UNIVERSITE DE KINSHASA

FACULTE POLYTECHNIQUE

DEPARTEMENT DE GENIE ELECTRIQUE ET INFORMATIQUE

TRAVAIL PRATIQUE DU COURS D'ALGORITHMIQUE ET PROGRAMMATION

Par :
MFWAMBA NDAYA (2GEI)
SHIMBA ILUNGA (2GEI)
BADIBANGA BADIBANGA (2GEI)

Dirigé par l'assistant **MOBISA**

Année académique 2021-2022

ANALYSE ALGORITHMIQUE ET RECURSIVITE

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

Réponse.

```
import matplotlib.pyplot as plt
import numpy as np
import math

# La valeur de n est a l'intersection de ces deux fonctions
x = [x/100 for x in range(1, 500)]
y1=list(map(lambda x: 1.11*math.log(x),x))
y2 =x
plt.plot(x,y1)
plt.plot(x,y2)
plt.show()
```

On peut chercher la valeur approximative de n_0 en representant graphiquement les deux fonctions $y = x$ et $y = \frac{10}{9} \log x$. La valeur approximative correspond ici à l'intersection de ces deux courbes.

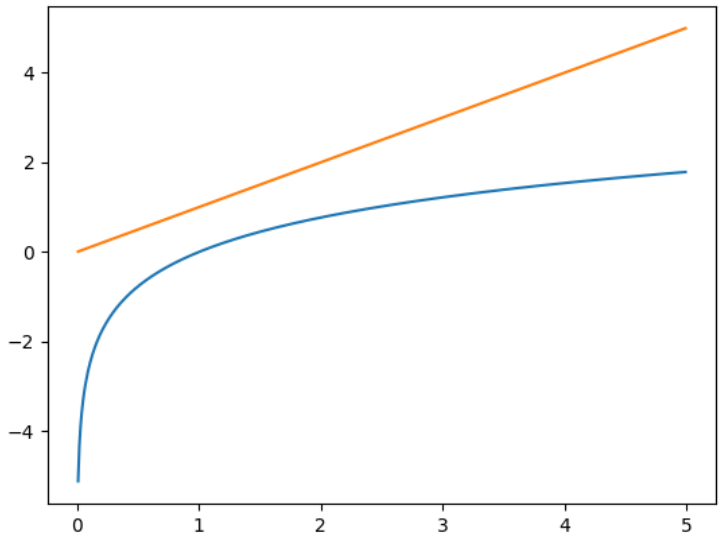

FIGURE 1 – Graphique de ces deux fonctions

Comme ces deux graphiues ne se croisent pas, on peut en conclure qu'il n'existe pas de valeurs de n_0 pour que A soit meilleur que B.

Question 2 Le nombre d'opérations primitives exécutées par les algorithmes A et B est $(140n^2)$ et $(29n^3)$, respectivement. Déterminez n0 tel que A soit meilleur que B pour tout $n \leq n_0$.

Utiliser Matlab ou Excel pour montrer les évolutions des temps d'exécution des algorithmes A et B dans un graphique.

Réponse. Nous avons : $140n^2 = 29n^3 \implies 29n = 140 \implies n_0 = \frac{140}{29}$ Le code matlab est le suivant :

```
n = 1:0.01:100;
A = 140 * n.^2;
B = 29 * n.^3 / 140;

plot(n,A,'r', n,B, 'b')
legend('A','B')
xlabel('n')
ylabel('Nombre d operations primitives')
```

Listing 1 – Implémentation matlab

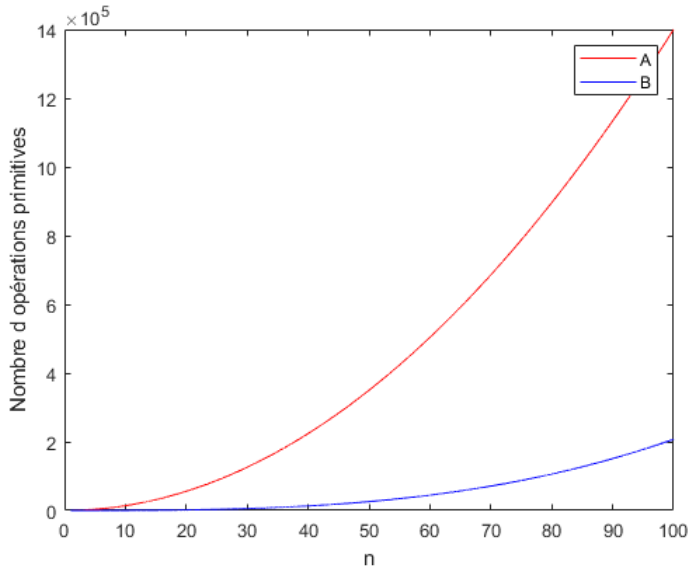

FIGURE 2 – Fonction associée au temps d'execution

Le fichier source est aussi annexé à ce présent document.

Question 3 Montrer que les deux énoncés suivants sont équivalents :

1. Le temps d'exécution de l'algorithme A est toujours $O(f(n))$.
2. Le temps d'exécution de l'algorithme A est toujours $O(f(n))$.

Réponse.

1. Si le pire cas d'éxécution est $O(f(n))$, alors il existe une constante c telle que $cf(n)$ >pire cas pour $n > n_0$.
   Puisque le pire cas est toujours >= à tous les autres cas, (pire cas d'exécution >= $g(n)$), alors $cf(n)$ >=pire cas>= tous les autres cas A Puisque $a >= b$ et $b >= c$, alors $a >= c$, ce qui signifie que c (l'algorithme A) est $O(f(n))$
2. Si le pire cas d'exécution est $O(f(n))$, alors il existe une constante c telle que $cf(n)$ >pire cas pour $n > n_0$. Puisque le pire cas est toujours >= à tous les autres cas, (pire cas d'exécution >= $g(n)$), alors $cf(n)$ >= pire cas >= tous les autres cas A. Puisque $a >= b$ et $b >= c$, alors $a >= c$, ce qui signifie que c (l'algorithme A) est$O(f(n))$.

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

Réponse.

Si $d(n)$ vaut$O(f(n))$, il existe une constante telle que $d(n) <= cf(n)$ pour tout $n > n_0$. Si on multiplie $d(n)$ par a, on obtient $(ad(n)) <= acf(n) = c'f(n)$ Essentiellement, la nouvelle constante sera $C\ old * a$, qui maintient toujours la condition d'origine de la notation O, qui sera vraie lorsque $n > a * n_0$.

Question 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 $O(f(n)g(n))$.

Réponse.

Si $d(n)$ est $O(f(n))$ et $e(n)$ est $O(g(n))$, alors $d(n) <= cf(n)$.
Pour $n_f > n_{f0}$ and $e(n) <= dg(n)$. Comme résultat, $d(n)e(n) <= (cf(n)) * (dg(n))$ et $n_f * ne > n_f0 * n_{e0}$.
Ce qui signifie qu'il y avait un nouveau $n' = n_f * n_e$ et $n'_0 = n_f0 * n_{e0}$, et $c' = c * d$ tel que $d(n)e(n) <= c'(f(n)g(n))$ pour $n' < n'_0$, Ce qui signifie que $d(n)e(n)$ est $O(d(n)g(n))$.

Question 6 Montrer que si $d(n)$ vaut $O(f(n))$, alors $(a \times d(n))$ vaut $O(f(n))$, pour toute constante $a > 0$.

Réponse.

Commme précedemment, si $d(n)$ est $O(f(n))$ and $e(n)$ est $O(g(n))$, alors $d(n) <= cf(n)$ pour $n_f > n_{f0}$ and $e(n) <= dg(n)$ pour $n_e > n_{e0}$.
Cela signifie qu'il y avaint un nouveau $d(n) + e(n) <= (cf(n)) + (dg(n))$ et $n > n_f0 + n_e0$.
Qui singifie qu'il y avait un nouveau $n' = n_f + n_e$ et $n'_0 = n_f0 + n_e0$, tel que : $d(n) + e(n) <= cf(n) + dg(n) for n > n'_0$; cependant ceci ne satisfait pas la notatiàon O.
Nous pouvons absorber c et d dans leur fonctions tel que $d(n) + e(n) <= f(cn) + g(dn)$.
En absorbant c, nous notons que $n'' = cn'$, alors $n' = n''/c$, qui signifie que $n''/c >= n''_0/c$.
Analoguement pour d, $n''/cd >= n"/cd$.
Cependant, il y avait une nouvelle valeur de n_0 tel que : $d(n) + e(n) <= (f(n) + d(n))$ pour $n >= n_0$", qui satisfait les conditions $O(f(n) + d(n))$

Question 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 $O(f(n) - g(n))$.

Réponse.

Si nous avons $d(n) = n$ et $e(n) = n$ avec $f(n) = n$ et $g(n) = n$, alors nous satisfaisons $d(n) <= C(f(n))$ pour $n >= 0$, et $e(n) <= C2(g(n))$ pour $n >= 0$.
Alors $f(n) - g(n) = 0$ et $d(n) - e(n) = n$.
Il n'y a pas de valeur pour $n > 0$ tel que $0 >= 0n$, qui signifie que $d(n) - e(n)$ n'est pas $O(f(n) - g(n))$.

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

Réponse.

Si $d(n)$ est $O(f(n))$ et $f(n)$ est $O(g(n))$, alors $d(n) <= cf(n)$ pour $n_f > n_{f0}$ et $f(n) <= dg(n)$ for $n_g > n_{g0}$.
Si cela est vraie, alors $d(n) <= cf(n) <= c(d(g(n))) = cd * g(n) = c'g(n)$ par substitution, c'est vrai pour $n_f * n_g > n_f0 * n_{g0}$, or $n > n'_0$.
C'est ainsi cela satisfait les conditions de $d(n)$ est $O(g(n))$.

Question 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 ?

Réponse.
L'algorithme **D** appelle l'algorithme **E** n fois.
Cependant, le pire cas de temps d'exécution est $O(D(n)) * (O(i))$, qui est traduit par $O(n*1) = O(n)$ depuis la description.

Question 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.

Réponse.
La notation O signifie qu'il y a une constante C tel que $f(n) < Cg(n)$. Cependant si l'algorithme de Alphonse est perfomant que $A(n \log n)$ et l'algorithme de Bob est plus perfomant que $B(n^2)$.
Nous pouvons résoudre pour la valeur de n où $An \log n = Bn^2$, que nous connaissons vrai lorsque $n = 100$. Ce qui signifie que $(A/B) = (100)/(log(100)) = 15.05$ (voir en dessous)
Cela signifie que le temps d'exécution de l'algorithme de Alphonse sur une seule itératon est 15 fois moins, mais il effectue moins d'opérations sur l'ensemble, il commence à mieux effectuer pour des grandes valeurs de n.
On peut visualiser sur cette petite implémentation

```
import math
print(100/math.log(100, 2))
15.05149978319906

```

Question 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.

Réponse.

```
import time
import random

def find_max(seq):
 """On cree la fonction pour prendre en entree une sequence d'entiers
   pour retourner l'element max de la sequence en utilisant une approche
  recursif"""
    if len(seq)==1
        return seq[0]
    else:
        max=find_max(seq[1:1])
         if seq[0]>max:
           return seq[0]
         else:
           return max

 #On mesure ensuite le temps d'execution de cet algorithme en creant une
  sequence d'entiers aleatoires et mesurer le temps necessaire pour
  trouver l'element maximal.
```

```
seq=[random.randint(0,100000) for i in range(10000)]
start=time.time()
max=find_max(seq)
end=time.time()
print("Max element :",max)
print("Time:",end-start)
```

On se sert ensuite de Matlab pour obtenir la fonction associée au temps d'exécution.

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

Réponse.

```
def min_max(S, index=0):
  """Fonction creer pour trouver le min et le max"""
  if index == len(S)-1:
    return S[index], S[index] #The current min and max values
  else:
    min_c, max_c = min_max(S, index+1)
    return min(S[index], min_c), max(S[index], max_c)


print(min_max([1,2,3,4,5,6,7,8]))
print(min_max([-45,2,774,5,6,7,8]))
print(min_max([8,7,6,5,4,3,2,1]))
```

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

Réponse.

```
VOWELLS = {'a','e','i','o','u'}
def c_or_v(S, index = 0):
  a = -1 if S[index] in VOWELLS else 1
  if index == len(S)-1:
    return a
  else:
  return (a+ c_or_v(S, index +1))

def check_vowells(S):
  ans = c_or_v(S)
  if ans>0:
    return (f'il y a plus de consonnes par {ans}')
  elif ans<0:
    return (f'il y a plus de voyelles par {ans}')
  else:
    return (f'il y a une egalite des nombres pour chacun {ans}')


strings_list = ['racecar', 'gohangasalamiimalasagnahog', 'notapalindrome'
 ,
'bob', 'a', 'hat', ]
for s in strings_list:
  print (check_vowells(s), s)
```

Résultat :

```
il y a plus de consonnes par 1 racecar
il y a plus de consonnes par2 gohangasalamiimalasagnahog
il y a plus de consonnes par 2 notapalindrome
il y a plus de consonnes par 1 bob
il y a plus de voyelles par -1 a
il y a plus de consonnes par 1 hat
```