

Algorithmes et Pensée Computationnelle

Consolidation 1

Les exercices de cette série sont une compilation d'exercices semblables à ceux vus lors des semaines précédentes. Le but de cette séance est de consolider les connaissances acquises lors des travaux pratiques des semaines 1 à 6.

Le code présenté dans les énoncés se trouve sur Moodle, dans le dossier **Code**.

1 Introduction et architecture des ordinateurs

Le but de cette section est de comprendre le fonctionnement d'un ordinateur. La série d'exercices sera axée autour de conversions en base binaire, décimale, hexadécimale, base 5 et de calcul de base en suivant le modèle Von Neumann.

1.1 Conversion

Question 1: (🕒 10 minutes) **Conversion**

1. Convertir le nombre $FFFFFF_{(16)}$ en base 10.
2. Convertir le nombre $4321_{(5)}$ en base 10.
3. Convertir le nombre $ABC_{(16)}$ en base 2.
4. Convertir le nombre $254_{(10)}$ en base 15.
5. Convertir le nombre $11101_{(2)}$ en base 10.

💡 Conseil

N'oubliez pas qu'en Hexadécimal, A vaut 10, B vaut 11, C vaut 12, D vaut 13, E vaut 14 et F vaut 15.

1.2 Conversion et arithmétique

Question 2: (🕒 5 minutes) **Conversion, addition et soustraction :**

Effectuez les opérations suivantes :

1. $10110101_{(2)} + 00101010_{(2)} = \dots_{(10)}$
2. $70_{(10)} - 10101010_{(2)} = \dots_{(10)}$

💡 Conseil

Convertissez dans une base commune avant d'effectuer les opérations.

1.3 Modèle de Von Neumann

Dans cette section, nous allons simuler une opération d'addition dans le **modèle de Von Neumann**, il va vous être demandé à chaque étape (FDES) de donner la valeur des registres.

État d'origine :

A l'origine, notre **Process Counter (PC)** vaut **00100001**.

Dans la mémoire, les instructions sont les suivantes :

Adresse	Valeur
00100001	00110100
00101100	10100110
01110001	111111101

Les registres sont les suivants :

Registre	Valeur
00	01111111
01	00100000
10	00101101
11	00001100

Les opérations disponibles pour l'unité de contrôle sont les suivantes :

Numéro	Valeur
00	ADD
01	XOR
10	MOV
11	SUB

Question 3: (🕒 5 minutes) Fetch

À la fin de l'opération **FETCH**, quelles sont les valeurs du **Process Counter** et de l'**Instruction Register** ?

💡 Conseil

Pour rappel, l'unité de contrôle (Control Unit) commande et contrôle le fonctionnement du système. Elle est chargée du séquençage des opérations. Après chaque opération **FETCH**, la valeur du Program Counter est incrémentée (valeur initiale + 1).

Question 4: (🕒 5 minutes) Decode

1. Quelle est la valeur de l'opération à exécuter ?
2. Quelle est l'adresse du registre dans lequel le résultat doit être enregistré ?
3. Quelle est la valeur du premier nombre de l'opération ?
4. Quelle est la valeur du deuxième nombre de l'opération ?

💡 Conseil

Pensez à décomposer la valeur de l'**Instruction Register** pour obtenir toutes les informations demandées.

Utilisez la même convention que celle présentée dans les diapositives du cours (Architecture des ordinateurs (Semaine 2) - Diapositive 15)

Les données issues de la décomposition de l'**Instruction Register** ne sont pas des valeurs brutes, mais des références. Trouvez les tables concordantes pour y récupérer les valeurs.

Question 5: (🕒 5 minutes) Execute

Quel est résultat de l'opération ?

💡 Conseil

Toutes les informations permettant d'effectuer l'opération se trouvent dans les données de l'**Instruction Register**.

2 Logiciels système

Question 6: (🕒 5 minutes) Sous Linux et MacOS, laquelle de ces commandes modifie le filesystem ?

1. `ls -la`
2. `sudo rm -rf ~/nano`
3. `sudo kill -9 3531`
4. `more nano.txt`
5. Aucune réponse n'est correcte.

💡 Conseil

Attention !

Certaines commandes listées ci-dessus peuvent avoir des conséquences irréversibles. Pour avoir une description détaillée d'une commande, vous pouvez ajouter `man` devant la commande sous Linux/MacOS ou ajouter `-h`, `--help` ou `/?` après la commande sous Windows.

3 Programmation de base

Question 7: (🕒 10 minutes)

1. Convertir $52_{(10)}$ en base 2 sur 8 bits.
2. Convertir $100_{(10)}$ en base 2 sur 8 bits.
3. Calculer en base 2 la soustraction de $01100100_{(2)}$ par $00110100_{(2)}$.
4. Déterminer au complément à deux l'opposé (multiplication par -1 en base 10) de $0110000_{(2)}$.

💡 Conseil

- Se référer aux techniques apprises dans la série 1 et la série 3
- Faire un tableau des puissances de 2 sur 8 bits.

4 Itération et récursivité

Question 8: (🕒 15 minutes) Itération et Récursivité

Créez une fonction itérative, puis une fonction récursive qui calculent le nombre de voyelles présentes dans un texte donné.

💡 Conseil

Pour la version itérative, parcourez toute la chaîne de caractère et incrémentez un compteur lorsque vous avez une voyelle.
Pour la version récursive, diminuez systématiquement la taille de votre chaîne de caractère. Si l'élément actuel est une voyelle, ajoutez 1, sinon, ajoutez 0.
Aidez vous d'une liste de toutes les voyelles et de la fonction `in` en Python (`List.contains()` en Java).

Voici les templates :

Python

```

1 def nb_voyelles_itérative(T,S) :
2     #TODO
3
4 def nb_voyelles_réursive(T,S) :
5     #TODO
6
7 voyelles = ['a','e','i','o','u','y']
8 texte = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean molestie elit ipsum, a tincidunt urna aliquet \
9 eget. Praesent et quam vitae justo hendrerit tristique. Ut malesuada ligula in mi ultricies tempor. Fusce blandit \
10 turpis sapien, in gravida orci aliquet et. Morbi in metus efficitur, volutpat purus sit amet, scelerisque massa. \
11 Vivamus vehicula justo quis leo feugiat fringilla. Maecenas sagittis ultrices accumsan. Cras libero est, gravida in \
12 eros ac, luctus ullamcorper nisi."
13
14 print(nb_voyelles_itérative(texte,voyelles))
15 print(nb_voyelles_réursive(texte,voyelles))

```

Java

```

1 import java.util.List;
2
3 public class Main {
4
5     public static int nb_voyelles_itérative(String S, List L){
6         //TODO
7     }
8
9     public static int nb_voyelles_réursive(String S, List L){
10        //TODO
11    }
12    public static void main(String[] args) {
13        List voyelles = List.of('a','e','i','o','u','y');
14        String texte = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean molestie elit ipsum, a tincidunt urna
15        aliquet" +
16        " eget. Praesent et quam vitae justo hendrerit tristique. Ut malesuada ligula in mi ultricies tempor. Fusce blandit" +
17        " turpis sapien, in gravida orci aliquet et. Morbi in metus efficitur, volutpat purus sit amet, scelerisque massa." +
18        " Vivamus vehicula justo quis leo feugiat fringilla. Maecenas sagittis ultrices accumsan. Cras libero est, gravida in"
19        +
20        " eros ac, luctus ullamcorper nisi.";
21
22        System.out.println(nb_voyelles_itérative(texte, voyelles));
23        System.out.println(nb_voyelles_réursive(texte, voyelles));
24    }
25 }

```

Question 9: (🕒 5 minutes) Lecture de code (Récursivité)

Qu'afficheront les programmes suivants ?

💡 Conseil

Ces deux programmes comportent des fonctions itératives, lisez bien le code de haut en bas et lorsque la fonction fait appel à elle-même, revenez au début de la fonction et effectuez de nouveau les instructions avec les nouveaux paramètres.
Une feuille de papier pourrait vous être utile !

Programme 1 :

```

1 def recursion_1(S) :
2     if len(S) == 1 :
3         return S[0]
4     else :
5         return S[0] + recursion_1(S[1:]) + S[0]
6
7 print(recursion_1("Python"))

```

Programme 2 :

```
1 def recursion_2(L) :
2     if len(L) == 1 :
3         print(L[0])
4     else :
5         recursion_2(L[1:])
6         print(L[0])
7         recursion_2(L[1:])
8
9 Liste = ["J","adore","Python"]
10 recursion_2(Liste)
```

5 Algorithmes et complexité

Question 10: (🕒 5 minutes) Complexité - Partie 1

Que fait le programme suivant ?

Estimez le nombre d'opérations qu'effectuera l'algorithme de la fonction **algo1** à chaque étape en fonction des paramètres qui lui seront assignés :

```
1 import math
2
3 # L est une liste d'entiers et x est un nombre entier
4 def algo1(L, x):
5     n = len(L)
6
7     for i in range(n):
8         L[i] = L[i] + math.pow(x, i)
```

💡 Conseil

Les opérations dans une boucle **for** sont répétées autant de fois que le nombre d'éléments sur lesquels nous itérons.

Question 11: (🕒 5 minutes) Complexité - Partie 2

Que fait le programme suivant ?

Estimez le nombre d'opérations qu'effectuera l'algorithme de la fonction **algo2** à chaque étape en fonction des paramètres qui lui seront assignés. Déterminez sa complexité.

```
1 # L liste de nombres entiers
2 def algo2(L):
3     n = len(L)
4
5     for i in range(n):
6         for j in range(n):
7             if i != j and L[i] == L[j]:
8                 return True
9
10    return False
```

💡 Conseil

Les opérations dans une boucle **for** sont répétées autant de fois que le nombre d'éléments sur lesquels nous itérons.

Question 12: (🕒 5 minutes) Complexité - Partie 3

Que fait le programme suivant ?

Estimez le nombre d'opérations qu'effectuera l'algorithme de la fonction **algo3** à chaque étape en fonction des paramètres qui lui seront assignés. Déterminez sa complexité.

```

1 # n un nombre entier
2 def algo3(n):
3     i = 1
4     s = 0
5     while i < n:
6         s += i
7         i *= 2
8     return s

```

💡 Conseil

Faites attention à l'évolution de la valeur de i . Celle-ci permettra de déterminer la complexité.

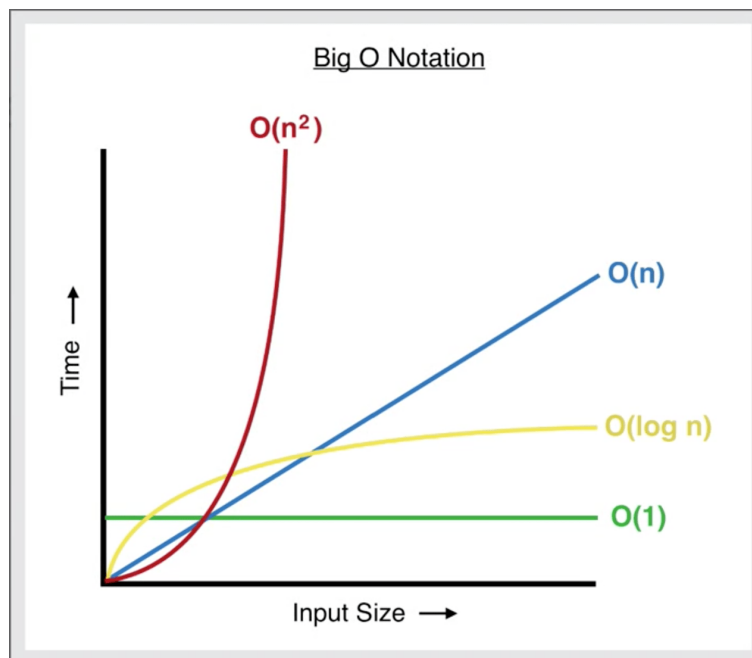


FIGURE 1 – Représentation de complexités temporelles

Question 13: (🕒 15 minutes) Tri fusion (Merge Sort)

1. Ecrire une fonction “merge” qui prend deux listes triées comme argument et retourne une liste fusionnée triée.
2. Quel est le nombre d'opérations effectuées ? Déterminer ensuite la complexité de la fonction, en posant n = longueur de la liste fusionnée.

Pour les tests utilisez les listes suivantes :

$l1 = [3, 10, 12]$ et $l2 = [5, 7, 14, 15]$.

💡 Conseil

- Cette fonction est une des deux parties de l'algorithme de tri fusion.
- Inspirez vous de la solution de l'exercice 8 (exercices basiques) de la série 5.
- N'hésitez pas à revoir le processus montré en cours (visualisation de l'algorithme dans les diapositives 84 à 108) pour comprendre comment marche concrètement le tri fusion.

6 Algorithmes de recherche

Question 14: (🕒 15 minutes) Recherche binaire

Dans cet exercice, vous devez retrouver l'élément d'une liste d'entiers triés qui est le plus proche d'un élément *e* donné. Pour ce faire, vous devez utiliser une version récursive de l'algorithme de recherche binaire.

Vous pouvez faire cet exercice aussi bien en Java qu'en Python.

>_ Exemple (Python)

```
L = [1, 11, 22, 33, 44, 55, 66, 77, 88, 99]
e = 73
```

```
print(plus_proche(L, e, recherche_binaire_recursive(L, 0, len(L)-1, e)))
```

Index retourné par la première fonction : 6
Résultat final : 77

💡 Conseil

Dans cet exercice, vous devez déclarer deux fonctions, et les combiner afin de retrouver l'élément de la liste qui est le plus proche de *e*.

La première fonction sera la fonction de recherche binaire récursive qui prendra comme paramètre la liste, l'index du premier élément de la liste, l'index du dernier élément de la liste et *e*. Cette fonction retournera l'index de l'un des éléments le plus proche de *e*. La deuxième fonction effectuera les comparaisons de différences entre *e* et les éléments se situant autour de l'élément correspondant à l'index retourné par la première fonction. Elle pourra ainsi déterminer lequel est le plus proche de *e*. Elle prendra en paramètre notre liste, *e*, et la valeur retournée par la première fonction.

Voici les templates :

Python

```
1 def recherche_binaire_recursive(L, s, r, e):
2     #TODO
3
4 def plus_proche(L,e,v):
5     #TODO
6
7 L = [1, 2, 5, 8, 12, 16, 24, 56, 58, 63]
8 s = 0
9 r = len(L)-1
10 e = 68
11 print(plus_proche(L, e, recherche_binaire_recursive(L, s, r, e)))
```

Java

```
1 import java.util.List;
2
3 public class Main {
4
5     public static int recherche_binaire_recursive(List L,int s,int r,int e){
6         //TODO
7     }
8
9     public static int plus_proche(List L,int e,int v){
10        //TODO
11    }
12
13    public static void main(String[] args) {
14        List L = List.of(1,2,5,8,12,16,24,56,58,63);
15        int s = 0;
16        int r = L.size()-1;
17        int e = 64;
```

```

18     System.out.println(plus_proche(L,e,recherche_binaire_recursive(L,s,r,e));
19 }
20 }

```

Question 15: (🕒 5 minutes) Python - Arbre : Recherche et insertion

Dans cet exercice, nous allons voir comment insérer des éléments dans un arbre binaire.

À l'aide de la fonction de recherche binaire (se référer aux exercices de la semaine 6, question 8), créez une boucle qui vérifie si chaque élément de liste se trouve dans l'arbre. Si un élément ne s'y trouve pas, il doit être inséré à l'intérieur de celui-ci.

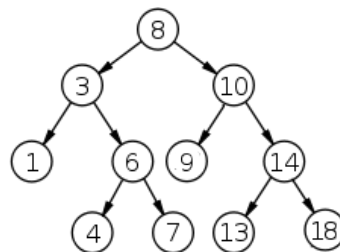
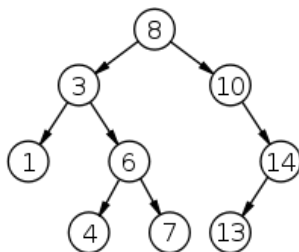
Voici l'arbre et les valeurs à rechercher :

```
tree = Arbre(8, 3, 1, 6, 4, 7, 10, 14, 13)
```

```
root = tree.root
```

```
liste = [3,6,9,18]
```

Voici la représentation graphique de l'arbre initial et de l'arbre attendu :



💡 Conseil

La fonction `insert(value, node)` permet d'insérer un élément dans un arbre. Exemple : `tree.insert(9,root)`

Téléchargez le fichier `question15.py` se trouvant dans le dossier `code` et écrivez votre code à partir de la ligne 66.

7 Quiz général

7.1 Python

Question 16: (🕒 2 minutes)

En Python, 'Hello' équivaut à "Hello".

1. - Vrai
2. - Faux

Question 17: (🕒 2 minutes)

Dans une fonction, nous pouvons utiliser les instructions `print()` ou `return`, elles ont le même rôle.

1. - Vrai
2. - Faux

Question 18: (🕒 2 minutes)

Lorsqu'on fait appel à une fonction, les arguments doivent nécessairement avoir le(s) même(s) noms tel(s) que définit dans la fonction. Exemple :


```

1 def recherche_lineaire(Liste, x):
2     for i in Liste:
3         if i == x:
4             return x in Liste
5     return -1
6
7 Liste = [1,3,5,7,9]
8 x = 3
9
10 recherche_lineaire(Liste,x)

```

1. - Vrai
2. - Faux

Question 19: (🕒 2 minutes)

Si le programme Python contient une erreur, celle-ci sera détectée avant l'exécution du programme.

1. - Vrai
2. - Faux

Question 20: (🕒 2 minutes)

Il est possible de faire appel à une fonction définie "plus bas" dans le code sans que cela ne pose problème.

```

1 import math
2
3 nombre_decimal_pi(4)
4
5 def nombre_decimal_pi(int):
6     return round(math.pi,int)

```

1. - Vrai
2. - Faux

Question 21: (🕒 5 minutes)

Les trois fonctions suivantes renvoient-elles systématiquement des résultats identiques ?

Les fonctions sont censées retourner le nombre `pi` avec le nombre de décimales (au moins une et au maximum 15) indiqué en paramètre.

<pre> 1 import math 2 3 def nombre_decimal_pi(int): 4 if int > 15: 5 int = 15 6 elif int < 0: 7 int = 1 8 resultat = round(math.pi,int) 9 return resultat 10 11 print(nombre_decimal_pi(-2)) 12 print(nombre_decimal_pi(4)) 13 print(nombre_decimal_pi(20)) </pre>	<pre> 1 import math 2 3 def nombre_decimal_pi(int): 4 if int > 15: 5 resultat = round(math.pi,15) 6 elif int < 0: 7 resultat = round(math.pi,1) 8 else: 9 resultat = round(math.pi,int) 10 return resultat 11 12 print(nombre_decimal_pi(-2)) 13 print(nombre_decimal_pi(4)) 14 print(nombre_decimal_pi(20)) </pre>	<pre> 1 import math 2 3 def nombre_decimal_pi(int): 4 if int > 15: 5 return round(math.pi,15) 6 elif int < 0: 7 return round(math.pi,1) 8 else: 9 return round(math.pi,int) 10 11 print(nombre_decimal_pi(-2)) 12 print(nombre_decimal_pi(4)) 13 print(nombre_decimal_pi(20)) </pre>
--	--	--

1. - Vrai
2. - Faux

7.2 Java

Question 22: (🕒 3 minutes)

Observez les deux programmes suivants en Java. Lequel a-t-il une bonne structure et peut être compilé sans erreur ?

```

1 //Programme A
2 public class Main {
3
4     public static void main(String[] args) {
5         ma_function();
6         autre_fonction();
7
8
9         static void ma_function(){
10             System.out.println("Voici ma fonction!");
11         }
12
13         static void autre_fonction(){
14             System.out.println("Une autre fonction!");
15         }
16     }
17 }

```

```

1 //Programme B
2 public class Main {
3
4     public static void main(String[] args) {
5         ma_function();
6         autre_fonction();
7     }
8
9     static void ma_function(){
10         System.out.println("Voici ma fonction!");
11     }
12
13     static void autre_fonction(){
14         System.out.println("Une autre fonction!");
15     }
16 }

```

1. Programme A
2. Programme B

Question 23: (🕒 2 minutes) Exercice 1

L'indentation des lignes de code en Java est aussi importante qu'en Python.

1. - Vrai
2. - Faux