Activité : Diviser pour régner

Introduction

L'expression **diviser pour régner** est très ancienne, que l'on retrouve dans les manuels d'histoire. C'est une stratégie d'affaiblissement d'un adversaire qui consiste à diviser ses membres et à les opposer pour semer la discorde et ainsi rendre la victoire plus facile.

En informatique, le concept est plus pacifique, mais la stratégie demeure. L'algorithme consiste à :

— Diviser, c'est à dire découper un problème initial en sous-problèmes ;

— Régner en résolvant les sous problèmes plus petits donc plus simples.

— Combiner les solutions des sous-problèmes pour donner une solution au problème initial.

Recherche dichotomique

1) La recherche d'un nombre dans une liste peut se faire en parcourant la liste jusqu'à trouver le nombre cherché.

a) Pour une liste de 20 nombres, combien de comparaisons seront au maximum nécessaires pour savoir si un nombre se trouve dans la liste ?

b) Dans le cas général, combien de comparaisons, dans le pire des cas, pour déterminer la présence ou non d'un nombre dans une liste de n nombres.

2) On peut améliorer la complexité de la recherche en effectuant la recherche sur une liste triée. La méthode repose sur le recherche dichotomique.

a) En quoi consiste cette méthode ?

b) Combien de comparaisons sont nécessaires (au pire des cas) pour une liste de 20 nombres ?

c) Qu'en est-il pour une liste de n nombres ?

d) On peut représenter graphiquement les valeurs avec le module **pyplot** de **matplotlib**.

```
1  # on importe le logarithme en base 2
2  from math import log2
3
4  # on importe le module matplotlib pour les
5  # représentations graphiques de données
6  from matplotlib import pyplot as plt
7
8  # On definit une liste d'abscisses de points
9  X=[i for i in range(1,1000)]
10
11 # On définit les ordonnées de chaque point
12 Y=[log2(i) for i in range(1,1000)]
13
14 # On représente graphiquement les points
15 plt.plot(X,Y)
```

Déterminer graphiquement la longueur de la liste pour avoir une recherche dichotomique supérieure à 10 au pire des cas.

e) On donne une version itérative, en Python, de cet algorithme :

```python
def dichotomie(liste,v):
    g=0
    d=len(liste)-1
    while g<=d:
        m=(g+d)//2
        if liste[m]<v:
            g=m+1
        elif liste[m]>v:
            d=m-1
        else:
            return True
    return False
```

En donner une version récursive.

f) L'algorithme récursif échoue dans le cas où le nombre maximal d'appels récursifs dépasse 3000 (par défaut dans python). Dans Python, la longueur maximale d'une liste est 2^{30}. ESt-ce que cela peut se produire ?

Le tri fusion

Le principe du tri fusion est une application du principe **diviser pour régner**. Vous trouverez des informations sur la page wikipedia suivante : `https://fr.wikipedia.org/wiki/Tri_fusion`

1) Appliquer le tri fusion sur la liste $L = [7, 4, 9, 1, 3, 5, 8]$ en donnant précisément les différentes étapes.

2) L'algorithme se divise en 2 parties. La première partie consiste à diviser la liste, et la seconde fusionne les listes triées.

On donne le script de la division :

```python
def tri_fusion(liste):
    if len(liste)<2:
        return liste
    else:
        liste1=tri_fusion(liste[0:len(liste)//2])
        liste2=tri_fusion(liste[len(liste)//2:len(liste)])
        return fusion_liste(liste1,liste2)
```

Écrire la fonction **fusion_liste** qui effectue la fusion de 2 listes triées (de préférence en itératif).

3) Créer des listes aléatoires de nombres entiers puis vérifier que le tri fusion est réalisé.

4) On va comparer l'efficacité du tri fusion par rapport au tri par sélection.

a) Recréer la fonction de tri par sélection d'une liste de nombres.

b) Effectuer des mesures sur le temps de tri pour chacune des méthodes (fusion, sélection) en prenant des tailles de listes de 100 à 1000 nombres avec un pas de 100.

c) Représenter sur un même graphique les mesures.