Recherche d'un élément par dichotomie

Contenus	Capacités attendues
Montrer la terminaison de la recherche dichotomique à l'aide d'un variant de boucle.	Des assertions peuvent être utilisées La preuve de la correction peut être présentée par le professeur.

La recherche dichotomique permet parfois de gagner beaucoup :
`https://youtu.be/3iIz-ONpLSI?t=2053`

I. Un peu d'histoire

- 220 av J.C.	Première utilisation d'une liste triée pour faciliter les recherches - Babylone (Mésopotamie, ancienne Irak)
1946	Exposé du principe de la recherche dichotomique par John Mauchly (co-concepteur de l'ENIAC)

II. Recherche d'un élément : méthode séquentielle

On a vu auparavant (Chapitre : Parcours séquentiel d'un tableau) l'algorithme de recherche dans un tableau non trié :

Algorithme : sequentielle(liste,x)

```
/* Algorithme de recherche séquentielle de x dans une liste                    */
```

Entrées :
x : l'entier que l'on cherche
liste : une liste d'entiers dans laquelle on cherche
Sorties : un booléen

```
1 début
2     pour chaque élément de la liste faire
3         si element = x alors
4             renvoyer Vrai
5         fin si
6     fin pour
7     renvoyer Faux
8 fin
```

Son implémentation en Python est :

```
def sequentielle(liste,x):
    """algorithme de recherche de x dans liste
    Renvoie True si x est dans la liste et False sinon

    Entrées :
    x, un entier
    liste, une liste d'entiers

    Sorties :
    un booléen

    Exemples d'exécution :
    >>> sequentielle([1, 8, 5], 8)
    True
    >>> sequentielle([1, 8, 5], 2)
    False
    """
    for elt in liste :
        if elt == x:
            return True
    return False
```

— L'algorithme fonctionne avec une liste triée ou non triée.

— Le temps d'exécution (la complexité en temps) s'accroît linéairement avec la longueur de la liste.

— On est assuré que l'algorithme se termine, car on utilise une boucle bornée (la boucle `for` parcourt chaque élément de la liste, et s'arrête lorsqu'elle atteint, au pire, le dernier élément).

Activité 1.
On considère la liste suivante :

-8	-6	-3	-1	4	7	12	13	18	20	31	33	48	59	72	73	80

1. Comptez le nombre de fois qu'il faudra évaluer la valeur d'une cellule du tableau ci-dessus avant d'y trouver la valeur 31.

2. Quelle est(sont) la(les) valeur(s) du tableau qui demandera(ont) le plus grand nombre d'itérations avant d'être trouvée(s). Et combien d'itérations sont nécessaires ?

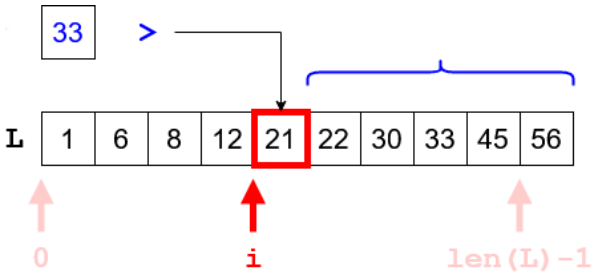
III. Principe de la recherche par dichotomie

La recherche dichotomique permet de rechercher une valeur dans une liste triée. Le principe est le suivant :

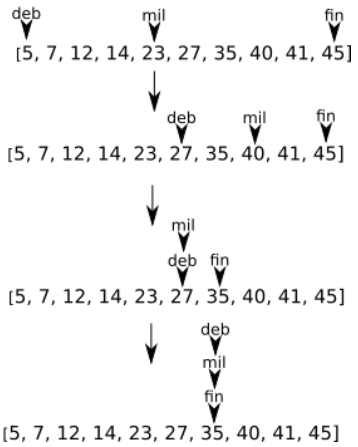
— Si x **est égal** à la valeur au milieu de la liste, alors on s'arrête.

— Si x **est inférieur** à la valeur du milieu de la liste, alors on recommence la recherche dans la première moitié, en excluant la valeur du milieu.

— Sinon on recommence la recherche dans la deuxième moitié, en excluant la valeur du milieu.



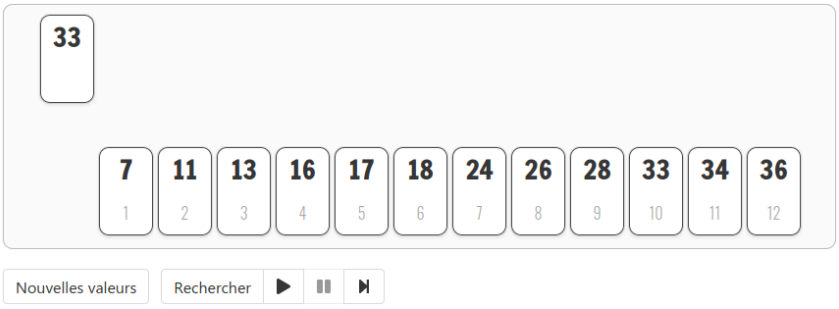
Exemple 1 (Recherche de la valeur 35 dans la liste)



ATTENTION

La recherche par dichotomie ne fonctionne qu'avec une liste triée.

Un exemple animé est disponible ici : `https://professeurb.github.io/articles/dichoto/`



Activité 2.

On considère la liste suivante :

-8	-6	-3	-1	4	7	12	13	18	20	31	33	48	59	72	73	80

On cherche si la valeur 31 est présente dans ce tableau.

1. **Premier tour** :
   - **a.** Encadrez la valeur située au milieu de la liste ci-dessous :

-8	-6	-3	-1	4	7	12	13	18	20	31	33	48	59	72	73	80

   - **b.** La valeur cherchée est-elle égale à la valeur située au milieu ?
   - **c.** Écrivez la liste dans laquelle on va poursuivre la recherche :

2. **Deuxième tour** :
   - **a.** Encadrez la valeur située au milieu de la liste ci-dessous :

20	31	33	48	59	72	73	80

   - **b.** La valeur cherchée est-elle égale à la valeur située au milieu ?
   - **c.** Écrivez la liste dans laquelle on va poursuivre la recherche :

3. **Troisième tour** :
   - **a.** Encadrez la valeur située au milieu de la liste ci-dessous :

20	31	33

   - **b.** La valeur cherchée est-elle égale à la valeur située au milieu ?
4. En combien d'itérations a-t-on trouvé la valeur recherchée. Comparez à la méthode séquentielle (voir paragraphe II.).
5. Comptez le nombre d'itérations pour trouver la valeur 80.
6. Comptez le nombre d'itérations pour trouver la valeur -8.
7. Quelle est(sont) la(les) valeur(s) du tableau qui demandera(ont) le plus grand nombre d'itérations avant d'être trouvée(s). Et combien d'itérations sont nécessaires ?

IV. Pseudo-code et implémentation Python

Algorithme : dichotomie(liste,x)

```
/* Algorithme de recherche par dichotomie de x dans une liste                  */
```

Entrées :
x : l'entier que l'on cherche
liste : une liste **triée** d'entiers dans laquelle on cherche
Sorties : un booléen

```
1  début
2      debut ← 0;
3      fin ← longueur(liste) - 1;
4      milieu ← int((debut + fin)/2);
5      tant que debut ⩽ fin faire
6          si x = liste[milieu] alors
7              renvoyer Vrai
8          sinon si liste[milieu] > x alors
9              fin ← milieu - 1
10         sinon
11             debut ← milieu + 1
12         fin si
13         milieu ← int((debut + fin)/2);
14     fin tq
15     renvoyer Faux
16 fin
```

Son implémentation en Python est :

```python
def dichotomie(liste,x):
    """algorithme de recherche de x dans liste
    Renvoie True si x est dans la liste et False sinon
    Entrées :
    x, un entier
    liste, une liste d'entiers

    Sorties :
    un booléen

    Exemples d'exécution :
    >>> dichotomie([1, 5, 8], 8)
    True
    >>> dichotomie([1, 5, 8], 2)
    False
    """
    debut = 0
    fin = len(liste) - 1
    milieu = int((debut + fin)/2)

    while debut <= fin:
        if x == liste[milieu]:
            return True
        elif liste[milieu] > x:
            fin = milieu - 1
        else :
            debut = milieu + 1
        milieu = int((debut + fin)/2)

    return False
```

V. Références bibliographiques

- ▷ Numérique et Sciences Informatiques, Serge Blays, ellipses, 2019, p. 288
- ▷ Informatique, Nicolas Audfray et al., Dunod, 2017, p.85
- ▷ `https://info.blaisepascal.fr/nsi-recherche-dichotomique`, 10/04/2020

VI. Exercices

QCM

1. Avec un algorithme de recherche par dichotomie, combien d'étapes sont nécessaires pour déterminer que 35 est présent dans le tableau [1, 7, 12, 16, 18, 20, 24, 28, 35, 43, 69] ?
   - ☐ 2 étapes
   - ☐ 1 étape
   - ☐ 9 étapes
   - ☐ 11 étapes
2. Pour pouvoir utiliser un algorithme de recherche par dichotomie dans une liste, quelle précondition doit être vraie ?
   - ☐ La liste doit être triée
   - ☐ La liste ne doit pas comporter de doublons
   - ☐ La liste doit comporter uniquement des entiers positifs
   - ☐ La liste doit être de longueur inférieure à 1024
3. Un algorithme de recherche dichotomique dans une liste triée de taille n nécessite exactement k comparaisons dans le pire des cas. Combien de comparaisons sont nécessaires avec le même algorithme pour une liste de taille 2n ?
   - ☐ 2k + 1 comparaisons.
   - ☐ 2k comparaisons.
   - ☐ k + 2 comparaisons.
   - ☐ k + 1 comparaisons.
4. Voici un code en Python :

```
x = 1
while x / 2 > 0:
    x = x/2
```

   Quelle affirmation est vraie ?
   - ☐ C'est une boucle infinie, le programme ne termine jamais.
   - ☐ Le programme s'arrête pendant l'exécution en signalant une erreur
   - ☐ Le code s'exécute sans erreur et la valeur finale de x est 0.
   - ☐ Le code s'exécute sans erreur et la valeur finale de x est le plus petit flottant strictement positif.

Exercice 1.

1. Écrivez une fonction Python *dichotomie_indice*(), prenant *liste* (une liste de nombres triées par ordre croissant) et *val* (la valeur recherchée) en argument, et renvoyant l'indice de l'élément cherché dans un liste triée.
2. Modifiez cette fonction pour qu'elle renvoie aussi le nombre d'itérations réalisées.

Exercice 2 (Résolution numérique d'une équation).
On considère une fonction mathématique définie par $f(x) = 3x^2 - 2x - 2000$, continue sur l'intervalle [a, b]. Pour simplifier, on considère que la fonction est croissante, c'est à dire que $f(a) < f(b)$ et que $f(a) < 0$ et $f(b) > 0$ (ce qui est bien le cas ici).

On cherche, dans cet exercice, à écrire une fonction python, renvoyant la valeur de x pour laquelle $f(x) = 0$. Pour éviter de comparer deux réels, on introduit une précision *epsilon*. On admet que la fonction s'arrête lorsque $f(x) < epsilon$.

Pour cela, on procède de la façon suivante :

- On détermine le milieu de l'intervalle [a, b] : $m = \dfrac{a+b}{2}$
- On regarde si $f(a)$ et $f(m)$ sont du même signe :
  - ▷ s'ils sont du même signe, on réduit la recherche à la deuxième moitié de l'intervalle,
  - ▷ s'ils sont de signe opposé, on réduit la recherche à la première moitié de l'intervalle.
- On recommence l'opération tant que $|a - b| < epsilon$.
- À la sortie de la boucle, on renvoie la valeur de *milieu*.

1. Comment, en utilisant une multiplication, peut-on savoir si $f(a)$ et $f(m)$ sont du même signe ?

2. Comment peut-on réduire la recherche à la première moitié de l'intervalle ?

3. Comment peut-on réduire la recherche à la deuxième moitié de l'intervalle ?

4. Déterminez les deux racines x_1 et x_2 de l'équation. Imaginez deux post-conditions utilisant ce résultat.
Aide : on pourra utiliser la fonction *round()* dont la documentation est disponible ici : `https://docs.python.org/fr/3.7/library/functions.html?highlight=round#round`.

5. Écrivez la fonction Python *recherche_racine*(), prenant les bornes de l'intervalle (a et b) et la précision *epsilon* en argument, et renvoyant la valeur pour laquelle la fonction f s'annule à *epsilon* près (ce qu'on appelle la racine).

6. Modifiez cette fonction pour qu'elle renvoie aussi le nombre d'itérations réalisées.

7. Combien d'itérations sont nécessaires, dans le cas où l'on recherche la racine dans l'intervalle [0, 100], avec une précision de $1 \cdot 10^{-3}$?

8. Dans le cas d'un parcours séquentiel, quel serait le pas d'itération ? Combien d'itérations seraient nécessaires pour la même recherche ?

On souhaite maintenant pouvoir résoudre une équation du type $f(x) = k$.

9. Comment faudrait-il procéder pour se retrouver dans le cas précédent ?

10. Quel serait l'argument supplémentaire à passer dans la fonction Python ?

11. Écrivez la fonction Python *resolution_equation*(), prenant les bornes de l'intervalle (a et b), la précision *epsilon* et la valeur du seconde membre k en argument, et renvoyant la valeur pour laquelle $f(x) = k$ à *epsilon* près.