Concours blanc

Option informatique, première année

Julien Reichert

Toutes les fonctions de ce devoir sont à écrire en Ocaml. Si la transcription en Ocaml d'un algorithme pose problème, il est cependant envisageable de l'écrire en pseudo-code.

1 Fonctions préliminaires

Exercice 1 : Écrire une fonction prenant en argument une liste et déterminant si elle est croissante.

Exercice 2 : Même exercice avec un tableau.

Exercice 3 : Écrire une fonction prenant en argument une liste d'entiers et un entier et déterminant le nombre d'éléments de la liste strictement inférieurs à l'entier en question. [1]

Exercice 4 : Même exercice avec un entier et un tableau d'entiers.

Exercice 5 : Énoncer le théorème de récurrence des partitions (« master theorem »). L'utiliser pour donner la complexité d'une recherche dichotomique dans un tableau (on ne demandera pas d'écrire le programme).

2 Tri fusion

On rappelle ici le principe du tri fusion : une structure de taille ≤ 2 est triée, sinon on la coupe en deux parts égales à un près de la façon que l'on souhaite, on trie les deux moitiés puis on les recombine en une structure triée en comparant à chaque fois les éléments minimaux et en récupérant le plus petit des deux, jusqu'à ce qu'une structure soit vide, après quoi on peut simplement récupérer l'autre en guise de reste.

Exercice 6 : Programmer le tri fusion sur des listes.

Exercice 7 : Même exercice sur des tableaux. On ne fera pas le travail en place et on ne se souciera pas de la complexité en espace, en particulier.

Exercice 8 : Dans les deux cas, déterminer la complexité du programme.

1. On notera qu'en pratique la fonction peut aussi marcher si on remplace tous les entiers par un autre type.

3 Inversions dans une liste / un tableau

Soit `s` une structure dont les éléments sont organisés d'une façon ou d'une autre (en pratique, une liste ou un tableau). Une inversion dans `s` est un couple de deux éléments `a` et `b` de `s` tels que `a < b` et `a` figure après `b` dans `s`. Par exemple, dans la liste `[2; 3; 2; 1]`, il y a quatre inversions : l'élément `1` est après les deux `2` et le `3`, et un des `2` est après le `3`. Les deux `2` ne forment pas une inversion car ils sont égaux.

Exercice 9 : Écrire une fonction qui détermine simplement le nombre d'inversions dans une liste. Déterminer la complexité.

Exercice 10 : Même exercice avec un tableau.

Pour améliorer la complexité, nous allons utiliser un algorithme de type diviser pour régner, qui s'inspire du tri fusion.

Le principe est le suivant : soit `s` une structure dont la moitié gauche est notée `s1` et la moitié droite est notée `s2`. Alors le nombre d'inversions dans `s` est le nombre d'inversions dans `s1` plus le nombre d'inversions dans `s2` plus la somme du nombre d'éléments de `s1` supérieurs à un élément de `s2` pour tous les éléments de `s2` (on peut aussi faire la somme dans l'autre ordre).

Exercice 11 : Quelle serait la complexité en suivant naïvement ce principe (et en déterminant simplement la somme mentionnée à la fin du paragraphe ci-avant ?)

Exercice 12 : Écrire une fonction qui prend en entrée une liste et qui retourne le couple formé par sa première et sa deuxième moitié. Attention à laisser les éléments dans le même ordre. Déterminer aussi la complexité de cette fonction.

Exercice 13 : Écrire une fonction qui utilise efficacement le principe décrit en début de section pour déterminer le nombre d'inversions dans une liste ou un tableau (au choix).

Exercice 14 : Déterminer la complexité de la fonction précédente.