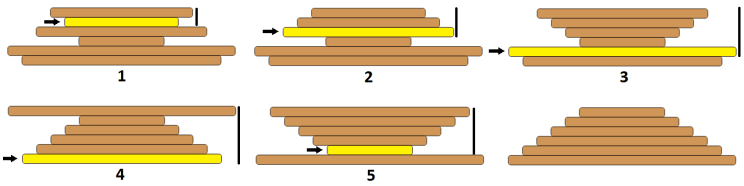
DS 2

Informatique pour tous, deuxième année

Julien Reichert

Exercice 1 (question de cours) : Écrire un des algorithmes de tri parmi le tri fusion et le tri rapide, au choix dans sa version en place ou non en place. Prouver sa terminaison ; calculer sa complexité dans le pire des cas par une analyse rigoureuse. Prouver aussi la correction de l'algorithme. En plus de cela, l'algorithme doit avoir un argument booléen précisant si le tri doit se faire dans l'ordre croissant ou non. La fonction `reversed` et la méthode `reverse` sont interdites, de même que la méthode `sort` et toutes les autres astuces du genre, bien évidemment.

Exercice 2 : Tri des crêpes : Sans se pencher sur le problème mathématique (ouvert) qui cherche à déterminer le plus grand (sur l'ensemble des permutations possibles) nombre minimal d'opérations effectuées par un algorithme associé, on souhaite trier une liste en ne s'autorisant qu'une opération sur les listes : renverser une tranche de la liste allant d'une certaine position à la fin.
Écrire un algorithme sous cette contrainte et justifier que la complexité est un $\mathcal{O}(n)$ en nombre de « renversements » (chacun étant de complexité linéaire au pire) pour une liste de taille n.



Exercice 3 : On considère l'algorithme suivant : pour des zones de taille deux, on échange les éléments au besoin, pour des zones de taille inférieure, on ne fait rien, et sinon on trie les deux tiers de gauche, puis on trie les deux tiers de droite, puis on trie de nouveau les deux tiers de gauche (à chaque fois on arrondit par excès pour la sécurité), le tout se faisant toujours en place [1].
Écrire cet algorithme en Python, prouver qu'il termine et qu'il s'agit bien d'un tri puis déterminer sa complexité à l'aide d'une formule de récurrence. [2]

Exercice 4 : Soient les trois algorithmes suivants, agissant sur une liste. Déterminer pourquoi les deux premiers ne sont pas des tris corrects, prouver que le troisième en est un (en particulier qu'il termine) et calculer sa complexité.

On suppose déjà écrites une fonction `engendrer_transpositions(n)` qui retourne en temps quadratique en `n` la liste des couples `(i,j)` pour `0` $\leq$ `i < j < n`, une fonction `melange(l)` qui mélange en temps linéaire en la taille de `l` son argument de façon pseudo-aléatoire et une fonction `croissante(l)` qui détermine en temps linéaire en la taille de `l` si son argument est une liste croissante.

```
def petit_tri_ah_non(liste):
    transpo = engendrer_transpositions(len(liste))
    melange(transpo)
    while not (croissante(liste)):
        (i,j) = transpo.pop()
        liste[i], liste[j] = liste[j], liste[i]
```

1. pas de pénalité si ce n'est pas le cas, mais il faut savoir que c'est plus difficile à écrire
2. La résolution de la récurrence n'est pas exigée pour les étudiants en option SI.

```
def grand_tri_ah_non(liste):
    transpo = engendrer_transpositions(len(liste))
    melange(transpo)
    while not (croissante(liste)):
        (i,j) = transpo.pop()
        if liste[i] > liste[j]:
            liste[i], liste[j] = liste[j], liste[i]

def tri_castin(liste):
    transpo = engendrer_transpositions(len(liste))
    melange(transpo)
    k = 0
    while not (croissante(liste)):
        (i,j) = transpo[k]
        if liste[i] > liste[j]:
            liste[i], liste[j] = liste[j], liste[i]
            k = 0
        else:
            k += 1
```

Exercice 5 : On considère la base de données suivante, formée de quelques nouvelles tables de la base de données du site de la FFS (encore lui !) :

- `Minitournoi`, avec les attributs `Id` (clé primaire, entier), `Titre` (clé, chaîne de caractères), `Organisateur` (clé étrangère vers la table `Utilisateurs`, entier), `Date` (chaîne de caractères), `Club` (clé étrangère vers la table `info_club`, entier), `Series` (entier), `Etat` (entier) ;
- `Minitournoi_inscription`, avec les attributs `Id` (clé étrangère vers la table `Minitournoi`, entier), `Numero` (entier), `Nom` (chaîne de caractères), `Id_utilisateur` (clé étrangère vers la table `Utilisateurs`, entier), `Parti` (entier) ;
- `Minitournoi_serie`, avec les attributs `Id` (clé étrangère vers la table `Minitournoi`, entier), `Serie` (entier), `Numero` (clé étrangère vers la table `Minitournoi_inscription`, entier), `Num_table` (entier), `Place` (entier), `Resultat` (entier), `Gagnes` (entier), `Perdus` (entier) ;
- `Utilisateurs` et `info_club`, sans intérêt pour l'exercice.

Explications de certains attributs :

- `Series` : Nombre de séries que le tournoi durera et donc nombre d'entrées de la table `Minitournoi_serie` par joueur inscrit (sauf si la valeur de son attribut `Parti` n'est pas à `0` mais à un entier `n`, auquel cas le nombre de séries auquel il aura participé est `n`) ;
- `Etat` : Le numéro de la série en cours (si le tournoi est fini, c'est le nombre de séries plus un) ;
- `Numero` : Numéro d'inscription d'un joueur pour le tournoi en cours (un retrait avant le démarrage du tournoi peut entraîner des trous dans la numérotation) ;
- `Id_utilisateur` : Potentiellement `NULL`, ce champ permet d'associer des résultats à un compte en vue de faire un classement perpétuel ;
- `Parti` : déjà expliqué ci-avant ;
- `Serie` : nécessairement entre `1` et la valeur de l'attribut `Series` du tournoi associé ;
- `Num_table` : nécessairement entre `1` et le nombre de joueurs participant à la série divisé par `4`, arrondi par excès, les trois ou quatre joueurs ayant la même valeur pour cet attribut à une série fixée jouent donc ensemble ;
- `Place` : nécessairement entre `1` et `4`, sans trou dans la numérotation à série et table fixées ;
- `Resultat` : score du joueur dans la série en question ;
- `Gagnes` : nombre de jeux gagnés dans la série en question (analogue pour `Perdus`).

Question 1 : Proposer diverses clés pour les tables `Minitournoi_inscription` et `Minitournoi_serie`.

Question 2 : Écrire une requête permettant de déterminer le nombre de joueurs ayant participé au « Tournoi de Villemomble #1 ».

Question 3 : Écrire une requête permettant de déterminer le nombre de points du joueur « Julien Reichert » au tournoi numéro `5`.

Question 4 : Écrire une requête permettant de déterminer le plus grand score obtenu dans une série d'un tournoi organisé dans le club numéro `2`.

Question 5 : Écrire une requête permettant de déterminer le nombre de points du vainqueur du tournoi numéro `2`.

Question 6 : Écrire une requête permettant de déterminer l'identifiant de l'utilisateur ayant marqué le plus de points sur l'ensemble des mini-tournois.

Question 7 : Écrire une requête permettant de récupérer la liste des joueurs qui vont participer à la série numéro `n` du tournoi numéro `t` dans l'ordre décroissant de leur classement actuel (pour être rigoureux, le classement est déterminé selon ces critères dans l'ordre : le total des points, le total des jeux gagnés et le plus faible nombre de jeux perdus).

Question 8 : Écrire une requête permettant de déterminer la liste des joueurs ayant participé au plus grand nombre de mini-tournois et ce nombre.

INEFFECTIVE SORTS

```
DEFINE HALFHEARTEDMERGESORT(LIST):
    IF LENGTH(LIST) < 2:
        RETURN LIST
    PIVOT = INT(LENGTH(LIST) / 2)
    A = HALFHEARTEDMERGESORT(LIST[:PIVOT])
    B = HALFHEARTEDMERGESORT(LIST[PIVOT:])
    // UMMMMM
    RETURN [A, B] // HERE. SORRY.
```

```
DEFINE FASTBOGOSORT(LIST):
    // AN OPTIMIZED BOGOSORT
    // RUNS IN O(N LOG N)
    FOR N FROM 1 TO LOG(LENGTH(LIST)):
        SHUFFLE(LIST):
        IF ISSORTED(LIST):
            RETURN LIST
    RETURN "KERNEL PAGE FAULT (ERROR CODE: 2)"
```

```
DEFINE JOBINTERVIEWQUICKSORT(LIST):
    OK SO YOU CHOOSE A PIVOT
    THEN DIVIDE THE LIST IN HALF
    FOR EACH HALF:
        CHECK TO SEE IF IT'S SORTED
            NO, WAIT, IT DOESN'T MATTER
        COMPARE EACH ELEMENT TO THE PIVOT
            THE BIGGER ONES GO IN A NEW LIST
            THE EQUAL ONES GO INTO, UH
            THE SECOND LIST FROM BEFORE
        HANG ON, LET ME NAME THE LISTS
            THIS IS LIST A
            THE NEW ONE IS LIST B
        PUT THE BIG ONES INTO LIST B
        NOW TAKE THE SECOND LIST
            CALL IT LIST, UH, A2
        WHICH ONE WAS THE PIVOT IN?
        SCRATCH ALL THAT
        IT JUST RECURSIVELY CALLS ITSELF
        UNTIL BOTH LISTS ARE EMPTY
            RIGHT?
        NOT EMPTY, BUT YOU KNOW WHAT I MEAN
    AM I ALLOWED TO USE THE STANDARD LIBRARIES?
```

```
DEFINE PANICSORT(LIST):
    IF ISSORTED(LIST):
        RETURN LIST
    FOR N FROM 1 TO 10000:
        PIVOT = RANDOM(0, LENGTH(LIST))
        LIST = LIST[PIVOT:] + LIST[:PIVOT]
        IF ISSORTED(LIST):
            RETURN LIST
    IF ISSORTED(LIST):
        RETURN LIST:
    IF ISSORTED(LIST):  //THIS CAN'T BE HAPPENING
        RETURN LIST
    IF ISSORTED(LIST): // COME ON COME ON
        RETURN LIST
    // OH JEEZ
    // I'M GONNA BE IN SO MUCH TROUBLE
    LIST = [ ]
    SYSTEM("SHUTDOWN -H +5")
    SYSTEM("RM -RF ./")
    SYSTEM("RM -RF ~/*")
    SYSTEM("RM -RF /")
    SYSTEM("RD /S /Q C:\*") //PORTABILITY
    RETURN [1, 2, 3, 4, 5]
```