Exercice 1. Tri par sélection

Question 1 –

```
def mini(L):
    assert L != []
    i_min = 0
    for i in range(1,len(L)):
        if L[i] < L[i_min]:
            i_min = i
    return i_min
```

```
def supprimer(L, i):
    assert 0 <= i and i < len(L)
    return L[:i] + L[i+1:]
```

```
def tri_selection(L):
    T = []
    for _ in range(len(L)):
        i = mini(L)
        T.append(L[i])
        L = supprimer(L, i)
    return T
```

Question 2 – Le temps d'exécution des fonctions `mini` et `supprimer` est en $\Theta(\texttt{len(L)})$. Donc la complexité de la fonction `tri_selection` est en $\Theta(\texttt{len(L)}^2)$.

Question 3 – Ce n'est pas un tri en place car une nouvelle liste est créée.

C'est bien un tri stable car dans la fonction `mini`, si le plus petit élément apparaît plusieurs fois, alors on renvoie le plus petit indice possible.

Exercice 2. Tri par insertion

Question 1 –

```
# Hypothèse: T est triée
def insertion(T, x):
    i = 0
    while i < len(T) and T[i] < x:
        i += 1
    return T[:i] + [x] + T[i:]
```

```
def tri_insertion(L):
    T = []
    for e in L:
        T = insertion(T, e)
    return T
```

Question 2 – La fonction `insertion` est en complexité $\Theta(\texttt{len(T)})$. La fonction `tri_insertion` est en complexité $\Theta(\texttt{len(L)}^2)$.

Question 3 – Ce n'est pas un tri en place car une nouvelle liste est créée.

Ce n'est pas un tri stable, par exemple pour `[1, 1.0]` on obtient `[1.0, 1]`. En effet dans la boucle `for` de la fonction `insertion`, si `x == T[i]` alors on renvoie `L + [x] + T[i:]` alors que `T[i]` est placé avant `x` dans `L`. Pour obtenir un tri stable, il faut changer la condition « `T[i] < x` » par « `T[i] <= x` ».

Question 4 – La recherche dichotomique dans une liste triée `T` est en complexité $\mathcal{O}$`(log(len(T)))`. Après avoir obtenu l'indice d'insertion, il faut insérer l'élément dans `T`, ce qui nécessite de faire une copie de la liste, soit une complexité en Θ`(len(T))`. Finalement, même avec une recherche dichotomique, le temps d'exécution est en Θ`(len(L)`^{2}`)`.

Exercice 3. Tri rapide

Question 1 –

```
# L[0] est le pivot
# Hypothèse: L est non vide
def separer(L):
    L1 = []; L2 = []
    for i in range(1,len(L)):
        if L[i] <= L[0]:
            L1.append(L[i])
        else:
            L2.append(L[i])
    return L1, L2
```

```
def tri_rapide(L):
    if L == []:
        return []
    else:
        L1, L2 = separer(L)
        T1 = tri_rapide(L1)
        T2 = tri_rapide(L2)
        return T1 + [L[0]] + T2
```

Question 2 – Dans le pire cas, à chaque appel récursif l'une des deux sous-listes est vide. Dans ce cas, le nombre d'appels récursifs est en Θ`(len(L))`. Comme la fonction `separer` est en complexité Θ`(len(L))`, la fonction `tri_rapide` est en complexité Θ`(len(L)`^{2}`)` dans le pire cas.

Question 3 – Ce n'est pas un tri en place car une nouvelle liste est créée.

Ce n'est pas un tri stable, par exemple pour `[1, 1.0]` on obtient `[1.0, 1]`. En effet, les éléments égaux au pivot (qui est `L[0]`) sont stockés dans la liste `L1` et apparaissent donc avant `L[0]` dans `T`.

Exercice 4. Tri par comptage

Question 1 –

```
# Hypothèse: L est non vide
def maxi(L):
    m = L[0]
    for i in range(len(L)):
        if L[i] > m:
            m = L[i]
    return m
```

```
def comptage(L):
    if L == []:
        return []
    M = [0 for _ in range(maxi(L)+1)]
    for e in L:
        M[e] += 1
    return M
```

```
def tri_comptage(L):
    T = []
    M = comptage(L)
    for i in range(len(M)):
        for _ in range(M[i]):
            T.append(i)
    return T
```

Question 2 –

```
# Hypothèse: L est non vide
def mini_bis(L):
    m = L[0]
    for i in range(len(L)):
        if L[i] < m:
            m = L[i]
    return m
```

```
def comptage_bis(L):
    if L == []:
        return [], []
    M_pos = [0 for _ in range(maxi(L)+1)]
    M_neg = [0 for _ in range(-mini_bis(L)+1)]
    for e in L:
        if e >= 0:
            M_pos[e] += 1
        else:
            M_neg[-e] += 1
    return M_pos, M_neg
```

```
def tri_comptage_bis(L):
    T = []
    M_pos, M_neg = comptage_bis(L)
    for i in range(len(M_neg)-1, -1, -1):
        for _ in range(M_neg[i]):
            T.append(-i)
    for i in range(len(M_pos)):
        for _ in range(M_pos[i]):
            T.append(i)
    return T
```

Exercice 5. Tri à bulles

Question 1 –

```
def etape_tri_bulles(L, i_max):
    for i in range(i_max+1):
        if L[i] > L[i+1]:
            tmp = L[i]
            L[i] = L[i+1]
            L[i+1] = tmp
```

```
def tri_bulles(L):
    for i_max in range(len(L)-2, -1, -1):
        etape_tri_bulles(L, i_max)
```

Question 2 – La fonction `etape_tri_bulles` est en complexité Θ(`i_max`). Donc la fonction `tri_bulles` est en complexité $\Theta(\texttt{len(L)}^2)$.

Question 3 – C'est bien un tri en place, puisque la quantité de mémoire utilisée est indépendante de la taille de la liste en entrée.

C'est un tri stable. Le point important est que dans la fonction `etape_tri_bulles` deux éléments égaux ne sont pas échangés. Ainsi, le tri ne serait pas en place si on remplaçait la condition « `L[i] > L[i+1]` » par « `L[i] >= L[i+1]` ».

Exercice 6. Le tri du singe

Question 1 –

```
def est_triee(L):
    for i in range(len(L)-1):
        if L[i] > L[i+1]:
            return False
    return True
```

```
# Cette fonction renvoie l'indice du a ème None dans S.
# Hypothèse: cet indice existe.
def a_eme_None(a,S):
    nb_None_vus = 0
    i = 0
    while nb_None_vus < a:
        if S[i] is None:     ## Ou (if S[i] == None:)
            nb_None_vus += 1
        i = i+1
    return i-1
```

```
def perm_alea(n):
    S = [None]*n
    for i in range(n):
        A = n-i
        a = random.randint(1,A)
        S[a_eme_None(a,S)] = i
    return S
```

```
def appliquer_perm(L,S):
    M = [None]*len(S)
    for i in range(len(S)):
        M[i] = L[S[i]]
    return M
```

```
def tri_singe(L):
    N = L[:]
    n = len(L)
    while not est_triee(N):
        S = perm_alea(n)
        N = appliquer_perm(N,S)
    return N
```

Question 2 – Ce n'est pas un tri en place car une nouvelle liste est créée.
Ce n'est pas un tri stable, puisque deux éléments égaux peuvent être échangés.