Devoir surveillé n°2 : concours blanc

MPSI Lycée Clemenceau : option informatique

Lundi 28 avril 2025

Vous avez 3 heures.

Il sera tenu compte de la présentation et de la rédaction. Les programmes (ou fonctions) devront être commentés et le plus souvent justifiés (terminaison et correction).
Lire l'annexe pour les fonctions prédéfinies et autorisées pour chaque exercice.

Exercice I : manipulations de listes

1) Ecrire une fonction `pas_nulle` qui prend en entrée une liste d'entier et retourne la même liste privée des zéros éventuels : `pas_nulle : int list -> int list`

Exemple :

```
# pas_nulle [4;2;0;5;-3;0;1;8];;
    - : int list = [4; 2; 5; -3; 1; 8]
```

2) Ecrire une fonction `intervalle` qui prend en donnée deux entier `a` et `b` (avec `a<b`) et une liste `lst` et renvoie un triplet de listes `(l1,l2,l3)` tel que `l1` contient les éléments de `lst` strictement inférieur à `a`, `l3` contient les éléments de `lst` strictement supérieur à `b` et `l2` le reste de la liste `l`.

Exemple :

```
# intervalle 5 10 [2;4;5;16;54;12;15;8;94;1;0;5;6;7;10];;
    - : int list * int list * int list = [2; 4; 1; 0], [5; 8; 5; 6; 7; 10], [16; 54; 12; 15; 94]
```

3) Ecrire une fonction `indiceTripleMax` qui renvoie le couple `(indice, somme)` d'une liste où `indice` est l'indice du premier élément du triplet de somme maximale.

`indiceTripleMax : int list -> int * int`

Exemple :

```
# indiceTripleMax [1;5;7;8;6;2;7;9;4];;
    - : int * int = 3, 21
```

Remarque : on parle d'indice d'un élément de la liste en commençant à 1, ce n'est pas un tableau.

Exercice II

On admettra que la multiplication de deux entiers naturels se fait en temps constant.

1) Ecrire une fonction `puissance` de type `int -> int -> int` qui calcule, pour deux entiers n et k non nuls, n^k.

Ce programme devra être récursif ou faisant appel à des fonctions auxiliaires récursives, et devra utiliser le principe de diviser pour régner.

Donner la complexité de ce programme.

2) On s'intéresse ici au problème de déterminer si un entier naturel est une puissance non triviale d'un nombre entier ou non : on dit qu'un nombre entier naturel n est une *puissance entière* s'il existe deux entiers naturels k et m tous deux > 1 tels que $n = m^k$.

a) Écrire la fonction :

`test_puissance : int -> int -> bool`

qui prend en entrée les entiers naturels $n > 1$ et $k > 1$ et renvoie le booléen `true` s'il existe un entier naturel m tel que $n = m^k$ et `false` sinon.

On proposera une version itérative et une version récursive.

b) **i)** Soit n un entier naturel. On suppose que n est une puissance entière : Soient k et m deux entiers naturels > 1 tels que $n = m^k$. Justifier que $k \leqslant \log_2(n)$.

ii) Écrire la fonction :

`test_puissance_entiere : int -> bool`

qui prend en entrée l'entier naturel $n > 1$ et renvoie le booléen `true` s'il existe deux entiers naturels k et m tous deux > 1 tels que $n = m^k$ et `false` sinon.

On proposera une version itérative et une version récursive.

iii) En déduire la fonction :

`liste1_puissances_entieres : int -> int list`

qui prend en entrée l'entier naturel $n > 1$ et renvoie la liste des entiers naturels compris entre 2 et n qui sont des puissances entières.

Exercice III : minimum local

Dans tout le problème on considère des tableaux de la forme $t = [|t_0; t_1; t_2; \ldots; t_{n-2}; t_{n-1}|]$ de longueur $n > 1$. On n'aura pas besoin de vérifier que les tableaux sont non vides. On cherche un minimum local de ce tableau, c'est-à-dire la valeur t_i avec i compris entre 0 et $n-1$ telle que t_i est inférieur ou égal à ses voisins ; on remarquera que t_i peut avoir 2 voisins mais n'en a qu'un seul si $i = 0$ ou $i = n-1$ ou même aucun si $n = 1$.

Exemples :

- t_1 et t_5 sont les minimums locaux de $[|14; 5; 8; 7; 4; 2|]$,
- t_1, t_2 et t_3 sont les minimums locaux de $[|14; 5; 5; 5; 8|]$,

On peut remarquer que le minimum d'un tableau est un minimum local.

1) Ecrire une fonction `minimum : 'a array -> 'a*int` qui reçoit un tableau pour paramètre et qui renvoie la valeur et l'indice d'un minimum (global).

2) Montrer que le premier minimum local d'un tableau `t` de longueur n est

- soit t_{n-1}
- soit t_i où i est le premier indice tel que $i < n-1$ et $t_i \leqslant t_{i+1}$.

Dans quel cas le premier minimum local vaut-il t_{n-1} ?

3) En déduire une fonction `premier_ml : 'a array -> 'a`, qui renvoie le premier minimum local d'un tableau.

4) Adapter la fonction précédente en une fonction `premier_ml_rec : ' list -> 'a` pour calculer un minimum local d'une liste.

5) k est un indice tel que $0 < k < n-1$ et t_k n'est pas un minimum local.

Prouver que :

- si $t_k > t_{k-1}$ alors un minimum local de $[|t_0; \ldots; t_{k-1}|]$ est un minimum local du tableau,
- si $t_k > t_{k+1}$ alors un minimum local de $[|t_{k+1}; \ldots; tn-1|]$ est un minimum local du tableau.

6) En déduire une fonction `minimum_local : 'a array -> a` utilisant la méthode diviser pour régner qui calcule un minimum local d'un tableau avec une complexité logarithme (on ne demande pas de justifier la complexité).

Exercice IV : sélection du $(k+1)^e$ plus petit élément

La sélection du $(k+1)^e$ plus petit élément d'une liste d'entiers `L`, non nécessairement triée, consiste à trouver le $(k+1)^e$ élément de la liste obtenue en triant `L` dans l'ordre croissant.
Par exemple, si `L = [9; 1; 2; 4; 7; 8]` le 3^e plus petit élément de `L` est 4. On pourra remarquer que si la liste `L` est triée dans l'ordre croissant, le $(k+1)^e$ plus petit élément est l'élément de rang k dans `L`.

On présente un algorithme permettant de résoudre ce problème de sélection avec une complexité temporelle linéaire dans le pire cas. Celui-ci est basé sur le principe de « diviser pour régner »et sur le choix d'un bon pivot pour partager la liste en deux sous-listes.

Étant donné un réel a, on note $\lfloor a \rfloor$ le plus grand entier inférieur ou égal à a.

Fonctions utiles

Dans cette section, on écrit des focntions auxiliaires qui sont utiles pour la fonction principale.

1) Ecrire une fonction récursive de signature : `longueur : 'a list -> int`, telle que `longueur lst` est la longueur de la liste `lst`.

2) Ecrire une fonction récursive de signature : `insertion : 'a list -> 'a -> 'a list`, telle que `insertion lst a` est la liste triée dans l'ordre croissant obtenue en ajoutant l'élément `a` dans la liste croissante `lst`.

3) En déduire une fonction récursive de signature : `tri_insertion : 'a list -> 'a list`, telle que `tri_insertion lst` est la liste obtenue en triant `lst` dans l'ordre croissant.

4) Écrire une fonction récursive de signature : `selection_n : 'a list -> int -> 'a`, telle que `selection_n lst n` est l'élément de rang n de la liste `lst`.
Par exemple, `selection_n [4;2;6;4;1;15] 3` est égal à 4.

5) Écrire une fonction récursive de signature : `paquets_de_cinq : 'a list -> 'a list list`, telle que `paquets_de_cinq lst` est une liste de listes obtenue en regroupant les éléments de la liste `lst` par paquets de cinq sauf éventuellement le dernier paquet qui est non vide et qui contient au plus cinq éléments.
Par exemple :
— `paquets_de_cinq []` est égal à `[]`,
— `paquets_de_cinq [2;1;2;1;3]` est égal à `[[2;1;2;1;3]]`,
— `paquets_de_cinq [3;4;2;1;5;6;3]` est égal à `[[3;4;2;1;5];[6;3]]`.

6) Écrire une fonction récursive de signature : `medians : 'a list list -> 'a list`, telle que `medians lst` est la liste `m` obtenue en prenant dans chaque liste lst_k apparaissant dans la liste de listes `lst`, l'élément médian de lst_k.
On convient que pour une liste `A` dont les éléments sont exactement $a_0 \leqslant a_1 \leqslant \cdots \leqslant a_{n-1}$, l'élément médian désigne $a_{\lfloor \frac{n}{2} \rfloor}$.
Dans le cas où la liste `L` n'est pas triée, l'élément médian désigne l'élément médian de la liste obtenue en triant `L` par ordre croissant.
Par exemple : `medians [[3;1;5;3;2];[4;3;1];[1;3];[5;1;2;4]]` est égal à `[3;3;3;4]`

7) Écrire une fonction de signature : `partage : 'a -> 'a list -> 'a list * 'a list * int * int`, telle que `partage p lst` est un quadruplet (l_1, l_2, n_1, n_2) où l_1 est la liste des éléments de `lst` plus petit que `p`, l_2 est la liste des éléments de `lst` strictement plus grand que `p`, n_1 et n_2 sont respectivement les longueurs de l_1 et l_2.

La fonction de sélection et sa complexité

On détaille la fonction de sélection :

Algorithme 1 - Sélection du $(k+1)^e$ plus petit élément

```
1  SELECTION L k :
   /* L est une liste, k est un entier positif                          */
2  début
3      n ←LONGUEUR L
4      si n ≤ 5 alors
5          M ← (TRI_INSERTION L)
6          retourner l'élément de rang k de M
7      fin
8      sinon
9          L_Cinq ← PAQUETS_DE_CINQ L
10         M ← MEDIANS L_Cinq
11         pivot ← SELECTION M ((n + 4) //5) //2
           /* L'opérateur // désigne le quotient d'entiers. Le rang ((n+4)//5)//2
              correspond au rang du médian de la liste M                 */
12         L1, L2, n1, n2 ← PARTAGE pivot L
13         si k < n1 alors
14             retourner SELECTION L1 k
15         fin
16         sinon
17             retourner SELECTION L2 (k − n1)
18         fin
19     fin
20 fin
```

8) Écrire une fonction récursive de signature : `selection : 'a list -> int -> 'a`, telle que `selection lst k` est le $(k+1)^e$ plus petit élément de la liste `lst`. L'écriture de la fonction sera une traduction en OCaml de l'Algorithme 1.

On cherche à déterminer la complexité en nombre de comparaisons de la fonction `selection`. Pour tout $n \in \mathbb{N}$, on note $T(n)$ le nombre maximum de comparaisons entre éléments lors d'une sélection d'un élément quelconque dans des listes `L`, sans répétition, de taille n.

En analysant l'Algorithme 1, il est possible de démontrer que :

$$\forall n \geqslant 55, T(n) \leqslant T\left(\left\lfloor \frac{n+4}{5} \right\rfloor\right) + T\left(\left\lfloor \frac{8n}{11} \right\rfloor\right) + 4n \qquad (I)$$

9) En admettant la proposition (I), montrer que pour tout entier n supérieur à 1, on a :

$$T(n) \leqslant (200 + T(55))n$$

Pour l'initialisation, on pourra remarquer que T est une fonction croissante.

Exercice V : le tri fusion

L'objectif de cet exercice est d'étudier une implantation particulière d'un algorithme de tri d'une séquence d'entiers.

Définition 1 Une séquence s de taille n de valeurs v_i avec $1 \leqslant i \leqslant n$ est notée $\langle v_1, ..., v_n \rangle$. Une même valeur v peut figurer plusieurs fois dans s, nous noterons $card(v, s)$ le cardinal de v dans s, c'est-à-dire le nombre de fois que v figure dans s avec : $card(v, \langle v_1, ..., v_n \rangle) = card\{i \in \mathbb{N} \mid 1 \leqslant i \leqslant n,\ v = v_i\}$

Une séquence d'entiers est représentée par le type séquence et une paire de séquences d'entiers est représentée par le type paire dont les définitions sont :

```
type sequence = int list;;
type paire = sequence * sequence;;
```

Soit le programme en langage CaML :

```
let rec eclater s =
 match s with
  | [] -> ([],[])
  | [e] -> ([e],[])
  | e1::e2::q ->
     let (p1,p2) = eclater q in
       (e1::p1,e2::p2);;

let rec fusionner s1 s2 =
 match s1,s2 with
  | (e1::ql,e2::q2) ->
      if (e1 <= e2)
       then e1::(fusionner q1 s2)
       else e2::(fusionner s1 q2)
  | (_,_) -> sl @ S2;;

let rec trier s =
  match s with
    | [] -> []
    | [e] -> [e]
    | _ ->
       let (s1,s2) = eclater s in
         fusionner (trier s1) (trier s2);;
```

Soit la constante `exemple` définie et initialisée par :

```
let exemple = [4; 3; 2; 1] ;;
```

Question 1 Détailler les étapes du calcul de (`trier exemple`) en précisant pour chaque appel aux fonctions `eclater`, `fusionner` et `trier`, la valeur du paramètre et du résultat.

Question 2 Soit la séquence `p` $= \langle p_1, ..., p_n \rangle$ de taille n, soient les séquences `a` $= \langle a_1, ..., a_r \rangle$ de taille r et `b` $= \langle b_1, ..., b_s \rangle$ de taille s telles que `(a,b) = (eclater p)`, montrer que :

$$\begin{array}{l} \forall i,\ 1 \leqslant i \leqslant n,\ card(p_i, p) = card(p_i, a) + card(p_i, b) \\ s = E(n/2) \\ r = n - E(n/2) \end{array}$$

Question 3 Soient les séquences `a` $= \langle a_1, ..., a_m \rangle$ de taille m et `b` $= \langle b_1, ..., b_n \rangle$ de taille n, soit la séquence `r` $= \langle r_1, ..., r_l \rangle$ de taille l telle que `r = (fusionner a b)`, montrer que :

$$\begin{array}{l} l = m + n \\ \forall k,\ 1 \leqslant k \leqslant l,\ card(r_k, r) = card(r_k, a) + card(r_k, b) \\ \left\{ \begin{array}{l} \forall i,\ 1 \leqslant i < m,\ a_i \leqslant a_{i+1} \\ \forall j,\ 1 \leqslant j < n,\ b_j \leqslant b_{j+1} \end{array} \right. \Rightarrow \forall k,\ 1 \leqslant k < l,\ r_k \leqslant r_{k+1} \end{array}$$

Question 4 Soit la séquence `p` $= \langle p_1, ..., p_m \rangle$ de taille m, soit la séquence `r` $= \langle r_1, ..., r_n \rangle$ telle que `r = (trier p)`, montrer que :

$$\begin{array}{l} m = n \\ \forall i,\ 1 \leqslant i \leqslant m,\ card(p_i, r) = card(p_i, p) \\ \forall i,\ 1 \leqslant i < n,\ r_i \leqslant r_{i+1} \end{array}$$

Question 5 Montrer que le calcul des fonctions `eclater`, `fusionner` et `trier` se termine quelles que soient les valeurs de leurs paramètres respectant le type des fonctions.

Question 6 Donner des exemples de valeurs du paramètre `s` de la fonction `trier` qui correspondent aux meilleur et pire cas en nombre d'appels récursifs effectués.

Calculer une estimation de la complexité dans les meilleur et pire cas des fonctions `eclater`, `fusionner` et `trier` en fonction du nombre de valeurs dans les séquences données en paramètre. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.

Annexe

Exercice I : pas de concaténation, ni de longueur de liste, seulement, si besoin, `List.hd` et `List.tl`
Exercice II : pas de changement de types
Exercice III :
— `Array.make n a` : créer un tableau de taille n dont tous les éléments sont égaux à `a`
— `Array.length` : donne la longueur d'un tableau
— `p.(i)` : donne l'élément du tableau d'indice `i` (l'indice commence à 0)
— `p.(i)<- a` : pour mettre la valeur de `a` dans le tableau à l'indice `i`
Pas de concaténations de listes, ni de longueur de liste, seulement, si besoin, `List.hd` et `List.tl`
Exercice IV : si besoin, `List.hd` et `List.tl`