Aix*Marseille université

Domaine Sciences et Technologies
Portail 1 René Descartes

Introduction à l'informatique : TD 5

Code UE : SPO1U57

Année 2023–2024

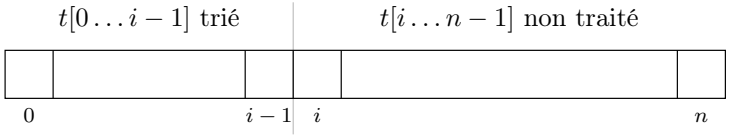
Tris

Exercice 1 (Permutations)

1. On se donne trois variables entières a, b et c. Quel algorithme permet d'effectuer une *permutation circulaire* vers la droite de ces trois variables, c'est-à-dire de faire en sorte qu'à la fin, a contienne la valeur originelle de c, b la valeur de a et c la valeur de b ?
2. Comment faire de même avec un tableau $t = [t[0], t[1], \ldots, t[n-1]]$ de n entiers, plutôt que trois variables ?
3. Évaluer la complexité en temps de cet algorithme.

Exercice 2 (Tri à bulles) La question de comment trier un tableau, c'est-à-dire ordonner ses éléments selon un ordre défini a priori, est très importante en informatique. En effet, très souvent, quand on récupère des données et qu'on les organise dans une structure de données linéaire telle qu'un tableau, un premier traitement consiste à simplement les insérer dans l'ordre où elles arrivent. Or, quand il s'agit de les utiliser pour réaliser un calcul, il peut être très utile d'y accéder de manière ordonnée. Dans ce cadre, de nombreuses recherches ont été menées par le passé sur la manière de trier un tableau, qui ont abouti à des algorithmes d'efficacités variées. Dans cet exercice, nous allons nous intéresser à l'un de ces algorithmes, fondé sur la méthode dite "des bulles".

Il s'agit d'une méthode qui opère par permutations successives sur le tableau à trier. Le principe est le suivant, en admettant qu'on procède à un tri par ordre croissant. Considérons un tableau $t[0 \ldots n-1]$. On le trie de gauche à droite. Voilà la situation, une fois qu'on a trié les i premiers éléments, avec $1 \leq i < n-1$:



L'objectif est d'augmenter la taille de la partie $t[0 \ldots i-1]$ triée. Pour ce faire, on parcourt le sous-tableau $t[i \ldots n-1]$ de la droite vers la gauche et, chaque fois que deux éléments consécutifs ne sont pas ordonnés, on les permute (on fait "remonter les bulles"). À titre d'exemple, si l'on prend le mot "BATEAUX" et qu'on le voit comme un tableau de 7 caractères, la méthode des bulles le trie selon l'ordre alphabétique de la manière suivante :

i	trié / non traité	min. partie non traitée	après "remontée des bulles"
0	/ BATEAUX	A	A / BATEUX
1	A / BATEUX	A	AA / BETUX
2	AA / BETUX	B	AAB / ETUX
3	AAB / ETUX	E	AABE / TUX
4	AABE / TUX	T	AABET / UX
5	AABET / UX	U	AABETU / X
6	AABETU / X		

1. Exécuter le tri à bulles du mot "INFORMATIQUE" selon l'ordre alphabétique (vous représenterez le résultat sous forme d'un tableau tel que celui donné ci-dessus).
2. Écrire l'algorithme décrivant de manière formelle le tri d'un tableau d'entiers selon l'ordre croissant par la méthode des bulles.
3. Évaluer la complexité en temps de cet algorithme : donner le nombre d'opérations (comparaisons, lecture d'un élément du tableau et affectations) réalisées dans le pire des cas sur un tableau de longueur n.

4. En regardant à nouveau l'exécution de l'algorithme sur BATEAUX et INFORMATIQUE, qu'observez-vous ? Pourrait-on l'améliorer afin d'éviter des comparaisons inutiles ? Comment ?
5. Proposer un nouvel algorithme de tri à bulles qui améliore significativement le précédent en tenant compte de vos observations précédentes. Quelle est sa complexité ? *Il est possible de faire s'arrêter une fonction en plaçant l'instruction* `return` *qui ne renvoie rien mais stoppe l'exécution où elle en est...*

Exercice 3 En cours, on a étudié un autre algorithme de tri, le tri par insertion, dont voici une description possible :

```
def trier_par_insertion(A):
    n = len(A)
    for i in range(1, n):
        x = A[i]
        j = i
        while j > 0 and x < A[j - 1]:
            A[j] = A[j - 1]
            j = j - 1
        A[j] = x
```

1. Rappelez la raison pour laquelle ce code ne retourne rien.
2. Que se passe-t-il si on remplace la troisième ligne de l'algorithme par `for i in range(n)` ?
3. Montrer que cet algorithme termine. *Indication : La seule raison pour laquelle il pourrait ne pas terminer est la boucle* `while`*, comme dans l'algorithme de recherche dichotomique. Montrer que cette boucle termine bien dans tous les cas.*
4. On a vu en cours que, dans le pire des cas, la complexité du tri par insertion est en $\mathcal{O}(n^2)$. On peut raisonnablement se poser la question de savoir si on a surestimé le nombre d'opérations élémentaires. En fait, il n'en est rien. Trouvez donc une suite de tableaux $(t_n)_{n\in\mathbf{N}}$ avec t_n un tableau de longueur n telle que le nombre C_n d'opérations élémentaires effectuées lors du tri du tableau t_n est de la forme $an^2 + bn + c$ avec a, b, c des constantes et $a > 0$.