Programmation récursive exercices

Nom :	Note : / 20
	Classe :

1 Conversion décimal binaire

Étudions pour commencer le principe d'une conversion décimale vers binaire. Celle-ci se fonde sur une propriété très simple la parité.

1.1 Analyse de l'algorithme :

Q1. Soit N le nombre à convertir et Nb sa représentation en base 2. Donner la valeur du bit de poids faible b_0 de Nb si :
- N est pair
- N est impair

Q2. Quelle opération mathématique simple va nous permettre de déterminer la parité de N ?

Q3. A partir de l'exemple donné ci-dessous calculer le résultat de la conversion de N=69 faire le tableau.

10	%	2	0
5	%	2	1
2	%	2	0
1	%	2	1

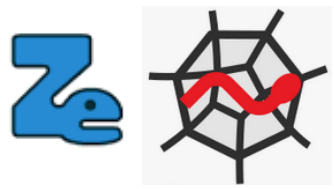
Le résultat 10_2 = 1 0 1 0

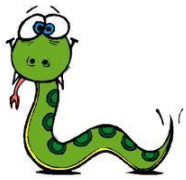
Q4. Quelle opération permet de passer d'une ligne du tableau à la ligne suivante ?

10	%	2	0
5	%	2	1
2	%	2	0
1	%	2	1

Q5. Les opérations successives s'arrêtent quel est le critère d'arrêt ?

A partir de l'étude précédente nous comprenons mieux comment l'algorithme récursif peut être construit.






Q6. Décrire 'à la main' le principe de cet algorithme tel qu'il apparaît dans la réflexion précédente. On précisera le cas de base et le cas récursif.

Q7. Donner la définition récursive, voir le format ci-dessous, de la fonction conversion_decimal_binaire. On ajoutera au raisonnement précédent le cas où N = 0

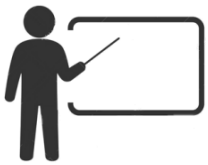
$$conversion(N) = \begin{cases} \\ \\ \end{cases}$$

Q8. Apporter la preuve de la terminaison de votre algorithme.

1.2 Mise en œuvre

Script_récursif_5. Coder votre algorithme à partir du script

Script_recursif_5_depart_eleve.py



2 Suite de Syracuse

2.1 Présentation

En mathématiques, on appelle suite de Syracuse une suite d'entiers naturels définie de la manière suivante : on part d'un nombre entier plus grand que zéro ; s'il est pair, on le divise par 2 ; s'il est impair, on le multiplie par 3 et on ajoute 1. En répétant l'opération, on obtient une suite d'entiers positifs dont chacun ne dépend que de son prédécesseur[1].

Q9. Donner une définition récursive de la suite qui est définie de la manière suivante : u_0 = N puis $u_{n+1} = u_n/2$ si u_n est pair et $u_{n+1} = 3 \cdot u_n + 1$ si un est impair.

$$u_0 = N \quad \text{pour } N > 0$$

$$u_{n+1} = \begin{cases} \\ \\ \end{cases}$$

La conjecture de Syracuse est l'hypothèse mathématique selon laquelle la suite de Syracuse de n'importe quel entier strictement positif atteint 1. Elle n'est pas démontrée ni infirmée à ce jour … avis aux amateurs !

[1] Source https://fr.wikipedia.org/wiki/Conjecture_de_Syracuse

Q10. Calculez les valeurs successives de la suite avec N = 15. Remplir le tableau ci-dessous :

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}
15										

u_{11}	u_{12}	u_{13}	u_{14}	u_{15}	u_{16}	u_{17}	u_{18}	u_{19}	u_{20}	u_{21}

Q11. Une fois atteint la valeur 1 pour la première fois que se passe-t-il ?

2.2 Mise en œuvre

Script_récursif_6. Coder la suite de Syracuse à partir du script : Script_recursif_6_depart_eleve.py

2.3 Amélioration

Ajoutons au script précédent le tracé des résultats avec la bibliothèque Matplotlib.

- Il faut faire l'appel des bibliothèques :

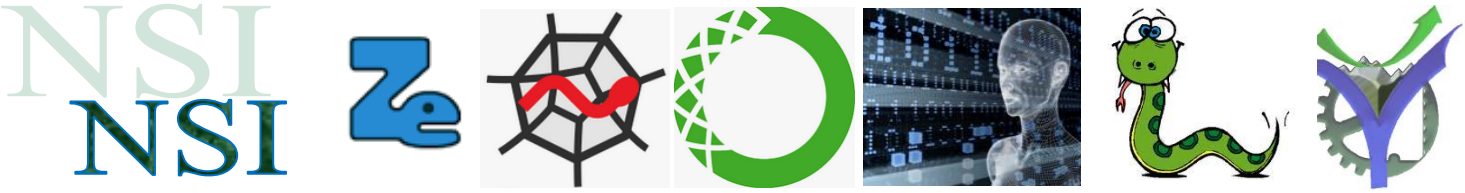
```
import matplotlib
import matplotlib.pyplot as plt
```

- Puis traiter l'affichage des valeurs avec un peu de mise en forme :

```
# Tracé des valeurs de la suite avec matplotlib
# Préparation des abscisses et ordonnées
abscisse = [ x for x in range(len(suite))]
ordonnee = [suite[x] for x in range(len(suite))]

# Préparation du graphique
titre = 'Représentation de la suite de Syracuse pour Uo=' + str(n)
plt.style.use('seaborn-whitegrid')
plt.title(titre)
texte_abscisse = "La première valeur à 1 apparaît pour n =" + str(len(suite)-1)
plt.xlabel(texte_abscisse)
plt.plot(abscisse,ordonnee)

# Tracé du graphique
plt.show()
```



3 Coefficient binomial, dénombrement, récursivité

3.1 Définition[2]

En mathématiques, les coefficients binomiaux, définis pour tout entier naturel n et tout entier naturel k inférieur ou égal à n, donnent le nombre de parties de k éléments, avec 0 ≤ k ≤ n, dans un ensemble de n éléments.

On les note : $\binom{n}{k}$ ou C_n^k (lu combinaison de k parmi n). $\binom{n}{k} = C_n^k = \frac{n!}{k!(n-k)!}$

3.2 Rappels

Nombre de permutations

Il existe n! permutations d'un ensemble E de n éléments.

Arrangements

Lorsque nous choisissons k objets parmi n, avec k ≤ n, objets et que l'ordre dans lequel les objets sont sélectionnés revêt une importance, nous pouvons les représenter par un k-uplet d'éléments distincts et on en constitue une liste ordonnée sans répétition possible, c'est-à-dire dans laquelle l'ordre des éléments est pris en compte (si l'on permute deux éléments de la liste, on a une liste différente, et un élément ne peut être présent qu'une seule fois). Une telle liste ordonnée est appelée un arrangement.

Le nombre d'arrangements que l'on peut faire est noté : A_n^k avec $A_n^k = \frac{n!}{(n-k)!}$

3.3 Exercices d'application

Q12. Combien y a-t-il d'anagramme du mot PATRICE ?

Q13. Combien peut-on former de numéro de téléphone à huit chiffres ?

Q14. Combien peut-on former de numéro de téléphone à huit chiffres ne contenant pas le chiffre 0?

Q15. Combien y a-t-il de permutations différentes du mot ERREUR, il faudra tenir compte du fait que la lettre R est présente 3 fois et la lettre E est présente 2 fois.

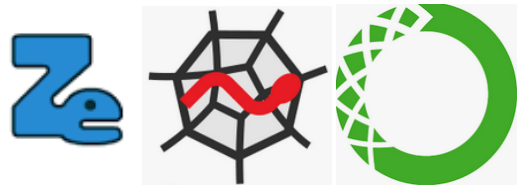
Q16. Un clavier de porte d'immeuble possède 9 touches. Chaque code est composé d'une lettre suivie de trois chiffres distincts ou non.

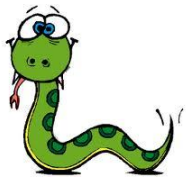
a) Combien de codes différents peut-on former ?

b) Combien y a-t-il de codes comportant des chiffres distincts ?

1	2	3
4	5	6
A	B	C

[2] https://fr.wikipedia.org/wiki/Coefficient_binomial






Q17. Un groupe de 3 élèves doit aller chercher des livres au CDI. Il y a 24 élèves dans la classe. Combien peut-on former de groupes ?

3.4 Calcul récursif du coefficient binomial

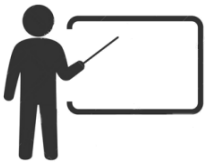
La formule de Pascal lie de manière récursive les coefficients binomiaux :

$$C_n^p = \begin{cases} 1 \text{ si } p = 0 \text{ ou } p = n \\ C_{n-1}^{p} + C_{n-1}^{p-1} \end{cases}$$

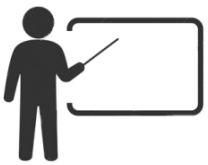
Q18. Écrire le code de la fonction récursive en Python calculCombinaison(n, p)

Script_récursif_7. Coder votre fonction calculCombinaison à partir du script : Script_recursif_7_depart_eleve.py

Observez la rapidité de votre programme. Comment peut-on l'améliorer ?



Script_récursif_8. Améliorer la rapidité de votre script précédent.



4 Récursivité et complexité

4.1 Calcul de la complexité de la fonction factorielle

La complexité C(n) de la fonction factorielle peut s'écrire de manière récursive comme ci-contre :

La complexité d'un algorithme récursif se fait par la résolution d'une équation de récurrence en éliminant la récurrence par substitution de proche en proche.

$$C(n) = \begin{cases} a & \text{si } n = 1, \\ b + C(n-1) & \text{sinon} \end{cases}$$

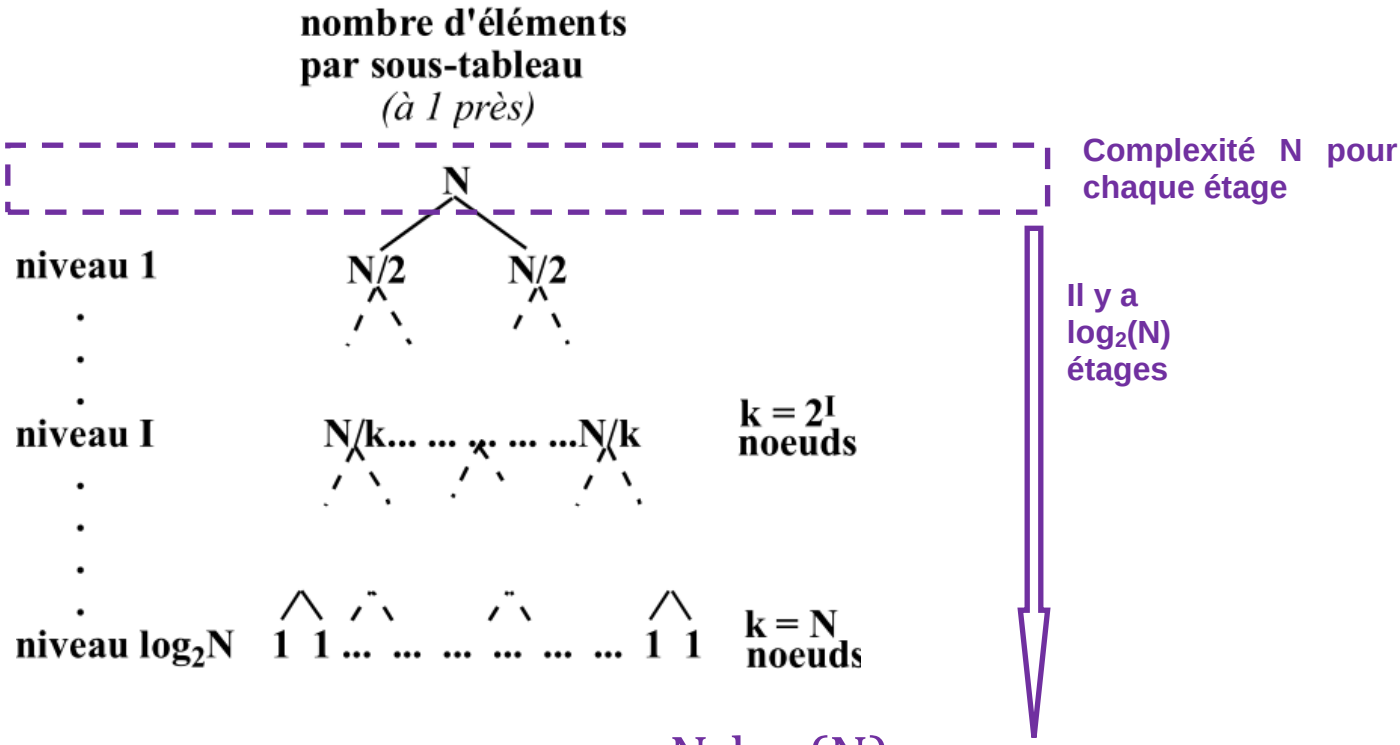
Q19. En utilisant ce procédé montrer que la complexité est égale à : C(n) = (n - 1) b + a

Q20. Conclure sur la complexité de la fonction factorielle exprimée en notation de Landau O(n).

4.2 Algorithme diviser pour régner et complexité[3] principe

Reprenons l'algorithme du tri-fusion. Nous avons une liste de n nombres à trier. Le problème de complexité N au départ se décompose en une succession de problèmes de complexité 2•N/2 puis 4• N/4 etc. ….

Pour atteindre la fin du processus récursif de division d'une liste en deux listes nous avons $\log_2(n)$ étapes, souvent noté log(n) en omettant la base 2.

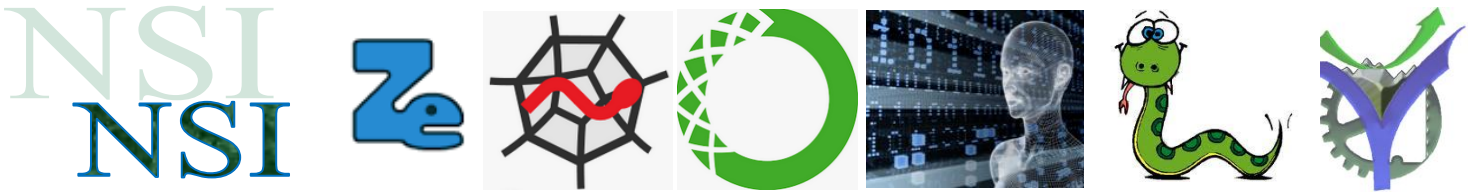


Complexité globale du tri fusion (merge sort) : $N \cdot \log(N)$

4.3 Quelques résultats usuels de complexité.

- Θ(1) : complexité constante
  (calcul d'une formule)
- Θ(log*n*) : complexité logarithmique
  (recherche dichotomique)
- Θ(n) : complexité linéaire
  (recherche séquentielle)
- Θ(*n* log*n*) : complexité subquadratique
  (tri optimal)
- Θ(n^2) : complexité quadratique
  (tri simple)
- Θ(n^k) : complexité polynomiale
- Θ(2^n) : complexité exponentielle
  (tours de Hanoï)

[3] D'après Le *Master Theorem*. Jean-Marc.*Vincent*@imag.fr. Université de Grenoble-Alpes, UFR IM2AG. L3 Informatique.



4.4 Exercices de complexité[4]

Q21. Une complexité décrite par une relation de récurrence de la forme : C(n) = C(n-1) + n avec C(1) = n est en :

☐ $\Theta(n)$ ☐ $\Theta(n^2)$

☐ $\Theta(2^n)$ ☐ $\Theta(n\log_2 n)$

Q22. Diviser pour régner est un paradigme de programmation :

☐ principalement utilisé avec un style de programmation itératif,

☐ principalement utilisé avec un style de programmation récursif,

☐ qui consiste à diviser un problème en sous problèmes dépendants les uns des autres,

☐ qui consiste à diviser un problème en sous problèmes indépendants les uns des autres.

Q23. Un algorithme diviser pour régner pourrait avoir une complexité de la forme :

☐ C(n) = C(n-1)+n

☐ C(n) = 2C(n/2) + n^2

☐ C(n) = 4C(n/3) + n

☐ C(n) = n / 2

Suite arithmétique

Définition : Une suite (u_n) est une suite arithmétique s'il existe un nombre r tel que pour tout entier n, on a : $u_{n+1} = u_n + r$.
Le nombre r est appelé raison de la suite.

Propriété : n est un entier naturel non nul alors on a : $1+2+3+...+n = \frac{n(n+1)}{2}$

Suite géométrique

Définition : Une suite (u_n) est une suite géométrique s'il existe un nombre q tel que pour tout entier n, on a : $u_{n+1} = q \times u_n$.
Le nombre q est appelé raison de la suite.

Propriété : n est un entier naturel non nul et q un réel différent de 1 alors on a :
$1+q+q^2+...+q^n = \frac{1-q^{n+1}}{1-q}$

[4] Sources diverses