Domaine Sciences et Technologies
Portail 1 René Descartes

Introduction à l'informatique : TD 5
Code UE : S14IN1I1

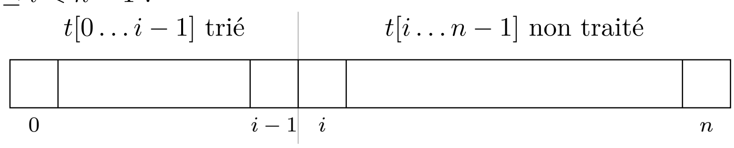
Année 2020–21 **Tris**

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 $t = \big[t[0], t[1], \ldots, t[n-1]\big]$ 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 :

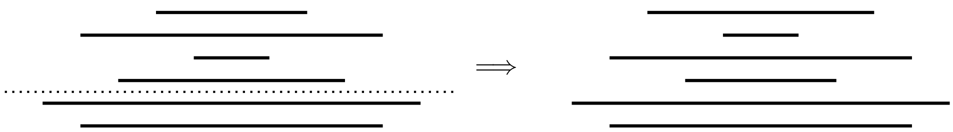
i	trié / non traité	min. partie non traitée	après "remontée des bulles"
0	/ BATEAUX	A	A / BATEUX
1	A / BATEUX	A	AA / BETUX
2	AA / BETUX	B	AAB / ETUX
3	AAB / ETUX	E	AABE / TUX
4	AABE / TUX	T	AABET / UX
5	AABET / UX	U	AABETU / X
6	AABETU / X		

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* `renvoyer()` *qui ne renvoie rien mais stoppe l'exécution où elle en est...*

Exercice 3 (Tri du crêpier psychorigide)

Dans une crêperie du Vieux Port, deux crêpiers se relaient en cuisine. Toutes les crêpes n'ont pas exactement le même diamètre. Au moment du changement de crêpier, s'il reste des crêpes déjà précuites, le crêpier sur le départ, un peu psychorigide sur les bords, veut laisser à son collègue une pile de crêpes triée de la plus grande en bas, jusqu'à la plus petite en haut. Oui, mais voilà : la cuisine est minuscule ! La seule possibilité pour le crêpier psychorigide est d'utiliser sa grande spatule, de la planter entre deux crêpes de la pile de crêpe et de retourner la totalité du tas de crêpes au-dessus, le tout sur la même pile de crêpe. Par exemple, dans la pile de crêpes à gauche ci-dessous, si le crêpier plante sa spatule à l'endroit des pointillés et retourne le tas du dessus, il obtient la pile de crêpes de droite.



1. Comment le crêpier doit-il s'y prendre pour trier sa pile de crêpes ? Décrire une méthode que le crêpier peut utiliser facilement : en particulier, notez qu'il est facile pour le crêpier de trouver le plus grande crêpe dans une pile de crêpes...
2. Décrire votre algorithme en pseudo-code en supposant que la pile de crêpe est représentée par le tableau `t[0...n−1]` des diamètres des crêpes en commençant par la crêpe la plus basse. Vous pourrez utiliser
   — la fonction `retourner_spatule(t, i)` pour dire au crêpier de planter sa spatule en-dessous de la `i`-ème crêpe du tas `t` et de retourner la pile au-dessus (l'exemple du début correspond donc à `retourner_spatule(t, 2)`) ;
   — la fonction `plus_grande_crêpe(t, i)` qui retourne la plus grande crêpe dans le tas `t` au-dessus de la `i`-ème crêpe, c'est-à-dire dans le sous-tableau `t[i...n−1]` : ainsi `plus_grande_crêpe(t, 0)` renvoie la plus grande crêpe du tas, alors que `plus_grande_crêpe(t, 1)` fait de même en ignorant la crêpe du bas.
3. Combien d'opérations élémentaires (recherche de la plus grande crêpe dans un sous-tas et retournement d'un sous-tas) effectue le crêpier s'il utilise votre algorithme dans le pire des cas, en fonction du nombre n de crêpes dans la pile ?
4. Proposer des algorithmes en pseudo-code pour réaliser les fonctions `plus_grande_crêpe` et `retourner_spatule`, en ne s'autorisant comme opérations élémentaires sur les tableaux que la lecture et l'écriture dans une case du tableau.
5. En supposant désormais qu'on compte comme opérations élémentaires les comparaisons d'éléments du tableau et les lectures et écritures dans le tableau, quelle est la complexité du tri du crêpier psychorigide ?

En cours, on a étudié un autre algorithme de tri, le tri par insertion, dont voici un pseudo-code possible :

```
fonction trier_par_insertion(tableau) :
    n := longueur(tableau)
    Pour i de 1 à n−1 faire
        x := tableau[i]
        # insérer x parmi les i premiers éléments
        j := i
        Tant que ((j > 0) et (x < tableau[j−1])) faire
            # décaler d'un élément
            tableau[j] := tableau[j−1]
            j := j−1
        FinTantQue
        # ici, x ≥ tableau[j−1] ou bien j=0
        tableau[j] := x
    FinPour
    # le tableau est trié !
```

Exercice 4

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 `Pour i de 0 à n−1 faire` ?
3. Montrer que cet algorithme termine. *Indication : La seule raison pour laquelle il pourrait ne pas terminer est la boucle* `Tant que`*, 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 $a > 0$.