TP2 : Algorithmes de tri

Exercice 1 (Tri par insertion par tableaux)

1. Écrire une fonction en OCaml `insere i t` qui, le tableau t vérifiant

$$t.(0) \leq t.(1) \leq \ldots \leq t.(i-1),$$

le modifie de telle façon qu'il vérifie :

$$t.(0) \leq t.(1) \leq \ldots \leq t.(i-1) \leq t.(i).$$

2. En déduire une fonction en OCaml `tri_insertion t` qui trie le tableau t d'après l'algorithme du tri par insertion.
3. Déterminer la complexité temporelle du tri par insertion par tableaux.

Exercice 2 (Tri par sélection par listes)

1. Écrire une fonction en OCaml `minimum_et_reste l` qui isole le minimum d'une liste des autres éléments. Par exemple :

```
minimum_et_reste [5;4;8;2;6;7;7;2;9];;
- : int * int list = 2 , [5;4;8;2;6;7;7;9]
```

2. En déduire une fonction en OCaml `tri_selection l` qui trie une liste l d'après l'algorithme du tri par sélection.
3. Déterminer la complexité temporelle du tri par sélection par listes.

Exercice 3 (Tri à bulles par listes)

1. Écrire une fonction en OCaml `une_etape l` effectuant la première étape du tri à bulles sur une liste l. Par exemple :

```
une_etape [5;4;8;2;6;7;7;2;9];;
- : bool * int list = true , [2;5;4;8;2;6;7;7;9]
une_etape [7;7;8;9];;
- : bool * int list = false , [7;7;8;9]
```

2. En déduire une fonction en OCaml `tri_bulles l` qui trie une liste l d'après l'algorithme du tri à bulles.
3. Déterminer la complexité temporelle du tri à bulles par listes.

Exercice 4 (Tri pancake)

On cherche à trier une série de n entiers avec la seule opération consistant à retourner l'ensemble des éléments jusqu'à un indice i. Voici une illustration de cette opération :

On peut imaginer un cuisinier qui, disposant d'une pile de pancakes, peut insérer sa spatule au milieu de la pile et retourner tous les pancakes se trouvant au-dessus. Il cherche à effectuer de tels retournements pour trier la pile de pancakes par diamètres croissants.

1. Montrer qu'une pile de pancakes peut être triée par ordre croissant en moins de $2n-3$ retournements.
2. Programmation par tableaux.
   (a) Écrire une fonction en OCaml `retourne` telle que `retourne t i` modifie un tableau t pour le "retourner" jusqu'à l'indice i inclus.
   (b) Écrire une fonction en OCaml `maximum` telle que `maximum t i` rend l'indice de l'élément maximum de t entre l'indice 0 et un indice i donné en argument.
   (c) En déduire une fonction en OCaml `tri_pancake` qui trie un tableau t par retournements successifs.
   (d) Déterminer la complexité temporelle du tri pancake par tableaux.
3. Programmation par listes.
   (a) Écrire une fonction en OCaml `retourne` pour les listes.
   (b) Écrire une fonction en OCaml `maximum` pour les listes.
   (c) En déduire une fonction en OCaml `tri_pancake` pour les listes.

Exercice 5 (Tri fusion par tableaux)

1. Écrire une fonction en OCaml `fusion t l m u` qui, le tableau t vérifiant

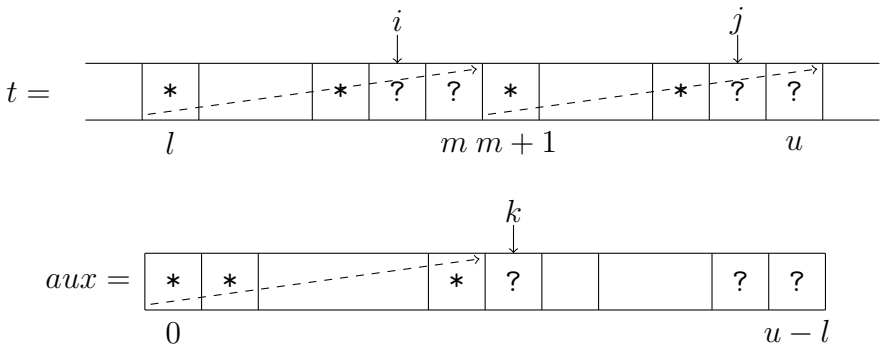
$$\begin{cases} t.(l) \leq t.(l+1) \leq \ldots \leq t.(m-1) \leq t.(m) \\ t.(m+1) \leq t.(m+2) \leq \ldots \leq t.(u-1) \leq t.(u) \end{cases},$$

   le modifie de telle façon à ce qu'il vérifie :

$$t.(l) \leq t.(l+1) \leq \ldots \leq t.(m) \leq \ldots \leq t.(u-1) \leq t.(u).$$

   L'idée peut s'illustrer de la manière suivante : vous avez deux paquets de cartes triés, les plus petites sur le dessus, chaque étape élémentaire consiste à choisir la carte la plus petite parmi les deux au sommet de chaque paquet et à la placer sur le paquet trié en construction.

   Plus formellement, nous respecterons l'invariant de boucle suivant :



où :

$$\begin{cases} \{aux.(p)\,,\, p \in [\![0, k-1]\!]\} = \{t.(p)\,,\, p \in [\![l, i-1]\!]\} \cup \{t.(p)\,,\, p \in [\![m, j-1]\!]\} \\ aux.(0) \leq aux.(1) \leq ... \leq aux.(k-1) \end{cases},$$

et nous terminerons en recopiant le tableau aux dans le tableau t.

2. Écrire une fonction en OCaml `tri_fusion t` qui trie le tableau t d'après l'algorithme du tri fusion.
3. Prouver la terminaison des fonctions `fusion` et `tri_fusion`.
4. Déterminer la complexité temporelle du tri fusion par tableau.
