TP n°2 d'informatique

```
for i in range(5):
  for j in range(2):
      print(i, j)
```

Renvoie :

```
0 0
0 1
1 0
1 1
2 0
2 1
3 0
3 1
4 0
4 1
```

Ici, `print` est appelé 4×3 fois, la complexité est de $m \times n$

```
def vandermonde(n, p):
    """Crée une matrice de vandermonde"""
    M = []
    for i in range(n):
        L = []
        for j in range(p):
            L.append(i**j)
        M.append(L)
    return M

print(vandermonde(3, 4))
```

Renvoie :

```
[[1, 0, 0, 0], [1, 1, 1, 1], [1, 2, 4, 8]]
```

```python
def tableau_triangulaire(n):
    """crée le tableau triangulaire de taille n avec m[i][j]=(i+1)*(j+1)"""
    M = []
    for i in range(n):
        L = []
        for j in range(i):  # décrit l'interval d'entier [[0, i-1]]
            L.append((i+1)**(j+1))
        M.append(L)
    return M

print(tableau_triangulaire(8))
```

Renvoie :

```
[[], [2], [3, 9], [4, 16, 64], [5, 25, 125, 625], [6, 36, 216, 1296, 7776],
[7, 49, 343, 2401, 16807, 117649], [8, 64, 512, 4096, 32768, 262144, 2097152]]
```

La complexité de cet algorithme est de $\frac{n(n+1)}{2}$ (triangle) $= n^2$

```python
def plus_proche(L):
    """détermine les deux valeurs les plus proches dans une liste"""
    n = len(L)
    ind = [0, 1]
    d = abs(L[0]-L[1])

    for i in range(n-1):
        for j in range(i+1, n):
            a = abs(L[j]-L[i])
            if a < d:
                d = a
                ind = [i, j]
    return ind, d

print(plus_proche([0,3,5,4,3,4]))
```

Renvoie :

```
([1, 4], 0)
```

La complexité de cet algorithme est de n^2 parce que de $\frac{n(n-1)}{2}$

```python
def recherche(texte, motif):
    n, p = len(texte), len(motif)
    if n < p:
        return False  # Logique
    for i in range(n-p+1):
```

```
        if texte[i] == motif[0]:
            j = 1
            while j < p and texte[i+j] == motif[j]:
                j += 1
            if j == p:
                return True
    return False

print(recherche("The quick brown fox jumps over the lazy dog", "og"))
```

Optimisation, si le mot est dans le texte, on peut vérifier tout sauf le mot +1 caractère (l'algorithme étendra la recherche dès que le mot commencera à être trouvé)

```
L = [3, 1, 4, 1, 5, 9, 2, 6, 5]
def tri_selection(L):
    n = len(L)
    for i in range(n):
        indmin = i
        mini = L[i]
        for j in range(i+1, n):
            if L[j] < mini:
                indmin = j
                mini = L[j]
        L[i], L[indmin] = L[indmin], L[i]

tri_selection(L)
print(L)
```

Renvoie :

```
[1, 1, 2, 3, 4, 5, 5, 6, 9]
```

```
L = [3, 1, 4, 1, 5, 9, 2, 6, 5]
def tri_selection(L):
    """Tri un peu comme les jeux de carte"""
    for i in range(len(L)):
        indice_du_minimum = i
        minimum = L[i]
        for j in range(i+1, len(L)):
            if L[j] < minimum:
                indice_du_minimum = j
                minimum = L[j]
        L[i], L[indice_du_minimum] = L[indice_du_minimum], L[i]
tri_selection(L)
print(L)
```

Renvoie :

```
[1, 1, 2, 3, 4, 5, 5, 6, 9]
```

```
def tri_à_bulle(L):
    """Tri à bulle"""
    for i in range(len(L)):
```

Voir le PDF
