Correction du TD 7

Julien Reichert

Exercice 1

Si le premier élément est toujours le maximum ou le minimum de la zone, la séparation laissera toujours une des deux zones vide. C'est malheureusement le cas si la séquence est triée, en particulier (croissante ou décroissante, par ailleurs).

Exercice 2

On évite l'écueil systématique dans le cas où la séquence est triée, mais il reste une probabilité non nulle d'avoir toujours un mauvais pivot (vaguement estimée à $\frac{2^{n-1}}{n!}$ avec n pour taille de la séquence). Ainsi, on va chercher une solution ailleurs.

Exercice 3

Une valeur qui n'est pas forcément égale à un élément de la séquence fait perdre l'opportunité d'avoir un élément d'ores et déjà bien placé. Comme le minimum et le maximum sont de toute manière séparés ainsi, la terminaison est au moins garantie, mais rien n'exclut que dans la version triée l'écart d'un élément à l'autre explose si vite que la séparation n'a aucune raison d'être équilibrée.

Exercice 4

Premier algorithme : compter pour chaque élément le nombre d'éléments supérieurs et inférieurs et trancher, la complexité est quadratique.

Deuxième algorithme : prendre l'élément du milieu de la version triée, avec un algorithme optimal la complexité est en $\Theta(n \log n)$ où n est la taille de la séquence.

Quoi qu'il en soit, dans l'optique de se servir de la médiane pour un tri, on évitera ces deux algorithmes.

Exercice 5

La complexité ayant de la pertinence quand la taille de l'entrée tend vers l'infini, pour des entrées de taille bornée il n'y a pas tant d'intérêt à optimiser (en pratique cela joue dans le facteur constant).

Exercice 6

La structure est de taille $\lceil \frac{n}{5} \rceil$. Jusqu'à ce que cette taille soit assez petite pour qu'on calcule la médiane directement (disons là aussi une taille de 5), on utilise l'algorithme en cours d'écriture (d'où une récursion) pour calculer la médiane, ce qui en fait un exemple de DPR.

Exercice 7

Supposons que M_5 soit la plus petite valeur du paquet de 5 où elle figure. Alors elle ne figurera pas parmi les $\lceil \frac{n}{5} \rceil$ médianes et n'aura en particulier aucune chance d'être M_5 .

Cependant, M_5 est inférieure à la moitié des $\lceil \frac{n}{5} \rceil$ médianes, chacune inférieure à deux valeurs. On en déduit que M_5 est, de manière sûre, grossièrement inférieure à trente pour cent des valeurs, mais aussi supérieure à trente pour cent des valeurs, là aussi de manière sûre.

Exercice 8

En se servant de M_5 pour filtrer les éléments de la séquence de départ, on élimine tous ceux qui sont inférieurs à M_5 ou tous ceux qui lui sont supérieurs, afin de maintenir la portion qui contient la médiane. Il reste alors entre trente et soixante-dix pour cent des éléments.

On a donc pour une séquence (de départ ou actuelle) de taille n , en notant c_n la complexité du calcul de la médiane :

- Séparer en paquets de cinq : $\Theta(n)$.
- Calculer la médiane de chacun des paquets : $\Theta(n)$ (c'est $\lceil \frac{n}{5} \rceil$ fois le coût borné du calcul de la médiane d'une séquence de taille au plus 5).
- Calculer la médiane de ces médianes : $c_{\frac{n}{5}}$.
- Éliminer les éléments qui ne peuvent plus être la médiane de la séquence de départ : $\Theta(n)$.
- Une borne supérieure de la complexité de l'appel récursif sur ce qui reste (selon le principe que la complexité est croissante donc la plus grande taille de l'argument donne la plus grande complexité) : $c_{\frac{7n}{10}}$.

On en déduit que $c_n \leq c_{\frac{n}{5}} + c_{\frac{7n}{10}} + \Theta(n)$, et même en remplaçant le $\leq$ par un $=$ on tomberait dans le cas du résultat admis dans l'énoncé, ce qui permet de conclure que $c_n = \Theta(n)$.

Exercice 9

En OCaml sur des tableaux, c'est plus pratique et ce serait pareil en C modulo la gestion de la mémoire :

```
exception Trouve of int;;
```

```

(* Taille paire : la première des deux à être trouvée. *)
let med5 t =
  let n = Array.length t in
  assert (n <= 5);
  try
    for i = 0 to n - 1 do
      let inf = ref 0 and sup = ref 0 in
      for j = 0 to n - 1 do
        if t.(j) < t.(i) then incr inf
        else if t.(j) > t.(i) then incr sup
      done;
      if !inf <= n / 2 && !sup <= n / 2 then raise (Trouve i)
    done;
    failwith "Ce cas ne devrait pas arriver !"
  with Trouve i -> t.(i);;

let rangement t debut fin m =
  assert (debut <= fin);
  let d = ref debut and f = ref fin in
  while !d <= !f do
    if t.(!d) = m then
      begin
        let buff = t.(debut) in t.(debut) <- t.(!d); t.(!d) <- buff; incr d
      end
    else if t.(!d) > m then
      begin
        let buff = t.(!f) in t.(!f) <- t.(!d); t.(!d) <- buff; decr f
      end
    else incr d
  done;
  let buff = t.(!f) in
  t.(!f) <- t.(debut); (* aussi m *)
  t.(debut) <- buff;
  !f;; (* position du pivot *)

```

```

(* Taille paire : médiane de gauche. *)
let rec medianerec t debut_init fin_init debut fin =
  assert (debut <= fin);
  let taille = fin - debut + 1 in
  if taille <= 5 then med5 (Array.sub t debut taille)
  else
  begin
    let nb_medianes = if taille mod 5 = 0 then taille / 5 else taille / 5 + 1 in
    let les_medianes = Array.make nb_medianes t.(0) in
    for i = 0 to nb_medianes - 1 do
      let taille_portion = ref 5 in
      if i = nb_medianes - 1 && taille mod 5 <> 0
      then taille_portion := taille mod 5;
      les_medianes.(i) <- medianetot (Array.sub t (debut + i * 5) !taille_portion)
    done;
    let m = medianetot les_medianes in
    let pos_m = rangement t debut fin m in
    match pos_m - debut_init - (fin_init - debut_init) / 2 with
    | 0 -> t.(pos_m)
    | x when x < 0 -> medianerec t debut_init fin_init (pos_m + 1) fin
    | _ -> medianerec t debut_init fin_init debut (pos_m - 1)
  end

and mediane t debut fin =
  medianerec t debut fin debut fin

and medianetot t =
  mediane t 0 (Array.length t - 1);;

```

Exercice 10

En OCaml :

```

let tri_rapide t =
  let rec aux debut fin =
    if debut < fin then
    begin
      let m = mediane t debut fin in
      let ind = rangement t debut fin m in
      aux debut (ind - 1);
      aux (ind + 1) fin
    end
  in aux 0 (Array.length t - 1);;

```

On note que le tri rapide agit par mutation, mais que le travail n'est cependant pas en place vu que la fonction `mediane` utilise de la mémoire annexe.

Pour se faire une idée, voilà ce que cela donnerait en C :

```
void tri_rapide(int tab[], int taille)
{
    void aux(int deb, int fin)
    {
        if (deb >= fin) return;
        int* tabbis = malloc((fin-deb+1)*sizeof(int));
        for (int i = 0 ; i <= fin-deb ; i += 1) tabbis[i] = tab[deb+i];
        int mm = mediane(tabbis, fin-deb+1);
        free(tabbis);
        int d = deb; int f = fin;
        int buff, posmed;
        while (d <= f)
        {
            if (tab[d] > mediane)
            {
                buff = tab[d]; tab[d] = tab[f]; tab[f] = buff;
                f -= 1;
            }
            else
            {
                if (tab[d] == mediane) posmed = d;
                d += 1;
            }
        }
        buff = tab[posmed]; tab[posmed] = tab[f]; tab[f] = buff;
        aux(deb, f-1);
        aux(f+1, fin);
    }
    aux(0, taille-1);
}
```

Remarque annexe : l'algorithme de calcul de la médiane peut être adapté pour se brancher directement sur le tri rapide, auquel cas on le ferait agir en ajoutant l'effet secondaire de répartir à chaque étape de part et d'autre de chaque médiane calculée les éléments inférieurs ou supérieurs, permettant de ne pas faire deux fois le même travail...