TD : Analyse d'algorithmes

Exercice 1. Réécrire la fonction de recherche simple du cours (dans une liste non triée) pour qu'elle utilise une boucle `while` et non une boucle `for` avec une instruction de sortie (`return`) dans le corps de boucle.

Exercice 2. `truc()` et `bidule()` sont deux fonctions quelconques, sans argument. Déterminer le nombre de fois qu'elles sont appelées dans les scripts ci-dessous.

```
for i in range(n):
    truc()
for i in range(n):
    bidule()
```

```
for i in range(n):
    truc()
    for j in range(n):
        bidule()
```

```
for i in range(n):
    truc()
    for j in range(i):
        bidule()
```

```
for i in range(n):
    truc()
    for j in range(i):
        bidule()
        for k in range(j):
            truc() ; bidule()
```

En supposant que les fonctions s'exécutent en $O(1)$, donner la complexité asymptotique de ces scripts.

Exercice 3. On considère la fonction suivante :

une fonction f

```
def f(n):
    assert n>=0
    p=1
    for i in range(1,n+1):
        p*=i
    return(p)
```

1. Que fait la fonction f ?
2. Prouver que le résultat est correct en exhibant un invariant de boucle.
3. Quelle est sa complexité en nombre d'opérations arithmétiques ?

Exercice 4. Cet exercice fait suite au précédent. On considère la fonction g suivante :

```
def g(n):
    assert n>=0
    s=0
    for i in range(n+1):
        s+=f(i)
    return s
```

1. Que calcule-t-elle ? Exhibez un invariant de boucle.
2. Quelle est sa complexité en nombre d'opérations arithmétiques ?
3. Proposez une idée d'amélioration.

Exercice 5. *C'est plus, c'est moins, mais dans l'autre sens*. Le but de cet exercice est d'écrire un programme qui joue au « c'est plus c'est moins » en vous posant des questions. Vous choisissez un nombre dans votre tête, supposé être entre 0 et 99. Le programme vous pose des questions du type « est-ce n ? » et vous répondez par « plus » « moins » ou « oui » s'il a gagné.

1. Décrire un algorithme bête qui fonctionnera (on ne demande pas de le coder). Si vous mentez, le programme devra s'en rendre compte. Combien pose-t-il de questions dans le pire cas ? (Le pire cas étant quand vous donnez un maximum de réponses crédibles, mais pas « oui »).
2. Proposez un algorithme inspiré de la recherche dichotomique. Combien de fois pouvez-vous vous permettre de ne pas répondre « oui » sans qu'un mensonge soit détecté ?
3. Codez effectivement l'algorithme.
4. On joue maintenant avec des nombres entre 0 et 100000. Environ combien de questions vous posera-t-il en plus ?

Exercice 6. *Plus petit entier naturel non présent dans une liste*. On s'intéresse ici au problème de déterminer le plus petit entier naturel non présent dans une liste constituée d'entiers.

1. Écrire une fonction `est_present(L,x)` renvoyant un booléen indiquant si `x` est présent dans la liste `L`.

2. En déduire une fonction `ppenp(L)` renvoyant le plus petit entier naturel non présent dans `L`. On utilisera une boucle `while`.
3. Estimer sa complexité dans le pire cas en fonction de la taille n de la liste passée en entrée.
4. Proposez une solution de complexité linéaire en la taille de la liste. *Indication : utiliser une liste de booléens de taille n ou $n+1$.*

Exercice 7. *Tri par comptage.* On s'intéresse au problème du tri d'une liste d'entiers naturels, constituée d'entiers strictement inférieurs à une borne k connue. Le tri par sélection déja vu en cours est de complexité $O(n^2)$ avec n la taille de la liste. Si k est petit, on peut donner une solution plus efficace.

1. On suppose dans cette question que $k = 2$. Écrire une fonction `tri_bits(L)` prenant en entrée une liste constituée de 0 et de 1 et renvoyant une nouvelle liste, équivalente à `L`, triée. Comme le nom de l'exercice l'indique, on procédera par comptage.
2. Quelle est la complexité en fonction de la taille n de la liste ?
3. Donner une fonction `tri_comptage(L,k)` basée sur le même principe, prenant également en entrée un entier k, la liste étant supposée contenir uniquement des entiers entre 0 et $k-1$. *Indication : utiliser une liste d'entiers de taille k.*
4. Estimer sa complexité en fonction de n et k.

Exercice 8. *Recherche de pic dans une liste.* On dit qu'un élément d'une liste est un pic s'il est supérieur à ses deux voisins (ou son unique voisin s'il est sur le bord). Par exemple, dans la liste `[10, 51, 32, 14, 78, 40, 82]`, les pics sont 51, 78 et 82. Cet exercice a pour but de trouver les indices des pics dans une liste : les indices des pics précédents sont 1, 4 et 6.

1. Écrire une fonction `indices_pics(L)` prenant en entrée une liste (qu'on pourra éventuellement supposer de taille au moins 2) et renvoyant la liste des indices de ses pics.

   Les questions qui suivent ont pour but d'écrire une fonction cherchant l'indice d'*un seul pic*, mais de manière plus efficace que la fonction précédente.
2. Soit $i \geq 1$ un indice de la liste. On suppose que `L[i]<L[i-1]`. Montrer que les pics de la liste `L[:i]` (liste constituée des éléments de `L` d'indice entre 0 et $i-1$) sont exactement ceux de `L` situés avant l'indice i exclus. Justifier que la liste `L[:i]` contient au moins un pic.
3. En vous inspirant de la question précédente et de la recherche dichotomique, écrire une fonction permettant de déterminer l'indice d'un pic de la liste `L` avec une complexité $O(\log n)$ où n est la taille de la liste. On ne fera pas d'extraction de portions de liste (coûteux en complexité).

Exercice 9. *Calcul de $\lfloor\sqrt{n}\rfloor$.* On considère la fonction suivante, prenant en entrée un entier au moins égal à 1.

```
def racine_int(n):
    i=1
    s=n
    while s-i>1:
        m=(i+s)//2
        if m*m<=n:
            i=m
        else:
            s=m
    return i
```

1. Montrer que $s-i$ est un *variant de boucle* (une quantité qui diminue strictement à chaque passage dans la boucle).
2. En déduire la terminaison de l'algorithme.
3. Montrer que $i^2 \leq n < s^2$ est un invariant de la boucle (pour $n > 1$).
4. En déduire la correction de l'algorithme.
5. Quelle est sa complexité arithmétique (nombre d'opérations effectuées) ?

Exercice 10. *Triplets Pythagoriciens.* Soit $N > 0$. On veut calculer tous les triplets d'entiers (a, b, c) tels que $1 \leq a \leq b < c \leq N$ et $a^2 + b^2 = c^2$. On ne travaillera qu'avec des entiers dans cet exercice (pas de flottants)

1. Proposer un algorithme naïf utilisant trois boucles imbriquées, renvoyant la liste des triplets convenables.
2. Quelle est sa complexité ?
3. En utilisant la fonction de l'exercice précédent, proposer une méthode de meilleure complexité.