UNIVERSITE DE KINSHASA

FACULTE POLYTECHNIQUE

Deuxième Graduat GCEIM

TP D'ALGO & PROGRAMMATION

PROJET 2 (Questions théoriques)

Dirigé par :

PROF OLAMBA

Assistant MOBISA

GROUPE 12 :

MAVINGA MAVINGA 2GC

MPUMBU MUYAYA 2GC

NGALAMULUME KABUE 2GC

ANNEE ACADEMIQUE 2021-2022

TP D'ALGORITHMIQUE ET PROGRAMMATION GROUPE 12 G2 2023

Fiche 2: Les fondements mathématiques de l'algorithmique et l'évaluation des algorithmes

QUESTIONS (13)

1. Le nombre d'opérations primitives exécutées par les algorithmes A et B est (50 n log n) et (45 n^2), respectivement. Déterminez n_0 tel que A soit meilleur que B, pour tout $n \geq n_0$.

2. Le nombre d'opérations primitives exécutées par les algorithmes A et B est (140 n^2) et (29 n^3), respectivement. Déterminez n_0 tel que A soit meilleur que B pour tout $n \geq n_0$. Utiliser Matlab ou Excel pour montrer les évolutions des temps d'exécution des algorithmes A et B dans un graphique.

3. Montrer que les deux énoncés suivants sont équivalents :
(a) Le temps d'exécution de l'algorithme A est toujours O(f(n)).
(b) Dans le pire des cas, le temps d'exécution de l'algorithme A est O(f(n)).

4. Montrer que si d(n) vaut O(f(n)), alors (a x d(n)) vaut O(f(n)), pour toute constante $a > 0$.

5. Montrer que si d(n) vaut O(f(n)) et e(n) vaut O(g(n)), alors le produit d(n)e(n) est O(f(n)g(n)).

6. Montrer que si d(n) vaut O(f(n)) et e(n) vaut O(g(n)), alors d(n)+e(n) vaut O(f(n)+g(n)).

7. Montrer que si d(n) est O(f(n)) et e(n) est O(g(n)), alors d(n)−e(n) n'est pas nécessairement O(f(n)−g(n)).

8. Montrer que si d(n) est O(f(n)) et f (n) est O(g(n)), alors d(n) est O(g(n)).

9. Étant donné une séquence de n éléments S, l'algorithme D appelle l'algorithme E sur chaque
élément S[i]. L'algorithme E s'exécute en un temps O(i) lorsqu'il est appelé sur l'élément S[i]. Quel est le pire temps d'exécution de l'algorithme D?

10. Alphonse et Bob se disputent à propos de leurs algorithmes. Alphonse revendique le fait que son algorithme de temps d'exécution O(n log n) est toujours plus rapide que l'algorithme de temps d'exécution O(n^2) de Bob. Pour régler la question, ils effectuent une série d'expériences. À la consternation d'Alphonse, ils découvrent que si $n < 100$,

l'algorithme de temps $O(n^2)$ s'exécute plus rapidement, et que c'est uniquement lorsque $n \geq 100$ est le temps $O(n \log n)$ est meilleur. Expliquez comment cela est possible.

11. Concevoir un algorithme récursif permettant de trouver l'élément maximal d'une séquence d'entiers. Implémenter cet algorithme et mesurer son temps d'exécution. Utiliser Matlab ou Excel pour "fitter" les points expérimentaux et obtenir la fonction associée au temps d'exécution. Calculer par la méthode des opérations primitives le temps d'exécution de l'algorithme. Comparer les deux résultats.

12. Concevoir un algorithme récursif qui permet de trouver le minimum et le maximum d'une séquence de nombres sans utiliser de boucle.

13. Concevoir un algorithme récursif permettant de déterminer si une chaine de caractères contient plus de voyelles que de consonnes.

REPONSES

1. On a $A = 50n\log n$ et $B = 45n^2$
Pour que A soit meilleur que B, on doit avoir

$$A \geq B \quad \rightarrow \quad 50n \log n \geq 45\ n\text{^}2 \quad \rightarrow \quad n\_0 = 7{,}9432 \ldots$$

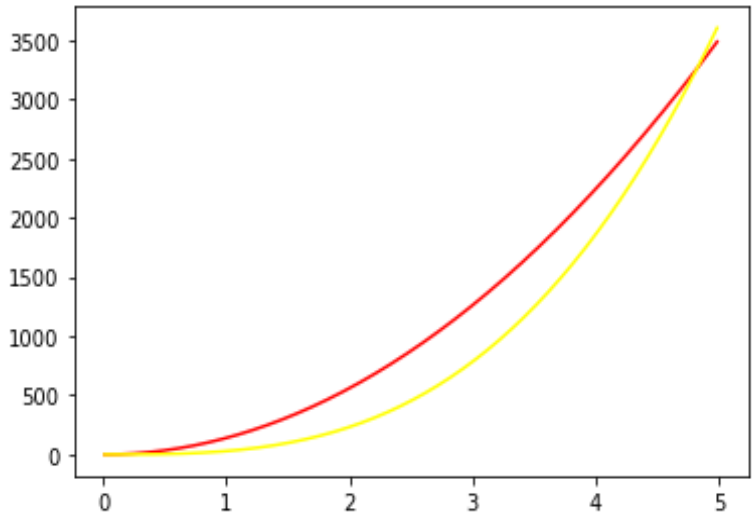
2. $A = 140n^2$ et $B = 29n^3$. Pour que A soit meilleur que B il suffit que
$A \geq B$
$140n^2 \geq 29n^3$
$no \leq 4.82757$

À l'aide de python et son module

Matplotlib, nous avons fait la représentation graphique de nos deux fonctions :

```
In [16]: x = [x/100 for x in range(1, 500)]
         y1 = list(map(lambda x: 140*x**2,x))
         y2 = list(map(lambda x: 29*x**3,x))
         plt.plot(x, y1, color="red")
         plt.plot(x, y2,color="yellow")

Out[16]: [<matplotlib.lines.Line2D at 0x1fb8532e9d0>]
```



3. A est $O(f(n))$.
   Si le temps d'exécution du pire des cas $O(f(n))$, il existe alors une constante c telle que
   $cf(n) >=$ au pire cas pour $n > no$

Puisque le pire de cas est toujours supérieur ou égal à tout autre cas
(le temps d'exécution du pire des cas $\rightarrow$ (g(n))),
$cf(n) \geq$ pire cas $\geq$ A

4. Si $d(n) = O(f(n)) \geq a * d(n) = O(f(n))$ pour $a > 0$.

Si d(n)vaut $O(f(n))$alors, il existe une constante c telle que
$d(n) \leq c\, f(n)$ pour tout $n > no$. On a alors $ad(n) \leq acf(n) = c'f(n)$.
La nouvelle constante sera donc "a" qui maintient toujours la condition originale de O qui sera Vrai

5. Si d(n) équivaut à O(f(n)) et e(n) vaut O(g(n)), alors $d(n) = cf(n)$ pour nF > nfo et $e(n) <$ dg (n) pour n_e > n_e0 en conséquence, d(n)e (n) <= (cf (n)) (dg (n)) et $n\_f * n\_en\_f0 * 0$

Ce qui signifie qu'il existe un nouveau n $n\_f * n\_e$ et et $no' = n\_f0 * n\_e0$,
et un $c' = c'd$ tel que $d(n)e(n) \leq e'(f(n)g(n))$
pour $n' >$ non', ce qui signifie que d(n)e(n) est O(f(n)g(n))
Et $n_0 * n_{f0} *$ n e0, et a $c = c'd$ tel que \nd (n)e (n) $\leq$ c' (f (n)g(n))
pour $n' >$ non', ce qui signifie que d(n) e (n) est O(f(n) g (n))

6. Comme précédemment, si d(n) vaut o(f(n)) et e(n) vaut o(g(n)), alors d(n) <=
cf (n) pour n_f > n_fo et e(n) <= dg (n) pour n_e > n_e0

Cela signifie que d(n) + e(n) <= (cf (n)) + (dg(n)) et
n > n_f0 + n_e0
ce qui signifie qu'il existe un nouveau n' = n_f + n_e et et no = n_f0 + n_e0, tel que :
d(n)e(n) ≤ cf (n) + dg (n)
pour n > non' ; cependant, cela ne satisfait toujours pas la notation O.
On peut absorber c et d dans leurs fonctions telles que d(n) e(n) <= f(cn) + g(dn)
Pour absorber c, on note que n' > no/ c cn', donc n = n/ c, ce qui signifie que n'/ c
de même pour d, n'/ cd >= no/ cd
Il existe donc de nouvelles valeurs de no telles que
d(n) + e(n) ≤ $\big((n) + d(n)\big)$pour n ≥ non, qui satisfait O$\big(1(n) + d(n)\big)$conditions

7. Le point clé ici est que ce n'est pas parce que quelque chose est O(n)
que cela doit être cette fonction

Par exemple, f(n)5 is O(n)est mathématiquement vrai,
bien que ce soit une mauvaise forme de le dire
Par conséquent, si nous avons d(n) = n et e (n) = n
avec f(n) = n et g(n) = n, alors nous vérifions d(n) <= C(f (n)) pour n >= 0, et e(n)
<= C2 (g(n)) pour n >= 0
F(n) − g(n) = 0 et d(n) − e(n) = n
Il n'y a pas de valeur pour n > 0 telle que 0 >
= n, ce qui signifie que d(n) − e(n) n'est pas 0(f(n) − g(n))

8. Comme précédemment, si d(n)vaut o$\big(f(n)\big)$et e(n)vaut o$\big(g(n)\big)$, alors d(n)
≤ cf (n)pour $n_f > n_{fo}$et e(n) ≤ dg (n)pour $n_e > n_{e0}$

9. L'algorithme E est appelé par l'Algorithme D n fois
Par conséquent, le temps d'exécution le plus défavorable est O(D(n)) ∗ (0(i)), 0(n ∗ 1)
= O(n) d'après la description

10. La notation O signifie qu'il existe une constante c telle que f(n) < Cg(n)
Donc, si les algorithmes de Alphonse fonctionnent mieux que A (nlogn) et que l'algorithme
De Bob fonctionne mieux que B(n^2), nous pouvons résoudre la valeur où Anlogn
= Bn^2, dont nous savons qu'elle est vraie Quand n
= 100. Cela signifie, (A/B) = (100) / (log (100)) = 15.05)
Cela signifie que le temps d'exécution de KIMBULU sur une seule itération
Est 15 fois plus lent mais comme il effectue globalement moins d'opération,
et commence à mieux performé à des grandes valeurs de n