# Correction TD3

MP2I Lycée Pierre de Fermat

## Exercice 1.

Dans l'ordre de la suite la plus lente à la plus rapide :

— $1 + (-1)^n$

— $3$

— $\log_2(n)$ et $ln(n^3)$

— $\sqrt{n}$

— $n$ et $(\log_2 n)^7 + n$

— $n \log_5(n)$

— $n^2 - 2n + 8$

— $n^{\log(n)}$

— $\pi^{\frac{n}{2}}$

— $e^n$

— $n!$

— $n^n$

— $e^{n^2}$

— $n^{n^n}$

Notons que “rapide” signifie ici ”qui grandit rapidement”, mais un algorithme en temps $\Theta(n^{n^n})$ serait extrêmement **lent**.

## Exercice 2.

**Q1.** Soit $u = (u_n)_{n\in\mathbb{N}}$ une suite réelle positive **croissante**. Soit $n \in \mathbb{N}$. Pour $i \in [\![0, n-1]\!]$, par croissance, $u_i \leq u_n$. Donc, en sommant :

$$\sum_{i=0}^{n-1} u_i \leq \sum_{i=0}^{n-1} u_n = nu_n$$

Et ce pour tout $n \in \mathbb{N}$. Donc, pour $n_0 = 0$ et $C = 1$, on a bien :

$$\forall n \geq n_0, \sum_{i=0}^{n-1} u_i \leq Cnu_n$$

c'est à dire $\sum_{i=0}^{n-1} u_i = \mathcal{O}(nu_n)$

**Q2.** La suite $u = (u_n)_{n\in\mathbb{N}}$ avec $u_0 = 1$ et $u_n = 0$ pour tout $n \geq 1$ fournit un contre exemple. La suite des inverses $v = (\frac{1}{n+1})_{n\in\mathbb{N}}$ fonctionne également.

# Exercice 3.

On peut donner un premier algorithme naïf qui regarde chaque carré et ajoute sa surface à la surface totale de sa couleur :

---

**Entrée(s) :** $l$ tableau de $n$ entiers, $L$ tableau de $m$ entiers
**Sortie(s) :** Triplet $(s_A, s_B, s_C)$ donnant la surface totale de couleur $A, B, C$

$s = [0, 0, 0]$ `// 0 pour` $A$`, 1 pour` $B$`, 2 pour` $C$
**pour** $i = 0$ **à** $n - 1$ **faire**
  **pour** $j = 0$ **à** $m - 1$ **faire**
    $s[(i + j) \mod 3] \leftarrow s[(i + j) \mod 3] + l_i \times L_j$;
**retourner** $s$

---

Cet algorithme est en $\mathcal{O}(nm)$ car la boucle intérieure effectue $m$ passages et la boucle extérieure en effectue $n$.

On peut obtenir un algorithme plus efficace en regroupant les carrés de même couleur. Commençons par noter, pour $c \in \{0, 1, 2\}$, $S_c$ la surface totale de couleur $c$. On a par exemple :

$$S_0 = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} (l_i \times L_j \text{ si } (i + j) = 0 \mod 3 \text{ et } 0 \text{ sinon})$$

On peut faire sortir le terme $l_i$ de la somme intérieure :

$$S_0 = \sum_{i=0}^{n-1} l_i \sum_{j\in[\![0,m-1]\!], i+j=0 \mod 3} L_j$$

Pour $i \in [\![0, n - 1]\!]$, intéressons nous au terme que l'on appellera $SL_i$ et défini par :

$$SL_i = \sum_{j\in[\![0,m-1]\!], i+j=0 \mod 3} L_j$$

Ce terme donne la largeur totale des cases de couleur $A$ sur la ligne de longueur $l_i$. Autrement dit, $l_i \times SL_i$ donne la surface totale de couleur $A$ sur la ligne $i$.

On remarque de plus que les valeurs de $SL_i$ se répètent par cycle de 3, car l'ensemble des $j \in [\![0, n-1]\!]$ tels que $i + j = 0 \mod 3$ ne dépend que du reste de $i$ modulo 3. Graphiquement, cela s'interprète par le fait que toutes les trois lignes, les cases de couleur $A$ sont aux même colonnes. On a donc :

$$\forall i \in [\![0, n - 1]\!], SL_i = SL_{i\%3}$$

où $i\%3$ est le reste de $i$ modulo 3.

Ainsi, on peut commencer par calculer $SL_0, SL_1$ et $SL_2$ en $\mathcal{O}(m)$, puis obtenir $S_0$ avec le calcul suivant :

$$S_0 = \sum_{i=0}^{n-1} l_i SL_{i\%3}$$

Comme on a déjà pré-calculé les $(SL_i)_{i=0,1,2}$, le calcul de $S_0$ se fait ensuite en $\mathcal{O}(n)$. Au total, la surface a pu être calculée en $\mathcal{O}(n+m)$. On peut ensuite appliquer la même méthode pour calculer $S_1$ et $S_2$. On a par exemple :

$$S_1 = \sum_{i=0}^{n-1} l_i SL_{(i+2)\%3}$$

# Exercice 4.

**Q1.** On cherche l'indice $i \in [\![0, N-2]\!]$ tel que $T[i] = 0$ et $T[i+1] = 1$. Alternativement, on peut aussi dire que l'on cherche $i \in [\![0, N-2]\!]$ maximal tel que $T[i] = 0$.
Jeter un téléphone de l'étage $i$ signifie "regarder la case $T[i]$", et casser un téléphone signifie "voir un 1 dans la case regardée".

**Q2.** Si l'on ne doit casser qu'un seul téléphone, on n'a pas le choix : on doit faire une recherche **linéaire**. On teste l'étage 0, puis l'étage 1, puis l'étage 2, etc... jusqu'à avoir trouvé un étage où le téléphone casse :

---

**Entrée(s) :** $T$ un tableau de $n$ booléens trié
**Sortie(s) :** $i \in [\![0, n-2]\!]$ tel que $T[i] = 0$ et $T[i+1] = 1$)

$i \leftarrow 0$;
`// Invariant de boucle:` $\forall j \in [\![0, i-1]\!], T[j] = 0$
**tant que** $T[i] = 0$ **faire**
  $i \leftarrow i+1$;
`// En sortie:` $T[i] = 1$
**retourner** $i$

---

L'algorithme termine car on sait qu'il existe une case de $T$ qui contient 1.

**Q3.** ...

**Q4.** Dans une recherche par dichotomie, on utilise deux variables, notons les $a$ et $b$, qui marquent les bornes de la zone de recherche. Lorsque l'on regarde la case du milieu de la zone délimitée, $T[m]$ avec $m = \lfloor \frac{a+b}{2} \rfloor$, selon que l'on voit 0 ou 1, on peut éliminer toutes les cases à gauche ou à droite de $m$.
Il y a plusieurs manières de gérer la dichotomie, on doit réfléchir à plusieurs questions :

— Quelle est la condition d'arrêt ?
— Est ce que l'on enlève la case du milieu dans le nouvelle intervalle de recherche ?
— Que renvoie t-on en sortie de boucle ?

La réponse est la même pour les trois questions : il faut réfléchir aux invariants que vérifient $a$ et $b$, et ça vous donne la réponse.
Par exemple, supposons que l'on pose comme invariants :

$$A : T[a] = 0$$
$$B : T[b] = 1$$

**Entrée(s) :** $T$ un tableau de $n$ booléens trié
**Sortie(s) :** $i \in [\![0, n-1]\!]$ tel que $T[i] = 0$ et $T[i+1] = 1$

$a \leftarrow 0$ `// borne inférieure de l'intervalle de recherche`
$b \leftarrow n-1$ `// borne supérieure (exclue) de l'intervalle de recherche`
`// Invariant:` $T[a] = 0, T[b] = 1$
**tant que** $b \neq a+1$ **faire**
  $m \leftarrow \lfloor \frac{a+b}{2} \rfloor$;
  **si** $T[m] = 1$ **alors**
    $b \leftarrow m$;
  **sinon**
    $a \leftarrow m$;
`// En sortie,` $a+1 = b$, $T[b] = 1$ `et` $T[a] = 0$
**retourner** $a$

Alors, la condition d'arrêt peut être $b = a+1$, qui exprime que les cases $a$ et $b$ se suivent. Dans ce cas, on peut renvoyer $a$. Mais alors, quand on regarde la case du milieu, on ne doit pas la sauter, sinon on risque d'enfreindre les invariants :
Montrons que $P$ : "$T[a] = 0, T[b] = 1$" est bien un invariant de boucle. On note $a_k, b_k$ les valeurs de $a$ et $b$ après $k$ passages de boucle.

- En entrée de boucle, $a_0 = 0$ et $b_0 = n-1$, et on sait que la première case contient 0 et que la dernière contient 1. $P(0)$ est vérifiée
- Supposons $P(k)$ pour un $k \in \mathbb{N}$, et considérons un $k+1$-ème passage. On note $m = \lfloor \frac{a_k+b_k}{2} \rfloor$.
  - Premier cas : si $T[m] = 1$ : Alors $b_{k+1} = m$, donc $T[b_{k+1}] = 1$, et $a_{k+1} = a_k$ et par HR $T[a_k] = 0$. Donc $P(k+1)$ est vérifiée.
  - Deuxième cas : totalement analogue.

$P$ est donc un invariant de boucle. En sortie, $a+1 = b$, donc $T[a] = 0$ et $T[a+1] = 1$ : l'algorithme est correct.

Calcul de complexité : Montrons que $b - a - 1$ est divisé par 2 à chaque tour de boucle. Plaçons nous à un tour $k$ quelconque, et montrons que :

$$b_{k+1} - a_{k+1} - 1 \leq \frac{b_{k+1} - a_{k+1} - 1}{2}$$

On a deux cas :

- Si $T[m] = 1$, alors $b_{k+1} = m$. Donc :

$$\begin{array}{rcll} b_{k+1} - a_{k+1} - 1 & = & \lfloor \frac{a_k+b_k}{2} \rfloor - a_k - 1 & \\ & \leq & \frac{a_k+b_k}{2} - a_k - 1 & \text{car } \lfloor x \rfloor \leq x \\ & \leq & \frac{b_k-a_k}{2} - 1 & \\ & \leq & \frac{b_k-a_k-1}{2} & \end{array}$$

- Sinon, $a_{k+1} = m$ :

$$\begin{array}{rcll} b_{k+1} - a_{k+1} - 1 & = & b_k - \lfloor \frac{a_k+b_k}{2} \rfloor - 1 & \\ & < & b_k - (\frac{a_k+b_k}{2} - 1) - 1 & \text{car } \lfloor x \rfloor + 1 > x \\ & < & \frac{b_k-a_k}{2} & \end{array}$$

Donc, $2(b_{k+1} - a_{k+1} - 1) < b_k - a_k$. Comme ce sont des entiers, on peut en déduire que $2(b_{k+1} - a_{k+1} - 1) \leq b_k - a_k - 1$. En redivisant par 2 :

$$b_{k+1} - a_{k+1} - 1 \leq \frac{b_k - a_k - 1}{2}$$

Ainsi, on fait au plus $\log_2(n-1)$ tours de boucle (la valeur de $b - a - 1$ en entrée de boucle). Chaque tour étant en temps constant, l'algorithme est bien en $\mathcal{O}(\log n)$.

**Q5.** L'idée est de procéder en deux phases. La première phase va tester des étages $u_0, u_1, \ldots$ jusqu'à ce que le téléphone casse à un étage $u_k$. Alors, on est sûr que l'étage limite se situe entre $u_{k-1} + 1$ et $u_k$. Comme il ne nous reste qu'un téléphone, on ne peut pas faire mieux qu'une recherche linéaire comme dans la question 1.
Cette méthode prendra au total $k + 1 + (u_{k+1} - u_k)$ étapes. Le but est donc d'équilibrer les deux phases. Plus la première phase est rapide, plus l'écart entre les $u_i$ est grand, et plus la deuxième phase est longue. Si l'on prend $u_i = i^2$, les deux phases prennent un temps $\mathcal{O}(\sqrt{n})$ :

---

**Entrée(s) :** $T$ un tableau infini de booléens trié contenant au moins un 1
**Sortie(s) :** $N \in \mathbb{N}$ tel que $T[N] = 0$ et $T[N+1] = 1$

$i \leftarrow 0$;
`// Première phase: encadrement de l'étage`
**tant que** $T[i^2] = 0$ **faire**
  $i \leftarrow i + 1$;
`// En sortie:` $N \in [\![i^2+1, (i+1)^2]\!]$
`// Deuxième phase: recherche linéaire`
**pour** $j = i^2 + 1$ **à** $(i+1)^2$ **faire**
  **si** $T[j] = 1$ **alors**
    **retourner** $j$

---

La première phase s'arrête lorsque $i = \lceil \sqrt{N} \rceil$, et prend donc $\mathcal{O}(\sqrt{N})$ étapes. La deuxième phase prend $(i+1)^2 - i^2 = 2i + 1$ étapes, soit à nouveau $\mathcal{O}(\sqrt{N})$.
La complexité totale est bien $\mathcal{O}(\sqrt{N})$.