Correction du devoir du Vendredi 2 Mai

Exercice 1 (Nombre d'inversions d'une permutation)

1. (a)
```
let fusion t d m f =
  let aux = Array.make (f-d+1) t.(0) and i = ref l
                    and j = ref (m+1) and k = ref 0 in
    while (!i <= m) && (!j <= f) do
      if (t.(!i) <= t.(!j))
        then
          begin
            aux.(!k) <- t.(!i);
            i := !i + 1
          end
        else
          begin
            aux.(!k) <- t.(!j);
            j := !j + 1
          end;
      k := !k + 1
    done;
    while (!i <= m) do
      aux.(!k) <- t.(!i);
      i := !i + 1;
      k := !k + 1
    done;
    while (!j <= f) do
      aux.(!k) <- t.(!j);
      j := !j + 1;
      k := !k + 1
    done;
    for x=0 to (!k-1) do
      t.(d+x) <- aux.(x)
    done
;;
```
   (b)
```
let tri_fusion t =
  let rec tri_fusion_aux t d f =
    if (d < f) then
      begin
        let m = (d+f)/2 in
          tri_fusion_aux t d m;
          tri_fusion_aux t (m+1) f;
          fusion t d m f;
      end
  in tri_fusion_aux t 0 (Array.length t - 1)
;;
```

2. (a)
```
let fusion_et_inversions t d m f =
  let aux = Array.make (f-d+1) t.(0) and i = ref d
```

```
    and j = ref (m+1) and k = ref 0 and inv= ref 0 in
    while (!i <= m) && (!j <= f) do
      if (t.(!i) <= t.(!j))
      then
        begin
          aux.(!k) <- t.(!i);
          i := !i + 1
        end
      else
        begin
          aux.(!k) <- t.(!j);
          inv:=!inv + m - !i+1;
          j := !j + 1
        end;
      k := !k + 1
    done;
    while (!i <= m) do
      aux.(!k) <- t.(!i);
      i := !i + 1;
      k := !k + 1
    done;
    while (!j <= f) do
      aux.(!k) <- t.(!j);
      j := !j + 1;
      k := !k + 1
    done;
    for x=0 to (!k-1) do
      t.(d+x) <- aux.(x)
    done;
    !inv
  ;;
```

(b)

```
let inversions t =
  let rec inversions_aux t d f =
    if (d=f) then
      0
    else
      begin
        let m = (d+f)/2 in
        let inv1=inversions_aux t d m and inv2=inversions_aux t (m+1) f in
        inv1+inv2+(fusion_et_inversions t d m f)
      end
  in inversions_aux t 0 (Array.length t - 1)
;;
```

La complexité de cette algorithme est celle du tri fusion, c'est-à-dire $O(n \log_2(n))$.

Exercice 2 (Nombre d'occurrences d'un élément dans un ensemble)

1. (a)

```
let occurences1 a x =
  let n = Array.length a and compt = ref 0 in
    for k = 0 to (n-1) do
      if a.(k) = x then compt := !compt+1
    done;
  !compt
;;
```

(b) Proposons l'invariant de boucle : Soit $k \in [\![1, n]\!]$,

$(\mathcal{H}_k)$ "A la fin de la k-ième itération, *compt* contient le nombre de valeurs égales à x parmi $\{a.(0), \dots, a.(k-1)\}$."

• $(\mathcal{H}_1)$: A la première itération, on teste si $a.(0) = x$. Si c'est le cas, la variable *compt*, qui était initialisée à 0, prend la valeur $0 + 1 = 1$. Sinon, la variable *compt* reste égale à 0. Dans tous les cas, à la fin de la première itération, *compt* contient le nombre de valeurs égales à x parmi $\{a.(0)\}$.

• Soit $k \in [\![1, n]\!]$. Supposons $(\mathcal{H}_{k-1})$, c'est-à-dire qu'au début de la k-ième itération, *compt* contient le nombre de valeurs égales à x parmi $\{a.(0), \dots, a.(k-2)\}$. On teste alors si $a.(k-1) = x$. Si c'est le cas, la variable *compt* prend la valeur *compt* + 1. Sinon, la variable *compt* reste inchangée. Dans tous les cas, à la fin de la k-ième itération, *compt* contient bien le nombre de valeurs égales à x parmi $\{a.(0), \dots, a.(k-1)\}$. Donc $(\mathcal{H}_k)$ est vraie.

• Par principe de récurrence, pour tout k appartenant à $[\![1, n]\!]$, $(\mathcal{H}_k)$ est vraie.

Notre algorithme est donc correct.

(c) On effectue n itérations avec à chaque fois une comparaison. On donc une complexité $C(n) = n = O(n)$ qui est linéaire.

2. (a)

```
let rec occurences2 a x = match a with
  | [] -> 0
  | t::q when (t=x) -> 1+(occurences2 q x)
  | t::q -> occurences2 q x
;;
```

(b) Nous proposons comme suite strictement décroissante la suite "longueur de la liste a". Si a est de longueur $n \in \mathbb{N}^*$, `occurences2 a x` amène un appel récursif sur la queue de a de longueur $n - 1 < n$.

L'ordre usuel sur $\mathbb{N}$ étant bien fondé, notre algorithme se termine.

(c) Choisissons comme taille de données n la longueur de la liste et comme opération fondamentale la comparaison. En notant $C(n)$ la complexité au pire, on obtient :

$$C(0) = 0 \text{ et } \forall n \in \mathbb{N}^*,\ C(n) = 1 + C(n-1).$$

Donc $C(n) = n = O(n)$. La complexité est linéaire.

3.

```
let occurrence3 a x =
  let rec aux a x acc = match a with
    | [] -> acc
    | t::q when (t=x) -> aux q x (acc+1)
    | t::q -> aux q x acc
  in aux a x 0
;;
```

Exercice 3 (Karatsuba avec des listes de flottants)

1. (a)
```
let rec normalise p=
  let rec que_des_zeros l=match l with
    |[] -> true
    |t::q -> t=0.0 && (que_des_zeros q) in
  match que_des_zeros p with
    |true -> []
    |false -> let t::q=p in t::(normalise q);;
```
   (b)
```
let rec ajout p1 p2=match p1,p2 with
    |_,[] -> p1
    |[],_ -> p2
    |t1::q1,t2::q2 -> normalise ((t1+.t2)::(ajout q1 q2));;
```
   (c)
```
let rec multscal p lambda= match p with
    |[] -> []
    |t::q -> normalise ((lambda*.t)::(multscal q lambda));;
```
   (d)
```
let rec soustr p1 p2=match p1,p2 with
    |_,[] -> p1
    |[],t2::q2 ->(-.t2)::(soustr [] q2)
    |t1::q1,t2::q2 -> normalise ((t1-.t2)::(soustr q1 q2));;
```

2.
```
let rec mult p1 p2=match p1,p2 with
  |_,[] -> []
  |[],_ -> []
  |t1::q1,t2::q2 -> let c1 = mult q1 q2 and
    c2 = ajout (multscal q2 t1) (multscal q1 t2) and c3=[t1*.t2] in
      ajout (ajout (0.::0.::c1) (0.::c2)) c3;;
```

3. (a)
```
let rec separe l k= match k with
    |0 -> [],l
    |_ -> let t::q = l in let l1,l2= separe q (k-1) in t::l1,l2;;
```
   (b)
```
let rec produit_x i p= match i with
    |0 -> p
    |_ -> 0.::(produit_x (i-1) p);;
```
   (c)
```
let rec karatsuba k p q=match k,p,q with
    |0,_,_ ->mult p q
    |_,[],_  -> []
    |_,_,[]  -> []
    |_     ->let a1,a2 = separe p k
```

```
        and b1,b2 = separe q k in
        let c1 = (karatsuba (k/2) a1 b1) and
          c2 = (karatsuba (k/2) a2 b2) in
        let a12 = (ajout a1 a2) and
          b12 = (ajout b1 b2) in
        let aux = (karatsuba (k/2) a12 b12) in
        let c3 = soustr (soustr aux c1) c2 in
        ajout c1 (ajout (produit_x k c3) (produit_x (2*k) c2));;

let n1=List.length p1 and n2= List.length p2 in
  karatsuba ((max n1 n2)/2) p1 p2;;
```
