UNIVERSITÉ PARIS-SUD 11

Examen final
Cours d'analyse, algorithmique
—Master 2 CCI—

Les documents manuscrits, sujets de travaux pratiques et dirigés ainsi que les supports de cours sont autorisées. Tous les autres documents tels que livres, calculatrices, téléphones portables et ordinateurs sont interdits.

Les exercices sont indépendants. Si l'on ne sait pas justifier une question, on peut néanmoins utiliser la réponse dans la suite.

Durée : 2 heures

▸ Exercice 1. (Tri par sélection/échange)

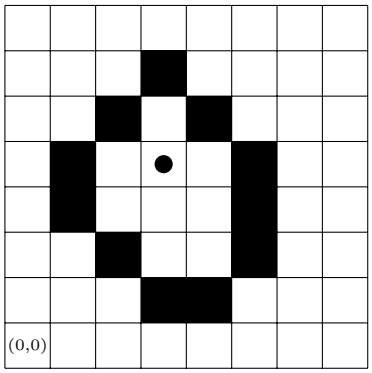
Soit T un tableau de taille n. Pour trier T, on peut procéder de la manière suivante :
- rechercher le plus petit élément de T, puis, s'il n'est pas en position 0, l'échanger avec l'élément d'indice 0 ;
- rechercher le second plus petit élément de T (c'est-à-dire le plus petit parmi les éléments d'indice plus grand que 1), puis, s'il n'est pas en position 1, l'échanger avec l'élément d'indice 1 ;
- rechercher le troisième plus petit élément de T (c'est-à-dire le plus petit parmi les éléments d'indice plus grand que 2), puis, s'il n'est pas en position 2, l'échanger avec l'élément d'indice 2 ;
- continuer de cette façon jusqu'à ce que le tableau soit entièrement trié.

Cette méthode est nommée tri par sélection/échange.

1. Écrire un algorithme pour chercher la position d'un plus petit élément d'un tableau (non trié).
2. En déduire un algorithme pour le tri par sélection/échange.
3. Montrer que l'algorithme vérifie l'invariant suivant : à la fin de la i-ème étape, les i premiers éléments du tableau sont les plus petits et ils sont triés. Plus précisément, à la fin de la i-ème étape on a
   - pour tout j tel que $0 \leq j < i-1$ on a $T[j] \leq T[j+1]$ et
   - pour tout j tel que $i \leq j < n$, on a $T[i-1] \leq T[j]$.
4. En déduire que, à la fin, le tableau est trié c'est-à-dire que, pour tout j tel que $0 \leq j < n-1$, on a $T[j] \leq T[j+1]$
5. Quel est le nombre de comparaisons d'éléments du tableau effectuées par cet algorithme ? Quelle est sa complexité ?
6. Quel est dans le cas le pire, le nombre de copies d'éléments du tableau effectuées par cet algorithme ?
7. Expliquer pourquoi cet algorithme est intéressant pour trier des éléments dont la comparaison est rapide mais la copie lente.

▸ **Exercice 2. (Fonction récursive)**

On considère un tableau à deux dimensions contenant des cases noires ou blanches. Le tableau est initialement le suivant (on considère que $(0, 0)$ est en bas à gauche).



On exécute l'algorithme `noircir` suivant :

```
Algorithme noircir(x, y)

si tab[x,y] = blanc alors
   tab[x,y] <- noir
   noircir(x, y-1)
   noircir(x, y+1)
   noircir(x-1, y1)
   noircir(x+1, y)
```

```
Algorithme noircir2(x, y)

si tab[x,y] = blanc alors
   noircir2(x, y-1)
   noircir2(x, y+1)
   noircir2(x-1, y1)
   noircir2(x+1, y)
   tab[x,y] <- noir
```

1. On appelle l'algorithme `noircir` avec les coordonnées de la case contenant un point. Quel est l'état du dessin à la fin ?
2. Que pensez vous de l'algorithme `noircir2` ?

Les questions suivantes concernent l'algorithme `noircir`.

3. Dire dans quel ordre les cases ont-elles été noircies.
4. Montrer que le nombre d'appels récursifs est $1 + 4n$ où n est le nombre de cases noircies.
5. En déduire la complexité de cet algorithme.

▸ **Exercice 3. (Palindromes)**

Un mot est un *palindrome* s'il peut être lu indifféremment de droite à gauche ou de gauche à droite (comme *été*, *laval*, ...). On étend cette définition aux phrases en ignorant les séparateurs, la ponctuation et la différence entre minuscules et majuscules (ex : *Esope reste et se repose.*). Le but de l'exercice est de déterminer si une phrase est un palindrome. Comment procéder pour ne pas transformer la phrase d'origine ?

On suppose connues les fonctions suivantes :
- `longueur(ch)` qui retourne la longueur de la chaîne de caractères `ch` ;
- `copie(ch)` qui copie la chaîne de caractères `ch`.
- `est_lettre(c)` qui répond si un caractère `c` est une lettre ou non ;
- `majuscule(c)` qui retourne la majuscule qui correspond à `c` si `c` est une minuscule et le caractère `c` lui même si sinon.

On ne se préoccupera pas des éventuelles allocations mémoires.

1. Écrire une fonction `miroir` qui reçoit une chaîne de caractères et la remplace par une la chaîne écrite en sens inverse.
2. Utiliser `miroir` pour déterminer si une chaîne est un palindrome.
3. Procéder directement sur la chaîne à tester sans la modifier.

▸ **Exercice 4. (Sélections dans un tableau)**

On manipule une collection d'elements du type `elem` stockée dans un tableau `Tab` : Le nombre d'élements est stocké dans la variable `taille`, on suppose donc que les éléments $0 \leq i <$ `taille` sont initialisés.

1. Rappeler l'algorithme de la fonction `supprime(Tab, i)` qui supprime un élément à une position $0 \leq$ `i` $<$ `taille` donnée.
2. Quel est la complexité de cette fonction ?
3. Soit $P(e)$ un prédicat ou e est un élément. Par exemple `elem` désigne les entiers et $P(e) =$ «est pair». En utilisant, la fonction précédente écrire une fonction qui supprime de `Tab` tous les élements de vérifiant pas le prédicat. Dans l'exemple précédent si on donne `Tab` $= [1, 4, 2, 3, 6, 4, ?, ?, ?]$ avec `taille` $= 6$, à la fin on a `Tab` $= [4, 2, 6, 4, ?, ?, ?, ?, ?]$ avec `taille` $= 4$.
4. Montrer que la complexité de cette fonction est $O(\texttt{taille}^2)$.
5. En s'inspirant des algorithmes de partition vu en cours, écrire un algorithme dont la complexité est $O(\texttt{taille})$.
6. Justifier la complexité de l'algorithme proposé.