Correction du TP2 : Algorithmes de tri

Exercice 1

1.

```
let insere i t =
  let k = ref (i-1) and aux = t.(i) in
    while (!k >= 0) && (t.(!k) > aux) do
      t.(!k+1) <- t.(!k);
      k := !k-1
    done;
    t.(!k+1) <- aux
;;
```

2.

```
let tri_insertion t =
  let n = Array.length t in
    for i=1 to n-1 do
      insere i t
    done;
    t
;;
```

3. — Complexité de `insere i t` :
La taille des données est i et les opérations fondamentales sont les comparaisons et les affectations. Notons $C_c(i)$ le nombre de comparaisons et $C_a(i)$ le nombre d'affectations.
A chaque itération, on fait 2 comparaisons et 1 affectation. Comme nous faisons i itérations, $C_c(i) = 2i$ et $C_a(i) = i+1$ (car on termine par l'affectation `t.(!k+1) <- aux`).

— Complexité de `tri_insertion t` :
La taille des données est la longueur n du tableau et les opérations fondamentales sont toujours les comparaisons et les affectations. Notons $T_c(n)$ le nombre de comparaisons et $T_a(n)$ le nombre d'affectations. Alors :

$$\begin{aligned} T_c(n) &= \sum_{i=1}^{n-1} C_c(i) = 2\sum_{i=1}^{n-1} i = n(n-1) = O(n^2), \\ T_a(n) &= \sum_{i=1}^{n-1} C_a(i) = \sum_{i=1}^{n-1}(i+1) = \sum_{i=2}^{n} i = \frac{n(n+1)}{2} - 1 = O(n^2). \end{aligned}$$

Exercice 2

1.

```
let rec minimum_et_reste l = match l with
  | [a] -> a , []
  | t::q -> let (m,r) = minimum_et_reste q in
              if m < t then (m , t::r) else (t,q)
;;
```

2.

```
let rec tri_selection l = match l with
  | [] -> []
  | _ -> let (m,r) = minimum_et_reste l in
           m::(tri_selection r)
;;
```

3. — Complexité de `minimum_et_reste l` :
La taille des données est la longueur n de la liste. Les opérations fondamentales sont les comparaisons et les ajouts en tête de liste. Notons $C_c(n)$ le nombre de comparaisons et $C_a(n)$ le nombre d'ajouts en tête de liste. Alors :

$$\begin{cases} C_c(1) = 0 \\ \forall n \in \mathbb{N} \setminus \{0,1\},\ C_c(n) = C_c(n-1) + 1 \end{cases}$$

Donc pour tout $n \in \mathbb{N}^*$, $C_c(n) = n - 1$.
Exactement de la même façon, on obtient pour tout $n \in \mathbb{N}^*$, $C_a(n) = n - 1$.
— Complexité de `tri_selection l` :
La taille des données est la longueur n de la liste. Les opérations fondamentales sont toujours les comparaisons et les ajouts en tête de liste. Notons $T_c(n)$ le nombre de comparaisons et $T_a(n)$ le nombre d'ajouts en tête de liste.

Pour le nombre de comparaisons :

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

Alors, pour tout $k \in \mathbb{N}^*$, $T_c(k) - T_c(k-1) = k - 1$. En sommant pour k alors de 1 à n, on a :
— d'une part, $\displaystyle\sum_{k=1}^{n} (T_c(k) - T_c(k-1)) = T_c(n)$ (par télescopage),
— d'autre part, $\displaystyle\sum_{k=1}^{n} (k-1) = \frac{(n-1)n}{2}$.
Donc, pour tout $n \in \mathbb{N}^*$, $T_c(n) = \dfrac{(n-1)n}{2} = O(n^2)$.

Pour le nombre d'ajout en tête de liste :

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

De même que précédemment, pour tout $n \in \mathbb{N}^*$, $T_a(n) = \dfrac{n(n+1)}{2} = O(n^2)$.

Exercice 3

1.

```
let rec une_etape l = match l with
  | [a] -> false,l
  | t::q -> let b,t1::q1 = une_etape q in
              if (t <= t1) then b,t::t1::q1
                           else true,t1::t::q1
;;
```

2.

```
let rec tri_bulles l = match l with
  | [] -> []
  | _ -> let b,t::q = une_etape l in
              if b then t::(tri_bulles q)
                   else l
;;
```

3. — Complexité de `une_etape l` :
La taille des données est la longueur n de la liste. Les opérations fondamentales sont les comparaisons et les ajouts en tête de liste. Notons $C_c(n)$ le nombre de comparaisons et $C_a(n)$ le nombre d'ajouts en tête de liste.

Pour le nombre de comparaisons :

$$\begin{cases} C_c(1) = 0 \\ \forall n \in \mathbb{N} \setminus \{0,1\},\ C_c(n) = C_c(n-1) + 1 \end{cases}$$

Donc pour tout $n \in \mathbb{N}^*$, $C_c(n) = n - 1$.

Pour le nombre d'ajouts en tête de liste :

$$\begin{cases} C_a(1) = 0 \\ \forall n \in \mathbb{N} \setminus \{0,1\},\ C_a(n) = C_a(n-1) + 2 \end{cases}$$

Donc pour tout $n \in \mathbb{N}^*$, $C_c(n) = 2(n-1)$.

— Complexité de `tri_bulles l` :
La taille des données est la longueur n de la liste. Les opérations fondamentales sont toujours les comparaisons et les ajouts en tête de liste. Notons $T_c(n)$ le nombre de comparaisons et $T_a(n)$ le nombre d'ajouts en tête de liste.

Pour le nombre de comparaisons :

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

Comme dans l'exercice précédent, pour tout $n \in \mathbb{N}^*$, $T_c(n) = \dfrac{(n-1)n}{2} = O(n^2)$.

Pour le nombre d'ajout en tête de liste :

$$T_a(0) = 0 \quad \text{et} \quad \forall n \in \mathbb{N}^*,\ T_a(n) = C_a(n) + T_a(n-1) + 1 = 2n - 1 + T_a(n-1).$$

De même que précédemment, pour tout $n \in \mathbb{N}^*$, $T_a(n) = n^2 = O(n^2)$.

Exercice 4

1. Si on a deux pancakes, une seule manipulation est éventuellement nécessaire. Si on dispose de plus de trois pancakes à trier, on glisse la spatule en dessous du plus grand, et on retourne tout le tas. Le plus grand se retrouve donc en haut de la pile. Il suffit alors de glisser la spatule en dessous du tas et de tout retourner. On réitère ce processus sur les $n-1$ pancakes restants. Le nombre d'opérations nécessaires vérifie donc :
   — $C(2) \leq 1$.
   — Si $n \geq 3$, $C(n) \leq 2 + C(n-1)$.
   Une récurrence rapide montre que $C(n) \leq 2n-3$.

2. (a) On utilise la fonction `echange` vu en cours.

```
let retourne t i =
  for j = 0 to i/2 do
    echange t j (i-j)
  done
;;
```

   (b)

```
let maximum t i =
  let m = ref 0 in
    for j = 1 to i do
      if t.(j) > t.(!m) then m := j
    done;
    !m
;;
```

   (c)

```
let tri_pancake t =
  for i = (Array.length t - 1) downto 1 do
    retourne t (maximum t i);
    retourne t i
  done;
  t
;;
```

   (d) La taille de données est la longueur n de la série à trier. Les opérations fondamentales sont les échanges et les comparaisons.
   Dans la fonction `retourne`, on fait $\left\lfloor \frac{i}{2} \right\rfloor$ échanges et 0 comparaison.
   Dans la fonction `maximun`, on fait 0 échange et i comparaisons.
   Dans la fonction `tri_pancake`, on fait une boucle indexée par i entre 1 et $n-1$ avec à chaque fois 2 appels à la fonction `retourne` (sur une taille $\leq i$ de données) et 1 appels à la fonction `maximum` (sur une taille i de données).
   Or on sait que $\displaystyle\sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} = O(n^2)$.

Donc le tri pancake trie une série de n données avec un nombre de comparaisons et un nombre d'échanges tous les deux égaux à $O(n^2)$.

3. (a)

```
let retourne l i =
  let rec retourne_aux l1 l2 i = match i with
    | 0 -> l2 @ l1
    | _ -> let t::q = l1 in retourne_aux q (t::l2) (i-1)
  in retourne_aux l [] i
;;
```

(b)

```
let maximum l i =
  let rec maximum_aux m j k l i = match k with
    | k when k>i -> j
    | _ -> let t::q=l in
             if t>m then maximum_aux t k (k+1) q i
                    else maximum_aux m j (k+1) q i
  in maximum_aux (List.hd l) 1 1 l i
;;
```

Dans la fonction auxiliaire, `m` est la valeur du maximum, `j` le numéro de l'élément de la liste `l` correspondant au maximum et `k` le numéro de l'élément de la liste où on en est dans la recherche.

Si `k > i`, on a parcouru les `i` premiers éléments et on a donc terminé notre recherche (le maximum est le `j`-ième élément).

Sinon, on teste si la tête de la liste est ou non le nouveau maximum et on modifie si nécessaire sa valeur (`t` ou `m`), le numéro de l'élément (`k` ou `j`), le numéro à partir duquel on fait la recherche (`k+1`) et on travaille sur la queue de la liste récursivement (`q`).

(c)

```
let tri_pancake l =
  let rec tri_pancake_aux l i = match i with
    | 1 -> l
    | _ -> let j = maximum l i in
           tri_pancake_aux (retourne (retourne l j) i) (i-1)
  in tri_pancake_aux l (List.length l)
;;
```

Exercice 5

1.

```
let fusion t l m u =
  let aux = Array.make (u-l+1) t.(0) and i = ref l
                    and j = ref (m+1) and k = ref 0 in
    while (!i <= m) && (!j <= u) 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 <= u) do
      aux.(!k) <- t.(!j);
      j := !j + 1;
      k := !k + 1
    done;
    for x=0 to (!k-1) do
      t.(l+x) <- aux.(x)
    done
;;
```

2.

```
let tri_fusion t =
  let rec tri_fusion_aux t l u =
    if (l < u) then
      begin
        let m = (l+u)/2 in
          tri_fusion_aux t l m;
          tri_fusion_aux t (m+1) u;
          fusion t l m u;
      end
  in tri_fusion_aux t 0 (Array.length t - 1)
;;
```

3. — Pour la fonction `fusion` :
   Nous avons trois répétitives conditionnelles. La terminaison de la première est assurée par la stricte décroissance de la suite d'entiers naturels $(m + u - i - j)$.

La terminaison de la seconde est assurée par la stricte décroissance de la suite d'entiers naturels $(m-i)$. La terminaison de la troisième est assurée par la stricte décroissance de la suite d'entiers naturels $(u-j)$.

— Pour la fonction `tri_fusion` :
Il suffit de prouver la terminaison de la fonction `tri_fusion_aux`. Pour cela, on prouve la stricte décroissance de la suite d'entiers naturels $(u-l)$. u et l étant deux entiers tels que : $l < u$, $u-l$ est un entier naturel et :
Premier cas : $l+u$ est pair, alors $l < m = \dfrac{l+u}{2} < u$, et $m-l < u-l$, et $u-(m+1) = u - \dfrac{l+u}{2} - 1 < u-l-1 < u-l$.
Deuxième cas : $l+u$ est impair, alors $l - \dfrac{1}{2} < m = \dfrac{l+u-1}{2} < u$, et les valeurs étant entières, $l \leq m = \dfrac{l+u-1}{2} < u$, donc $m-l < u-l$ et $u-(m+1) \leq u-l-1 < u-l$.
Dans tous les cas, la suite $(u-l)$ est une suite strictement décroissante d'entiers naturels.

4. — Pour la fonction `fusion` :
La taille de données est le couple (l,u). Les opérations fondamentales sont les comparaisons. On note $C(l,u)$ le nombre de comparaisons.
Le placement d'un terme dans *aux* demande au plus 3 comparaisons, aucune pour le placement dans t. Sur un total de $u-l+1$ éléments, cela donne donc :

$$C(l,u) = 3(u-l+1).$$

— Pour la fonction `tri_fusion` :
La taille de données est la longueur de la sous-série de données $p = u-l+1$. Les opérations fondamentales sont toujours les comparaisons. On note $T(p)$ le nombre de comparaisons. Alors :

$$T(p) = 0 + 2T(p/2) + 3p = 2T(p/2) + O(p).$$

En appliquant le théorème maître de complexité, avec $q=2$ et $\gamma = 1$, on obtient $T(p) = O(p\log_2(p))$.
Pour la série complète de longueur n, on a donc $T(n) = O(n\log_2(n))$.