Algorithmique avancée : Feuille de TD n^o 1

I Complexité des algorithmes

1 Complexité asymptotique

On considère les fonctions suivantes :

1) $f(n) = 2n^2 + 5n + 10$
2) (Travail Personnel) $f(n) = \frac{1}{5} n \log_2(n) + n$ où $\log_2$ est le logarithme en base 2. On rappelle que pour tous réels $a, b > 0$, et tout $x > 0$, il y a un rapport constant entre $\log_a(x)$ et $\log_b(x)$ ($\log_a(x) = \frac{\log_b(x)}{\log_b(a)}$), c'est pour cela que dans les équivalents asymptotiques on utilisera la notation log sans indiquer de base particulière.
3) (Travail Personnel) $f(n) = \sum_{i=0}^{d} c_i n^i$ avec $\forall i \in [0, d], c_i \geq 0$ et $c_d > 0$

Pour chaque fonction f ci-dessus, proposez une fonction g_1, la plus simple possible, telle que $f \in \Theta(g_1)$ et démontrez ce résultat. Donnez aussi une fonction g_2 telle que $f \in o(g_2)$ en le démontrant.

2 Analyse d'un algorithme simple

Pour les trois formulations suivantes du problème de calcul du plus grand diviseur d'un nombre entier $n \geq 2$, exprimer l'ordre de grandeur de la complexité temporelle maximale et de la complexité spatiale maximale en fonction de n[1]. En conclure l'algoritme le plus rapide pour le calcul du plus grand diviseur d'un nombre.

1) Recherche descendante du plus grand diviseur

   Notons pgd(n) le plus grand diviseur de n. On a les propriétés suivantes :
   - $1 \leq \text{pgd}(n) \leq n - 1$
   - $\text{pgd}(n) = 1 \Leftrightarrow n$ est premier.

   On déduit de ces propriétés l'algorithme suivant :

   Require: $n \geq 2$
   Ensure: $r = \texttt{pgd}(n)$
   $r \leftarrow n - 1$
   while n *mod* $r \neq 0$ **do** $r \leftarrow r - 1$
   return r

2) (Travail Personnel) Recherche ascendante du plus petit diviseur ≥ 2

   Nous pouvons remarquer que $\text{pgd}(n) = \frac{n}{p}$ où p est le plus petit diviseur de n qui soit ≥ 2, noté $\text{ppd}(n)$.

   On déduit de cette remarque l'algorithme suivant :

   Require: $n \geq 2$
   Ensure: $r = \texttt{pgd}(n)$
   $k \leftarrow 2$
   while n *mod* $k \neq 0$ **do** $k \leftarrow k + 1$
   $r \leftarrow n/k$
   return r

1. D'habitude la complexité est exprimée en fonction de la taille des données, ici on demande de l'exprimer en fonction de la valeur du nombre n

3) (Travail Personnel)Réduction de l'espace de recherche

   On peut remarquer que n non premier $\Rightarrow 2 \leq \text{ppd}(n) \leq \text{pgd}(n) \leq n - 1$. D'où, comme $ppd(n) \times pgd(n) = n$, n non premier $\Rightarrow (\text{ppd}(n))^2 \leq n$. On en déduit que si n ne possède pas de diviseurs compris entre 2 et $\lfloor \sqrt{n} \rfloor$, c'est qu'il est premier. Nous pouvons donc utiliser cette remarque pour réduire l'espace de recherche et proposer l'algorithme suivant :

   Require: $n \geq 2$
   Ensure: $r = \texttt{pgd}(n)$
   $k \leftarrow 2$
   while *(n mod $k \neq 0$) and ($k \leq n/k$)* **do** $k \leftarrow k + 1$
   if $k > n/k$ **then**
   $\quad r \leftarrow 1$
   else
   $\quad r \leftarrow n/k$
   return r

3 Analyse de l'algorithme de tri fusion (von Neumann 1945)

L'algorithme du tri fusion MergeSort peut s'exprimer de la manière récursive suivante, dans laquelle la fonction merge(T, i, j, k) intercale linéairement les élements des deux sous-tableaux triés $T[i:j]$ et $T[j+1:k]$ pour fournir le sous-tableau trié $T[i:k]$.

Function MergeSort(T, i, k)

Input: T : array
i, k integer so that $T_m \leq i \leq k \leq T_M$ with T_m, T_M the bounds of T
Ensure: `T is sorted between i and k`
if $k > i$ **then**
$\quad j \leftarrow (k+i)/2$
$\quad$ MergeSort(T, i, j)
$\quad$ MergeSort$(T, j+1, k)$
$\quad$ merge(T, i, j, k)

Function merge(T, p, q, r)

Input: T : array s.t. $T[p..q]$ and $T[(q+1)..r]$ are sorted
p, q, r integer s.t. $T_m \leq p \leq q \leq r \leq T_M$ with T_m, T_M the bounds of T
Ensure: `T is sorted between p and r`
$n_1 \leftarrow q - p + 1$; $n_2 \leftarrow r - q$; create the arrays $L[1..(n_1+1)]$ and $R[1..(n_2+1)]$
for $i \leftarrow 1$ *to* n_1 **do** $L[i] \leftarrow T[p+i-1]$
for $j \leftarrow 1$ *to* n_2 **do** $R[j] \leftarrow T[q+j]$
$i \leftarrow 1$; $j \leftarrow 1$; $L[n_1+1] = \infty$; $R[n_2+1] = \infty$
for $k \leftarrow p$ *to* r **do**
$\quad$ **if** $L[i] \leq R[j]$ **then** $T[k] \leftarrow L[i]$; i++
$\quad$ **else** $T[k] \leftarrow R[j]$; j++

1) Dérouler cet algorithme sur le tableau

40	10	30	45	10

2) Exprimer la complexité temporelle et spatiale en pire cas de la fonction merge en fonction de la taille des données.
3) Exprimer la complexité temporelle et spatiale en pire cas de MergeSort pour un tableau de taille n.