Exercice 1 :

1/ (1, 5) (2, 9, 4) (3, 7, 10, 6) (8)

2/ (2, 9, 4) → ordre = 3

3/ 8

4/ σ = (1,5) ∘ (2, 9, 4) ∘ (3, 7, 10, 6)

5/ σ^{-1} = (1, 5) ∘ (2, 9, 4) ∘ (3, 6, 10, 7)

(a,b,c,d) = (a,d) ∘ (a,c) ∘ (a,b) = (b,a) ∘ (b,d) ∘ (b,c) = (a,b) ∘ (b,c) ∘ (c,d)
(On fait des circuits pour tourner en rond)

a → b	a → c	a → d	Premier cycle
b → a	b → b	b → b	
c → c	c → a	c → c	
d → d	d → d	d → d	

6/

σ =	(1,5) ∘	(2,4) ∘ (4,9) ∘	(3,6) ∘ (3,10) ∘ (3,7)
		(9,2) ∘ (2,4) ∘	(6,10) ∘ (6,7) ∘ (6,3)
		(4,9) ∘ (9,4) ∘	(10,7) ∘ (10,3) ∘ (10,6)
			(7,3) ∘ (7,6) ∘ (7,10)
σ =	(1,5) ∘	(4,2) ∘ (2,9) ∘	(7,10) ∘ (10,6) ∘ (6,3)
		(2,9) ∘ (9,4) ∘	(3,7) ∘ (7,10) ∘ (10,6)
		(9,4) ∘ (4,2) ∘	(10,6) ∘ (6,3) ∘ (3,7)
			(6,3) ∘ (3,7) ∘ (7,10)

7/
(1,4) (1,5) (1,6) (1,9)
(2,3) (2,4) (2,5) (2,6) (2,8) (2,9) (2,10)
(3,4) (3,5) (3,6) (3,9) (3,10)
(4,5)
(7,8) (7,9) (7,10)
(8, 9) (8,10)

8/
Il est ouf le prof faut pas chercher à comprendre !!! La réponse est n !

9/ Ordre inférieur ou égal à 12
σ^{12} = id

Exercice 2 :

```
def inversions(T):
    if len(T)<2 : return (T,0)
    gauche, inv_gauche = inversions(T[:len(T)//2])
    droite, inv_droite = inversions(T[len(T)//2:])
    liste,nb = fusion(gauche, droite)
    return liste, nb + inv_gauche +inv_droite
```

```
def fusion(gauche, droite):
    if len(gauche)==0 : return (droite,0)
    if len(droite)==0 : return (gauche,0)
    if gauche[0] < droite[0] :
        liste, nb = fusion(gauche[1:], droite)
        return [gauche[0]] + liste, nb
    else :
        liste, nb = fusion(gauche, droite[1:])
        return [droite[0]] + liste, nb +len(gauche)
```

Exercice 3 :

1/
CI : Avant le premier tour de boucle T[:1] = T[0] est trié et contient le même élément qu'initialement, T[:1] est trié.
On veut montrer que la propagation de l'invariant : s'il et vrai à la fin du tour i, il l'est à la fin du tour i+1.
Soit i >= 0, on suppose l'invariant vérifié pour i
En entrant dans le tour i+1, T[:i+1]
On veut montrer qu'à la fin de la boucle sur j, T[i+1] a été correctement inséré dans T[:i+1]
Lemme : A chaque itération sur j, $T[j] = T^0[i+1]$, $T[j-1] = T^0[j-1]$
Vrai quand j = i, et par propagation ensuite
La boucle sur j s'arrête lorsque $T[j-1] < T[j] = T^0[i+1]$ donc pour j-1 = k, ie lorsque $T^0[i+1]$ est bien placé.

2/
1 échange = 3 affectations
Tri(σ) fait Inv(σ) échanges donc 3 Inv(σ) affectations
→ On peut se contenter de

$$\sum_{i=1}^{len(t-1)} Inv(\sigma,i)+2 \quad \delta[\sigma(i)<\sigma(j)]\exists j<i$$

$\sum = Inv(\sigma) \rightarrow \; Inv(\sigma) \in \Theta(n^2) \qquad Inv(\sigma)+2n$
$2 \# \{ i \mid \exists j < i \;\; \sigma(j) > \sigma(i)\} \leq 2len(\sigma)$
Pour chaque élément on fait : Inv(σ, i) + 1 comparaisons
(+ 1 sauf si $\forall j < i \; \sigma(j) > \sigma(i)$)
$Inv(\sigma, i) \rightarrow \#\{ j < i \mid \sigma(j) > \sigma(i) \}$
$\sum = Inv(\sigma) + n$
Donc avec cette amélioration on gagne un facteur sur la complexité, mais elle reste $\Theta(n^2)$ en moyenne.

Exercice 4 :

```
def unionRec(E,F):
    if len(E) == 0 : return F[:]
    if len(F) == 0 : return E[:]
    if E[0] == F[0] : return [E[0]] +  unionRec(E[1:], F[1:])
    if E[0] < F[0] : return  [E[0]] +  unionRec(E[1:], F)
    else : return [F[0]] +  unionRec(E, F[1:])

def interRec(E, F):
```

```
    if len(E) == 0 : return []
    if len(F) == 0 : return []
    if E[0] == F[0] : return [E[0]] +  interRec(E[1:], F[1:])
    if E[0] < F[0] : return  interRec(E[1:], F)
    else : return interRec(E, F[1:])

def difRec(E,F):
    if len(E) == 0 : return []
    if len(F) == 0 : return []
    if E[0] == F[0] : return difRec(E[1:], F[1:])
    if E[0] < F[0] : return  [E[0]] +  difRec(E[1:], F)
    else : return difRec(E, F[1:])

def difSymRec(E,F) :
    if len(E) == 0 : return F[:]
    if len(F) == 0 : return E[:]
    if E[0] == F[0] : return difSymRec(E[1:], F[1:])
    if E[0] < F[0] : return  [E[0]] +  difSymRec(E[1:], F)
    else : return [F[0]] +  difSymRec(E, F[1:])
```

Complexité : m = len(E), n = len(F), le nombre d'appels récursifs est borné par m+n