Dans cette série d'exercice, vous allez être amené à travailler sur différents algorithmes de recherche, ainsi qu'étudier leur efficacité.

// Exercice 1 - Recherche séquentielle

Q1. Définissez une fonction `recherche_sequentielle(liste, valeur)` qui recherche si `valeur` est présent dans la liste, si oui, la fonction retourne sa position dans liste,sinon, elle retourne `-1`.

Si l'élément est présent plusieurs fois dans la liste, il faut retourner celui de plus petit indice.

```
def recherche_sequentielle(liste, valeur):
    """
    Effectue une recherche séquentielle d'une valeur dans une liste.
    Paramètres :
    - liste (list): La liste dans laquelle effectuer la recherche.
    - valeur (any): La valeur à rechercher dans la liste.
    Retourne :
    int: l'indice de l'élément dans la liste, ou -1 s'il n'est pas trouvé
    """
```

Exemple

```
>>> recherche_sequentielle([1, 2, 3, 4, 5], 3)
2
```

Q2. Nous cherchons a estimer le nombre de comparaisons **dans le pire des cas** de la recherche séquentielle. Autrement dit **Combien de fois fais t-on une comparaison de valeurs dans le pire des cas en fonction de la taille de la liste ?**

// Exercice 2 - Recherche dichotomique

L'objectif de cet exercice est de vous faire implémenter la recherche dichotomique. À la différence de l'exercice précédent, vous allez désormais faire une recherche **dans une liste triée**.

Qu'est-ce que la recherche dichotomique ?

La recherche dichotomique est une technique efficace pour trouver un élément dans une liste triée. L'idée est de diviser la liste en deux moitiés et de comparer l'élément à trouver avec l'élément au milieu de la liste. Selon le résultat de la comparaison, on sait dans quelle moitié de la liste chercher, et on recommence le processus jusqu'à ce que l'élément soit trouvé ou que la sous-liste soit vide.

Conseils

- Pensez à utiliser deux indices pour suivre les limites de la sous-liste dans laquelle vous cherchez : debut et fin.
- Faites attention aux cas limites (liste vide, élément au début ou à la fin de la liste, etc.)

```
def recherche_dichotomique(liste, element):
    """
    Recherche `element` dans `liste` en utilisant la méthode de recherche dichotomique.
    Parameters:
    - liste (list): une liste triée d'éléments
    - element: l'élément à rechercher
    Returns:
    int: l'indice de l'élément dans la liste, ou -1 s'il n'est pas trouvé
    """
```

Exemple

```
>>> recherche_dichotomique([1, 2, 3, 4, 5], 3)
2
```

Q2. Nous cherchons a estimer le nombre de comparaisons **dans le pire des cas** de la recherche dichotomique. Autrement dit **Combien de fois fais t-on une comparaison de valeurs dans le pire des cas en fonction de la taille de la liste ?**

// Exercice 3 - La cible

Q1. Ecrivez une fonction `cible(liste, c)` qui return `True` s'il est possible d'obtenir la valeur `c` en faisant la somme de deux nombres appartenant à `liste` , `False` sinon.

```
def cible(liste, c):
    """Recherche si une paire d'éléments dans la liste donne une somme égale à 'c'.
    Parameters:
    - liste (list): une liste d'éléments numériques
    - c (int or float): la valeur cible pour la somme
    Returns:
    bool: True si une paire d'éléments de la liste a une somme égale à 'c', False sinon"""
```

Exemple

```
>>> cible([14, 19, 22, 27, 27, 29, 40, 44, 5, 50], 51)
True
```

Commentaire

Nous pouvons obtenir 51 en faisant 22 + 29 = 51 donc on peut atteindre la cible.

Exemple

```
>>> cible([14, 19, 22, 27, 27, 29, 40, 44, 5, 50], 101)
False
```

Commentaire

Il n'est pas possible d'atteindre la cible, car il n'existe aucune combinaison de 2 nombres dans la liste permettant d'atteindre 101. Cela peut se montrer assez facilement : le plus grand nombre de la liste est 50, donc le plus grand nombre cible possible est 100 (on prend deux fois 50), comme 101 est plus grand, il n'existe donc pas de solution.

Q2. Exactement le même problème que précédemment, sauf que vous disposez cette fois d'une information supplémentaire : La liste en paramètre est **triée**.

En tenant compte de cette information, vous devez optimiser votre fonction. La méthode de l'exercice précédent ne sera pas assez rapide pour valider celui-ci.

```
def cible2(liste, c):
    """Recherche si une paire d'éléments dans la liste donne une somme égale à 'c'.
    Parameters:
    - liste (list): une liste d'éléments numériques
    - c (int or float): la valeur cible pour la somme
    Returns:
    bool: True si une paire d'éléments de la liste a une somme égale à 'c', False sinon"""
```

Exemple

```
>>> cible2([5, 14, 19, 22, 27, 27, 29, 40, 44, 50], 51)
True
```

Commentaire Pas de changement dans les règles, 51 peut toujours s'obtenir en fait 22+29=51