Algorithmes de tri et complexité

I) Problématique

Comment ranger des données afin de faciliter leur accès ?

Dans un dictionnaire, les mots sont rangés dans un ordre logique, l'ordre alphabétique qui permet de ne pas devoir parcourir tout l'ouvrage pour retrouver une définition.
De même, un joueur de cartes va ranger son jeu selon un ordre intuitif, afin de limiter le temps de recherche pendant le déroulement de la partie.
Ranger des données pour faciliter leur accès futur revient à les « trier », nous allons donc introduire la notion de tri (avec plusieurs sens distincts : séparer, ordonner, choisir), puis d'étudier quelques algorithmes de tri.

II) Les algorithmes de tri

L'un des problèmes qui se posent lorsque l'on commence la programmation est celui qui consiste à trier un tableau de valeurs. Comme ce type de problème revient très souvent, des méthodes de tri efficaces ont été recherchées et découvertes. Ces méthodes sont bien entendues aujourd'hui disponibles dans les bibliothèques de tous les langages de programmation mais l'objectif de l'activité est de tenter d'en redécouvrir certaines et de les mettre en œuvre dans le langage Python.
L'importance de ce problème fait que plusieurs dizaines d'algorithmes différents ont été proposés pour trier des objets : ***tri par sélection, tri par fusion, tri par insertion, tri rapide, tri bulle…***
Le but d'un algorithme de tri est donc de calculer un nouveau tableau, ou de modifier le tableau initial, de manière à ce qu'il contienne les mêmes nombres que le tableau initial, mais que ces éléments soient ordonnés.

III) Mesure et estimation du temps d'exécution d'un algorithme

Il n'est pas possible de déterminer à l'avance le temps exact que prendra un ordinateur pour exécuter un algorithme : cette caractéristique dépend de trop nombreux paramètres, tant matériels que logiciels. En revanche, il est souvent possible d'évaluer l'ordre de grandeur du temps d'exécution en fonction des paramètres de l'algorithme. Par exemple, si n désigne la taille du paramètre d'entrée, un algorithme sera qualifié de linéaire lorsque le temps d'exécution croîtra proportionnellement à n et de quadratique lorsque le temps d'exécution croîtra proportionnellement à n^2…

Pour chacun des algorithmes de tri suivants, vous devez :

a) Implémenter l'algorithme sous Python et tester son bon fonctionnement (au pire des cas et dans des cas quelconques) sur des tableaux de différentes tailles (grandes ou petites), et pour une taille donnée, choisir des listes aléatoires ou non, triées dans l'ordre croissant ou décroissant.
b) Tracer les courbes de complexité, puis les comparer.

1) Tri par insertion

Principe : c'est la méthode de tri du joueur de cartes

- *Séparer le tableau en 2 parties : partie déjà triée, partie non triée*
- *Prendre un élément non trié et l'insérer à sa place dans l'ensemble des éléments triés*
- *Recommencer le processus jusqu'à ce qu'il n'y ait plus d'éléments de la liste initiale à insérer*

http://lwh.free.fr/pages/algo/tri/tri_rapide.html

Programme en python :

```
def tri_insertion(tableau):
    for i in range(1,len(tableau)):
        en_cours = tableau[i]
        j = i
        #décalage des éléments du tableau
        while j>0 and tableau[j-1]>en_cours:
            tableau[j]=tableau[j-1]
            j = j-1
        #on insère l'élément à sa place
        tableau[j]=en_cours
```

Activité 1 :

1) Effectuer à la main un tri de la liste suivante : T = {6,9, 5, 0, 3, 2 ,7}

i	Tableau T avant l'itération sur i								Tableau T après l'itération sur i							
1																
2																
3																
4																
5																
6																
7																

2) **Tri bulle**

Principe :

- *comparer deux à deux les éléments e1 et e2 consécutifs d'un tableau et d'effecteur une permutation si e1 > e2.*
- *continuer de trier jusqu'à ce qu'il n'y ait plus de permutation.*

Programme en python :

```
def tri_bulle(tableau):
    permutation = True
    passage = 0
    while permutation == True:
        permutation = False
        passage = passage + 1
        for en_cours in range(0, len(tableau) - passage):
            if tableau[en_cours] > tableau[en_cours + 1]:
                permutation = True
                # On echange les deux elements
                tableau[en_cours], tableau[en_cours + 1] = \
                tableau[en_cours + 1],tableau[en_cours]
return tableau
```

Activité 2 :

1) Effectuer à la main un tri de la liste suivante : T = {6,9, 5, 0, 3, 2 ,7}

i	Tableau T avant l'itération sur i									Tableau T après l'itération sur i								
1																		
2																		
3																		
4																		
5																		
6																		
7																		

3) Le tri rapide

Principe :

La méthode consiste à placer un élément du tableau (appelé pivot) à sa place définitive, en permutant tous les éléments de telle sorte que tous ceux qui sont inférieurs au pivot soient à sa gauche et que tous ceux qui sont supérieurs au pivot soient à sa droite.
Cette opération s'appelle le partitionnement. Pour chacun des sous-tableaux, on définit un nouveau pivot et on répète l'opération de partitionnement. Ce processus est répété récursivement, jusqu'à ce que l'ensemble des éléments soit trié.

Concrètement, pour partitionner un sous-tableau :

- *le pivot est placé à la fin (arbitrairement), en l'échangeant avec le dernier élément du sous-tableau ;*
- *tous les éléments inférieurs au pivot sont placés en début du sous-tableau ;*
- *le pivot est déplacé à la fin des éléments déplacés*

Programme en python :

```
def tri_rapide(tableau):
    if not tableau:
        return []
    else:
        pivot = tableau[-1]
        plus_petit = [x for x in tableau     if x <  pivot]
        plus_grand = [x for x in tableau[:-1] if x >= pivot]
        return tri_rapide(plus_petit) + [pivot] + tri_rapide(plus_grand)
```

4) Le tri fusion

Principe :

Il s'agit d'un tri basé sur la stratégie « diviser pour régner ». Le principe du tri fusion est le suivant :

- *On divise en deux moitiés la liste à trier (en prenant par exemple, un élément sur deux pour chacune des listes).*
- *On trie chacune d'entre elles.*
- *On fusionne les deux moitiés obtenues pour reconstituer la liste triée.*

Programme en python :

```
def fusion(gauche,droite):
    resultat = []
    index_gauche, index_droite = 0, 0
    while index_gauche < len(gauche) and index_droite < len(droite):
        if gauche[index_gauche] <= droite[index_droite]:
            resultat.append(gauche[index_gauche])
            index_gauche += 1
        else:
            resultat.append(droite[index_droite])
            index_droite += 1
    if gauche:
        resultat.extend(gauche[index_gauche:])
    if droite:
        resultat.extend(droite[index_droite:])
    return resultat

def tri_fusion(m):
    if len(m) <= 1:
        return m
    milieu = len(m) // 2
    gauche = m[:milieu]
    droite = m[milieu:]
    gauche = tri_fusion(gauche)
    droite = tri_fusion(droite)
    return list(fusion(gauche, droite))
```