Devoir surveillé 1 b

Centrale 2012

Option informatique MP1, MP2 & MP3

Choix du pivot dans le tri rapide

L'objet de ce problème est le choix d'un pivot dans l'algorithme du tri rapide.
On s'intéresse au tri en ordre croissant d'un tableau d'entiers.
Les entiers seront supposés distincts.
On rappelle que la longueur d'un tableau est donnée par `Array.length` en Ocaml.

Le sujet a été modifié afin d'en corriger quelques imprécisions. Cependant les dernières questions (24 à 27) demandent explicitement des réponses "raisonnablement" convaincantes : il conviendra d'essayer de donner une idée de la démonstration en précisant clairement les hypothèses que l'on admet.

I Introduction

Question 1
Écrire une fonction `echange` telle que `echange t i j`
réalise l'échange des deux éléments d'indices i et j du tableau t.

```
echange : int -> int -> int array -> unit
```

Question 2
Décrire un algorithme simple de tri ; donner sa complexité dans le pire des cas en fonction du nombre de comparaisons. Le programmer.

```
tri : 'a array -> unit
```

II Tri rapide

L'algorithme du tri rapide sur un tableau consiste en une première étape où on permute les éléments du tableau de sorte que les éléments plus petits (respectivement plus grands) qu'un pivot p choisi dans le tableau soient placés avant (respectivement après) cet élément p dans le tableau.
On exécute ensuite récursivement le tri rapide sur les deux parties du tableau ainsi séparées par p.
Le sujet discute du choix du pivot.
Dans un premier temps le pivot sera le premier élément de la partie à séparer.
Par exemple, le tableau `[|19; 7; 17; 14; 22; 5; 26; 21; 2; 12|]` peut devenir après la première étape, 19 étant le pivot, `[|7; 17; 14; 12; 5; 2; 19; 21; 26; 22|]`.
Notons que les éléments inférieurs (resp. supérieurs) à 19 peuvent être permutés entre eux.
Le point crucial est qu'ils soient situés avant (resp. après) 19 dans le tableau.
Un des intérêts importants de cet algorithme est qu'il peut être réalisé en place : le tableau ne sera jamais recopié. Pour cela, les appels récursifs prendront comme arguments les indices délimitant la partie à trier.

On propose la fonction

```
let separer t a b =
  let min = ref a and max = ref b and p = t.(a) in
    while !min  < !max do
      if p < t.(!min+1)
      then (echange (!min+1) !max t; max := !max -1)
      else (echange (!min+1) !min t; min := !min+1) done;
  !min;;
```

Question 3
Quel est le résultat de `separer t 2 6` avec
`t = [|1; 3; 15; 14; 17; 20; 11; 29; 31; 24|]` ? Que vaut alors `t` ?

Question 4
Prouver qu'on a, à chaque début de la boucle, `t.(!min)` qui vaut p.
Que renvoie la fonction ?

Question 5
Prouver que la fonction termine (on pourra proposer un variant de boucle simple).
Combien de comparaison sont effectuées ?

Question 6
Que peut-on dire des éléments compris entre les indices `a` et `!min -1` ?
En déduire que la fonction sépare bien les éléments entre les indices a et b.

Question 7
Écrire une fonction `tri_rapide` réalisant le tri d'un tableau en appliquant l'algorithme décrit plus haut. On n'en demande pas l'analyse.

```
tri_rapide : 'a array -> unit
```

III Étude de complexité

On s'intéresse maintenant à la complexité du tri rapide dans deux cas particuliers.

Question 8
Donner un ordre de grandeur du nombre de comparaisons effectuées dans le tri rapide lorsque le tableau est trié dans l'ordre décroissant.

Question 9
On suppose ici que n est de la forme $n = 2^k - 1$ et que, lors d'une exécution du tri rapide, chaque séparation du tableau coupe le tableau en deux parts égales : ces parties auront, à chaque étape, un nombre d'éléments de la forme $2^k - 1$. On note a_k le nombre de comparaisons effectuées pour trier une portion de tableau de taille $2^k - 1$.
Prouver qu'on a $a_k \leq (k-1)(2^k - 1)$.
En déduire un majorant $C(n)$ en fonction de n et de $\log_2(n)$.

Question 10
On ne suppose plus que n est de la forme $n = 2^k - 1$ mais on suppose que lors d'une exécution du tri rapide, chaque séparation du tableau coupe le tableau en deux parts égales ou dont les tailles ne diffèrent que de 1. On admet que, dans ce cas le nombre de nombre de comparaisons vérifie, pour $n \leq m$, $C(n) \leq 2C(m)$.
Prouver que $C(n) = \mathcal{O}\big(n \log_2(n)\big)$.

IV Recherche d'une pseudo médiane

Ce qui précède suggère que le choix d'un pivot « proche de la médiane » permet d'améliorer les performances du tri rapide.
Il existe un algorithme permettant de trouver cette médiane en un temps linéaire en la taille du tableau, mais cet algorithme difficile ne sera pas discuté ici.
Une façon simple pour lutter contre le pire des cas consiste à choisir le pivot de façon aléatoire, mais le gain obtenu est relativement subtil et le pire des cas reste quadratique.

Une méthode intermédiaire consiste à rechercher une **pseudo médiane**.
Lorsque $\alpha \in]0, 1[$, une α-pseudo médiane d'un tableau est une valeur présente dans le tableau telle qu'au moins $K_1 n^\alpha$ (respectivement $K_2 n^\alpha$) éléments du tableau lui sont inférieurs (respectivement supérieurs), avec K_1 et K_2 deux constantes strictement positives.

Pour un tableau de taille $n = 3^k$, on utilisera l'algorithme de recherche d'une pseudo médiane suivant :

- si $k = 0$, on retourne directement le seul élément considéré,
- sinon, on regroupe les éléments du tableau par 3, on calcule les 3^{k-1} médianes de ces groupes, puis on applique récursivement l'algorithme à ces 3^{k-1} valeurs.

On rappelle que les différents éléments du tableau peuvent être supposés distincts.

IV.1 Calcul de la pseudo-médiane en place

Question 11
Écrire une fonction prenant en entrée un tableau et trois indices distincts et retournant la position de la médiane, parmi les trois éléments du tableau dont on a donné les indices.

```
medianePlace : 'a array -> int -> int -> int -> int
```

Question 12
Écrire une fonction calculant une pseudo médiane.
Cette fonction travaillera obligatoirement dans le tableau initial, sans en créer de nouveau, et en maintenant globalement invariant l'ensemble des valeurs présentes dans le tableau.
On pourra supposer le tableau de taille 3^k, placer dans une première étape les médianes de blocs de trois en positions $3i$, puis prendre les médianes de ces médianes et les placer en position $9i$, etc.
La pseudo médiane sera placée en position 0 dans le tableau.

```
pseudo_mediane : 'a array -> 'a
```

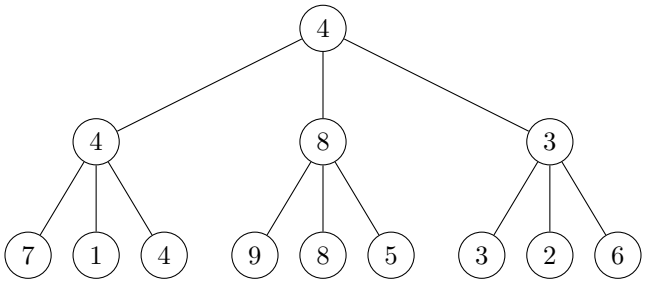
IV.2 Calcul de la pseudo-médiane à l'aide d'un arbre ternaire

On propose de construire un arbre ternaire : les feuilles sont étiquetées par les entrées du tableau et chaque nœud interne est la médiane de ses fils.

```
type a3 =F of int | N of int*a3*a3*a3;;

let racine arbre =
  match arbre with
  |F(x) -> x
  |N(x, _, _, _) - >x;;
```

Voici l'arbre construit sur le tableau `[|7; 1; 4; 9; 8; 5; 3; 2; 6|]`.



Question 13
Écrire une fonction calculant la médiane de trois entiers distincts.

```
mediane : 'a -> 'a -> 'a -> 'a
```

Question 14
Écrire une fonction prenant en entrée trois arbres ternaires et retournant l'arbre ternaire dont la racine est étiquetée par la médiane des trois racines des arbres donnés en entrée et a pour fils ces trois arbres.

```
construire_arbre : a3 -> a3 -> a3 -> a3
```

Question 15
Écrire une fonction récursive `construire t i j` construisant l'arbre ternaire associé aux valeurs du tableau `t` comprises entre les indices i et j compris.
On supposera qu'il existe un entier k tel que $j = i + 3^k - 1$ (on a sélectionné 3^k éléments).

```
construire : int array -> int -> int -> a3
```

Question 16
En déduire une fonction qui admet pour paramètre un tableau de taille 3^k et qui renvoie une pseudo médiane des valeurs du tableau. On devra utiliser un arbre ternaire.

```
pseudo_mediane : int array -> int
```

IV.3 Étude théorique

Question 17
Donner un ordre de grandeur du temps d'exécution de l'algorithme de calcul d'une pseudo médiane.

Pour un tableau de taille 3, la pseudo médiane calculée par l'algorithme ci-dessus est exactement la médiane. On note $n = 3^k$ la taille des tableaux utilisés.

Question 18
Montrer que, pour $k \geq 2$, il existe au moins 2^k éléments qui sont majorés par la pseudo médiane.

Question 19
Montrer que, pour tout $k \geq 2$, il existe un tableau pour lequel il y a exactement 2^k éléments du tableau qui sont majorés par la pseudo médiane.

Question 20
Prouver que cet algorithme permet de calculer une α-pseudo médiane, avec $\alpha = \dfrac{\ln(2)}{\ln(3)}$.

Question 21
Expliquer comment adapter l'implémentation de l'algorithme si le tableau a une longueur qui n'est pas une puissance de 3 ?

IV.4 Extensions

Question 22
Si on modifie l'algorithme en considérant des blocs de 5 éléments plutôt que 3, que dire (en supposant que la longueur du tableau est une puissance de 5) du résultat retourné et du temps de calcul de ce nouvel algorithme ?

Question 23
Montrer que pour tout $\varepsilon > 0$, il existe un algorithme s'exécutant en un coût linéaire et permettant de calculer une $(1-\varepsilon)$-pseudo médiane d'un tableau.

V Gain apporté par la pseudo médiane

On s'intéresse enfin au gain qu'apporte l'utilisation de pseudo médianes dans le tri rapide.
On suppose ici (sauf à la dernière question) que le tri rapide est exécuté en utilisant à chaque étape une 1/2-pseudo médiane avec des constantes $K_1 = K_2 = 1$.
On note $C(n)$ le temps de calcul dans le pire des cas pour appliquer le tri rapide dans ce cas à un tableau de taille n.
Ici encore, une analyse précise et rigoureuse de la complexité est délicate, mais on souhaite obtenir une évaluation de façon raisonnablement convaincante.
Dans cette évaluation en première approximation, on s'autorise à écrire $C(x)$ même lorsque x n'est pas entier : ce sera un raccourci pour $C\big(\lceil x \rceil\big)$.

Question 24
exo :a Justifier qualitativement le fait que C vérifie une inégalité de la forme

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

Question 25
exo :b Montrer que $C(n) = C(n/2) + \mathcal{O}\big(n^{3/2}\big)$.
Indication : on pourra étudier la suite définie par $\alpha_0 = n$ et $\alpha_{k+1} = \alpha_k - \sqrt{\alpha_k}$ si $\alpha_k > 1$ (et $\alpha_{k+1} = 0$ sinon. En particulier on considérera les entiers p tels que $\alpha_p \leq n/2$.

Question 26
exo :c Conclure.

Question 27
Que peut-on raisonnablement espérer comme complexité dans le pire des cas, pour un tri rapide effectué en calculant une $\frac{\ln(2)}{\ln(3)}$-pseudo médiane avec l'algorithme de la partie 4 ?

VI Solutions

Solution de la question 1

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

Solution de la question 2

On peut, par exemple, utiliser le tri par insertion : on insère les éléments du tableau du début jusqu'à la fin en produisant des débuts de tableaux triés. On sait que, dans le pire des cas, la complexité est en $\mathcal{O}(n^2)$.

On commence par une fonction d'insertion qui insère l'élément d'indice i en supposant que les termes de 0 à $i-1$ sont triés.

```
let inserer i tableau =
  let k = ref i in
  while !k > 0 && t.(!k -1) > t.(!k) do
    echange !k (!k - 1) tableau;
    k := !k - 1 done;;

let tri_insertion tableau =
  let n = Array.length tableau in
  for i = 1 to (n-1) do inserer i tableau done;;
```

Solution de la question 3

Le pivot est 15, `t` est transformé en
`t = [|1; 3; 14; 11; 15; 20; 17; 29; 31; 24|]` et la valeur renvoyée est 4 ;

Solution de la question 4

On a le pivot en a initialement avec `!min` valant a.

À chaque étape on ne change la valeur de **`min`** en `!min + 1` qu'après une permutation des position `!min` et `!min + 1` du tableau donc on a toujours p en `t.(!min)`.

Ainsi la fonction renvoie la position du pivot.

Solution de la question 5

`!max - !min` diminue de 1 à chaque étape et reste positif quand la boucle est effectuée : l'algorithme termine et fait `b - a` comparaisons car la valeur initiale de `!max - !min` est `b - a` et sa valeur finale est 0.

Solution de la question 6

On n'effectue une permutation des position `!min` et `!min + 1` que lorsque la valeur de `t.(!min + 1)` est strictement inférieure au pivot donc les éléments compris entre les indices `i` et `!min -1` sont strictement inférieurs au pivot.

De même les éléments entre les indices `!max + 1` et `b` sont strictement supérieurs au pivot ils ne peuvent être égaux au pivot car les valeurs sont distinctes.

En sortie de la boucle on a les valeur de **`min`** et de **`max`** qui sont égales, on note k leur valeur.

Le pivot est à la position k, les éléments entre les indices a et $k-1$ sont inférieurs strictement au pivot et les éléments entre les indices $k+1$ et b sont supérieurs strictement au pivot : on a bien séparé.

Solution de la question 7

```
let tri_rapide t =
  let rec auxTri a b =
    if b > a
     then let p = separation t a b in
          (auxTri a (p-1); auxTri (p+1) b) in
  auxTri 0 (Array.length t -1);;
```

Solution de la question 8
Lorsque la portion de tableau auquel on applique `separation` est décroissante, la fonction de séparation envoie le pivot en dernière place et décale les autres éléments en conservant la décroissance. Ainsi une des partie issue de la séparation est vide et l'autre sera décroissante.
Le nombre de comparaisons effectuées par le tri rapide appliqué à un tableau décroissant de longueur n est donc $C(n) = (n-1) + (n-2) + \cdots + 1 + C(0) = \dfrac{n(n-1)}{2} \simeq \dfrac{n^2}{2}$.
Le tri rapide est donc quadratique quand le tableau est décroissant.

Solution de la question 9
On a $C(1) = 0$ donc $a_1 = 0$: on a bien $a_1 \leq (1-1)(2^1-1)$.
De plus $a_{k+1} = 2a_k + 2^{k+1} - 2$ car on sépare en deux parties de taille $2^k - 1$ et la séparation effectue $n-1$ comparaisons avec $n = 2^{k+1} - 1$. Si la majoration est vraie au rang k on a alors

$$a_{k+1} \leq 2(k-1)(2^k-1) + 2^{k+1} - 2 = (k-1)2^{k+1} - 2k + 2 + 2^{k+1} - 2 = k2^{k+1} - 2k \leq k2^{k+1}$$

donc la majoration est valide au rang $k+1$. Elle est vraie pour tout k.
On a donc $C(n) = a_p \leq (p-1)(2^p-1) = (p-1)n$ avec

$$\log_2(n) = \log_2(2^p - 1) = p + \log_2(1 - 2^{-p}) \geq p + \log_2(1 - 2^{-1}) = p - 1$$

donc $C(n) \leq n \log_2(n)$.

Solution de la question 10
On suppose qu'on a $2^p \leq n < 2^{p+1} - 1 = m$. On a $C(n) \leq 2C(m) = 2a_{p+1} \leq 2p2^{p+1} = 4p2^p$.
Or $2^p \leq n$ d'où $p \leq \log_2(n)$: $C(n) \leq 4n \log_2(n)$.
Cette inégalité est valide aussi pour $n = 2^p - 1$ donc on a bien $C(n) = \mathcal{O}\big(n \log_2(n)\big)$.

Solution de la question 11

```
let medianePlace t a b c=
  if t.(a) < t.(b)
  then if t.(b) < t.(c)
       then b
       else if t.(a) < t.(c) then c else a
  else if t.(a) < t.(c)
      then a
      else if t.(b) < t.(c) then c else b;;
```

Solution de la question 12
On calcule récursivement les pseudos-médianes.

```
let pseudo_mediane t =
  let rec auxMed a b =
    if b=a+2
    then let i = mediane t a (a+1) b in echange a i t
    else let r =(b+1-a)/3 in
         begin
           auxMed a         (a + r - 1);
           auxMed (a + r)   (a + 2*r - 1);
           auxMed (a + 2*r) b;
           let i = mediane t a (a + r) (a + 2*r) in
           echange a i t
        end in
  auxMed 0 (Array.length t -1);
  t.(0);;
```

Solution de la question 13

```
let mediane a b c=
  if a < b
  then if b < c
       then b
       else if a < c then c else a
  else if a < c
       then a
       else if b < c then c else b;;
```

Solution de la question 14

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

Solution de la question 15

```
let rec construire t i j =
  if i = j
  then F(t.(i))
  else let r = (j + 1 - i)/3 in
        let t1 = construire t i          (i + r - 1) in
        let t2 = construire t (i + r)    (i + 2*r -1) in
        let t3 = construire t (i + 2*r) j in
        construire_arbre t1 t2 t3;;
```

Solution de la question 16

```
let pseudo_mediane t =
  let ter = construire t 0 (Array.length t) in
  racine ter;;
```

Solution de la question 17

Si a est le nombre d'instructions, y compris celles nécessaires à `construire_arbre` mais hors appels récursifs, de `construire` alors, en notant $T(k)$ la complexité pour un tableau de taille 3^k, on a $T(k) = 3T(k-1) + a$.

On pose alors $t_k = \frac{T(k)}{3^k}$ et on a $t_k = t_{k-1} + \frac{a}{3^k}$ avec $T(0) = 1$ d'où

$$t_k = 1 + \sum_{p=1}^{k} \frac{a}{3^p} = 1 + \frac{a}{3}\frac{1 - \frac{1}{3^k}}{1 - \frac{1}{3}} = 1 + \frac{a}{2} - \frac{a}{2.3^k}.$$

Ainsi $T(k) = 3^k t_k = nt_k \leq \left(1 + \frac{a}{2}\right)n$: la complexité est linéaire.

Solution de la question 18

Prouvons par récurrence qu'il y a au moins 2^k valeurs du tableau supérieures à la pseudo médiane pour un tableau de taille 3^k.

- Pour un tableau de taille $3^0 = 1$ la pseudo-médiane est l'unique valeur : la propriété est vraie.
- On suppose la propriété vraie pour les tableaux de taille 3^k.
  Si on cherche la pseudo médiane d'un tableau de taille 3^{k+1} alors on le divise en 3 tableaux de taille 3^k. Pour chacun d'eux la pseudo médiane est minorée par au moins 2^k éléments.
  La pseudo médiane de l'ensemble est la médiane des 3 pseudo médianes des sous-tableaux, elle est minorée par 2 d'entre elles donc par au moins $2^k + 2^k = 2^{k+1}$ éléments : la propriété vraie pour les tableaux de taille 3^{k+1}.

Solution de la question 19

On va construire par récurrence sur k un tableau de taille 3^k dont les éléments sont les entiers de 0 à $3^k - 1$ et qui contient exactement 2^k éléments inférieurs ou égaux à sa pseudo médiane. Celle-ci doit donc être le 2^k-ième élément : $2^k - 1$.

Pour $k = 0$ `[|0|]` convient.

On suppose construit un tableau `t` de taille 3^k vérifiant les propriétés. On note $d = 3^k$.

On construit alors un tableau `tt` de taille 3^{k+1} avec, pour i variant de 0 à $d - 1$,
`tt.(i) <- 2*t.(i)`, `tt.(i+d) <- 2*t.(i)+1` et `tt.(i+2*d) <- t.(i)+2*d`.

Les pseudos médianes des 3 sous-tableaux de taille d de `tt` sont, en notant $p = 2^k - 1$, $2p$, $2p + 1$ et $p + 2d$.

On a $2p + 1 = 2^{k+1} - 1$ et $p + 2d = 2^k - 1 + 2.3^k \geq 2^k + 3^k \geq 2^{k+1}$ donc $2p \leq 2p + 1 \leq p + 2d$.

Ainsi la pseudo médiane de `tt` est $2p + 1 = 2^{k+1} - 1$.

Le nouveau tableau contient tous les entiers de 0 à $3^{k+1} - 1$ donc contient exactement 2^{k+1} éléments inférieurs à la pseudo médiane.

On a donc prouvé le résultat par récurrence.

```
let exemple n =
  let rec aux n =
    match n with
   |0 -> [|0|], 1
   |k -> let t, d = aux (k - 1) in
         let tt = Array.make (3*d) 0 in
         for i = 0 to (d-1) do
           tt.(i) <- 2*t.(i);
           tt.(i+d) <- 2*t.(i) + 1;
           tt.(i+2*d) <- t.(i) + 2*d done;
         tt, 3*d in
  let t, _ = aux n in t;;

exemple 3;;
[| 0;  4;  8;  2;  6; 10; 12; 14; 16;
   1;  5;  9;  3;  7; 11; 13; 15; 17;
  18; 20; 22; 19; 21; 23; 24; 25; 26|]
```

Solution de la question 20
De la même façon on prouve qu'il y a dans un tableau de taille 3^k au moins 2^k éléments supérieurs ou égaux à la pseudo médiane.
Si $n = 3^k$ alors, en posant $\alpha = \frac{\ln(2)}{\ln(3)}$, $2^k = 3^{k\alpha} = n^\alpha$.
Ainsi la pseudo médiane calculée est une α-pseudo médiane.

Solution de la question 21
Pour un tableau de taille n, on définit k tel que $3^k \leq n < 3^{k+1}$.
On applique l'algorithme sur la portion du tableau contenant les 3^k premiers éléments.
Si d est la pseudo médiane obtenue, d est supérieure a au moins 2^k éléments de la portion donc a au moins 2^k éléments du tableau, de même d est inférieure à au moins 2^k éléments du tableau.
On a $2^k = (3^k)^\alpha \geq \left(\frac{n}{3}\right)^\alpha = Kn^\alpha$ avec $K = \frac{1}{3^\alpha} = \frac{1}{2}$.
Ainsi on a toujours une α-pseudo médiane avec des constantes $K_1 = K_2 = \frac{1}{2}$.

Solution de la question 22
Si on cherche une pseudo médiane en coupant par paquets de 5 les mêmes raisonnements montrent qu'il y aura, si $5^k \leq n < 5^{k+1}$, au moins $\frac{1}{3}n^\beta$ éléments supérieurs (resp. inférieurs) à la pseudo-médiane calculée avec $\beta = \frac{\ln(3)}{\ln(5)}$.
On trouve donc une β-pseudo médiane. La complexité est encore linéaire

Solution de la question 23
En coupant par bloc de $2N+1$ on trouve une α_N-pseudo médiane avec $\alpha_N = \frac{\ln(N+1)}{\ln(2N+1)}$ qui peut être choisi aussi proche de 1 que l'on veut.

Solution de la question 24
La complexité du calcul de la pseudo médiane et de la séparation est linéaire.
Ainsi, si la séparation coupe en deux paquets de taille p et q on a $C(n) \leq C(p) + C(q) + An$.
On a $p + q = n - 1$ et $p \geq \sqrt{n}$, $q \geq \sqrt{n}$:
on veut donc majorer $C(x) + C(n-1-x)$ pour $x \in [\sqrt{n}; n - 1 - \sqrt{x}]$.
On fait l'hypothèse (raisonnable ?) que ce maximum est atteint au bord.
On a donc $C(n) \leq C\left(n - \sqrt{n}\right) + C\left(\sqrt{n}\right) + An$.
De plus on a vu que la complexité est, au pire, quadratique donc $C\left(\sqrt{n}\right) \leq B\left(\sqrt{n}\right)^2 = Bn$.
Pour $K = A + B$ on obtient bien $C(n) \leq C\left(n - \sqrt{n}\right) + Kn$.

Solution de la question 25

1. Si $\alpha_k > 1$ alors $\sqrt{\alpha_k} > 1$ donc $\alpha_{k+1} < \alpha_k - 1$.
   En particulier $\alpha_p \leq 1$ ou $\alpha_p \leq \alpha_0 - p \leq \frac{n}{2}$ pour $p \geq \frac{n}{2}$.
   On considère le premier entier q tel que $\alpha_q \leq \frac{n}{2}$, on le note q_0.
2. Pour $0 \leq k < q_0$,on a $\alpha_k \geq \frac{n}{2}$ d'où $\alpha_{k+1} \leq \alpha_k - \sqrt{\frac{n}{2}}$.
   On en déduit que, dans ce cas, $\alpha_k \leq \alpha_0 - k\sqrt{\frac{n}{2}} = n - k\sqrt{\frac{n}{2}}$.
   On a alors l'encadrement $\frac{n}{2} \leq \alpha_k \leq n - k\sqrt{\frac{n}{2}}$ donc $k \leq \dfrac{n - \frac{n}{2}}{\sqrt{\frac{n}{2}}} = \sqrt{\dfrac{n}{2}}$.
   Ainsi $k \leq q_0 - 1$ implique $\leq \sqrt{\frac{n}{2}}$ d'où $q_0 - 1 \leq \sqrt{\frac{n}{2}}$.
3. On a $C(\alpha_i) \leq C(\alpha_{i+1}) + K\alpha_i$ donc $C(\alpha_0) \leq C(\alpha_{q_0}) + K(\alpha_0 + \alpha_1 + \cdots + \alpha_{q-0-1})$.
   On fait l'hypothèse (raisonnable ?) qu'on peut déduire $C(\alpha_{q_0}) \leq C\left(\frac{n}{2}\right)$ de $\alpha_{q_0} \leq \frac{n}{2}$.
   On a alors $C(n) = C(\alpha_0) \leq C\left(\frac{n}{2}\right) + K\sum_{i=0}^{q_0-1} \frac{n}{2}$ donc
   $0 \leq C(n) - C\left(\frac{n}{2}\right) \leq Kq_0\frac{n}{2} \leq K\left(1 + \sqrt{\frac{n}{2}}\right)\frac{n}{2}$: $C(n) - C\left(\frac{n}{2}\right) = \mathcal{O}\left(n^{3/2}\right)$.

Solution de la question 26
On en déduit que $C(n) \leq C\left(\frac{n}{2^p}\right) + K'n^{3/2}\left(1 + \frac{1}{2^{2/3}} + \cdots + \frac{1}{2^{2(p-1)/3}}\right)$.
Si on choisit p assez grand on a $C\left(\frac{n}{2^p}\right) = C(1) = 0$. Si on pose $r = \frac{1}{2^{2/3}}$ on a donc

$$C(n) \leq K'n^{3/2}\left(1 + r + r^2 + \cdots + r^{p-1}\right) = K'n^{3/2}\frac{1 - r^p}{1 - r} \leq \frac{K'}{1 - r}n^{3/2} = \mathcal{O}\left(n^{3/2}\right)$$

Solution de la question 27
Alors là ... Cela devient de la poésie.

- On suppose qu'on a une α-pseudo médiane en temps linéaire.
- On a donc $C(n) \leq C\big(n - Kn^\alpha\big) + C\big(Kn^\alpha\big) + An$ avec la même hypothèse raisonnable qu'à la question **??**.
- Majorer $C\big(Kn^\alpha\big)$ par un $Bn^{2\alpha}$ n'est pas judicieux car on arriverait à une complexité quadratique quand α tend vers 1.
- On va faire une hypothèse qui peut sembler raisonnable : on suppose qu'on peut arriver à une complexité $C(n) = \mathcal{O}\big(n^\beta\big)$ et on cherche un bon β.
- On obtient $C(n) \leq C\big(n - Kn^\alpha\big) + C\big(Kn^\alpha\big) + An = C\big(n - Kn^\alpha\big) + \mathcal{O}\big(n^{\alpha\beta}\big) + An$
  On espère (raisonnablement) qu'on a $\alpha\beta \leq 1$: $C(n) \leq C\big(n - Kn^\alpha\big) + \mathcal{O}\big(n\big)$.
- Comme à la question **??**, la suite $u_0 = n$ et $u_{p+1} = u_p - Ku_p^\alpha$ atteint une valeur inférieure à $\frac{n}{2}$ en N étapes avec $N \leq \dfrac{n}{Bn^\alpha}$ donc $C(n) \leq C\big(\frac{n}{2}\big) + Dn^{2-\alpha}$.
- Comme à la question **??**, on en déduit $C(n) = \mathcal{O}\big(n^{2-\alpha}\big)$ donc $\beta = 2 - \alpha$.
- On vérifie qu'on a bien $\alpha\beta = 2\alpha - \alpha^2 = 1 - (1-\alpha)^2 \leq 1$.

En gros, on suppose que ça marche et on en déduit que ça marche,
c'est du magiquement raisonnable ici !