INFO-H-100 - Programmation
TP 17 — Calcul de complexité

Ex. 1. Indiquez la complexité de chaque étape de l'algorithme suivant ainsi que la complexité de la fonction entière.

```
def sequence(x):
    x = x + 1
    x = x + 4
    x = x * 4
    x = x ** 10
    x = x // 2
    return x
```

Ex. 2. Indiquez la complexité de chaque étape de l'algorithme suivant ainsi que la complexité de la fonction entière.

```
def boucle(x):
    x = x + 1
    i = 0
    while i < x:
        i = i + 1
    x = x * x
    return x
```

Ex. 3. Indiquez la complexité de chaque étape de l'algorithme suivant ainsi que la complexité de la fonction entière.

```
def boucle(x):
    x = x + 1
    i = 1
    while i < x:
        i = 2 * i
    x = x * x
    return x
```

Ex. 4. Indiquez la complexité de chaque étape de l'algorithme suivant ainsi que la complexité de la fonction entière.

```
def nesting(n):
    sum = 0
    i = 0
    while i < n:
        j = 0
        while j < i:
            sum += 1
            j += 1
        i += 1
    return sum
```

Ex. 5. Indiquez la complexité de chaque étape de l'algorithme suivant ainsi que la complexité de la fonction entière.

```
def nesting(n):
    sum = 0
    i = 0
    j = 0
    while i < n:
        while j < i:
            sum += 1
            j += 1
        i += 1
    return sum
```

Ex. 6. Indiquez la complexité de chaque étape de l'algorithme suivant ainsi que la complexité de la fonction entière.

```python
def nesting(n):
    m = n
    sum = 0
    i = 0
    while i < n:
        j = 0
        while j < n:
            k = 0
            while k < m:
                sum += 2
                k += 1
            j += 1
        i += 1
    return sum
```

Ex. 7. Indiquez la complexité de chaque étape de l'algorithme de recherche suivant ainsi que la complexité de la fonction entière.

```python
def recherche(s,x):
    """ Desc : Donne l'indice ou se trouve la valeur de x dans s
    INPUT : s (list), x(int)
    OUTPUT : (int) indice de position de l'element (-1 si l'element n'existe pas)
    """
    i = 0
    while i < len(s) and s[i] != x:
        i = i+1
    if i == len(s):
        i = -1
    return i
```

Ex. 8. Indiquez la complexité de chaque étape de l'algorithme "Deep Copy" suivant ainsi que la complexité de la fonction entière.

```python
def deep_copy(li):
    new = []
    for e in li:
        if type(e) == list:
            new.append(deep_copy(e))
        else:
            new.append(e)
    return new
```

Ex. 9. Indiquez la complexité de chaque étape de l'algorithme de tri par sélection suivant ainsi que la complexité de la fonction entière.

```python
def tri_selection(s):
    """ Desc : Trie la liste s par selection
    INPUT : s (list)
    OUTPUT : None
    """
    n = len(s)
    for i in range(n-1):
        min = i
        for j in range(i+1,n):
          if  s[j] < s[min]:
              min = j
        s[min],s[i] =  s[i],s[min]
```

Ex. 10. Indiquez la complexité de chaque étape de l'algorithme de tri par insertion suivant ainsi que la complexité de la fonction entière. Que vaut la complexité de cette algorithme dans le cas ou la liste est déjà triée ?

```python
def tri_insertion(s):
    """ Desc : Trie la liste s par insertion
    INPUT : s (list)
    OUTPUT : None
    """
    n = len(s)
    for i in range(1,n):
        Save = s[i]
        j = i-1
        while j>=0 and s[j] > Save:
```

```
            s[j+1] = s[j]
            j=j-1
        s[j+1] = Save
```

Ex. 11. Indiquez la complexité de chaque étape de l'algorithme "equality_list" suivant ainsi que la complexité de la fonction entière.

```
def equality_list(a,b):
    res = False
    if len(a)==len(b):
        res = True
        for i in a:
            if i in b:
                a.remove(i)
                b.remove(i)
            else:
                res = False
    return res
```

Ex. 12. Indiquez la complexité de chaque étape de l'algorithme "f1" suivant ainsi que la complexité de la fonction entière. Expliquez aussi ce que fait cet algorithme.

```
from math import sqrt
def f1(nmax):
    nums = list(range(2,nmax+1))
    for i in range(2, int(sqrt(nmax))+1):
        f = lambda x: x == i or x % i != 0
        nums = list(filter(f, nums))
    return nums
```

Ex. 13. Indiquez la complexité de chaque étape de l'algorithme "f2" suivant ainsi que la complexité de la fonction entière. Expliquez aussi ce que fait cet algorithme.

```
def f2(a):
    a= a.split()
    b= {}
    num =0
    for i in a:
        b[i] = b.get(i,0) + 1
        if b[i] == 1:
            num += 1
    return num
```

INFO-H-100 - Programmation
TP 17 — Calcul de complexité
Corrections

Solution de l'exercice 1:

Indiquez la complexité de chaque étape de l'algorithme suivant ainsi que la complexité de la fonction entière.

```
def sequence(x):   # O(1)
    x = x + 1      # O(1)
    x = x + 4      # O(1)
    x = x * 4      # O(1)
    x = x ** 10    # O(log(10)) = O(1)
    x = x // 2     # O(1)
    return x       # O(1)
```

Solution de l'exercice 2:

Indiquez la complexité de chaque étape de l'algorithme suivant ainsi que la complexité de la fonction entière.

```
def boucle(x):     # O(x)
    x = x + 1      # O(1)
    i = 0          # O(1)
    while i < x:   # O(x)
        i = i + 1  # O(1)
    x = x * x      # O(1)
    return x       # O(1)
```

Solution de l'exercice 3:

Indiquez la complexité de chaque étape de l'algorithme suivant ainsi que la complexité de la fonction entière.

```
def boucle(x):     # O(x)
    x = x + 1      # O(1)
    i = 1          # O(1)
    while i < x:   # O(log(x))
        i = 2 * i  # O(1)
    x = x * x      # O(1)
    return x       # O(1)
```

Solution de l'exercice 4:

Indiquez la complexité de chaque étape de l'algorithme suivant ainsi que la complexité de la fonction entière.

```
def nesting(n):       # O(n^2)
    sum = 0           # O(1)
    i = 0             # O(1)
    while i < n:      # O(n)
        j = 0         # O(1)
        while j < i:  # O(n)
            sum += 1  # O(1)
            j += 1    # O(1)
        i += 1        # O(1)
    return sum        # O(1)
```

Solution de l'exercice 5:

Indiquez la complexité de chaque étape de l'algorithme suivant ainsi que la complexité de la fonction entière.

```
def nesting(n):          # O(n)
    sum = 0              # O(1)
    i = 0                # O(1)
    j = 0                # O(1)
    while i < n:         # O(n)
        while j < i:     # O(1)
            sum += 1     # O(1)
            j += 1       # O(1)
        i += 1           # O(1)
    return sum           # O(1)
```

Solution de l'exercice 6:

Indiquez la complexité de chaque étape de l'algorithme suivant ainsi que la complexité de la fonction entière.

```
def nesting(n):          # O(n^2.m) = O(n^3)
    m = n                # O(1)
    sum = 0              # O(1)
    i = 0                # O(1)
    while i < n:         # O(n)
        j = 0            # O(1)
        while j < n:     # O(n)
            k = 0        # O(1)
            while k < m: # O(m) = O(n)
                sum += 2 # O(1)
                k += 1   # O(1)
            j += 1       # O(1)
        i += 1           # O(1)
    return sum           # O(1)
```

Solution de l'exercice 7:

```
def recherche(s,x):                    # O(n)
    i = 0                              # O(1)
    while i < len(s) and s[i] != x:    # O(n)
        i = i+1                        # O(1)
    if i == len(s):                    # O(1)
        i = -1                         # O(1)
    return i                           # O(1)
```

Solution de l'exercice 9:

```
def tri_selection(s):                    # O(n^2)
   n = len(s)                            # Assignation en O(1), len(s) en O(1)
   for i in range(n-1):                  # O(n)
      min = i                            # O(1)
      for j in range(i+1,n):             # O(n-i)
        if  s[j] < s[min]:               # O(1)
           min = j                       # O(1)
      s[min],s[i] =  s[i],s[min]         # Deux assignations en O(1) chacune
```

Solution de l'exercice 10:

Cas général : $O(n^2)$ Si la liste est déjà triée : $O(n)$

```
def tri_insertion(s):                   # O(n^2)
   n = len(s)                            # O(1)
   for i in range(1,n):                  # O(n)
      Save = s[i]                        # O(1)
      j = i-1                            # O(1)
      while j>=0 and s[j] > Save:        # O(n-i)
         s[j+1] = s[j]                   # O(1)
         j=j-1                           # O(1)
      s[j+1] = Save                      # O(1)
```

Solution de l'exercice 12:

Cette fonction renvoie le liste de tous les nombres premiers inférieurs ou égal à nmax. La complexité totale est en $O(\sqrt{nmax} * len(nums))$ soit en $O(nmax^{3/2})$ car len(nums) <= nmax.

```
def f1(nmax):                                  # O(sqrt(nmax)*len(nums))
    nums = list(range(2,nmax+1))               # O(nmax)
    for i in range(2, int(sqrt(nmax))+1):      # O(sqrt(nmax)) * O(len(nums))
        f = lambda x: x == i or x % i != 0     # O(1)
        nums = list(filter(f, nums))           # O(len(nums))
    return nums                                # O(1)
```

Solution de l'exercice 13:

Renvoie le nombre de "mots" différents d'un texte. Au pire cas, tous les mots sont différents et on obtient $O((\text{nombre de mots différents dans a})^2)$.

```
def f2(a):
   a = a.split()                  # O(len(a))
   b = {}                         # O(1)
   num = 0                        # O(1)
   for i in a:                    # O(len(a)) * O(len(b))
      b[i] = b.get(i,0) + 1       # O(len(b))
      if b[i] == 1:               # O(len(b))
         num += 1                 # O(1)
   return num                     # O(1)
```