Exercice 4

1)
a) O(nlog(n))
b) Tri selection/insertion/bulle. Cout $O(n^2)$

2)
[7, 4, 2, 1, 8, 5, 6, 3]
[7, 4, 2, 1]
[7, 4]
[2, 1]
[8, 5, 6, 3]
[8, 5]
[6, 3]

3) 4)
```
def moitie_droite(L):
    return L[len(L)//2:]

def moitie_gauche(L):
    return L[:len(L)//2]

def fusion(L1, L2):
    L = []
    n1 = len(L1)
    n2 = len(L2)
    i1 = 0
    i2 = 0
    while i1 < n1 or i2 < n2:
        if i1 >= n1:
            L.append(L2[i2])
            i2 = i2 + 1
        elif i2 >= n2:
            L.append(L1[i1])
            i1 = i1 + 1
        else:
            e1 = L1[i1]
            e2 = L2[i2]
            if e1 < e2:
                L.append(e1)
                i1 = i1 + 1
            else:
                L.append(e2)
                i2 = i2 + 1
    return L

def tri_fusion(L):
    n = len(L)
    if n <= 1:
        return L
    print(L)
    mg = moitie_gauche(L)
    md = moitie_droite(L)
    L1 = tri_fusion(mg)
    L2 = tri_fusion(md)
    return fusion(L1, L2)

tri_fusion([7, 4, 2, 1, 8, 5, 6, 3])
```

Exercice 1 :

1)

```
def plus_proche_voisin(t, cible):
    dmin = distance(t[0], cible)
    idx_ppv = 0
    n = len(t)
    for idx in range(1, n) :
        if distance(t[idx]) < dmin :
            dmin = distance(t[idx])
            idx_ppv = idx
    return idx_ppv
```

2) O(n)

3)

a) On garde le résultat dans une variable et on réutilise cette variable. Exemple : d = distance(obj, cible)

b) Cela permet de savoir que le plus proche voisin ayant la plus grande distance est toujours en première position de la liste et de ne pas à avoir à le chercher dans la liste à chaque fois.

c)

```
def insertion(kppv, idx, d) :
    for i in range(len(kppv)):
        if kppv[i][1] < d:
            kppv.insert(i, (idx, d))
            return
    kppv.append( (idx, d) )
```