# Corrigé du DS1

## Sur les permutations

### Partie A – Ordre d'une permutation

**A.1**

```
# let composer t t' = let n = vect_length t in
        let res = make_vect n 0 in
                for i = 0 to n - 1 do res.(i) <- t.(t'.(i)) done;
        res;;
composer : int vect -> int vect -> int vect = <fun>

# composer [|0; 1; 2; 4; 5; 3|] [|1; 3; 2; 5; 4; 0|];;
- : int vect = [|1; 4; 2; 3; 5; 0|]
```

**A.2**

```
# let inverser t = let n = vect_length t in
        let inv = make_vect n 0 in
                for i = 0 to n - 1 do inv.(t.(i)) <- i done;
        inv;;
inverser : int vect -> int vect = <fun>

#inverser [|0; 2; 1; 4; 5; 3|];;
- : int vect = [|0; 2; 1; 5; 3; 4|]
```

**A.3** $\mathrm{Id}_{E_n}$ est (la seule permutation de $E_n$) d'ordre 1.
$\varphi : x \in E_n \mapsto (x+1)[n]$ est d'ordre $n$.

**A.4** On définit d'abord une fonction qui construit $\mathrm{Id}_{E_n}$ (on pouvait faire sans).

```
# let id n = let res = make_vect n 0 in
        for i = 1 to n - 1 do res.(i) <- i done;
        res;;
id : int -> int vect = <fun>

# let rec ordre_temp = fun
        | n t accu when accu = (id (vect_length t)) -> n
        | n t accu -> ordre_temp (n + 1) t (composer t accu);;
ordre_temp : int -> int vect -> int vect -> int = <fun>

# let ordre t = ordre_temp 1 t t;;
ordre : int vect -> int = <fun>
```

### Partie B – Manipuler les permutations

**B.1**

```
# let periode t i = let temp = ref t.(i) and res = ref 1 in
        while !temp <> i
                do temp := t.(!temp); res := !res + 1 done;
        !res;;
periode : int vect -> int -> int = <fun>
```

**B.2**

```
# let estDansOrbite t i j =
        let k = periode t i and temp = ref i and l = ref 0 and res = ref false in
                while !l < k && not !res
                        do temp := t.(!temp); res := !temp = j; l := !l + 1 done;
        !res;;
estDansOrbite : int vect -> int -> int -> bool = <fun>
```

**B.3** On compte le nombre de points de $E_n$ non fixés par $t$, et on teste si ce nombre vaut 2.

```
#let estTransposition t = let n = vect_length t and infixes = ref 0 in
        for i = 0 to n - 1 do
                if t.(i) <> i then infixes := !infixes + 1
        done;
        !infixes = 2;;
estTransposition : int vect -> bool = <fun>estTransposition : int vect -> bool = <fun>
```

**B.4** On empile les périodes non triviales dans une liste, puis on teste si le nombre de ces périodes est égal à la période du premier terme.

```
# let estCycle t = let n = vect_length t and test = ref [] in
        for i = 0 to n - 1 do
                if periode t i <> 1 then test := (periode t i) :: !test
                done;
        !test <> [] && hd !test = list_length !test;;
estCycle : int vect -> bool = <fun>
```

## Partie C – Opérations efficaces sur les permutations

**C.1** Si on n'a pas rempli la $i$-ième case, on la remplit par la valeur periode t i, ainsi que toutes les cases correspondant à l'orbite de $i$ : il y a donc bien autant d'appels à periode que d'orbites.

```
# let periodes t = let n = vect_length t in let p = make_vect n 0 in
 for i = 0 to n - 1 do if p.(i) = 0 then
        let k = periode t i and u = ref i in
                for j = 1 to k do p.(!u) <- k; u := t.(!u) done;
        done;
        p;;
periodes : int vect -> int vect = <fun>
```

**C.2**

```
# let itererEfficace t k = let n = vect_length t and p = periodes t in
        let res = make_vect n 0 in
                for i = 0 to n - 1 do
                        let r = k mod p.(i) and temp = ref i in
                                for j = 1 to r do temp := t.(!temp) done;
                        res.(i) <- !temp;
                done;
        res;;
itererEfficace : int vect -> int -> int vect = <fun>
```

**C.3** La permutation [|1; 2; 3; 0; 5; 6; 4|] est d'ordre 12.

**C.4**

```
# let rec pgcd = fun
        | a 0 -> a
        | a b -> pgcd b (a mod b);;
pgcd : int -> int -> int = <fun>
```

**C.5**

```
# let ppcm a b = a * b / (pgcd a b);;
ppcm : int -> int -> int = <fun>
```

**C.6** On se fonde sur une stratégie « diviser pour régner » :

```
# let rec ppcmTableau p = match vect_length p with
        | 1 -> p.(0)
        | n -> ppcm (ppcmTableau (sub_vect p 0 (n / 2)))
                   (ppcmTableau (sub_vect p (n / 2) (n - (n / 2) )));;
ppcmTableau : int vect -> int = <fun>

# let ordreEfficace t = ppcmTableau (periodes t);;
ordreEfficace : int vect -> int = <fun>
```

**C.7** On initialise un tableau de booléens test en le remplissant par des false, et si $j$ figure dans $t$, alors on remplace test.(j) par true. On teste si le tableau obtenu n'a que des true.

```
# let estPermutation t =
        let n = vect_length t and temp = ref true and i = ref 0 in
                let test = make_vect n false in
                        for i = 0 to n - 1 do
                                if t.(i) < n && t.(i) >= 0
                                        then test.(t.(i)) <- true
                        done;
                test = make_vect n true;;
estPermutation : int vect -> bool = <fun>
```