Recherche dichotomique

Objectif :

Le programme au BO indique dans la partie contenu : « recherche dichotomique dans un tableau trié ». Il se peut que la notion de dichotomie ait été vue en mathématiques en classe de seconde (exemple d'algorithme : pour une fonction dont le tableau de variations est donné, algorithmes d'approximation numérique d'un extremum (balayage, dichotomie) mais cela ne suffit bien sûr pas à la considérer comme acquise.
L'algorithme est intéressant par lui-même mais il va servir de prétexte à découvrir d'autres notions largement utilisées et qui seront approfondies par la suite :

- étude du coût d'un algorithme,
- preuve de la terminaison d'un programme à l'aide de la notion de variant,
- preuve de la correction d'un algorithme à l'aide la notion d'invariant.

Modalité de mise en œuvre :

Environnement python,

Dictionnaires,

(Variante : cartes avec des mots).

Pré-requis :

Boucles, conditions, tableaux en algorithmique, fonctions

Implémentation en langage python

Bibliothèque spécifique à python étudiée dans d'autres disciplines (matplotlib).

Compétences :

- anticiper (terminaison d'un algorithme)
- évaluer (notion de complexité)
- décomposer (simplifier le problème)
- abstraire (concevoir un algorithme)

Séance 1 (2 heures)

- **Partie 1** – Mise en situation – 10 min
- **Partie 2** – Algorithme papier et implémentation python – 45 min
- **Partie 3** – Exploitation des résultats – 30 min

Le reste de la séance pour remédiation, diagnostic, exercices, corrections…

Séance 2 (1 heure)

- **Partie 4** - Terminaison et correction des algorithmes

Séance facultative

- **Partie 5** – Exercices complémentaires

Partie 1 Mise en situation (activité débranchée)

Les élèves cherchent la page où un mot est présent sur un dictionnaire selon diverses stratégies.

- Le 1er groupe recherche le mot en commençant par la 1ère lettre de l'alphabet (stratégie linéaire)
- Le 2nd groupe partage le dictionnaire en 2 parties égales, élimine la partie où le mot n'est pas, et recommence (stratégie dichotomique)
- Méthode aléatoire

Partie 2 Le nombre secret

Afin de quantifier plus facilement les programmes, on passe à une liste de nombres ordonnée à la place des mots, ce qui facilite l'implémentation en Python (pas de dictionnaire à créer).
Vous devez réaliser un algorithme (sur papier) permettant à l'ordinateur de deviner le nombre compris entre 1 et n que vous avez choisi. Vous pouvez utiliser l'une ou l'autre des stratégies vues plus haut (ou les deux si vous êtes rapides). Puis on passe sur l'ordinateur, implémentation en Python.

```
liste=[]
for i in range(1,10001):
    liste.append(i)

# fonction lineaire pour 10000 valeurs
def lineaire(nbre_secret):
    for i in range(10000):
        if liste[i]==nbre_secret:
            return "trouvé!"
```

```
# fonction dicho pour 10000 valeurs
def dicho(nbre_secret):
    a = liste[0]
    b=liste[9999]
    m=(a+b)//2
    while m!=nbre_secret:
        if nbre_secret<m:
            b=m
            m=(a+b)//2
        else:
            a=m
            m=((a+b)//2)+1
    return "trouvé!"
```

Partie 3 Exploitation des résultats

lineairedicho.py

On va chercher avec quelle vitesse on obtient les résultats

- en exécutant les fonctions **linéaire_print** et **dicho_print**
- en exécutant les fonctions **linéaire_etapes** et **dicho_etapes**
- en exécutant les fonctions **linéaire_print** et **dicho_print**
- en exécutant la fonction **graphes**

```
# fonction lineaire_print pour 10000 valeurs
def lineaire_print(nbre_secret):
    for i in range(10000):
        print("test n° ", i+1)
        if liste[i]==nbre_secret:
            return "trouvé!"

# fonction lineaire_etapes pour 10000 valeurs
def lineaire_etapes(nbre_secret):
    nb_etapes=0
    for i in range(10000):
        nb_etapes=nb_etapes+1
        if liste[i]==nbre_secret:
            return "trouvé! en ", nb_etapes, " étapes"

# fonction lineaire_time pour 10000 valeurs
def lineaire_time(nbre_secret):
    start_time = time.time()
    for i in range(10000):
        if liste[i]==nbre_secret:
            return "trouvé! en " , time.time() - start_time
```

```
# fonction dicho_print pour 10000 valeurs
def dicho_print(nbre_secret):
    a = liste[0]
    b=liste[9999]
    m=(a+b)//2
    cpt=0
    while m!=nbre_secret:
        cpt=cpt+1
        if nbre_secret<m:
            print("test n° ", cpt, "sur intervalle de", a, "à", m)
            b=m
            m=(a+b)//2
        else:
            print("                                                test n° ", cpt, "sur intervalle de", m, "à", b)
            a=m
            m=((a+b)//2)+1
    print("test n° ",cpt+1)
    return "trouvé!"
```

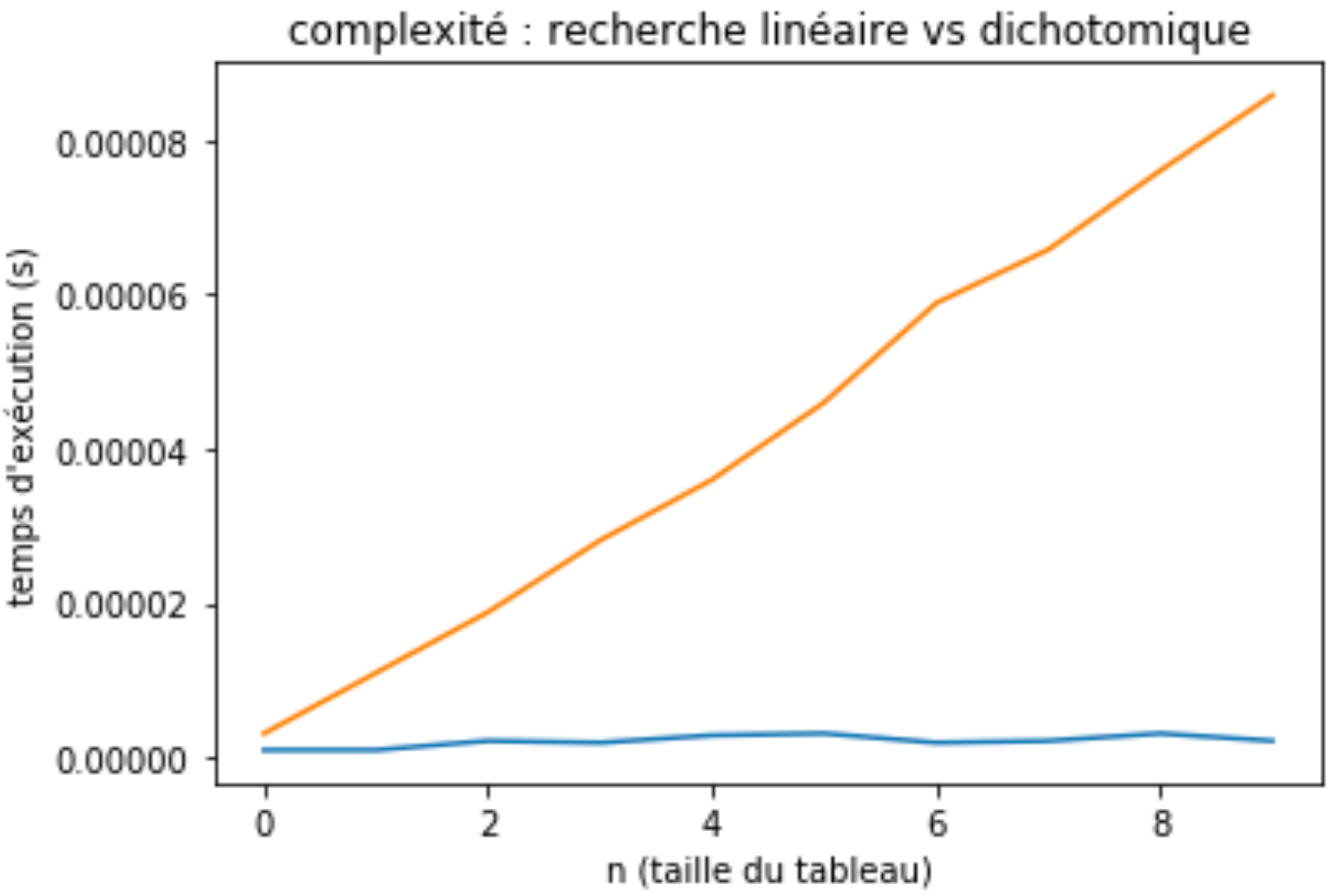
```
# fonction dicho_etapes pour 10000 valeurs
def dicho_etapes(nbre_secret):
    a = liste[0]
    b=liste[9999]
    m=(a+b)//2
    nb_etapes=0
    while m!=nbre_secret:
        nb_etapes=nb_etapes+1
        if nbre_secret<m:
            b=m
            m=(a+b)//2
        else:
            a=m
            m=((a+b)//2)+1
    nb_etapes=nb_etapes+1
    return "trouvé! en ", nb_etapes, "étapes"
```

```
# fonction dicho_time pour 10000 valeurs
def dicho_time(nbre_secret):
    start_time = time.time()
    a = liste[0]
    b=liste[9999]
    m=(a+b)//2
    while m!=nbre_secret:
        if nbre_secret<m:
            b=m
            m=(a+b)//2
        else:
            a=m
            m=((a+b)//2)+1
    return "trouvé! en ", time.time()-start_time
```

```
# fonctions lineaire_tps, dicho_tps et graphes pour n valeurs
def lineaire_tps(taille,nbre_secret):
    start_time = time.time()
    n=taille
    for i in range(n):
        if liste[i]==nbre_secret:
            return time.time() - start_time
```

```
def dicho_tps(taille,nbre_secret):
    start_time = time.time()
    n=taille
    a = liste[0]
    b=liste[n-1]
    m=(a+b)//2
    while m!=nbre_secret:
        if nbre_secret<m:
            b=m
            m=(a+b)//2
        else:
            a=m
            m=((a+b)//2)+1
    return time.time()-start_time
```

```
def graphes(taille):
    global liste
    liste=[]
    for i in range(1,taille+1):
        liste.append(i)
    liste_tlin=[]
    liste_tdich=[]
    for t in range(10,taille+1,100):
        n_secret=t
        liste_tlin.append(lineaire_tps(t,n_secret))
        liste_tdich.append(dicho_tps(t,n_secret))
    plt.title("complexité : recherche linéaire vs dichotomique")
    plt.xlabel("n (taille du tableau)")
    plt.ylabel("temps d'exécution (s)")
    plt.plot(liste_tdich)
    plt.plot(liste_tlin)
    plt.show()
    return "Fini"
```


complexité : recherche linéaire vs dichotomique
temps d'exécution (s)
0.00008
0.00006
0.00004
0.00002
0.00000
0
2
4
6
8
n (taille du tableau)

Séance 1 Bilan

Les algorithmes de recherche n'ont pas tous la même efficacité on parle de complexité. Certains sont plus efficaces que d'autres comme la recherche dichotomique consistant à rechercher l'élément dans un sous-ensemble réduit de moitié à chaque étape.

Grâce à la méthode par dichotomie, par exemple pour un ensemble de 10000 éléments, la machine trouvera à coup sûr le nombre secret en 14 tests maximum(2^{14}=16384).

Ce nombre de tests qui dépend de la taille n de l'ensemble est le nombre de fois qu'il faut diviser n par 2 pour obtenir un nombre inférieur ou égal à 1. (résolution de l'équation et notation $\log_2(n)$ en Terminale)

Partie 4 Terminaison et correction des algorithmes

```
liste=[]
for i in range(1,10001):
    liste.append(i)

def dicho(nbre_secret):
    debut = liste[0]
    fin = liste[9999]
    m=(debut+fin)//2
    cpt=0
    if nbre_secret<0 or nbre_secret>10000:
        return False
    while m!=nbre_secret:
        cpt=cpt+1
        if nbre_secret<m:
            print("test n° ", cpt, "sur intervalle de", debut, "à", m)
            fin = m
            m=(debut+fin)//2
        else:
            print("test n° ", cpt, "sur intervalle de", m, "à", fin)
            debut=m
            m=((debut+fin)//2)
    print("test n° ",cpt+1)
    print("Le nombre choisi était ", m)
    return True
```

Ce programme vous parait-il satisfaisant ?

- Notre algorithme va-t-il se terminer dans tous les cas ? Notion de terminaison
- Notre algorithme est-il correct, c'est à dire qu'il va bien nous donner le nombre secret ? Notion de preuve.

- Pourquoi un programme ne se terminerait-il pas ?
- Que proposeriez-vous pour répondre à ces questions ?

Montrer la terminaison de la recherche dichotomique à l'aide d'un variant de boucle :

Pour démontrer la terminaison d'une boucle, il faut par exemple trouver une quantité, fonction des paramètres de la boucle, qui est un entier positif et qui décroît strictement à chaque itération. Cette quantité est appelée **variant de boucle**.

Quel pourrait être ici le variant à tester ?

Si le tableau ne contient qu'un nombre, c'est forcément celui-là et on est sûr de sortir de la boucle.
On va donc montrer que le nombre d'éléments du tableau diminue jusqu'à la taille 1.

Montrer la correction de la recherche dichotomique à l'aide d'un invariant de boucle :

Pour démontrer qu'un algorithme produit le résultat attendu, on utilise un invariant de boucle, c'est-à-dire une propriété qui est vérifiée avant d'entrer dans la boucle. Si cette propriété est vérifiée avant une itération, elle est vérifiée après celle-ci. Lorsqu'elle est vérifiée en sortie de boucle, elle permet (combinée avec la condition de sortie de boucle) d'en déduire que le programme produit le résultat attendu.

Pour cela, montrons que la propriété P_i suivante est un invariant de la boucle « tant que ».
P_i : « Le nombre recherché appartient au tableau délimité par les indices début et fin avant l'entrée dans la boucle.

La démonstration va se faire en 3 étapes :

- On montre que l'invariant de boucle est vrai avant la première itération de la boucle, c'est la phase d'initialisation.

- On montre que si l'invariant de boucle est vrai avant une itération de la boucle, il le reste avant l'itération suivante, c'est la phase de conservation.

- Enfin, on montre qu'après les passages dans la boucle, l'invariant fournit une propriété utile qui aide à montrer la correction de l'algorithme

Partie 5 Exercices complémentaires

➢ Algorithme dichotomique récursif

L'algorithme de recherche dichotomique peut-être résolu par l'utilisation de la récursivité.
Une fonction est dite récursive si elle comporte, dans son corps, au moins un appel à elle-même en général avec un sous-ensemble des données.
Le but est alors d'écrire une fonction dichotomique qui dans sa définition fera appel a elle-même deux fois :

- une fois pour la recherche à gauche si le nombre mystère a trouver est dans le tas de gauche
- ou alors a droite

Voici une ébauche de solution :

Def ***dichotomie***(<liste ordonnée de nombres>, nb_mystere) :

    <traitement …>

    Si nb_mystere est dans le tas de gauche alors j'appelle dichotomie(<sous-liste ordonnée de nombre à gauches>)

    Si ***nb_mystere est dans le tas de droite alors j'appelle dichotomie***(<sous-liste ordonnée de nombre à droite>)

    <traitement …>

fin de la définition de la fonction dichotomie

Solution à l'exercice voir fichier **dichotomie_recursive.py**

Partie 5 Exercices complémentaires

- Activité sur la complexité

activité

- Exercices sur la terminaison

fiche_prof

Partie 5 Exercices complémentaires

QCM 7.1

A désignant un entier, lequel des codes suivants ne termine pas ?

Réponses possibles

A

```
i = A + 1
while i < A:
    i = i-1
```

B

```
i = A + 1
while i < A:
    i = i + 1
```

C

```
i = A - 1
while i < A:
    i = i - 1
```

D

```
i = A - 1
while i < A:
    i = i + 1
```