Correction du TD 6

Julien Reichert

Exercice 1

```
void tri_rapide(int tab[], int taille)
{
  void aux(int deb, int fin)
  {
    if (deb >= fin) return;
    int d = deb+1;
    int f = fin;
    int buff;
    while (d <= f)
    {
      if (tab[d] > tab[deb])
      {
        buff = tab[d];
        tab[d] = tab[f];
        tab[f] = buff;
        f -= 1;
      }
      else d += 1;
    }
    buff = tab[deb];
    tab[deb] = tab[f];
    tab[f] = buff;
    aux(deb, f-1);
    aux(f+1, fin);
  }
  aux(0, taille-1);
}

int main() // Pour tester
{
  int tab[] = { 3 , 7 , 4 , 1 , 5 , 2 , 6 };
  tri_rapide(tab, 7);
  for (int i = 0 ; i < 7 ; i += 1) printf("%d ", tab[i]);
  return 0;
}
```

Version non en place en OCaml :

```
let rec separe x l = match l with
| [] -> [], []
| a::q -> let lg, ld = separe x q in if a < x then a::lg, ld else lg, a::ld;;

let rec tri_rapide l = match l with
| [] -> []
| [a] -> [a]
| a::q -> let lg, ld = separe a q in (tri_rapide lg)@(a :: tri_rapide ld);;
```

Exercice 2

```
let taille_entier n = String.length (string_of_int n);; (* C'est dans l'esprit ! *)

let rec taille_max_entiers l = match l with
| [] -> failwith "Liste vide"
| [a] -> taille_entier a
| a::q -> max (taille_entier a) (taille_max_entiers q);;
(* Peut se faire avec un fold ! *)

let i_eme_chiffre i k =
  let kbis = ref k in
  for j = 1 to i do kbis := !kbis / 10 done;
  !kbis mod 10;;
(* i valant 0 = chiffre des unités *)

let maj_base bases indice k =
  let j = i_eme_chiffre indice k in
  bases.(j) <- k::bases.(j);;

let tri_base l =
  if l = [] then []
  else
    let n = taille_max_entiers l in
    let etape indice liste_actuelle =
      let tab = Array.make 10 [] in
      List.iter (maj_base tab indice) (List.rev liste_actuelle); tab
    in let rec aux indice liste =
      if indice = n then liste
      else let tab = etape indice liste in
      aux (indice+1) (Array.fold_right (@) tab [])
      (* merci à Cyprien R. pour le signalement que left va moins vite que right *)
    in aux 0 l;;
```

Exercice 3

Le squelette reste le même, on change simplement quelques détails.

```
let taille_max_chaines l =
  List.fold_left (fun accu s -> max accu (String.length s)) 0 l;;
(* Je l'avais dit ! *)

let maj_basebis bases indice s =
  if indice >= String.length s then bases.(0) <- s::bases.(0)
  else
    let j = int_of_char s.[indice] mod 32 in
    bases.(j) <- s::bases.(j);;

let tri_base_chaines l =
  if l = [] then []
  else
    let n = taille_max_chaines l in
    let etape indice liste_actuelle =
      let tab = Array.make 27 [] in
      List.iter (maj_basebis tab indice) (List.rev liste_actuelle); tab
    in let rec aux indice liste =
      if indice = (-1) then liste
      else let tab = etape indice liste in
      aux (indice-1) (Array.fold_right (@) tab [])
    in aux (n-1) l;;
```

Exercice 4

```
int* permute(int* tab, int taille, int* permutation)
{
  int* resultat = malloc(taille * sizeof(int));
  for (int i = 0 ; i < taille ; i += 1)
  {
    resultat[permutation[i]] = tab[i];
  }
  return resultat;
}
```

Exercice 5

Une seule permutation est triée, et pour l'obtenir à partir de σ, seule la réciproque de celle-ci convient. La structure de groupe de $\mathfrak{S}_n$ suffit en tant que justification.

Exercice 6

Si `seq[i]` est inférieur à `seq[j]`, alors en considérant une permutation σ telle que $\sigma(i) > \sigma(j)$, la permutation σ ne peut pas transformer la séquence en une séquence triée, car l'élément initialement à la position `i` est envoyé à droite de l'élément initialement à la position `j` à tort.

Ainsi, les permutations sont réparties en deux catégories, dont l'une est éliminable de la recherche de la permutation qui trie la liste. En ce qui concerne la première comparaison, elle élimine exactement la moitié des permutations (en supposant évidemment que $n \geq 2$).

Exercice 7

À la lumière du dernier paragraphe de la réponse à l'exercice précédent, il s'agit au sens étymologique de procéder par dichotomie. On peut même voir une illustration du minmax vu en deuxième année pour choisir des comparaisons pertinentes qui séparent l'ensemble des permutations restantes en deux parts aussi équilibrées que possible, et en anticipant que les prochaines séparations puissent maintenir l'équilibre (pour ce contexte particulier, ce n'est pas un problème, mais il est bon de garder en tête le principe de voir au-delà du tour actuel).

Exercice 8

On a $n! \sim (\frac{n}{e})^n \sqrt{2\pi n}$, et comme les équivalents passent au logarithme (exercice de mathématiques), $\log(n!) \sim n \log n - n \log e + \frac{1}{2} \log(2\pi n)$, d'où par croissances comparées $\log(n!) \sim n \log n$, ceci étant valable quelle que soit la base du logarithme, ici implicitement 2.

Exercice 9

Dans le pire des cas, lorsqu'une comparaison est effectuée, le résultat est que parmi les deux catégories de permutations constituées en fonction de la comparaison, c'est la plus grande qui demeure. Ainsi, le meilleur algorithme garde toujours des parts équilibrées afin que la moitié (arrondie par défaut) soit éliminée. Pour passer de l'ensemble des permutations à une seule permutation possible, il lui faut alors diviser $n!$ par deux de l'ordre de $n \log n$ fois pour n assez grand.

Une autre façon de voir les choses est que l'algorithme explore une arborescence d'arité deux dont les nœuds représentent des comparaisons et les feuilles sont la permutation qui trie la séquence une fois qu'elle aura été trouvée. Chaque arbre correspond à un algorithme (on peut voir ceci comme un arbre de décision, en pratique) et le pire des cas correspond à la plus grande profondeur, autrement dit la hauteur de l'arbre. La hauteur d'un arbre binaire ayant $n!$ feuilles (donc de l'ordre du double de nœuds) est au moins le logarithme de sa taille, ce qui prouve également le résultat.

Exercice 10

Soit une permutation σ.

Considérons un tableau représentant la réciproque de σ de la même manière que ce qui s'est fait dans ce TD.

Le tri à bulles n'agit qu'en échangeant deux éléments consécutifs de ce tableau. Il s'agit donc de transpositions de la forme annoncée dans l'énoncé, et leur composée donne une permutation qui fait passer du tableau, au tableau représentant la fonction identité, donc la réciproque de la permutation représentée par le tableau, donc σ.