Calculabilité & Complexité

TD2

Exo 1 :

a. Ce programme cherche les doublons
b. $\sum_{i=1}^{n-1}(n-i) = \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i = (n-1)n - \frac{(n-1)n}{2} = \frac{n(n-1)}{2} \rightarrow \mathcal{O}(n^2)$
c. Pour obtenir un algorithme de meilleur complexité, on trie le tableau et on cherche 2 entiers consécutifs égaux -> On cherche $\mathcal{O}(n \log n)$

Exo 2 :

1. $T(1) = a, n > 1, T(n) = 2T\left(\frac{n}{2}\right) + bn$ avec a et b réels strictement positifs

   a. $T(n+1) \geq T(n)$ par induction
   $T(n+1) = 2T\left(\frac{n+1}{2}\right) + b(n+1) \geq 2\,T\left(\frac{n}{2}\right) + bn = T(n)$
   On a donc prouvé que $T(n)$ est croissant

   b. $n = 2^k$
   $T(n) = T(2^k) = 2T(2^{k-1}) + 2^k b$
   $2T(2^{k-1}) = 2^2\,T(2^{k-2}) + 2^k b + 2^k b$
   $\vdots$
   $2T(1) = 2^k T(1) + \sum_{i=1}^{k} 2^k b$

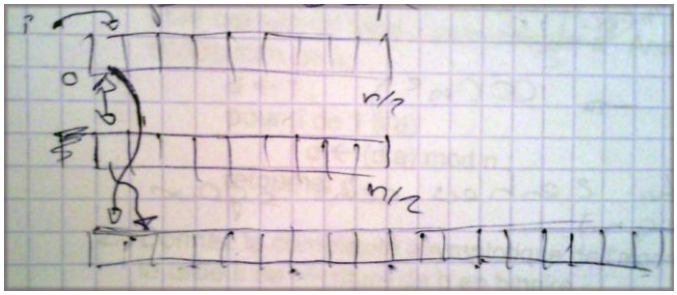
   $T(2^k) = a2^k + k2^k b$

   $\mathcal{O}(k2^k) = \mathcal{O}(n \log_2 n)$

   c. Montrer que $k > 1, (k+1) * 2^{k+1} < 4 * k * 2^k$
   $(k+1) * 2^{k+1} = 2(k+1) * 2^k \leq 4k2^k$

2. Diviser pour régner est en $\mathcal{O}(n \log_2 n)$ car on utilise une division successive du tableau en 2

3. Création d'un algorithme basé que diviser pour régner, à l'aide d'un tableau temporaire pour effectuer la fusion.
   On parcourt les 2 sous vecteurs en comparant les 2 éléments t on les copie dans le bon ordre par merge (tri)



Exo 3 :

1. Complexité asymptotique en fonction de b: $\mathcal{O}(b)$
   Mais b est écrit en nombre de bit ce qui donne un algorithme exponentiel

2. Complexité en fonction de la taille de l'écriture de b en binaire appelé k : $\mathcal{O}(2^k)$

3. a. Complexité asymptotique en fonction de la taille k de l'écriture de b en binaire :
   $\mathcal{O}(\log b) \rightarrow \mathcal{O}(\log 2^k)$