DS 4 d'option informatique - MP* - Choix du pivot dans le tri rapide (Sujet Centrale 2012) - Correction

Question *I.A* –

```
(* echange : int -> int -> int array -> unit *)
(* Les entiers en entrée sont les indices des éléments à échanger.
   Hypothèse: ces deux entiers sont des indices valides *)
let echange (i: int) (j: int) (tab: int array): unit =
  let tmp = tab.(i) in
  tab.(i) <- tab.(j);
  tab.(j) <- tmp;;
```

Question *I.B* – Un algorithme de tri simple est le tri par sélection. L'idée est de rechercher le plus petit élément du tableau, puis le deuxième plus petit, puis le troisième plus petit, et ainsi de suite.

Plus précisément, le tri par sélection se décompose en n étapes où n est la taille du tableau. Si on numérote ces étapes par $k \in [\![0, n-1]\!]$, alors à l'étape numéro k :

→ On recherche le minimum parmi :

$$\texttt{tab.(k)}, \texttt{tab.(k+1)}, \texttt{tab.(k+2)}, \ldots, \texttt{tab.(n-1)}$$

→ On échange ce minimum avec `tab.(k)`.

Pour la complexité, à l'étape numéro $k \in [\![0; n-1]\!]$, il faut rechercher un minimum parmi $n-k$ éléments, ce qui nécessite $n-k-1$ comparaisons. Le nombre total de comparaisons est donc :

$$\sum_{k=0}^{n-1}(n-k-1) = \sum_{\ell=0}^{n-1}\ell = \frac{n(n-1)}{2}$$

Ainsi :

La complexité du tri par sélection en terme de comparaisons est $\mathcal{O}(n^2)$ avec n la taille du tableau en entrée.

```
(* mini : int array -> int -> int *)
(* Renvoie l'indice i >= k qui minimise tab.(i) *)
let mini (tab: int array) (k: int): int =
  let i_min = ref k in
  for i = k + 1 to Array.length tab - 1 do
    if tab.(i) < tab.( !i_min ) then i_min := i
  done;
  !i_min;;
```

```
(* tri_selection : int array -> unit *)
let tri_selection (tab: int array): unit =
  for k = 0 to Array.length tab - 1 do
    let i_min = mini tab k in
    echange k i_min tab;
  done;;
```

Question I.C.1) – Soit `tab` un tableau de taille `m`, et deux entiers `i1, i2` tels que $0 \leqslant$ `i1` $<$ `i2` $\leqslant$ `m` $- 1$. On note `tab.(i1,i2)` le tableau contenant les éléments d'indices `k` $\in [\![$`i1; i2`$]\!]$:

```
tab.(i1,i2) = [|tab.(i1), tab.(i1+1), ..., tab.(i2-1), tab.(i2)|]
```

Supposons que l'on cherche à séparer le tableau `tab.(i1,i2)` de taille `n = i2 - i1 + 1` en choisissant pour pivot le premier élément `tab.(i1)`. L'idée est de diviser `tab.(i1,i2)` en quatre régions à l'aide de deux entiers `f` et `k` tels que `i1` $\leqslant$ `f` $<$ `k` $\leqslant$ `j` :

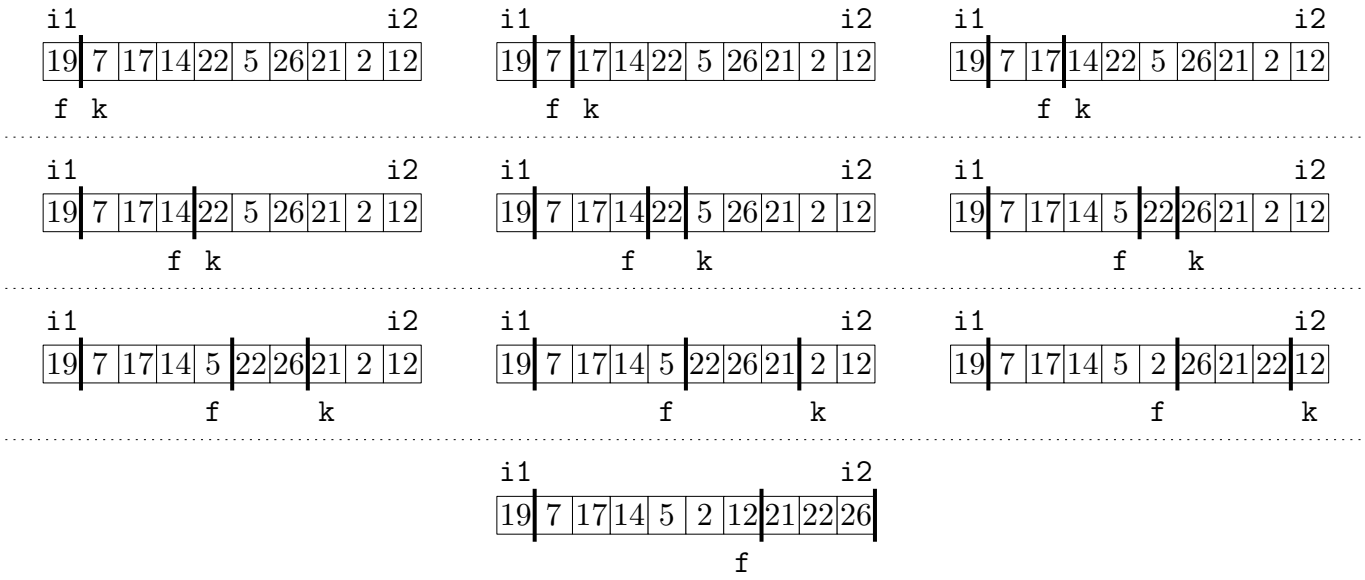
→ La première région `tab.(i1,i1)` contient uniquement le pivot.

→ La deuxième région `tab.(i1+1, f)` contient les éléments strictement inférieurs au pivot. Notons que cette région est vide lorsque `f = i1`.

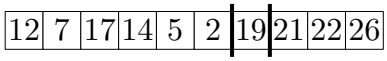
→ La troisième région `tab.(f+1, k-1)` contient les éléments strictement supérieurs au pivot. Notons que cette région est vide lorsque `k = f+1`.

→ La quatrième région `tab.(k, i2)` contient les éléments qui n'ont pas encore été comparés au pivot.

Initialement, les régions 2 et 3 sont vides, alors que la quatrième région contient `n-1` éléments. À chaque étape, on considère le premier élément de la quatrième région et on le place dans la région 2 ou 3 en fonction du résultat de sa comparaison avec le pivot. Le schéma ci-dessous montre l'évolution du tableau de l'énoncé :



À la fin de la procédure, il reste à échanger le pivot avec le dernier élément de la deuxième région pour obtenir :



Question I.C.2) –

```
(* separation : int array -> int -> int -> int *)
let separation (tab: int array) (i1: int) (i2: int): int =
  let f = ref i1 in
  for k = i1 + 1 to i2 do
    if tab.(k) < tab.(i1) then begin
        echange k (!f+1) tab;
        incr f;
      end
  done;
  echange i1 !f tab;
  !f;;
```

Question I.C.3) –

```
(* tri_rapide : int array -> unit *)
let tri_rapide (tab: int array): unit =
  let rec aux i1 i2 =
    if i1 < i2 then begin
        let j = separation tab i1 i2 in
        aux i1 (j-1);
        aux (j+1) i2;
      end in
  aux 0 (Array.length tab - 1);;
```

Question *II.A* –

⋆ Lorsque le tableau `tab` est déjà trié dans l'ordre croissant, on remarque que la fonction `separation` fait `n-1` comparaisons, ne modifie pas `tab` et renvoie `i1` (c'est à dire que le pivot reste en position `i1` dans `tab`). Ainsi, lorsqu'on s'intéresse à la fonction `tri_rapide`, un appel à la fonction `aux` sur un sous-tableau de taille `n` $\geqslant$ `2` fait :

→ Un appel récursif sur un sous-tableau vide qui s'exécute en temps constant.

→ Un appel récursif sur un sous-tableau de taille `n-1`.

Dès lors :

$$\begin{cases} C(1) = 0 \\ C(n) = n - 1 + C(n-1). \end{cases}$$

Une récurrence immédiate sur `n` $\in \mathbb{N}^*$ donne :

$$C(n) = \frac{n(n-1)}{2}.$$

Donc :

Lorsque le tableau est trié par ordre croissant, on obtient $C(n) = \Theta(n^2)$, c'est à dire que $v(n) = n^2$ convient.

⋆ Lorsque le tableau `tab` est déjà trié dans l'ordre décroissant, l'analyse est similaire : dans la fonction `aux`, un appel récursif se fait sur un tableau vide et l'autre sur le tableau privé du pivot. En revanche le sous-tableau à trier n'est pas systématiquement trié dans l'ordre décroissant. Voici par exemple l'évolution des sous-tableaux en entrée de `aux` lors de l'évaluation de « `tri_rapide [|10;9;8;7;6;5;4;3;2;1|]` » (les appels récursifs sur le sous-tableau vide ne sont pas représentés) :

`[|10;9;8;7;6;5;4;3;2;1|]` → `[|1;9;8;7;6;5;4;3;2|]` → `[|9;8;7;6;5;4;3;2|]` → `[|2;8;7;6;5;4;3|]` → `[|8;7;6;5;4;3|]` → `[|3;7;6;5;4|]` → `[|7;6;5;4|]` → `[|4;6;5|]` → `[|6;5|]` → `[|5|]`

Le sous-tableau à trier alterne donc entre un tableau trié dans l'ordre décroissant, et un tableau commençant par son élément minimum puis trié dans l'ordre décroissant. Le nombre de comparaisons reste le même que lorsque le tableau est trié par ordre croissant :

Lorsque le tableau est trié par ordre décroissant, on obtient $C(n) = \Theta(n^2)$, c'est à dire que $v(n) = n^2$ convient.

Remarque. Ici, on a interprété la formulation "nombre de comparaisons" comme le nombre de comparaisons entre éléments du tableau. En particulier, le test « `i1 < i2` » de la fonction `tri_rapide` n'est pas compté dans $C(n)$.

Remarque. La difficulté de cette question dépend en grande partie de l'implémentation choisie dans la question I.C.2). Le rapport du jury précise que cette question "nécessite de vérifier ce que deviennent les tableaux après la phase de séparation. Seuls quelques candidats ont vérifié ce point."

Question II.B.1) – Si on suppose que chaque séparation a coupé le tableau en deux parties égales, alors pour tout $n \in \mathbb{N}^*$:

→ Un tableau de taille $2n$ est coupé en un tableau de taille $n-1$ et un tableau de taille n.

→ Un tableau de taille $2n+1$ est coupé en deux tableaux de taille n.

Si on note $C(n)$ le nombre de comparaisons effectuées pour un tableau de taille $n \in \mathbb{N}$, on obtient :

$$C(0) = C(1) = 0 \text{ et } \forall n \in \mathbb{N}^* : \begin{cases} C(2n) = C(n) + C(n-1) + 2n - 1 \\ C(2n+1) = 2C(n) + 2n \end{cases}$$

Soit $M(n) \in \mathbb{N}$ l'entier défini par récurrence sur $n \in \mathbb{N}$ de la manière suivante :

$$M(0) = M(1) = 0 \text{ et } \forall n \in \mathbb{N}^* : \begin{cases} M(2n) = 2M(n) + 2n \\ M(2n+1) = 2M(n) + 2n \end{cases}$$

Une première récurrence sur $n \in \mathbb{N}$ montre que $M(n) \leqslant M(n+1)$. Une seconde récurrence sur $n \in \mathbb{N}$ montre que $C(n) \leqslant M(n)$. D'où le résultat avec $\alpha = 2$.

Question II.B.2) – D'après la question précédente :

$$M(1) = 0 \text{ et } \forall n \in \mathbb{N}^* : M(2n) = 2M(n) + 2n$$

On montre alors facilement par récurrence sur $k \in \mathbb{N}$ que $M(2^k) = k2^k$. Ainsi :

Lorsque $n = 2^k$ avec $k \in \mathbb{N}$, on a $M(n) = n\log_2(n)$

Question II.B.3) – Supposons que $(M(n))_{n\in\mathbb{N}}$ est croissante. Pour tout $n \in \mathbb{N}^*$, soit $k \in \mathbb{N}$ l'unique entier tel que $2^k \leqslant n < 2^{k+1}$. On a alors :

$$k \leqslant \log_2(n) < k+1 \qquad \text{donc} \qquad k = \left\lfloor \log_2(n) \right\rfloor \leqslant \log_2(n)$$

Ainsi :

$$M(n) \leqslant M(2^{k+1}) = (k+1)2^{k+1} \leqslant (\log_2(n)+1)2^{\log_2(n)+1} = 2n + \frac{2}{\ln 2} n\ln n = \mathcal{O}(n\ln n)$$

Question III.A.1) –

```
(* mediane3_QIIIA1 : 'a array -> int -> int -> int -> int *)
let mediane3_QIIIA1 tab i j k =
  if tab.(i) < tab.(j) then
    if tab.(k) < tab.(i) then i
    else if tab.(j) < tab.(k) then j
    else k
  else
    if tab.(i) < tab.(k) then i
    else if tab.(k) < tab.(j) then j
    else k;;
```

Question III.A.2) –

```
(* pseudo_mediane_QIIIA2 : int array -> int *)
(* Suppose que le tableau est de taille 3^k avec k >= 0.
 * La variable m prend les valeurs 3^0, 3^1, 3^2, 3^3, ... Pour chaque valeur de
 * m, on s'intéresse à des blocs de taille 3*m et les medianes de ces blocs sont
 * placées en positions (3*m)*i. *)
let pseudo_mediane_QIIIA2 tab =
  let m = ref 1 in
  let n = Array.length tab in
  while !m < n do
    let i = ref 0 in
    while !i < n do
      let j = mediane3_QIIIA1 tab !i (!i + !m) (!i + !m*2) in
      echange !i j tab;
      i := !i + !m*3;
    done;
    m := !m*3
  done;
  tab.(0);;
```

Question III.B.1) –

```
(* mediane3 : 'a -> 'a -> 'a -> 'a *)
let mediane3 i1 i2 i3 =
  let tab = [|i1; i2; i3|] in
  let k = mediane3_QIIIA1 tab 0 1 2 in
  tab.(k);;
```

Question III.B.2) –

```
(* construire3 : ternaire -> ternaire -> ternaire -> ternaire *)
let construire3 (a1: ternaire) (a2: ternaire) (a3: ternaire): ternaire =
  let r = mediane3 (racine a1) (racine a2) (racine a3) in
  N (r, a1, a2, a3);;
```

Question III.B.3) –

```
(* construire : int array -> int -> int -> ternaire *)
(* Suppose que j-i+1 est de la forme 3^k avec k >= 0 *)
let rec construire tab i j =
  if j < i then failwith "construire: j < i";
  if i = j then F tab.(i)
  else begin
      let n = j-i+1 in
      let a1 = construire tab i (i + n/3 - 1) in
      let a2 = construire tab (i + n/3) (i + 2*n/3 - 1) in
      let a3 = construire tab (i + 2*n/3) (i + n - 1) in
      construire3 a1 a2 a3
    end;;
```

Question III.B.4) –

```
(* pseudo_mediane_QIIIB4 : int array -> int *)
(* Suppose que le tableau est de taille 3^k avec k >= 0 *)
let pseudo_mediane_QIIIB4 tab =
  let a = construire tab 0 (Array.length tab - 1) in
  racine a;;
```

Question III.C.1) – On note $T(k)$ le temps d'exécution pour un tableau de taille 3^k et $C(k)$ le temps d'exécution sans les appels récursifs. D'après la description de l'algorithme donnée au début de la partie III :

- Si $k = 0$, alors il n'y a pas d'appel récursif.
- Sinon :
  - → On calcule les médianes de 3^{k-1} groupes de trois éléments en temps :
$$3^{k-1} \times \Theta(1) = \Theta(3^k).$$
  - → On applique récursivement l'algorithme sur un tableau de taille 3^{k-1}.

Ainsi :

$$C(k) = \Theta(3^k) \qquad \text{et} \qquad \begin{cases} T(0) = C(0) \\ \forall k \in \mathbb{N}^* : T(k) = C(k) + T(k-1) \end{cases}$$

Une récurrence immédiate sur $k \in \mathbb{N}$ donne :

$$T(k) = \sum_{i=0}^{k} C(i).$$

Donc :

$$T(k) = \Theta\left(\sum_{i=0}^{k} 3^i\right) = \Theta(3^k)$$

En conclusion :

L'algorithme de calcul d'une pseudo médiane s'exécute en temps $\Theta(n)$ où n est la taille du tableau.

Question III.C.2) – On le montre par récurrence sur $k \in \mathbb{N}$.

Initialisation. Pour $k = 0$, la valeur renvoyée est l'unique élément du tableau. Il y a donc exactement $1 = 2^0$ élément du tableau majoré par la valeur renvoyée.

Hérédité. Soit $k \in \mathbb{N}^*$. Supposons la propriété vraie au rang $k-1$ et montrons la au rang k.

Soit A l'ensemble des éléments du tableau (que l'énoncé suppose distincts deux à deux). L'algorithme regroupe les éléments de A par 3 et calcule les 3^{k-1} médianes de ces groupes. Ainsi :

$$A = \biguplus_{i=0}^{3^{k-1}-1} G_i \qquad \text{où les } G_i \text{ sont les groupes vérifiant} \qquad \operatorname{card}(G_i) = 3.$$

Pour chaque i, notons m_i la médiane de G_i. L'algorithme calcule ensuite récursivement la médiane des m_i que nous noterons m. Par l'hypothèse de récurrence, m majore au moins 2^{k-1} éléments parmi les m_i. Chaque m_i majore exactement 2 éléments de G_i et donc m majore au moins $2 \times 2^{k-1} = 2^k$ éléments du tableau.

Question III.C.3) – Étant donné un entier $k \in \mathbb{N}$ et T un tableau de taille $n = 3^k$, on note $\phi(T)$ le tableau de taille $3n$ où chaque élément t de T a été remplacé par les trois éléments $2t$, $2t+1$ et $t+2.3^k$. Par exemple, pour $k = 1$ si $T = [|t_0, t_1, t_2|]$, alors :

$$\phi(T) = [|\ 2t_0, 2t_0+1, t_0+6, 2t_1, 2t_1+1, t_1+6, 2t_2, 2t_2+1, t_2+6\ |]$$

Soit T_k le tableau défini par récurrence sur $k \in \mathbb{N}$:

$$\begin{cases} T_0 = [|0|] \\ \forall k \in \mathbb{N} : T_{k+1} = \phi(T_k) \end{cases}$$

Montrons que T_k convient.

$T_0 = [|0|]$
$T_1 = [|0, 1, 2|]$
$T_2 = [|0, 1, 6, 2, 3, 7, 4, 5, 8|]$
$T_3 = [|0, 1, 18, 2, 3, 19, 12, 13, 24, 4, 5, 20, 6, 7, 21, 14, 15, 25, 8, 9, 22, 10, 11, 23, 16, 17, 26|]$
...

⋆ Une récurrence immédiate sur $k \in \mathbb{N}$ montre que :

→ T_k est de taille 3^k

→ Les éléments de T_k appartiennent à $[\![0\,;3^k-1]\!]$.

→ Les éléments de T_k sont distincts deux à deux.

Le tableau T_k contient donc chaque élément de $[\![0\,;3^k-1]\!]$ une et une seule fois.

⋆ Montrons par récurrence sur $k \in \mathbb{N}$ que l'algorithme appliqué à T_k renvoie $2^k - 1$.

Initialisation. Pour $k = 0$, l'algorithme appliqué à $T_0 = [|0|]$ renvoie $0 = 2^0 - 1$.

Hérédité. Soit $k \in \mathbb{N}$ tel que l'algorithme appliqué à T_k renvoie $2^k - 1$. Lorsqu'on applique l'algorithme à T_{k+1} on a deux étapes :

→ L'algorithme regroupe les éléments par 3 puis calcule les 3^k médianes de ces groupes. Notons T' le tableau contenant ces 3^k médianes. En utilisant la définition de T_k, on remarque que chaque groupe contient $2t$, $2t+1$ et $t + 2.3^k$ où t est un élément de T. La médiane de ces trois éléments étant $2t+1$, on en déduit que T' est égal T où chaque élément t est remplacé par $2t+1$.

→ L'algorithme renvoie le résultat de l'appel récursif sur T'. Par hypothèse de récurrence, l'algorithme appliqué sur T_k renvoie $2^k - 1$. Par conséquent, l'algorithme appliqué sur T' renvoie $2 \times (2^k - 1) + 1 = 2^{k+1} - 1$.

Conclusion. L'algorithme appliqué sur T_k renvoie $2^k - 1$ qui majore exactement 2^k éléments de T_k.

Question III.C.4) − Soit T un tableau de taille $n = 3^k$ et m la valeur renvoyée par l'algorithme appliqué à T. D'après la question III.C.2), au moins 2^k éléments de T sont majorés par m. Une preuve similaire à celle de la question III.C.2) montre qu'au moins 2^k éléments de T sont minorés par m. De plus, si $\alpha = \ln 2/\ln 3$:

$$n^\alpha = (3^k)^{\ln 2/\ln 3} = 2^k$$

Ainsi :

La valeur renvoyée est bien une α-pseudo médiane avec $\alpha = \ln 2/\ln 3$ en prenant $K_1 = K_2 = 1$.

Question III.C.5) − Soit T un tableau de taille $n \in \mathbb{N}^*$ quelconque. Pour calculer une α-pseudo médiane où $\alpha = \ln 2/\ln 3$, on applique la procédure suivante :

→ On calcule $k \in \mathbb{N}$ l'unique entier tel que $3^k \leqslant n < 3^{k+1}$.

→ On calcule $T' = T[0..3^k - 1]$.

→ On exécute l'algorithme sur T' qui renvoie une certaine valeur m.

→ On renvoie m.

À l'aide de la question III.C.1), on remarque que cette procédure s'exécute en temps $\Theta(n)$. De plus, d'après la question III.C.4), au moins $3^{\alpha k}$ éléments de T' sont inférieurs (resp. supérieurs) à m. Or :

$$3^{\alpha k} = \frac{1}{3^\alpha} 3^{(k+1)\alpha} \geqslant \frac{1}{3^\alpha} n^\alpha = \frac{1}{2} n^\alpha$$

Si on pose $K_1 = K_2 = 1/2$, alors au moins $K_1 n^\alpha$ (resp. $K_2 n^\alpha$) éléments de T sont inférieurs (resp. supérieurs) à m. Donc cette procédure renvoie bien une α-pseudo médiane avec $\alpha = \ln 2/\ln 3$.

Question III.D.1) − Le même raisonnement que dans la question III.C.1) montre que :

La complexité de cet algorithme est en $\Theta(n)$ avec n la taille du tableau.

Si on note T le tableau et m la valeur renvoyée, le même raisonnement que dans la question III.C.2) montre qu'au moins 3^k éléments de T sont majorés (resp. minorés) par m. Ainsi, le même calcul que dans la question III.C.4) montre que :

Cet algorithme renvoie une α-pseudo médiane avec $\alpha = \ln 3/\ln 5$

Question III.D.2) – Pour tout $\ell \in \mathbb{N}^*$, on modifie l'algorithme en considérant des blocs de $2\ell+1$ éléments. Notons $\mathcal{A}_\ell$ cet algorithme modifié. Comme dans la question précédente, on peut montrer (en supposant que la longueur du tableau est une puissance de $2\ell+1$) que :

→ La complexité de $\mathcal{A}_\ell$ est en $\Theta(n)$ avec n la taille du tableau.

→ $\mathcal{A}_\ell$ renvoie une α-pseudo médiane avec $\alpha = \ln(\ell+1)/\ln(2\ell+1)$.

Comme dans la question III.C.5), on peut adapter $\mathcal{A}_\ell$ si le tableau a une longueur qui n'est pas une puissance de $2\ell+1$. Notons $\mathcal{A}'_\ell$ cet algorithme adapté.

On remarque que le limite croissante de $\alpha : \ell \mapsto \dfrac{\ln(\ell+1)}{\ln(2\ell+1)}$ est 1^-. Ainsi, pour tout $\varepsilon > 0$, il existe $\ell_\varepsilon \in \mathbb{N}^*$ tel que $1-\varepsilon \leqslant \alpha(\ell_\varepsilon) < 1$. L'algorithme $\mathcal{A}'_{\ell_\varepsilon}$ s'exécute bien en un coût linéaire et permet de calculer une $\alpha(\ell_\varepsilon)$-pseudo médiane d'un tableau. Étant donné que $1-\varepsilon \leqslant \alpha(\ell_\varepsilon)$, une $\alpha(\ell_\varepsilon)$-pseudo médiane est aussi une $(1-\varepsilon)$-pseudo médiane ; d'où le résultat.

Question *IV.A* – Lorsqu'on applique l'étape de séparation sur un tableau de taille n, on obtient deux tableaux de tailles n_1 et n_2 tels que $n = 1 + n_1 + n_2$. Qualitativement :

→ L'exécution est rapide lorsqu'à chaque étape $|n_1 - n_2|$ est petite.

→ L'exécution est lente lorsqu'à chaque étape $|n_1 - n_2|$ est grande.

En particulier, le pire cas est atteint lorsqu'à chaque étape, l'un des deux tableaux est vide. En utilisant les mêmes arguments que dans la question *II.A*, la complexité pire cas est en $\Theta(n^2)$ et donc :

$$C(n) = \mathcal{O}(n^2).$$

Lorsque le pivot est une 1/2-pseudo médiane, il existe K_1 et K_2 deux constantes strictement positives telles que :

$$n_1 \geqslant K_1\sqrt{n} \qquad\qquad n_2 \geqslant K_2\sqrt{n}$$

Étant donné qu'on cherche à majorer $C(n)$, on s'intéresse au pire cas. La quantité $|n_1 - n_2|$ est maximale lorsque $n_1 = K_1\sqrt{n}$ ou $n_2 = K_2\sqrt{n}$. Traitons le cas $n_1 = K_1\sqrt{n}$ (l'autre cas se traite de la même façon) :

→ Le calcul de la 1/2-pseudo médiane se fait en temps $\mathcal{O}(n)$.

→ La séparation du tableau se fait en temps $\mathcal{O}(n)$.

→ L'appel récursif sur le tableau de taille $n_1 = K_1\sqrt{n}$ se fait en temps $C\Big(K_1\sqrt{n}\Big) = \mathcal{O}\Big((K_1\sqrt{n})^2\Big) = \mathcal{O}(n)$.

→ Qualitativement, la fonction C est croissante. Ainsi, l'appel récursif sur le tableau de taille $n_2 = n - K_1\sqrt{n} - 1$ se fait en temps :

$$C\Big(n - K_1\sqrt{n} - 1\Big) \leqslant C\Big(n - K_1\sqrt{n}\Big)$$

En résumé :

$$C(n) = C\Big(n - K_1\sqrt{n}\Big) + \mathcal{O}(n) \tag{1}$$

Qualitativement, comme le $\mathcal{O}$ absorbe les constantes :

$$C(n) = C\Big(n - \sqrt{n}\Big) + \mathcal{O}(n) \tag{2}$$

Notons que l'égalité (2) se montre formellement à partir de l'égalité (1) en utilisant le même genre de raisonnement que dans la question *IV.B*.

En conclusion, par définition de la notation $\mathcal{O}$, il existe une constante K telle que :

$$C(n) \leqslant C\Big(n - \sqrt{n}\Big) + Kn$$

Question *IV.B* – Soit $n \in \mathbb{N}$.

⋆ Comme le suggère l'énoncé, on définit la suite $(\alpha_k)_{k\in\mathbb{N}}$ par :

$$\alpha_0 = n \text{ et } \forall k \in \mathbb{N} : \alpha_{k+1} = \begin{cases} \alpha_k - \sqrt{\alpha_k} & \text{si } \alpha_k \geqslant 1 \\ 0 & \text{sinon} \end{cases}$$

Soit $k \geqslant \sqrt{n/2}$ un entier et supposons par l'absurde que $\alpha_k > n/2$. La suite $(\alpha_k)_{k\in\mathbb{N}}$ étant à valeurs dans $\{0\} \cup [\,1\,;+\infty[$ et étant décroissante, pour tout $i \in [\![0, k-1]\!]$:

$$\alpha_{i+1} = 0 \qquad \text{ou} \qquad \alpha_{i+1} = \alpha_i - \sqrt{\alpha_i} \leqslant \alpha_i - \sqrt{\alpha_k} < \alpha_i - \sqrt{\frac{n}{2}}$$

S'il existe $i \in [\![0, k-1]\!]$ tel que $\alpha_{i+1} = 0$, alors $\alpha_k = 0$ ce qui contredit $\alpha_k > n/2$. Sinon, une récurrence immédiate sur $i \in [\![0, k]\!]$ montre que :

$$\alpha_i \leqslant n - i\sqrt{\frac{n}{2}}$$

En particulier pour $i = k$:

$$\alpha_k \leqslant n - k\sqrt{\frac{n}{2}} \leqslant n - \sqrt{\frac{n}{2}}\sqrt{\frac{n}{2}} = \frac{n}{2}$$

Ce qui contredit $\alpha_k > n/2$.

⋆ En exploitant l'inégalité de la question précédente, une récurrence sur $k \in \mathbb{N}$ montre que :

$$C(n) \leqslant C(\alpha_k) + K\sum_{i=0}^{k-1} \alpha_i$$

En particulier, avec $k = \left\lceil \sqrt{n/2} \right\rceil$:

$$\begin{aligned} C(n) &\leqslant C(\alpha_k) + K\sum_{i=0}^{k-1} \alpha_i \\ &\leqslant C(n/2) + K\sum_{i=0}^{k-1} n \\ &= C(n/2) + Knk \\ &= C(n/2) + \mathcal{O}(n^{3/2}). \end{aligned}$$

Question *IV.C* – Le théorème maître appliqué à la relation de récurrence de la question précédente donne :

$$C(n) = \mathcal{O}(n^{3/2})$$

L'utilisation d'une 1/2-pseudo médiane permet d'obtenir un tri rapide avec une complexité pire cas en $\mathcal{O}(n^{3/2})$.

Question *IV.D* – Lorsqu'on utilise une α-pseudo médiane avec $\alpha = \dfrac{\ln 2}{\ln 3}$, on peut raisonnablement espérer que la complexité est meilleure qu'avec une 1/2-pseudo médiane. Si on note $C'(n)$ le temps d'exécution, on obtient :

$$C'(n) = \mathcal{O}(n^{3/2}).$$

En utilisant cette borne et le même raisonnement que dans les questions précédentes, montrons la borne plus précise :

$$C'(n) = \mathcal{O}(n^{2-\alpha})$$

Le même raisonnement que dans la question $IV.A$ donne :

$$\begin{aligned} C'(n) &= C'(n - n^\alpha) + C'(n^\alpha) + \mathcal{O}(n) \\ &= C'(n - n^\alpha) + \mathcal{O}(n^{3\alpha/2}) + \mathcal{O}(n). \end{aligned}$$

Or $3\alpha/2 \approx 0,94 < 1$ et donc :

$$C'(n) \leqslant C'(n - n^\alpha) + K'n \text{ où } K' \text{ est une constante.}$$

Le même raisonnement que dans la question $IV.B$ montre que :

$$C'(n) = C(n/2) + \mathcal{O}(n^{2-\alpha})$$

Le même raisonnement que dans la question $IV.C$ montre que :

$$C'(n) = \mathcal{O}(n^{2-\alpha})$$

En conclusion :

Pour un tri rapide effectué en calculant une α-pseudo médiane où $\alpha = \ln 2 / \ln 3$, on peut raisonnablement espérer une complexité en $\mathcal{O}(n^{2-\alpha})$ où $2 - \alpha \approx 1,37$.