TP NSI – Recherche dichotomique et Élément majoritaire

Objectifs

- Comprendre et implémenter une recherche dichotomique dans un tableau trié.
- Savoir déterminer si un tableau contient un élément majoritaire.
- Développer l'autonomie, la rigueur algorithmique et la capacité à comparer des approches.

Ce TP propose deux parcours :

- **Parcours A** : pour progresser à son rythme avec de l'accompagnement.
- **Parcours B** : pour aller plus vite ou plus loin avec plus de liberté.

→ Tu pourras changer de parcours en cours de TP si tu te sens prêt(e) !

Parcours A – Accompagnement progressif

1. Écrire une fonction `recherche_lineaire(tab, valeur)` qui renvoie l'indice de `valeur` dans `tab` (ou `-1` si absent).
2. Quelle est la complexité de cet algorithme ?
3. Écrire une fonction `recherche_dichotomique(tab, valeur)` pour un tableau **trié** :
   - On peut s'aider de ce squelette :

```
def recherche_dichotomique(tab,
valeur):
    debut = 0
    fin = len(tab) - 1
    while debut <= fin:
        milieu = (debut + fin) // 2
        # Complète ici
```

4. Tester avec des cas simples (élément présent / absent).
5. Ajouter un **compteur d'itérations** pour comparer les deux méthodes.

Parcours B – Codage autonome

1. Implémenter directement `recherche_dichotomique(tab, valeur)` en version récursive.
2. Écrire une fonction `comparaison_recherches()` qui :
   - Génère un tableau trié de 1 à 1000.
   - Compare la recherche linéaire et dichotomique pour différents éléments (début, milieu, fin, absent)
3. Afficher le nombre de comparaisons ou d'appels effectués.
4. Que peut-on conclure à partir de ces résultats ?

Exercice passerelle (A → B)

- Si tu as terminé ta version itérative sans aide : passe au parcours B.
- Si tu bloques, demande de l'aide ou change temporairement de groupe pour échanger.

Partie 2 - Recherche d'un élément majoritaire

1. Écrire une fonction `compte_occurrences(tab, valeur)` qui compte combien de fois `valeur` apparaît dans `tab`.
2. En s'aidant de cette fonction, écrire `element_majoritaire(tab)` :
   - Pour chaque élément du tableau, utiliser `compte_occurrences(...)`.
   - Si un élément apparaît strictement plus de `len(tab) // 2`, il est majoritaire.
3. Tester la fonction avec plusieurs exemples.

1. Écrire une fonction `element_majoritaire_dico(tab)` qui utilise un **dictionnaire** pour compter les occurrences.
   - Complexité : `O(n)`
2. Implémenter l'**algorithme de Boyer-Moore** :

- Phase 1 : déterminer un candidat.
- Phase 2 : vérifier que c'est un vrai majoritaire.

1. Comparer les trois versions sur :

- Des tableaux avec et sans majoritaire.
- Des tableaux très grands (`n = 10 000`).

Exercice passerelle (A → B)

Tu as bien compris element_majoritaire() ? **Essaie la version avec dictionnaire !**

Partie 3 - Pour aller plus loin

Ces activités sont à faire si tu as terminé le TP principal.

Défis “niveau expert” :

- Modifier `recherche_dichotomique(tab, valeur)` pour qu’elle fonctionne aussi **sur un tableau trié décroissant**.
- Écrire une version **générique** de la recherche dichotomique qui accepte un paramètre `croissant = True`.
- Implémenter une fonction `element_majoritaire_trie(tab)` optimisée pour un tableau trié (piste : regarde le milieu).

Bonus réflexion :

- On doit chercher un mot dans un dictionnaire (papier). Pourquoi la dichotomie est-elle utile ici ?
- Comment la ferait-on sur un dictionnaire numérique en Python ? (type `dict`)

Pour finir

1. Quelles sont les **forces et limites** de la recherche dichotomique ?
2. Comment peut-on améliorer un algorithme naïf ?
3. Quels sont les liens entre structure de données et performance des algorithmes ?

Tests unitaires proposés

```
assert recherche_dichotomique([1, 3, 5, 7, 9], 5) == 2
assert recherche_dichotomique([1, 3, 5, 7, 9], 4) == -1

assert element_majoritaire([1, 2, 3, 3, 3]) == 3
assert element_majoritaire([1, 2, 3]) == None
```

En cas de doute, travaille en binôme ou demande un indice à ton professeur. N’oublie pas que tu peux toujours changer de parcours si tu veux accélérer ou consolider.