Kinshasa, le 22/02/2023

PAR :

AHOSAKADI KWALAKANA (2GC)
MULAMBA LUBANZA (2GEI)
TSHISENSE NGANDU (2GC)

DEVOIR D'ALGORITHME ET PROGRAMMATION

SOLUTIONS

1. La situation présente un algorithme A qui possède un nombre d'opérations primitives exécutées (50 n log n) et l'algorithme B (45 n^2),

De ce qui précède, dire que A est meilleur que B, montre que son temps d'exécution est inférieur à celui de B, c'est-à-dire théoriquement :

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

Nous pouvons pour se faire, traçons cette courbe sur Python pour approximer la valeur de **n** vérifiant cette expression, on trouve pour notre cas une zone hachurée ou cela est vérifiée.

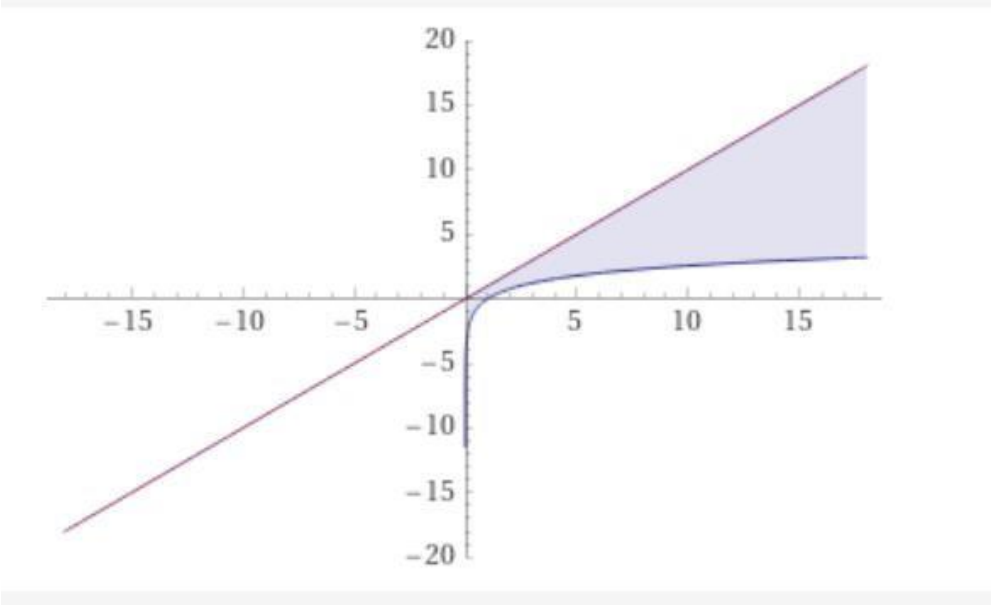
```python
# -*- coding: utf-8 -*-
"""
Created on Wed Feb 22 21:28:53 2023

@author: Dominique TSHISENSE
"""

import numpy as np
import matplotlib.pyplot as plt
from math import *

s = np.linspace(-15,15,100000)
t = (50/40)*np.log(s)

plt.plot(s,t)
plt.plot(s,s)
plt.title("graphique -01-")
```



2. Nous avons deux algorithmes A et B, ayant respectivement le nombre d'opérations primitives exécutées $140n^2$, et B : $29n^3$

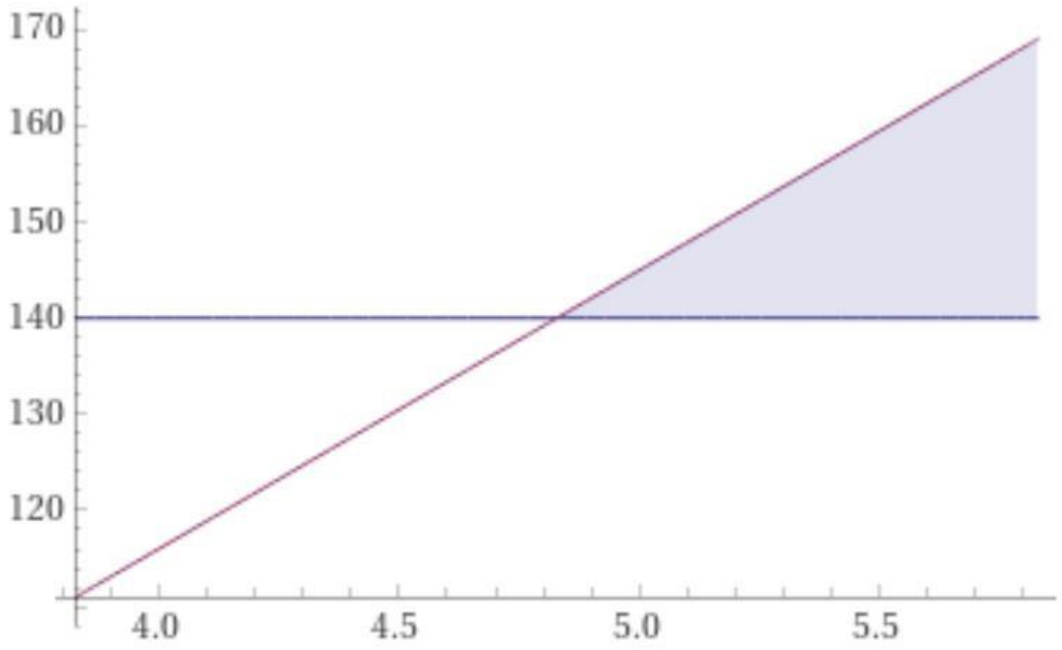
Partant de résultat de la question précédente, nous pouvons dire que A est meilleur que B si son nombre d'opérations primitives exécutées est inférieur à celui de B,

Cela sous-entend que $140n^2 < 29n^3$

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

D'où $n > \frac{140}{29}$

Le graphique ci-dessous nous fournie une idée concrète sur la question, en ce qui concerne la comparaison de ces deux algorithmes relativement à leur temps d'exécution.



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*f(n) est supérieur au cas pire pour n >no, car théoriquement et pratiquement dans le pire de cas, on observe un temps d'exécution supérieur.
On peut trivialement déduire que le pire de 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 >=b et b >=k, alors a >=k, ce qui signifie que k est O(f(n))

4. Montrons que si d(n) vaut O(f(n)), alors (a x 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) <= k*f(n) pour tout n>n0.

En multipliant d(n) par a, on obtient (a*d(n)) <= a*k*O(f(n)) = k' *O(f(n))
On déduira aussi trivialement, qu'on a obtenu une nouvelle constante, ce qui sera essentiellement k * a qui maintiendra la condition originale de notation O qui sera vraie quand n>a*n0

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)<= k*f(n) pour n_f>n_f0 et e(n)<= l*g(n) pour n_e>n_e0

Ceci veut dire que, d(n)*e(n) <= (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 et n0' = n_f0*n_e0, et un k' = k*l tel que d(n)*e(n) <= k'(f(n)*g(n)) pour n'>n0', 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 va par raisonnement mathématique, si d(n) a comme expression asymptotique
O(f(n)) et e(n) O(g(n)), alors d(n)<= k*f(n) pour n_f>n_f0 et e(n)<= h*g(n) pour n_e>n_e0

Cela signifie que d(n) + e(n) <= (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 et n0' = n_f0+n_e0, tel que d(n) + e(n) <= k*f(n) + h*g(n) pour n>n0' ;

En effet, on peut continuer en disant que d(n) + e(n) <= f(k*n) + g(h*n), ce qui va nous conduire à d(n) + e(n) <= 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) <= k*(f(n)) pour n>=0, et e(n) <= 2*k*(g(n)) pour n>=0

f(n)- g(n) = 0 et d(n) - e(n) = n-n , pas de valeur pour n>0 telle que
0>=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)<= k*f(n) pour n_f>n_f0

  et f(n)<= h*g(n) pour n_g>n_g0

- Si c'est vrai, alors d(n)<=k*f(n)<=k*(d(g(n))) = k*d*g(n) = k'*g(n) par substitution, ce qui est
- Vrai pour n_f*n_g>n_f0*n_g0, ou n>n0'

- 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 ?

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 $f(n) < C*g(n)$
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.

L'algorithme pour trouver l'élément maximum peut bien évidement 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.

```
Created on Wed Feb 22 21:59:07 2023

@author: Dominique TSHISENSE
"""

def maxRecursif(sequence, j, k):

    if j == len(sequence):

        print("Le max est :" + str(k))

        return k

    else:
        if j == 0:

            k = sequence[0]

        elif k < sequence[j]:

            k = sequence[j]

        print("Appel fonction recursive=" + str((j +1)))

        return maxRecursif(sequence, j+1, k)

sequeDom = [78,0,-5,98,64,12,765,-555,99,1008,7,86,75]

print("\n\n" , sequeDom)

maxRecursif(sequeDom,0,0)
```

```
.continuum                    04/02/2023 10:10

Help  Variable Exp

Console 1/A ×

In [18]: runfile('C:/Users/Dominique TSHISENSE/untitled1.py',

 [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

In [19]:
```

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