Proposition de corrigé du TP 3

Recherche dans un tableau

```python
L=[4,7,8,7,8,9,9,7,2,0,5]

def distance_min(L):   # On cherche min |Ti-tj| pour i<j
    n=len(L)
    min=abs(L[1]-L[0])
    for i in range(n):
        for j in range(i+1,n):
            if abs(L[j]-L[i])<min:
                min=abs(L[j]-L[i])
    return(min)

def indices_distance_min2(L):   # On cherche min |Ti-tj| pour i<j
    n=len(L)
    min=abs(L[1]-L[0])
    p,q=0,1
    for i in range(n):
        for j in range(i+1,n):
            if abs(L[j]-L[i])<min:
                min=abs(L[j]-L[i])
                p,q=i,j
    return p,q

def indices_distance_min3(L):
    D={}
    n=len(L)
    for i in range(n-1):
        m=abs(L[i+1]-L[i])
        j=i+1
        for k in range(i+1,n):
            if abs(L[k]-L[i])<m:
                m=abs(L[k]-L[i])
                j=k
        D[str(i)]=[j,m]
    p,q=0,D["0"][0]
    min=D["0"][1]
    for elt in D.items():
    # elt de la forme ["3",[4,min {|Lk-L3|, k>4}]
        if elt[1][1]<min:
            p,q=int(elt[0]),elt[1][0]
            min=elt[1][1]
    return(p,q)
```

On obtient pour chaque i compris entre 0 et $n-1$, $n-i-1$ comparaisons.
Au total, on en a $n**2-n(n-1)/2-n$ lequel est equivalent a $n**2/2$.
D'où une complexité quadratique.

Recherche d'un mot dans un texte

```
def est_ici(texte, motif, i):
    p=len(motif)
    j=0
    while j<=p-1 and motif[j]==texte[i+j]:
            j=j+1
    return(j==p)

def est_ici2(texte, motif, i):
    p=len(motif)
    j=0
    test=True
    while j<=p-1 and test:
        test=(motif[j]==texte[i+j])
        j=j+1
    return(test)

def est_sous_mot(texte, motif):
    n,p=len(texte), len(motif)
    i=0
    while i<=n-p and not est_ici(texte, motif, i):
        i=i+1
    return(i<=n-p)

def position_sous_mot(texte, motif):
    n,p=len(texte), len(motif)
    L=[]
    for i in range(n-p+1):
        if est_ici(texte, motif, i):
            L.append(i)
    return(L)
```

Tri à bulles

```python
def est_trie(T):
    """Teste si T est tri  ou pas"""
    test=True
    n=len(T)
    i=0
    while i<=n-2 and test:
        test=(T[i]<=T[i+1])
        i=i+1
    return(test)

def TriBulles(T):
    """tri du tableau T avec le tri   bulles"""
    n=len(T)
    for i in range(1,n):    # num ro du parcours
        for k in range(n-i):
            if T[k]>T[k+1]:
                T[k],T[k+1]=T[k+1],T[k]
        print(T)
    return(T)

def TriBulles2(T):
    """tri du tableau T avec le tri   bulles"""
    n=len(T)
    i=1
    echange=True
    while i<=n-1 and echange:
        echange=False
        for k in range(n-i):
            if T[k]>T[k+1]:
                T[k],T[k+1]=T[k+1],T[k]
                echange=True
        i+=1
        print(T)
    return(T)


import random as r

n=15
T=[r.randint(0,100) for i in range(n)]
```

La première boucle k exécute $n-1$ opérations. Pour chaque k, il y a $n-k$ boucles i et une comparaison pour chacune. Au total, il y a une complexite de $\sum_{k=1}^{n-1}\sum_{i=0}^{n-k} 1 = n(n-1)/2 = O(n**2)$: On obtient une complexité quadratique.

```
def TriCocktail(T):
    n=len(T)
    for k in range(1,(n-1)//2):
        for i in range(k-1,n-k):   # Parcours des cases k-1     n-k-1
            if T[i]>T[i+1]:
                T[i],T[i+1]=T[i+1],T[i]
        print(T)
        for i in range(n-k-1,k-2,-1):    # Parcours des cases n-k-1     k-1
            if T[i]>T[i+1]:
                T[i],T[i+1]=T[i+1],T[i]
        print(T)
    return(T)
```

Cette fonction est a privilégier lorsque le tableau contient des éléments minima en fin de tableau, notamment lorsque le tableau est décroissant par exemple.
Néanmoins, dans le pire des cas, le nombre de comparaison est la somme sur k variant de 1 à $(n-1)//2$ de $(n-k-1)-(k-1)+1+(n-k-1)-(k-1)+1 = 2(n-2k+1)$. Cela donne encore une complexité quadratique car équivalent à un terme de la forme $a.n**2$.