TP Algorithmes de tri corrigé

[4]:
```python
# 1.
def croissant(L):
    for i in range(len(L)-1):
        if L[i] > L[i+1]:
            return False
    return True

croissant([1,2,3,4,5]) and not croissant([1,2,4,3,5]) # test
```

[4]: True

[10]:
```python
# 2.
def appartient(e, L, i):
    if i == len(L):
        return False
    return L[i] == e or appartient(e, L, i+1)

appartient(1, [1, 2, 3], 0) and not appartient(1, [1, 2, 3], 1) # test
```

[10]: True

[11]:
```python
# 3.
def doublon(L):
    for i in range(len(L)):
        if appartient(L[i], L, i+1):
            return True
    return False

not doublon([1, 2, 3, 4, 5]) and doublon([1, 2, 3, 4, 4]) # test
```

[11]: True

[4]:
```python
# 4. Voir cours : complexité O(nlog(n))
def fusion(L1, L2, L):
    i1, i2 = 0, 0
    while i1 + i2 < len(L):
        if i1 >= len(L1):
            L[i1 + i2] = L2[i2]
            i2 = i2 + 1
        elif i2 >= len(L2):
            L[i1 + i2] = L1[i1]
            i1 = i1 + 1
        elif L[i1] < L[i2]:
            L[i1 + i2] = L1[i1]
            i1 = i1 + 1
        else:
            L[i1 + i2] = L2[i2]
            i2 = i2 + 1
```

```
def fusion(L1, L2):
    if len(L1) == 0: return L2
    if len(L2) == 0: return L1
    if L1[-1] > L2[-1]: m = L1.pop()
    else: m = L2.pop()
    L = fusion(L1, L2)
    L.append(m)
    return L

def tri_fusion(L):
    if len(L) <= 1: return L
    L1, L2 = L[: len(L)//2], L[len(L)//2 :]
    return fusion(tri_fusion(L1), tri_fusion(L2))

L = [5, 1, 3, 8, 2, 4, 9, 7, 6]
tri_fusion(L) == [1, 2, 3, 4, 5, 6, 7, 8, 9] # test
```

[4]: True

[3]:
```
# 5.
def doublon_triee(L):
    for i in range(len(L) - 1):
        if L[i] == L[i+1]:  # O(len(L))
            return True
    return False

not doublon_triee([1, 2, 3, 4, 5]) and doublon_triee([1, 2, 3, 4, 4]) # test
```

[3]: True

[5]:
```
# 6.
def doublon2(L):
    return doublon_triee(tri_fusion(L))

not doublon2([1, 2, 3, 4, 5]) and doublon2([1, 2, 3, 4, 4]) # test
```

[5]: True

Comparons le temps d'exécution de doublon et doublon2 :

[20]:
```
%%timeit
import sys
sys.setrecursionlimit(10000)
L = list(range(5000))
doublon(L)
```

2.24 s ± 76.1 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

[21]:
```
%%timeit
L = list(range(5000))
doublon2(L)
```

10.2 ms ± 274 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)