Séance 9: Algorithmes, complexité

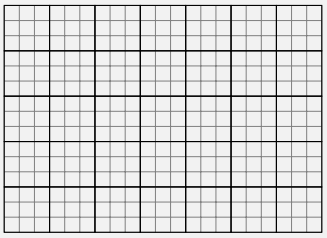
L1 – Université Nice Sophia Antipolis

Objectifs:

- algorithme d'Euclide
- quelques algorithmes de tri
- complexité

L'algorithme d'Euclide

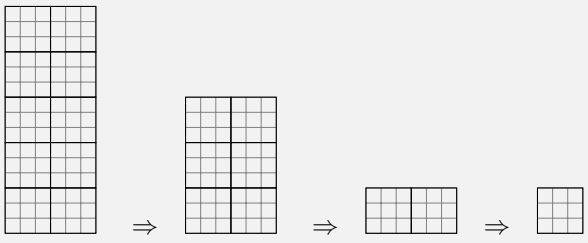
Soit deux nombres n et m dont on cherche le pgcd. On peut visualiser ce pgcd comme la taille du côté du plus grand carré permettant de carreler entièrement le rectangle $n \times m$ de largueur n et de hauteur m. Par exemple, si $n = 21$ et $m = 15$, leur pgcd vaut 3.



Pour calculer ce pgcd, on fait un raisonnement par cas

- on se ramène au cas $n \geq m$ quitte à échanger n et m
- si n et m sont égaux, ils sont égaux à leur pgcd.
- sinon, le pgcd de n et m est égal au pgcd de $n - m$ et m

Par exemple, partant du rectangle 21×15 précédent, on se ramène successivement aux rectangles 6×15 puis 6×9 puis 6×3 puis 3×3.



On peut « accélérer » l'algorithme en passant non pas de n à $n - m$ mais de n à $n - qm$, où q est le quotient de la division euclidienne de n par m. Dans la version accélérée, on aurait « sauté » l'étape du rectangle 6×9.

Exercice 1 (Algorithme d'Euclide, $\star$)

1. *Écrivez une fonction* récursive `affiche_et_calcule_pgcd(n,m)` *qui affiche les étapes du calcul du pgcd et renvoie le résultat. Par exemple,*

```
print('Le pgcd de 21 et 15 est {}.'.format(affiche_et_calcule_pgcd
    (21,15)))
```

affiche

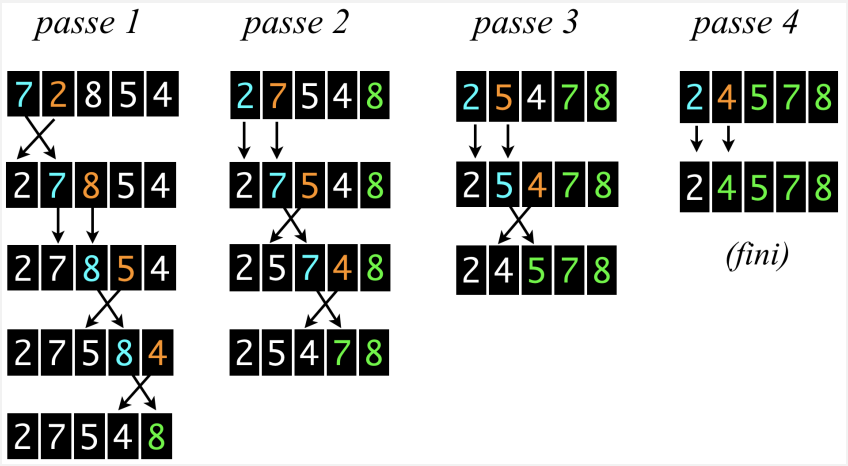
```
calcule le pgcd de n=21 et m=15
calcule le pgcd de n=6 et m=15
calcule le pgcd de n=15 et m=6
calcule le pgcd de n=9 et m=6
calcule le pgcd de n=3 et m=6
calcule le pgcd de n=6 et m=3
calcule le pgcd de n=3 et m=3
Le pgcd de 21 et 15 est 3.
```

2. *Modifez la fonction pour effectuer le calcul dans sa version accélérée*
3. *Modifiez la fonction en remplaçant les appels récursifs par une boucle* `while`

□

L'algorithme de tri à bulles

Le tri à bulles est un algorithme de tri qui repose sur l'idée que tant que deux éléments adjacents ne sont pas dans le bon ordre, on doit les échanger pour se rapprocher d'une liste triée. Il s'agit donc d'un tri *en place* (on ne crée pas une nouvelle liste), où les seuls échanges se font entre des valeurs adjacentes. Plus précisément, on effectue $n-1$ passes (où n est la longueur de la liste à trier) et dans chaque passe on parcourt les couples de valeurs adjacentes en partant du début de la liste et on les échange si elles ne sont pas dans le bon ordre.



On peut donc voir le tri à bulle comme une variante du tri par sélection, où on commence par positionner la valeur la plus grande à sa position correcte, puis on répète le procédé sur la sous-liste qui s'arrête juste avant ce dernier élément. Notons enfin que si durant une passe on n'a pas échangé deux éléments, la liste est triée, et on peut donc omettre les passes restantes.

Exercice 2 (Tri à bulles, ⋆⋆)

1. *Écrivez une fonction* `compare_et_echange(L,i)` *qui prend en argument une liste d'entiers* `L` *de longueur* $n \geq 2$ *et un entier* `i`$\leq n-2$ *et qui échange les entiers aux indices* `i` *et* `i+1` *si ils ne sont pas dans le bon ordre. Par exemple, on aura*

```
L = [3,4,5,1]
compare_et_echange(L,0)   # -> ne change rien
compare_et_echange(L,2)   # -> echange 5 et 1
```

```
# désormais L == [3,4,1,5]
```

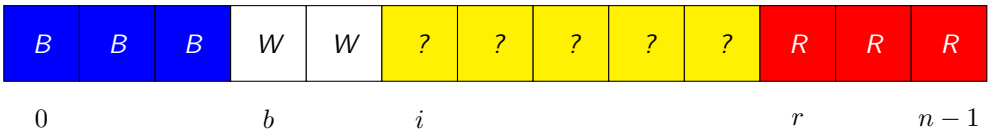
2. *Écrivez une fonction* `passe_tri_bulles(L)` *qui prend en argument une liste d'entiers* `L` *de longueur* $n \geq 2$ *et qui effectue une passe de l'algorithme du trie à bulle.*
3. *En déduire une fonction* `tri_bulles(L)` *qui trie en place la liste* `L` *par l'algorithme de tri à bulles.*
4. *Optionnel : permettez à votre fonction de terminer avant la passe* $n-1$ *si durant une passe aucun échange n'est effectué.*

□

Exercice 3 (Boucle while : le drapeau hollandais, ⋆⋆)

Le problème du drapeau hollandais est un classique des exercices de programmation promu par Edgser W. Dijkstra, un des « pères fondateurs » de l'informatique en tant que discipline scientifique, qui a laissé son nom dans des domaines aussi variés que l'algorithmique et les systèmes d'exploitations.

Le problème est le suivant : on dispose d'une liste `L` *contenant trois types de valeurs : 'B' (bleu), 'W' (blanc), et 'R'(rouge). Le but est d'écrire une fonction qui trie la liste* `L` *en place*[1] *de sorte que les couleurs apparaissent dans l'ordre du drapeau hollandais (*`'B'<'W'<'R'`*). L'algorithme consiste à diviser la liste en quatre zones consécutives, voir dessin ci-dessous, et à réduire à chaque étape la zone "jaune" des valeurs qui ne sont pas encore triées.*



Plus précisément, votre fonction utilisera en une boucle `while` *et manipulera les variables* i, b *et* r. *À chaque tour de boucle, il faudra réaliser l'une des actions ci-dessous, selon la couleur de la valeur de* `L[i]` *:*

- *si* `L[i]` *est bleu, on échange L[i] avec L[b], on avance* b *et on avance i*
- *si* `L[i]` *est rouge, on échange L[i] avec le dernier jaune, et on recule* r
- *si* `L[i]` *est blanc, on avance i*

1. *Comment sont initialisées les variables* b, i, *et* r *? Quelle est la condition de sortie de la boucle ?*
2. *Écrivez la fonction* `sort_dutch(L)` *en lui demandant d'afficher les étapes intermédiaires.*

□

Exercice 4 (Tri fusion, ⋆⋆)

L'algorithme de tri fusion est un algorithme classique de tri de liste par récurrence. Il est basé sur une approche "diviser pour régner" et de complexité asymptotique dans le pire cas $O(n \cdot \log(n))$. *L'algorithme peut être décrit ainsi :*

- *soit* L *la liste à trier.*
- *si* L *a moins de* 2 *éléments, elle est triée*
- *sinon on note* L_1 *et* L_2 *les deux demi-listes obtenues en coupant* L *en son milieu, avec* L_1 *contenant l'élément de position médiane si* L *est de longueur impaire. Par exemple, si* $L = [5,1,3,4,2]$, $L_1 = [5,1,3]$ *et* $L_2 = [4,2]$
- *on trie* L_1 *et* L_2 *récursivement (par exemple,* $L_1 = [1,3,5]$ *et* $L_2 = [2,4]$*).*
- *on fusionne* L_1 *et* L_2 *(par exemple,* $[1,2,3,4,5]$*).*

1. *Écrivez une fonction* `fusion(L1,L2)` *qui prend en arguments deux listes triées* L_1 *et* L_2 *et qui renvoie une nouvelle liste contenant les contenus de* L_1 *et* L_2 *fusionnés.*
2. *En déduire une fonction définie par récurrence* `tri_fusion(L)` *qui renvoie une nouvelle liste contenant le résultat du tri de* L.

1. On pourrait envisager un tri par comptage pour ce problème, mais ce ne serait pas un tri en place (le comptage requiert de créer une nouvelle liste). L'originalité de l'algorithme présenté dans cet exercice est de faire un tri en place de même complexité que le tri par comptage, i.e. linéaire en la taille de la liste, le nombre de valeurs différentes dans la liste étant considéré comme une constante (ici 3).

□

Exercice 5 (Exponentiation rapide, ⋆⋆⋆)
On considère la fonction suivante pour le calcul de x puissance n

```
def power(x,n) :
    acc = 1
    for i in range(n) :
        acc = acc * x
    return acc
```

1. *Combien de multiplications sont nécessaires pour calculer x^n avec cette méthode ?*
2. *Écrivez une fonction qui calcule x^n en $O(\log(n))$ multiplications.*

□