Concours commun Centrale-Supelec 2012

Corrigé de l'épreuve d'informatique

I Tri rapide d'un tableau

I.A

```
let echange i j t = let x=t.(i) in t.(i) <- t.(j); t.(j) <- x;;
```

I.B On peut facilement programmer le tri bulle : si t est un tableau de longueur n, on commence par faire varier un indice j de la valeur 0 à la valeur $n-2$, en échangeant $t.(j)$ et $t.(j+1)$ quand $t.(j) > t.(j+1)$. Après cette première boucle, la case $t.(n-1)$ contient la valeur maximale contenue initialement dans t. On recommence ensuite, mais avec j qui varie de 0 à $n-3$, et ainsi de suite jusqu'à ce que j varie de 0 à 0 : le tableau est alors trié ; le calcul se fait en temps quadratique (on effectue exactement $(n-1)+(n-2)+\cdots+1 = n(n-1)/2$ comparaisons).

```
let tri_bulle t = let n = vect_length t in
   for i=2 to n do
      for j=0 to n-i do
         if t.(j)>t.(j+1) then
            echange j (j+1) t
      done;
   done;;
```

I.C.1) Pour séparer le tableau $t[a..b]$ par rapport à la valeur $\alpha = t.(a)$, on utilise la méthode suivante :

- on crée deux pointeurs i et j initialisés aux valeurs $a+1$ et b ; nous allons maintenir, en incrémentant i ou en décrémentant j, l'invariant de boucle suivant :
  - $t.(a)$ contient toujours la valeur α ;
  - les valeurs $t.(k)$ pour $a < k < i$ sont inférieures ou égales à $t.(a)$;
  - les valeurs $t.(k)$ pour $j < k < b$ sont supérieures strictement à $t.(a)$;
  - $t[a..b]$ contient globalement les mêmes valeurs au cours du calcul.

- tant que $i \leq j$, deux cas se produisent :
  - si $t.(i) \leq t.(a)$, on incrémente i ;
  - sinon, on échange $t.(i)$ et $t.(j)$ et on décrémente j.

Comme $j-i$ diminue d'une unité à chaque boucle, on arrive à la situation $j = i-1$. Deux cas sont alors possibles :

- si $j > a$, on échange dans t les valeurs $t.(a)$ et $t.(j)$;

- si $j = a$, c'est que tous les éléments de $t[a+1..b]$ étaient strictement plus grands que $t.(a)$.

Dans les deux cas, α est en position j dans le tableau final.

I.C.2) La méthode exposée ci-dessus (en remarquant que les deux derniers cas peuvent se regrouper, puisque l'appel `echange j j t` ne modifie pas le tableau) donne directement la fonction :

```
let separation t a b = let i = ref (a+1) and j = ref b in
   while (!i)<= (!j) do
      if t.(!i)<= t.(a) then
         incr i
      else
         begin
         echange (!i) (!j) t;
         decr j
         end
   done;
   echange a (!j) t;
   !j;;
```

I.C.3) La fonction utilise une fonction auxiliaire `tri` qui, appliquée à deux entiers i et j, trie le sous-tableau $t[i..j]$.

```
let tri_rapide t =
   let rec tri i j =
      if i<j then
         begin
         let k = separation t i j in
            tri i (k-1);
            tri (k+1) j
         end
         in
      tri 0 ((vect_length t)-1) ;;
```

II Étude de complexité

II.A Pour répondre à cette question, nous sommes obligés de faire l'hypothèse qu'après séparation d'un tableau croissant (resp. décroissant) $t[i_1..i_2]$ par rapport à la valeur $t[i_1]$, le tableau $t[i_1+1..i_2]$ (resp. $t[i_1..i_2-1]$) est croissant ou décroissant. Le temps de calcul ne dépend alors que de la longueur du tableau : nous pouvons noter $C(n)$ le nombre de comparaisons effectuées par le tri rapide appliqué à un tableau "monotone" de longueur n. La séparation découpe un tel tableau en un tableau de taille nulle et un tableau de taille $n-1$ monotone en effectuant $n-1$ comparaisons, d'où :

$$C(n) = (n-1) + (n-2) + \cdots + 1 + C(0) = \frac{n(n-1)}{2} = \Theta(n^2).$$

Le tri rapide est donc quadratique quand le tableau est déjà trié, dans l'ordre croissant ou dans l'ordre décroissant.

II.B.1) L'énoncé est ici très approximatif, puisque le tableau de taille n est découpé en deux sous-tableaux de tailles n_1 et n_2 avec $n_1 + n_2 = n - 1$. On pourrait évidemment choisir $n_1 + n_2 = n$, en laissant le pivot dans l'un des deux sous-tableaux, mais ce ne serait pas très heureux puisque le pivot est à coup sûr bien placé après l'opération de séparation. Les hypothèses faites permettent d'écrire, pour tout $n \geq 0$:

$$\begin{cases} C(2n+1) = 2C(n) + 2n \\ C(2n+2) = C(n) + C(n+1) + 2n + 1 \end{cases}$$

où $C(n)$ est le nombre de comparaisons effectuées par l'algorithme dans le cas où chaque séparation coupe le tableau en deux parts égales (à une unité près). En admettant que C est croissante, la seconde égalité donne :

$$C(2(n+1)) \leq 2C(n+1) + 2(n+1)$$

La suite $\big(M(2^k)\big)_{k\geq 0}$ définie par $M(1) = 0$ et $M(2^{k+1}) = 2M(2^k) + 2 \times 2^k$ pour tout $k \geq 0$ vérifie bien la relation proposée, avec $\alpha = 2$, et une récurrence immédiate donne $C(2^k) \leq M(2^k)$ pour tout $k \geq 0$.

II.B.2) Montrons par récurrence sur k que $M(2^k) = 2^k M(1) + \alpha\, k\, 2^{k-1}$ pour tout $k \geq 0$.

- l'égalité est vérifiée quand $k = 0$;
- soit $k \geq 0$ et supposons que l'égalité est vraie au rang k. Alors :

$$M(2^{k+1}) = 2M(2^k) + \alpha\, 2^k = 2^{k+1}M(1) + \alpha\, k\, 2^k + \alpha\, 2^k = 2^{k+1}M(1) + \alpha\,(k+1)\, 2^k$$

  et l'égalité est vérifiée au rang $k+1$.

II.B.3) Les $M(n)$ ne sont pas définis quand n n'est pas une puissance de 2 : revenons donc à la suite (C_n), toujours supposée croissante. Pour $n \geq 1$, notons k l'unique entier naturel tel que $2^k \leq n < 2^{k+1}$. nous avons :

$$C(n) \leq C(2^{k+1}) \leq M(2^{k+1}) = 2^{k+1}M(1) + \alpha\,(k+1)\, 2^k = O(n \ln n)$$

car $2^k = O(n)$ et $k = O(\ln n)$.

III Recherche d'une pseudo médiane

III.A.1) On obtient facilement :

```
let mediane t i j k =
   if t.(i)<=t.(j) then
      if t.(j)<= t.(k) then
         j
      else
         if t.(k)<=t.(i) then
            i
         else
            k
   else
      if t.(j)>=t.(k) then
         j
      else
         if t.(k)>=t.(i) then
            i
         else
            k;;
```

III.A.2) Nous utilisons une variable locale u qui est multipliée par 3 à chaque appel à la fonction auxiliaire récursive `calcul`. Au début du calcul, u prend la valeur 1 et on applique `calcul` tant que $u < n$. Quand on atteint l'égalité $u = n$, la pseudo-médiane est dans la case $t.(0)$.

La fonction `calcul` n'a pas d'argument : elle calcule la position j de la médiane des valeurs $t.(0)$, $t.(u)$, $t.(2u)$ et échange les contenus $t.(0)$ et $t.(j)$; elle calcule ensuite la position j de la médiane des valeurs $t.(3u)$, $t.(4u)$, $t.(5u)$ et échange les contenus $t.(3u)$ et $t.(j)$, et ainsi de suite jusqu'à avoir parcouru tout le tableau t, par saut de longueur u. À la fin de ce parcours, u est alors changé en $3u$. Cela donne :

```
let pseudo_mediane_1 t =
   let n=vect_length t and u=ref 1 in
   let calcul() = let i = ref 0 in
      while (!i)<n do
         let j = mediane t (!i) ((!i)+(!u)) ((!i)+2*(!u)) in
         echange (!i) j t;
         i := (!i)+3*(!u);
      done;
      u := 3*(!u) in
   while (!u)<n do
      calcul()
   done;
   t.(0);;
```

III.B.1) La fonction reprend le schéma de celle du III.A :

```
let mediane3 (a,b,c) =
   if a<= b then
      if b<= c then
         b
      else
         max a c
   else
      if c<= b then
         b
      else
         min a c;;
```

III.B.2) Cette fonction ne pose aucune difficulté :

```
 let construire_arbre t1 t2 t3 = let a = mediane3 (racine t1,racine t2,racine t3) in
    N(a,t1,t2,t3);;
```

III.B.3) La fonction s'écrit facilement (on suppose que la longueur $j - i + 1$ de la sous-liste est une puissance de 3) :

```
let rec construire t i j = match j-i+1 with
   1 -> F(t.(i))
   | m -> let a=m/3 in
          let a1=construire t i (i+a-1) in
          let a2=construire t (i+a) (i+2*a-1) in
          let a3=construire t (i+2*a) (i+3*a-1) in
            construire_arbre a1 a2 a3;;
```

III.B.4) On suppose que la longueur n de t est une puissance de 3 : la pseudo médiane est alors la valeur stockée à la racine de l'arbre associé à t.

```
let pseudo_mediane_2 t = racine (construire t 0 ((vect_length t)-1) );;
```

III.C.1) Notons $T(k)$ le temps de construction de l'arbre ternaire associé à un tableau de longueur 3^k. Nous avons $T(0) = 1$ et $T(k) = 1 + 3\,T(k-1)$, d'où $T(k) = 3^k - \dfrac{1}{2}$: le temps de calcul d'une pseudo-médiane est inéaire par rapport à la longueur du tableau.

III.C.2 Montrons par récurrence qu'un arbre ternaire de hauteur k possède au moins 2^k feuilles dont les étiquettes sont supérieures ou égales à la valeur stockée à la racine de l'arbre :

- si $k = 0$, l'arbre est réduit à une feuille et le résultat est évident ;
- soit $k \geq 0$ et supposons que le résultat est vrai pour tous les arbres ternaires de hauteur k ; soit $a = N(\alpha, a1, a2, a3)$ un arbre ternaire de hauteur $k+1$. Comme α est la médiane des racines α_i des arbres a_i, il existe au moins deux indices i tels que $\alpha \geq \alpha_i$: par symétrie, nous supposerons que $\alpha \geq \alpha_1$ et $\alpha \geq \alpha_2$. Par hypothèse de récurrence, a_1 (resp. a_2) possède au moins 2^k feuilles supérieures ou égales à α_1 (resp. à α_2), et donc également supérieures à α : nous obtenons ainsi au moins 2^{k+1} feuilles de a supérieures à α, et la propriété est vraie au rang $k+1$.

III.C.3) Pour $k \in \mathbb{N}$, nous définissons les arbre a_k, b_k par récurrence sur k :

$$a_0 = F(1),\ b_0 = F(0),\ a_{k+1} = N(1, a_k, a_k, b_k) \ \text{ et } b_{k+1} = N(0, b_k, b_k, b_k).$$

Le tableau t_k est alors le tableau obtenu en rangeant les feuilles de a_k de gauche à droite. Il est clair que a_k est l'arbre construit par l'algorithme à partir de t_k : la pseudo médiane calculée par l'algorithme appliqué à t_k est donc égale à 1, et t_k contient exactement 2^k valeurs supérieures ou égales à 1(t_k contient 2^k fois la valeur 1 et $3^k - 2^k$ fois la valeur 0).

III.C.4) Une preuve identique permet de montrer qu'il y a également au moins 2^k éléments du tableau inférieurs ou égaux à la valeur retournée : comme $2^k = 2^{\ln_3 n} = n^{\ln 2/\ln 3}$, l'algorithme retourne bien une $\frac{\ln 2}{\ln 3}$ - pseudo médiane (avec $K_1 = K_2 = 1$).

III.C.5) Si t est de longueur n, on note k l'unique entier tel que $3^k \leq n < 3^{k+1}$. Nous avons donc :

$$\frac{\ln n}{\ln 3} - 1 < k \leq \frac{\ln n}{\ln 3}$$

On calcule alors par l'algorithme précédent une pseudo médiane m du tableau $t_1 = t[0..3^k - 1]$. Il existe au moins 2^k éléments de t_1 supérieurs ou égaux à m, donc :

$$\text{Card}\ \{i,\ t[i] \geq m\} \geq 2^k \geq 2^{\frac{\ln n}{\ln 3} - 1} = \frac{1}{2}\, n^{\ln 2/\ln 3}$$

De même, $\text{Card}\ \{i,\ t[i] \leq m\} \geq \frac{1}{2}\, n^{\ln 2/\ln 3}$ et m est une $\frac{\ln 2}{\ln 3}$ - pseudo médiane de t (avec $K_1 = K_2 = 1/2$).

Les algorithmes de la question B fonctionnent également avec des tables de longueurs quelconques. En effet, dans la fonction `construire`, on coupe le tableau en 3 en oubliant éventuellement un ou deux éléments à chaque appel récursif : on obtient ainsi à la fin du calcul l'arbre associé à un tableau $t_2$extrait de t qui est de longueur 3^k et une $\ln 2/\ln 3$ - pseudo médiane pour t_2 l'est également pour t. Par exemple, quand $n = 13$, on oublie $t[12]$, puis $t[3]$, $t[7]$ et $t[11]$:

```
# let t = [|0;1;2;3;4;5;6;7;8;9;10;11;12|] in construire t 0 12;;
- : ternaire =
 N (5, N (1, F 0, F 1, F 2), N (5, F 4, F 5, F 6), N (9, F 8, F 9, F 10))
```

et 5 est une $\ln 2/\ln 3$ - pseudo médiane pour $[|0;1;2;3;4;5;6;7;8;9;10;11;12|]$.

III.D.1) On montre de la même façon que dans le cas précédent que le temps de calcul est linéaire et qu'un tableau de longueur $n = 5^k$ possède au moins 3^k valeurs supérieures ou égales (resp. inférieures ou égales) à la valeur retournée par l'algorithme (la médiane de cinq valeurs est supérieure ou égale (resp. inférieure ou égale) à au moins trois des cinq valeurs). On en déduit que l'algorithme renvoie une $\ln 3/\ln 5$ - pseudo médiane (et la généralisation à un tableau de longueur quelconque se fait comme à la question C.5).

III.D.2) Plus généralement, en utilisant des blocs de $(2N+1)$ éléments, on obtient en temps linéaire une α_N - pseudo médiane avec $\alpha_N = \ln(N+1)/\ln(2N+1)$ pour des tableaux de longueur quelconque. Comme α_N tend vers 1^- quand N tend vers l'infini, il est possible de choisir N tel que $1-\varepsilon \leq \alpha_N$: on obtient ainsi en temps linéaire une $(1-\varepsilon)$ - pseudo médiane (si $\beta \leq \alpha$, une α - pseudo médiane est également une β - pseudo médiane).

IV Gain apporté par la pseudo médiane

Cette partie est rédigée de façon très vague et nous essaierons de donner des arguments "raisonnablement convaincants".

IV.A Soit t un tableau de longueur n. Notons n_1 et n_2 les tailles des deux listes auxquelles on applique récursivement l'algorithme de tri rapide ; nous avons donc :

$$n = n_1 + n_2 + 1,\ n_1 \geq K_1\sqrt{n} \text{ et } n_2 \geq K_2\sqrt{n}.$$

On en déduit donc que $n_1 \leq n - K_2\sqrt{n} - 1$ et $n_2 \leq n - K_1\sqrt{n} - 1$. Le pire des cas est sans doute atteint quand n_1 (ou n_2) est maximal (cela resterait à justifier !), i.e. quand $n_1 = n - K_2\sqrt{n} - 1$ ou $n_2 \leq n - K_1\sqrt{n} - 1$. Pour obtenir l'inégalité proposée, on va supposer que $K_1 = K_2 = 1$ (oubli de l'énoncé ?) et négliger le terme -1 : on peut donc estimer que l'on a

$$C(n) = C(n - \sqrt{n}) + C(\sqrt{n} - 1) + O(n)$$

où le terme $O(n)$ correspond au temps mis à calculer la $1/2$ - pseudo médiane et au temps de séparation par rapport à cette pseudo médiane. Le temps de traitement de la "petite" liste de longueur $\sqrt{n}$ est, dans le pire des cas, quadratique en $\sqrt{n}$, donc $C(\sqrt{n} - 1) = O(n)$ et :

$$C(n) \leq C(n - \sqrt{n}) + O(n)$$

Ainsi, on peut raisonnablement conjecturer qu'il existe une constante K telle que

$$\forall n \in \mathbb{N},\ C(n) \leq C(n - \sqrt{n}) + Kn.$$

On remarquera qu'une relation correcte serait plutôt :

$$C(n) \leq Kn + \max_{K_1\sqrt{n} \leq n_1 \leq n - K_2\sqrt{n} - 1} (C(n_1) + C(n - 1 - n_1))$$

la suite C étant alors majorée par la suite M définie par :

$$\begin{cases} M(n) = C(n) & \text{pour } n \leq n_0 \\ M(n) = Kn + \max\limits_{K_1\sqrt{n} \leq n_1 \leq n - K_2\sqrt{n} - 1} (M(n_1) + M(n - 1 - n_1)) & \text{pour tout } n > n_0 \end{cases}$$

où n_0 est assez grand pour que la définition soit correcte ... cela explique pourquoi une analyse rigoureuse est délicate !

IV.B) La suite (α_k) est décroissante. Si elle ne stationnait pas à la valeur 0, cette suite aurait une limite $\ell \geq 1$ telle que $\ell = \ell - \sqrt{\ell}$, ce qui est absurde. On en déduit donc que $\alpha_k = 0$ à partir d'un certain rang. Ainsi, il existe un entier $k_0 \geq 0$ tel que

$$n = \alpha_0 > \alpha_1 > \ldots > \alpha_{k_0} > \frac{n}{2} \geq \alpha_{k_0+1}.$$

En notant $f : \ x \longmapsto x - \sqrt{x}$ et en remarquant que f est croissante sur $[1, +\infty[$, nous avons :

$$\alpha_{k_0+1} = f(\alpha_{k_0}) > f\left(\frac{n}{2}\right) = \frac{n}{2} - \sqrt{\frac{n}{2}}$$

D'autre part :

$$\begin{aligned}
\alpha_{k_0+1} &= (\alpha_{k_0+1} - \alpha_{k_0}) + (\alpha_{k_0} - \alpha_{k_0-1}) + \cdots + (\alpha_1 - \alpha_0) + \alpha_0 \\
&= -\sqrt{\alpha_{k_0}} - \sqrt{\alpha_{k_0-1}} - \cdots - \sqrt{\alpha_0} + \alpha_0 \\
&< -(k_0+1)\sqrt{\frac{n}{2}} + n
\end{aligned}$$

Nous en déduisons :

$$\frac{n}{2} - \sqrt{\frac{n}{2}} < -(k_0+1)\sqrt{\frac{n}{2}} + n$$

ce qui s'écrit :

$$k_0 < \sqrt{\frac{n}{2}}.$$

Nous avons ainsi démontré que pour tout entier k tel que $k \geq \sqrt{\frac{n}{2}}$, on a $k \geq k_0+1$ et donc $\alpha_k \leq \alpha_{k_0+1} < n/2$.

Il reste à fixer k tel que $k \geq \sqrt{n/2} > k-1$ pour obtenir :

$$\begin{aligned}
0 \leq C(n) - C(n/2) &= C(\alpha_0) - C(n/2) \\
&= C(\alpha_1) - C(n/2) + K\alpha_0 \\
&= C(\alpha_2) - C(n/2) + K(\alpha_1 + \alpha_0) \\
&\vdots \\
&= \underbrace{C(\alpha_k) - C(n/2)}_{\leq 0} + K(\alpha_{k-1} + \cdots + \alpha_1 + \alpha_0) \\
&\leq K\,k\,\alpha_0 \\
&\leq K\left(1 + \sqrt{\frac{n}{2}}\right) n = O(n^{3/2})
\end{aligned}$$

qui est le résultat demandé.

IV.C Il existe ainsi une constante K_1 telle que $C(n) \leq C(n/2) + K_1 n^{3/2}$ pour tout n pair. On obtient alors facilement, en notant $q = 2^{3/2}$:

$$\forall\, k \geq 0,\ C(2^k) \leq C(2^0) + K_1(q + q^2 + \cdots + q^k) = C(1) + \frac{qK_1}{q-1}(q^k - 1) \leq C(1) + \frac{qK_1}{q-1}\,q^k$$

Pour n entier, on fixe ensuite k tel que $2^k \leq n < 2^{k+1}$ et on obtient (en admettant que C est croissante) :

$$C(n) \leq C(2^{k+1}) \leq C(1) + \frac{q^2 K_1}{q-1}\,q^k \leq C(1) + \frac{q^2 K_1}{q-1}\,q^{\ln_2(n)} = O(n^{\ln_2(q)}) = O(n^{3/2})$$

L'utilisation d'une 1/2- pseudo médiane permet donc de remplacer le temps quadratique dans le pire des cas par un temps en $O(n^{3/2})$.

Remarque : un petit calcul fait avec Maple montre que la suite (M_n) définie à la fin de la question IV.A (avec $K_1 = K_2 = 1$ et $n_0 = 6$) semble bien être de l'ordre de $n^{3/2}$:

```
> M := proc(n)
  option remember;
  if n<=6 then
     1
  else
     n+max([seq(M(i)+M(n-1-i),i=ceil(sqrt(n))..floor(n-sqrt(n)-1))])
  fi
  end:

> seq(evalf(M(2^k)/(2^k)^(3/2)),k=8..12);
```

$$0.6965332032,\ 0.7060709802,\ 0.7104492189,\ 0.7110126142,\ 0.7107276915$$

IV.D) Essayons de généraliser cette preuve déjà approximative au cas d'une α- pseudo médiane, avec $\alpha = \frac{\ln 2}{\ln 3}$. La même démarche que précédemment, avec les mêmes objections, donne :

$$C(n) \leq C(n - n^\alpha) + C(n^\alpha - 1) + O(n)$$

mais en écrivant que $C(n^\alpha) = O(n^{2\alpha})$ (le pire des cas est quadratique), nous obtenons (car $1/2 < \alpha$) :

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

et la méthode utilisée à la question IV.B permet d'obtenir la majoration

$$C(n) = C(n/2) + O(n^{1+\alpha}$$

qui donne comme à la question IV.C la majoration

$$C(n) = O(n^{1+\gamma})$$

qui n'a aucun intérêt, puisque $C(n)$ devrait décroître quand α croît ! C'est donc que la majoration $C(n^\alpha) = O(n^{2\alpha})$ est mauvaise et que l'analyse doit être un peu plus fine. Pour obtenir un résultat, nous allons faire l'hypothèse qu'il existe une constante γ telle que $C(n) = O(n^\gamma)$ et $\alpha\gamma \leq 1$. Nous aurons alors $C(n^\alpha) = O(n^{\alpha\gamma}) = O(n)$ et il existera une constante K telle que :

$$C(n) \leq C(n - n^\alpha) + Kn$$

En introduisant, comme à la question IV.C, la suite (α_k) définie par $\alpha_0 = n$ et la récurrence $\alpha_{k+1} = \alpha_k - \alpha_k^\alpha$, on démontre que

$$\forall\, k \geq \left(\frac{n}{2}\right)^{1-\alpha},\ \alpha_k \leq \frac{n}{2}$$

En fixant k tel que $k \geq \left(\frac{n}{2}\right)^{1-\alpha} > k - 1$, nous avons alors

$$0 \leq C(n) - C(n/2) \leq K\, k\, n = O(n^{2-\alpha}$$

On termine alors comme à la question IV.C, en notant $q = 2^{2-\alpha}$:

$$C(n) \leq C(2^{k+1}) \leq C(2^k) + K_1\, q^{k+1} \leq C(1) + \frac{q^2 K_1}{q-1}\, q^k \leq C(1) + \frac{q^2 K_1}{q-1}\, q^{\ln_2 n} = O(n^{2-\alpha})$$

en remarquant que la valeur $\gamma = 2 - \alpha$ vérifie bien $\alpha\gamma \leq 1$, ce qui permet de ne pas rejeter l'hypothèse faite (sans pour autant la valider).

Pour tout $\varepsilon > 0$, on peut donc raisonnablement conjecturer que le choix d'une $(1-\varepsilon)$ - pseudo médiane permette au tri rapide de fonctionner en un temps $O(n^{1+\varepsilon}$ dans le pire des cas. Ainsi, un tri rapide effectué en calculant une $\frac{\ln 2}{\ln 3}$ - pseudo médiane s'effectuera en un temps dans le pire des cas en $O(n^{1.36907\ldots})$.

Il est facile de programmer cette méthode : la fonction `pseudo_mediane` modifie le tableau $t[a..b]$ de sorte que la $\ln 2/\ln 3$-pseudo médiane se retrouve en position a après calcul (même méthode qu'à la question III.A.2).

```
 let pseudo_mediane t a b=
   let u=ref 1 in
   let calcul() = let i = ref a in
      while (!i)+2*(!u)<b do
      let j = mediane t (!i) ((!i)+(!u)) ((!i)+2*(!u)) in
      echange (!i) j t;
      i := (!i)+3*(!u);
      done;
      u := 3*(!u) in
   while (!u)<b-a+1 do calcul() done;;

let tri_rapide_pseudo_mediane t =
  let rec tri i j=
      if i<j then
         begin
         pseudo_mediane t i j;
         let k = separation t i j in
            tri i (k-1);
            tri (k+1) j
         end
         in
      tri 0 ((vect_length t)-1) ;;
```

On pourra se convaincre du gain d'efficacité en lançant le calcul ci-dessous, sans oublier que dans le cas d'une liste quelconque, le tri rapide basique prend moins de temps :

```
# let n=10000000 ;;
n : int = 100000

# let t=make_vect 100000 0;;
t : int vect =
[|0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
   0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
   ...|]

# for k=0 to n-1 do t.(k)<- k done;;
- : unit = ()

# tri_rapide t;;
- : unit = ()

# tri_rapide_pseudo_mediane t;;
- : unit = ()
```

```
# let t1=make_vect 100000 0;;
t1 : int vect =
[|0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
   0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
   ...|]

# for k=0 to n-1 do
      t.(k)<- random__int 10000000;
      t1.(k) <- t.(k) done;;
- : unit = ()

# tri_rapide t;;
- : unit = ()

# tri_rapide_pseudo_mediane t1;;
- : unit = ()
```