Devoir surveillé d'informatique

Consignes

- Les programmes demandés doivent être écrits en C et on suppose que les librairies standards usuelles (`<stdio.h>`, `<stdlib.h>`, `<stdbool.h>`) sont déjà importées.
- On pourra toujours librement utiliser une fonction demandée à une question précédente même si cette question n'a pas été traitée.
- Veillez à présenter vos idées et vos réponses partielles même si vous ne trouvez pas la solution complète à une question.
- La clarté et la lisibilité de la rédaction et des programmes sont des éléments de notation.

❑ Exercice 1 : *Algorithme du drapeau hollandais*

> « *Le problème du drapeau hollandais est un problème de programmation, présenté par Edsger Dijkstra, qui consiste à réorganiser une collection d'éléments identifiés par leur couleur, sachant que seules trois couleurs sont présentes (par exemple, rouge, blanc, bleu, dans le cas du drapeau des Pays-Bas). Étant donné un nombre quelconque de balles rouges, blanches et bleues alignées dans n'importe quel ordre, le problème est à les réarranger dans le bon ordre : les bleues d'abord, puis les blanches, puis les rouges.* »
>
> (Wikipedia)

On suppose déjà écrite la fonction `echange` de prototype `void echange(int tab[], int i, int j)` qui échange les éléments d'indice `i` et `j` dans le tableau `tab` et on considère dans la suite que cet échange s'effectue en temps constant. On donne ci-dessous une implémentation de l'algorithme du drapeau hollandais en langage C permettant de réarranger les éléments d'un tableau ne contenant *que* les trois valeurs entières 1, 2 et 3 :

```c
// Prend en entrée un tableau (ne contenant que les valeurs 1, 2 et 3)
// Ne renvoie rien
// Réarrange les éléments du tableau de façon à avoir les 1 puis les 2 et les 3
void drapeau_hollandais(int tab[], int taille)
{
    int i1 = 0;
    int i2 = taille - 1;
    int i3 = taille - 1;
    while (i1 <= i2){
        if (tab[i1] == 1){
            i1 = i1 + 1;}
        else{
            if (tab[i1] == 2){
                echange(tab, i1, i2);
                i2 = i2 - 1;}
            else{
                echange(tab, i1, i2);
                echange(tab, i2, i3);
                i3 = i3 - 1;
                i2 = i2 - 1;}
        }
    }
}
```

1. Etude de l'algorithme du drapeau hollandais.
   a) Faire fonctionner cet algorithme sur le tableau `tab = {1, 3, 2, 2, 3, 1}`, et donner le contenu de `tab` ainsi que celui des variables `i1`, `i2` et `i3` lors du déroulement de l'algorithme en recopiant

et complétant le tableau suivant :

	`tab`	`i1`	`i2`	`i3`
Initialisation	`{1, 3, 2, 2, 3, 1}`	`0`	`5`	`5`
Etape 1	...	...	...	...
Etape 2	...	...	...	...
Etape 3	...	...	...	...
Etape 4	...	...	...	...
Etape 5	...	...	...	...

b) Prouver la terminaison de cet algorithme.

c) Prouver la correction de cet algorithme.

🛟 *Indication : on pourra noter n la taille du tableau et :*

— P_1 *la tranche du tableau* `tab` *comprise entre les indices 0 et* `i1` *(exclu)*

— P_2 *la portion du tableau* `tab` *comprise entre les indices* `i2` *et* `i3` *(exclu)*

— P_3 *la portion du tableau* `tab` *comprise entre les indices* `i3` *et* $n-1$ *(exclu)*

Et prouver l'invariant suivant : « P_1 ne contient que des 1, P_2 que des 2 et P_3 que des 3 ».

d) Justifier brièvement que l'algorithme du drapeau hollandais a une complexité temporelle linéaire.

2. Comparaison avec le tri par insertion

a) Rappeler (sans justification), la complexité de l'algorithme du tri par insertion.

b) On a mesuré qu'en utilisant l'algorithme du tri par insertion un ordinateur trie une liste de dix million d'éléments en 5 secondes. Quel est le temps prévisible approximatif pour trier une liste contenant un milliard d'éléments ?

c) On a mesuré qu'en utilisant l'algorithme du drapeau hollandais un ordinateur trie une liste de dix million d'éléments en 0.2 secondes. Quel est le temps prévisible approximatif pour un trier une liste contenant un milliard d'éléments ?

❑ Exercice 2 : *Représentation des ensembles d'entiers*

En OCaml, on représente une partie de $\mathbb{N}$ par la liste **triée** de ses éléments. Par exemple l'ensemble $\{2, 9, 1, 6, 8\}$ sera représenté par la liste `[1; 2; 6; 8; 9]`. Le but de l'exercice est d'étudier deux méthodes permettant de calculer l'union de deux ensembles ainsi représentés. Dans l'étude de la complexité, on notera n_1 la longueur de la première liste et n_2 celle de la seconde.

1. Union en ajoutant chaque élément successivement

a) Ecrire une fonction `ajoute int -> int list -> int list` qui prend en argument un entier `n`, une liste triée `l` et insère `elt` dans `l` si `elt` n'y figure pas déjà et sinon ne fait rien. Par exemples :

— `insere 3 [2; 6; 7]` donne `[2; 3; 6; 7]`,

— `insere 4 [1; 4; 5]` donne `[1; 4; 5]`.

b) Ecrire une fonction `union` qui calcule l'union de deux listes en insérant successivement chacun des éléments de la première liste dans la seconde à l'aide de la fonction écrite à la question précédente.

c) Déterminer la complexité temporelle en fonction de n_1 et n_2 dans le pire des cas de cette fonction.

2. Deuxième méthode

a) Recopier et compléter la fonction ci-dessous (pointillés des lignes 3, 4 et 5) qui à l'aide d'une correspondance de motifs sur les deux listes calcule directement leur union :

```
1 let rec union l1 l2 =
2   match l1, l2 with
3   | l1, [] -> ....
4   | [], l2 -> ....
5   | h1::t1, h2::t2 -> ......
```

b) Prouver que cette fonction termine.
c) Donner sa complexité temporelle en fonction de n_1 et n_2.

❑ **Exercice 3** : *Evaluation d'un polynôme par la méthode de Horner*
En Ocaml, on représente un polynôme par la liste de ses coefficients (le coefficient de plus haut degré en premier). On suppose dans toute la suite de l'exercice qu'il s'agit de coefficients entiers. Par exemple le polynôme $x^2 - 11x + 30$ est représenté par la liste `[1, -11, 30]`. D'autre part on donne la fonction `puissance : int -> int -> int` ci-dessous qui prend en argument deux entiers naturels `a` et `n` et calcule `a` puissance `n`

```
(* Calcule a puissance n pour a et n entiers naturels*)
let rec puissance a n =
  if n=0 then 1 else a * (puissance a (n-1));;
```

1. Evaluation naïve
   a) Justifier rapidement qu'avec la fonction `puissance`, le calcul de a^n demande n multiplications.
   b) On donne ci-dessous la fonction `naif` permettant de calculer un polynôme en `x` en donnant la liste de ses coefficients `lcoeff` :

```
(* Calcule la valeur en x du  du polynome de coefficient lcoeff*)
  let rec naif lcoeff x =
    match lcoeff with
    | [] -> 0
    | ak::t-> ak*puissance x (List.length t) + (naif t x );;
```

   La fonction `naif` utilise la fonction `puissance` donnée en début d'exercice.
   Justifier rapidement la terminaison de cette fonction.
   c) Prouver la correction de la fonction `naif`.
   🛟 ; *Indication : penser à utiliser une identité mathématique entre le polynôme à calculer et celui calculé par l'appel récursif*
   d) Déterminer la complexité la fonction `naif`.
   🛟 *Indication : on pourra noter $C(n)$ le nombre de multiplications pour une liste de n coefficients puis établir que $C(n) = C(n-1) + n$ (utiliser le résultat de la question 1.a)*
2. Méthode de Horner
   a) Montrer que $\sum_{k=0}^{n} a_k x^k = a_0 + x \times (a_1 + x \times (\cdots + a_n))$
   b) En déduire un algorithme récursif pour calculer la valeur d'un polynôme sans calculer expliciterment les puissances de la variable.
   c) Ecrire une implémentation en OCaml de cet algorithme.
   d) Justifier rapidement que cet algorithme a une complexité linéaire.
3. **Bonus** : Rappeler (sans justification) la complexité de l'algorithme d'exponentiation rapide. Quelle sera la complexité de la fonction de la question 1.b si on utilise l'exponentiation rapide pour le calcul des puissances ?