# 1 Problématique

Rechercher un élément dans un tableau est une opération courante. Cette tâche a un coût qui dépend de la taille du tableau. Cependant, si le tableau est déjà trié est-il possible d'accélérer la recherche?

Comment implémenter une recherche efficace dans un tableau trié?

# 2 Recherche classique dans un tableau

#### 2.1 Génération des données

Imaginons un supermarché qui référence chaque article par un entier. Les références, au nombre de cent mille, sont contenues dans un tableau.

**Activité 1 :** Construire par compréhension un tableau de cent mille entiers compris entre 0 et 1000000.

#### 2.2 Recherche dans les données

Pour assigner une nouvelle référence, il faut d'abord vérifier que l'entier n'est pas déjà utilisé. Il faut parcourir le tableau élément par élément.



FIGURE 1 – Parcours séquentiel

# À retenir

Dans le pire des cas la complexité temporelle de la recherche dépend du nombre d'éléments. La complexité temporelle est **linéaire**.

### Activité 2:

- 1. Écrire la fonction recherche\_classique(tab: list, cherche: int)  $\rightarrow$  bool qui renvoie True si l'entier cherche est présent dans le tableau.
- 2. Tester la fonction : vérifier si le nombre 575000 est présent dans le tableau.
- 3. Dans le programme principal, créer une variable **COMPTEUR** initialisée à 0. Cette variable de test sera utilisée dans la fonction pour compter le nombre d'itérations de la boucle de recherche. On parle alors de **variable globale** car elle n'est pas propre à la fonction. Il faudra ajouter le code 1 au début de la fonction.

global COMPTEUR

Code 1 – Déclaration d'une variable globale



## 3 Recherche dans un tableau trié

#### 3.1 Des données ordonnées

Considérons maintenant que les références sont triées par ordre croissant au fur et à mesure de leur ajout dans le tableau de données.

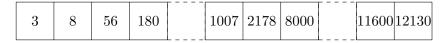


FIGURE 2 – Références triées

Activité 3 : Pour simplifier nous allons utiliser la méthode sort pour trier les données.

- 1. Construire par compréhension un tableau de cent mille entiers compris entre 0 et 1000000.
- 2. Trier le tableau.

### 3.2 Recherche dichotomique

Les données étant triées, le principe de la dichotomie, pour chercher la présence d'un élément, consiste à :

- couper le tableau en deux parties égales,
- ne garder que la partie contenant l'élément,
- répéter l'opération jusqu'à trouver l'élément ou avoir une partie vide.

Cherchons 302 dans le tableau suivant :

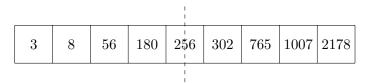


Figure 3 – Séparons les données en deux parties



FIGURE 4 – 256 n'est pas le nombre recherché et il est inférieur à 302

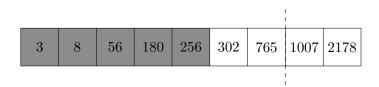


FIGURE 5 – Séparons les données restantes en deux parties



FIGURE 6 – Nous pouvons éliminer la partie supérieure à 302



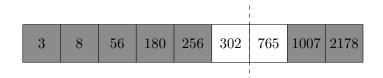


FIGURE 7 – Dernière séparation



FIGURE 8 - 302 a été trouvée en trois itérations

Activité 4 : Écrire la fonction recherche\_dicho(tab: list, cherche: int)  $\rightarrow$  bool qui applique le principe de la dichotomie. Pour séparer les données en deux parties (à peu près) égales il faudra calculer l'indice médian de la partie encore valide.

#### 3.3 Efficacité

#### Activité 5:

- 1. En utilisant une variable COMPTEUR, compter le nombre d'itérations de la boucle de recherche dichotomique.
- 2. Tester pour différentes tailles de tableau.

À chaque itération la quantité de données (notée  $\mathbf{n}$ ) à étudier est divisée par deux. Dans le pire des cas, on divise jusqu'à ce que la taille de la partie restante soit inférieure ou égale à 1.

$$\frac{n}{2^x} = 1$$

$$\Leftrightarrow n = 2^x$$

## Activité 6:

- 1. Encadrer la valeur de x entre deux entiers, si le tableau contient n=10000 éléments.
- 2. Effectuer le même encadrement pour cent mille, un million d'éléments.

# À retenir

La complexité temporelle de la recherche dichotomique est logarithmique :

$$\log_2 n = x$$

