Analyse d'algorithme

Montrer la terminaison, la correction, et étudier la complexité :

— Exponentiation rapide

— Tri insertion

— Tri sélection

— Dichotomie

Exercice 1. *Tri par gnome*

L'algorithme de tri par gnome, ou **gnome_sort**, est décrit par le pseudo-code suivant :

Algorithme 1 : gnome_sort

Entrée(s) : T tableau de n éléments de E
Sortie(s) : Rien, T est trié

```
1 i ← −1;
2 tant que i < n − 1 faire
3     si i ≥ 0 et T[i] > T[i + 1] alors
4         Échanger T[i] et T[i + 1];
5         i ← i − 1;
6     sinon
7         i ← i + 1;
```

Q1. Exécuter cet algorithme sur un tableau exemple et vérifier que le tableau est trié en sortie.

Q2. Montrer la correction de cet algorithme.

On s'intéresse maintenant d'une part à la preuve que cet algorithme termine, et d'autre part à l'étude de sa complexité. Les deux problèmes vont passer par la recherche d'un variant de boucle.

Pour T un tableau de taille n, et $k \in [\![0, n-1]\!]$. Le **nombre d'inversions de T en k**, noté $\mathbf{inv}_k(T)$ est

$$\mathbf{inv}_k(T) = \mathbf{card}(\{i \in [\![0, n-1]\!] \mid (i < k \text{ et } T[i] > T[k]) \text{ ou } (i > k \text{ et } T[i] < T[k])\})$$

Le **nombre d'inversions de T**, noté $\mathbf{inv}(T)$ est

$$\mathbf{inv}(T) = \sum_{k=0}^{n-1} \mathbf{inv}_k(T)$$

Q3. Donner le nombre d'inversions des tableaux suivants :

a) $T_a = [1, 2, 5, 3, 4, 6]$
b) $T_b = [1, 2, 3, 4, 5]$
c) $T_c = [7, 2, 6, 1, 4, 3, 5]$

Q4. Proposer un variant de boucle pour le tri par gnome.

Q5. En déduire la complexité asymptotique de l'algorithme.

Exercice 2. *Tableaux triés*

Quand un tableau est trié, de nombreuses opérations s'effectuent plus simplement que sur les tableaux non-triés. Pour les problèmes suivants, donnez un algorithme pour le cas général, et un algorithme spécialisé aux tableaux triés, ayant une meilleure complexité

Q1. Étant donné un tableau T, déterminer son maximum.

Q2. Étant donné un tableau T de taille n et un nombre x, déterminer si x est un élément de T.

Q3. Étant donné un tableau T de taille n, déterminer les indices $i \neq j \in [\![0, n-1]\!]$ tels que :

$$\forall k \neq l, |T[k] - T[l]| \geq |T[i] - T[j]|$$

i.e. les indices des deux éléments du tableau étant les plus proches.

Q4. Étant donné T un tableau de taille n d'entiers positifs et x un entier positif, déterminer s'il existe i, j tels que $x = T[i] + T[j]$.

Exercice 3. *Complexité précise*

On s'intéresse au problème suivant : étant donné un tableau T d'éléments d'un ensemble ordonné, déterminer le maximum de T. La seule opération élémentaire considérée ici est la comparaison, c'est à dire le fait de tester une inégalité ($<, \leq, >, \geq$) entre deux éléments.

Q1. Donner un algorithme en $\mathcal{O}(n)$. Combien de comparaisons fait-il exactement ?

On veut maintenant obtenir les deux éléments maximaux de T.

Q2. Proposer un algorithme pour résoudre ce problème, et donner le nombre exact de comparaisons dans le pire des cas.

Exercice 4. *Interrupteurs*

Vous vous trouvez dans une pièce. Au mur se trouvent N interrupteurs, pouvant chacun avoir deux positions, que l'on notera 0 et 1. Au départ, tous les interrupteurs sont en position 0.
La pièce ne comporte qu'une porte, qui ne s'ouvre que si les interrupteurs sont dans la bonne configuration parmi les 2^N possibilités.
Mettez au point une stratégie pour trouver la combinaison, en appuyant sur le moins d'interrupteurs possible.

Exercice 5. *Hasta la vistar*

On considère un groupe de n personnes. Chaque personne $i \in [\![0, n-1]\!]$ peut connaître ou non chaque autre personne $j \in [\![0, n-1]\!]$ (avec $i \neq j$).

On appelle $\star$***Star***$\star$ une personne qui ne connaît personne, mais que tout le monde connaît.

On s'intéresse au problème suivant :

Étant donné un groupe de n personnes, contient-il une $\star$***Star***$\star$?

Pour résoudre ce problème, on considère des algorithmes qui peuvent poser des questions du type "***Est ce que*** i ***connaît*** j ***?***".

On note $i \to j$ une telle question, et on compte la complexité d'un algorithme en fonction du nombre de questions qu'il pose.

Q1. Combien peut-il y avoir de stars dans un groupe ?

Q2. Comment modéliser cette situation ? Autrement dit, quelle(s) structure(s) simple(s) peut-on utiliser pour représenter les données du problème ? Avec cette modélisation, à quoi correspond une question ?

Q3. Donner un algorithme en $\mathcal{O}(n^2)$ permettant de trouver une star dans un groupe.

Q4. On considère une question $i \to j$ avec $i, j \in [\![0, n-1]\!]$. Montrer que quelle que soit la réponse, on peut éliminer i ou j comme potentielles $\star$***Stars***$\star$.

Q5. Donner un algorithme en $\mathcal{O}(n)$ permettant de trouver une star dans un groupe.

Q6. Dans le pire cas, quel est le nombre **exact** de questions posées par votre algorithme, en fonction de n ?

Q7. Trouver un algorithme permettant de trouver une star en au plus $3n - \lfloor \log(n) \rfloor - 3$ questions.

On peut montrer qu'il n'existe pas d'algorithme posant strictement moins de questions !

Exercice 6. *Tri à bulle*

Étudier la correction et la complexité de l'algorithme suivant :

Algorithme 2 : Tri à bulle

Entrée(s) : T tableau de taille n
Sortie(s) : Rien, T est trié dans l'ordre croissant

```
1 pour i = 0 à n − 1 faire
2     pour j = 0 à n − 2 faire
3         si T[j] > T[j + 1] alors
4             Échanger T[j] et T[j + 1];
```

Exercice 7. *Fausse pièce*

On dispose de n pièces de monnaie identiques. Cependant, l'une d'entre elle est une fausse, et pèse **plus lourd** que les vraies pièces. L'objectif du problème est de déterminer la fausse pièce le plus efficacement possible. Pour cela, on dispose de deux balances à plateaux, la balance **simple** et la balance **complexe**.

— La balance simple permet de comparer deux pièces a et b entre elles, et dit si la pièce a est plus lourde que b, moins lourde, ou si elles font le même poids.

— La balance complexe permet de comparer deux **tas** de pièces A et B, et dit lequel des deux est le plus lourd, ou bien si ils sont de même poids.

L'utilisation d'une balance est l'unique opération élémentaire du problème. La complexité des algorithmes s'exprimera donc en fonction du nombre de pesées.

Q1. Déterminer un algorithme en $\mathcal{O}(n)$ n'utilisant que la balance simple.

Q2. Déterminer un algorithme en $\mathcal{O}(\log n)$ utilisant la balance complexe.

Q3. On suppose maintenant que l'on ne sait pas si la fausse pièce est plus lourde ou plus légère. Déterminer un algorithme en $\mathcal{O}(n)$ avec la balance simple, et un algorithme en $\mathcal{O}(\log n)$ avec la balance complexe.

Exercice 8. *Occurrences*

Q1. Proposer un algorithme quadratique permettant de déterminer l'élément ayant le plus d'occurrences dans un tableau T donné.

Q2. Proposer une amélioration simple de l'algorithme précédent réduisant la complexité à $\mathcal{O}(n \log n)$.

Soit $K \in \mathbb{N}$. On dit qu'un tableau T est **borné par** K si ses valeurs sont comprises entre 0 et $K-1$. Étant donné un tableau T borné par K de taille n , On appelle **tableau des occurrences** le tableau U de taille K tel que :

$$\forall k \in [\![0, K-1]\!], U[k] = \textbf{card}(\{i \in [\![0, n-1]\!] \mid T[i] = k\}$$

i.e. dont la case k contient le nombre d'occurrences de k dans U.

Q3. Pour $K = 7$, donner le tableau des occurrences de $[5, 0, 2, 2, 1, 3, 5, 1]$.

Q4. Proposer un algorithme en $\mathcal{O}(N)$ permettant de construire le tableau des occurrences d'un tableau T borné par K de taille N. Ce tableau prendra en entrée T ainsi que l'entier K, et devra vérifier que T est bien borné par K.

Q5. En utilisant le tableau des occurrences, proposer un algorithme en $\mathcal{O}(N + K)$ permettant de déterminer l'élément ayant le plus d'occurrences dans un tableau T borné par K de taille N.

Q6. Proposer un algorithme permettant de trier un tableau T borné par K de taille N en temps $\mathcal{O}(N + K)$.