valeurs de `liste[i] <= 4`.
2. Pour une liste et un élément quelconque et , on suppose l'existence d'un indice i tel que `liste[i] > elt`. Que dire de l'expression `elt in liste[i:]` ?
3. En déduire un algorithme de recherche séquentielle effectuant en général moins de comparaisons que l'algorithme de recherche séquentielle présenté en cours (en donner une version récursive et itérative).
4. Pour chacune des versions de votre algorithme, donner un variant et une propriété invariante permettant d'établir qu'il est correct.

5 Pour comprendre la recherche dichotomique

0	2	4	6	8	10	12	14	16	18	20	22

1. Donnez la suite des tranches de liste parcourue lors de la recherche dichotomique de $x = 11$ dans la liste t ci-dessus. Combien de comparaisons d'éléments de la liste sont-elles effectuées ?
2. Même question avec $x = 12$ puis $x = 10$.
3. Dessinez l'arbre des recherches possibles pour une recherche dichotomique dans une liste de longueur 12.

6 Une petite amélioration

L'algorithme de recherche dichotomique tel qu'il a été présenté en cours est décrit par une boucle `tant que` dont la seule condition est $d < f$, d et f désignant les indices de début et de fin de la tranche courante de la recherche. Cependant il se peut que l'élément recherché soit trouvé avant que cette condition ne soit plus réalisée.

1. Proposez une nouvelle version de l'algo de recherche dichotomique qui tient compte de cette remarque.

7 Recherche dichotomique renvoyant un indice

1. Réalisez une implantation de la recherche dichotomique qui renvoie un indice et non une valeur booléenne.
2. Dans le cas où l'élément recherché est présent plusieurs fois dans la liste, quel est l'indice est renvoyé par votre fonction ?

8 Recherche dans une liste d'étudiants

On suppose dans cet exercice que les dates sont représentées par des objets de la classe `Date` et que cette classe définit, comme dans le TP, les méthodes permettant les comparaisons de deux dates. Son constructeur prend en paramètre trois entiers `(jour, mois, annee)` et initialise les attributs de même nom.

On dispose également une classe `Etudiant` dont le constructeur initialise les attributs `nip: int, nom: str, prenom: str, naissance: Date` et une variable `etudiants` contenant une liste d'étudiants ci-dessous.

```
etudiants = [Etudiant(99998132, "Calbuth", "Raymond", Date(12, 12, 1987)),
             Etudiant(99994451, "Talon", "Achille", Date(7, 11, 1963)),
             Etudiant(99996348, "Calbuth", "Monique", Date(29, 7, 1987)),
             Etudiant(99995433, "Blanc-Sec", "Adèle", Date(17, 4, 1976)),
             Etudiant(99997674, "Brisefer", "Benoît", Date(15, 12, 1960)),
             Etudiant(99998324, "Lagaffe", "Gaston", Date(28, 2, 1957))]
```

1. En utilisant la fonction `recherche_seq` vue en cours et sans la modifier, expliquer comment savoir si, oui ou non, il existe :

- un étudiant dont le prénom est `Timoléon` ;
- un étudiant dont le nom commence par `Cal` ;
- un étudiant né le 29 janvier 1965 ;
- un étudiant né en 1960.

Pour chaque cas, vous donnerez la fonction de comparaison.

2. Pour chacun des points précédents, dire si la recherche est fructueuse ou non sur l'exemple et combien d'appels à la fonction de comparaison ont eu lieu.