

UNIVERSITE DE KINSHASA

FACULTÉ POLYTECHNIQUE



Algorithme et Programmation

Date : 22/02/2022

Par :

KAMBALE MARUBA EXAUCE (2GEI)

MURHABAZI MUTUZI CEDRIC (2GC)

TWITE ILUNGA SYLVANO (2GEI)



1. L'algorithme A a comme notation ($50 \log(n)$) et l'algorithme B ($45n^2$), Dire que A est meilleur que B, signifie que son temps d'exécution est inférieur à celui de B, ceci veut dire :

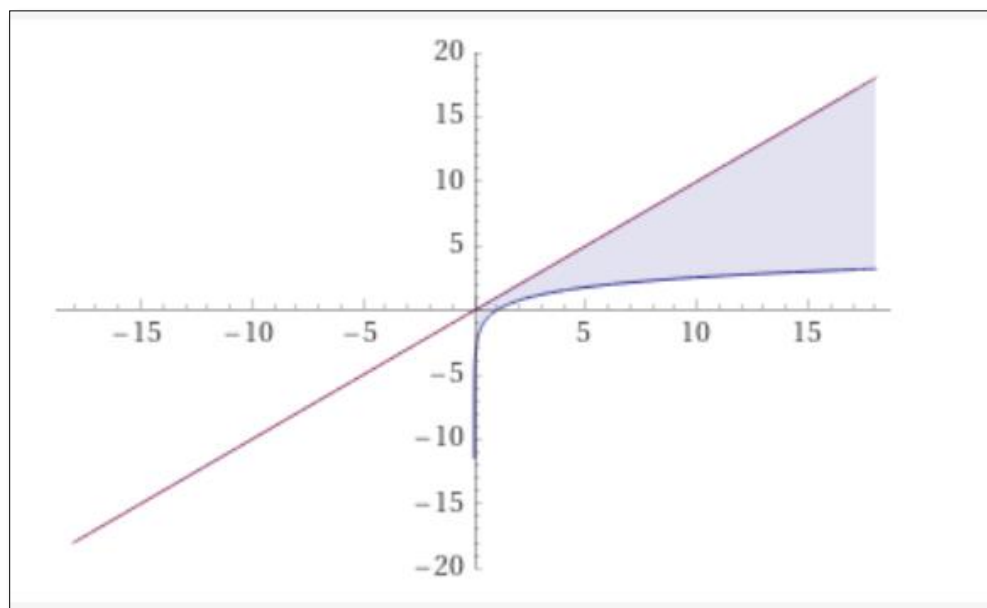
$$50n \log(n) < 45n^2 \text{ ie } \frac{50}{45} \log(n) < n \text{ ssi } \frac{10}{9} \log(n) < n$$

D'où cette expression nous génère la solution $\frac{10 \log(n)}{9} < n$ et le graphique suivant sur Python, ne prenant en compte que la zone hachurée.

```

2
3 import numpy as np
4 import matplotlib.pyplot as plt
5 from math import *
6
7 x = np.linspace(-12,12)
8 y = (50/45)*np.log(x)
9
10 plt.plot(x,y)
11 plt.plot(x,x)
12 plt.title("Question 1")
13
14

```

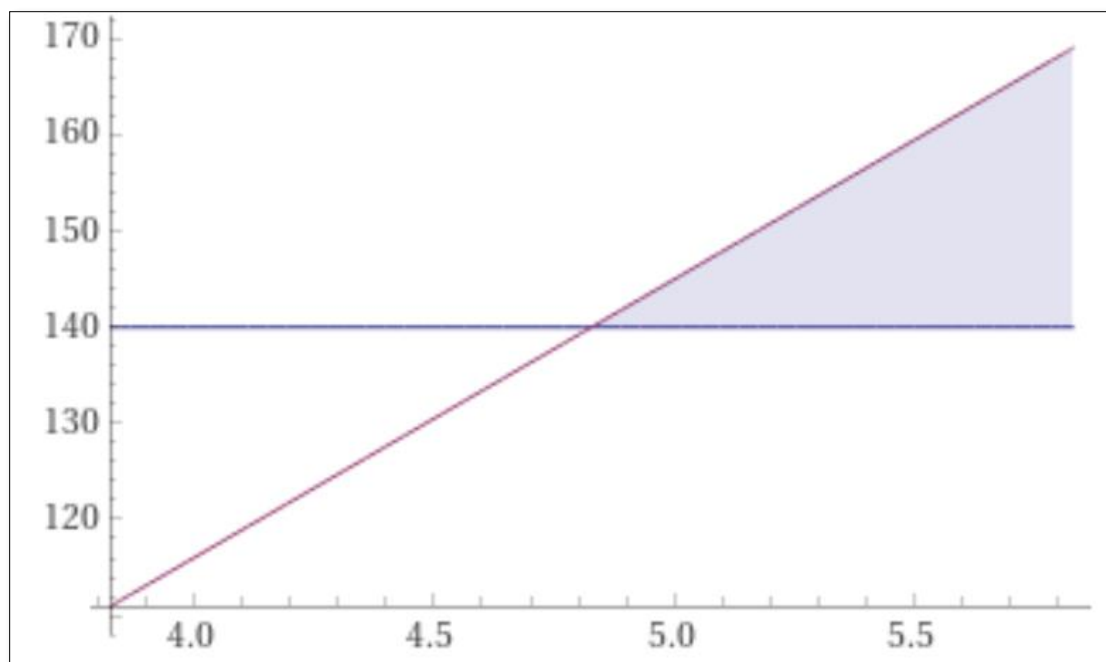


2. L'algorithme A a pour notation $140n^2$, et B : $29n^3$

En gardant la même logique, on dira que A est meilleur que B si son nombre d'opérations primitives exécutées est inférieur à celui de B, C'est à dire $140n^2 < 29n^3$,

On trouve que $140 < 29n$ en divisant les deux membres par n^2

D'où $n > \frac{140}{29}$



3. Montrons 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 $O(f(n))$.

On sait que si le temps d'exécution du cas favorable ou défavorable est $O(f(n))$, alors il devra exister une constante k telle que $k \cdot f(n)$ est supérieur au cas pire pour $n > n_0$, car théoriquement et pratiquement dans le pire des cas, on observe un temps d'exécution supérieur.

On peut trivialement déduire que le pire des cas de l'algorithme A est aussi $O(f(n))$, ceci car la forme asymptotique ne sera juste qu'un facteur multiplicatif de k du cas le plus favorable.

Comme $a \geq b$ et $b \geq k$, alors $a \geq k$, ce qui signifie que k est $O(f(n))$

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

Si $d(n)$ a comme expression asymptotique $O(f(n))$, alors il existe une constante k tel que $d(n) \leq k \cdot f(n)$ pour tout $n > n_0$.

En multipliant $d(n)$ par a , on obtient $(a \cdot d(n)) \leq a \cdot k \cdot O(f(n)) = k' \cdot O(f(n))$ On déduira aussi trivialement, qu'on a obtenue une nouvelle constante, **ce qui sera essentiellement $k \cdot a$ qui maintiendra la condition originale de notation O qui sera vraie quand $n > a \cdot n_0$**

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

En haut on a démontré que si $d(n)$ a comme expression de temps d'exécution $O(f(n))$ et $e(n) O(g(n))$, alors $d(n) \leq k*f(n)$ pour $n_f > n_{f0}$ et $e(n) \leq l*g(n)$ pour $n_e > n_{e0}$

Ceci veut dire que , $d(n)*e(n) \leq (c*f(n))*(d*g(n))$ et $n_f*n_e > n_{f0}*n_{e0}$

ce qui signifie en d'autres termes qu'il existe une nouvelle constante $n' = n_f*n_e$ et $n_{0'} = n_{f0}*n_{e0}$, et un $k' = k*l$ tel que $d(n)*e(n) \leq k'(f(n)*g(n))$ pour $n > n_{0'}$,

On conclut aussi que $d(n)*e(n)$ est $O(f(n)*g(n))$

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

On y vas par raisonnement mathématique, si $d(n)$ a comme expression asymptotique $O(f(n))$ et $e(n) O(g(n))$, alors $d(n) \leq k*f(n)$ pour $n_f > n_{f0}$ et $e(n) \leq h*g(n)$ pour $n_e > n_{e0}$
Cela signifie que $d(n) + e(n) \leq (k*f(n)) + (h*g(n))$ et $n > n_{f0}+n_{e0}$

En se référant du même raisonnement que précédemment, on peut dire qu'il existe un nouveau $n' = n_f+n_e$ et $n_{0'} = n_{f0}+n_{e0}$, tel que $d(n) + e(n) \leq k*f(n) + h*g(n)$ pour $n > n_{0'}$;

En effet, on peut continuer en disant que $d(n) + e(n) \leq f(k*n)+ g(h*n)$, ce qui va nous conduire à $d(n) + e(n) \leq O(f(n)+ g(n))$

7. Montrons que si $d(n)$ est $O(f(n))$ et $e(n)$ est $O(g(n))$, alors $d(n) - e(n)$ n'est pas nécessairement $O(f(n) - g(n))$.

On se contente à dire que si $d(n)$ a pour notation $O(f(n))$, et $e(n)$ notation $O(g(n))$,
Avec comme exemple $d(n) = n$ et $e(n) = n$ avec $f(n) = n$ et $g(n) = n$,

Alors on peut encore dire $d(n) \leq k \cdot f(n)$ pour $n \geq 0$, et $e(n) \leq 2 \cdot k \cdot g(n)$ pour $n \geq 0$
 $f(n) - g(n) = 0$ et $d(n) - e(n) = n - n$, pas de valeur pour $n > 0$ telle que $0 \geq n$, **ce qui signifie que $d(n) - e(n)$ n'est pas $O(f(n) - g(n))$**

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

Si $d(n)$ vaut $O(f(n))$ et $f(n)$ vaut $O(g(n))$, alors $d(n) \leq k \cdot f(n)$ pour $n_f > n_{f0}$ et $f(n) \leq h \cdot g(n)$ pour $n_g > n_{g0}$

Si c'est vrai, alors $d(n) \leq k \cdot f(n) \leq k \cdot (h \cdot g(n)) = k \cdot d \cdot g(n) = k' \cdot g(n)$ par substitution,

ce qui est vrai pour $n_f \cdot n_g > n_{f0} \cdot n_{g0}$, ou $n > n_0$

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 ?

Il s'agit ici d'un algorithme constitué de deux blocs, D et E parcourant une séquence S de n éléments. Le temps d'exécution de E est $O(i)$, il correspondra au temps d'exécution $O(n)$ de l'algorithme E , car n est la taille de notre séquence,

Nous pouvons donc conclure trivialement, que le pire temps d'exécution de D est $O(n^2)$, car la séquence a n éléments sera appelé par l'algorithme D qui appellera à son tour l'algorithme E n fois de suite.

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.

D'après nos résultats en haut, il existe C telle que $f(n) < C \cdot g(n)$

Par conséquent, si l'algorithme de Alphonse $A(n \log n)$ fonctionne mieux que celui de Bob $B(n^2)$,

On peut s'amuser à résoudre l'expression $n \log n = n^2$ Le rapport entre $n \log n / n^2$ donne $100 / 100 \log(100) = 15.5$ s

Ce qui signifie que l'algorithme d'alphonse est 15 fois lents sur une itération, mais ceci puisqu'il effectue moins d'opérations.

Plus l'algorithme d'Alphonse exécute plus d'opérations, plus il est meilleur que celui de BOB.

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.

Solution :

L'algorithme pour trouver l'élément maximum peut bien évidemment être rendu récursif. Dans cette tâche, l'algorithme doit examiner chaque élément de la liste, il n'y a donc qu'un seul cas de base : lorsque la fin de la liste est atteinte.

Une entrée doit être ajoutée pour garder une trace de l'élément maximum entre les appels récursifs.

Le cas récursif est presque identique à la recherche linéaire, il déplace simplement le processus vers l'élément suivant de la liste.

<pre> 2 """ 3 Created on Wed Feb 22 13:21:52 2023 4 5 @author: marex 6 """ 7 8 9 def maxRecurisif(sequence, i, j): 10 if i == len(sequence): 11 print("Le max est : " + str(j)) 12 return j 13 else: 14 if i == 0: 15 j = sequence[0] 16 elif j < sequence[i]: 17 j = sequence[i] 18 print("Appel fonction recursive= " + str((i +1))) 19 return maxRecurisif(sequence, i+1, j) 20 21 22 sequeMarex = [78,0,-5,98,64,12,765,-555,99,1008,7,86,75] 23 24 print("\n\n",sequeMarex) 25 maxRecurisif(sequeMarex,0,0) </pre>	<pre> [78, 0, -5, 98, 64, 12, 765, -555, 99, 1008, 7, 86, 75] Appel fonction recursive= 1 Appel fonction recursive= 2 Appel fonction recursive= 3 Appel fonction recursive= 4 Appel fonction recursive= 5 Appel fonction recursive= 6 Appel fonction recursive= 7 Appel fonction recursive= 8 Appel fonction recursive= 9 Appel fonction recursive= 10 Appel fonction recursive= 11 Appel fonction recursive= 12 Appel fonction recursive= 13 Le max est : 1008 </pre>
---	--

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

Solution :

Ici, on se réfère à l'exercice précédent où on avait déjà traité le cas de la recherche du maximum en utilisant l'approche direct d'implémentation du code avec Python.

Attaquons donc l'autre point, celui de rechercher le minimum.

```

"""
Created on Wed Feb 22 13:30:02 2023

@author: marex
"""

def minRecuratif(sequence, elt):
    if (elt == 1):
        return sequence[0]
    return min(A[elt - 1], minRecuratif(sequence, elt - 1))

A = [60, 4, 45, 6, -50, 10, 2, 54, 98, 76, 64, 422, 876, 998]
n = len(A)
print("\n\n\n")
print(minRecuratif(A, n))

```

-50

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

```

"""
Created on Wed Feb 22 13:30:02 2023

@author: marex
"""

def voyelle(ch):
    return ch.upper() in ['A', 'E', 'I', 'O', 'U']

def nbVoyelles(str, n):
    if (n == 1):
        return voyelle(str[n - 1]);

    return (nbVoyelles(str, n - 1) +
            voyelle(str[n - 1]));

nom = "Maruba Exauce";
voy = nbVoyelles(nom, len(nom))
print("\n\n\n")
print("Nombre Voyelles ", voy, "\n Consonnes ", len(nom) - voy)

```

Nombre Voyelles 7
Consonnes 6