

ÉPREUVE REGROUPÉE**Discipline: INFORMATIQUE****Date:** 19/12/2025**Cours: 2 INDP****Durée:** 105 minutes**Maître: L. Hofer, J.C. Barros, H. Garcia****Matériel autorisé: aucun (aide-mémoire inclus à la fin de l'épreuve)****Nom et prénom:****Total: 53 points + 4 bonus****Ex 1:** Programme récursif complexe

(7 pts.)

On considère la fonction `algo_tri`, définie ci-dessous en Python ([consulter l'aide-mémoire au besoin](#)) :

```
1 def algo_tri(liste):
2     if len(liste) <= 1:
3         return liste
4     milieu = len(liste)//2
5     gauche = algo_tri(liste[:milieu])
6     droite = algo_tri(liste[milieu:])
7     liste_triee = []
8     i = j = 0
9     while i < len(gauche) and j < len(droite):
10        if gauche[i] < droite[j]:
11            liste_triee.append(gauche[i])
12            i += 1
13        else:
14            liste_triee.append(droite[j])
15            j += 1
16    return liste_triee + gauche[i:] + droite[j:]
17
18 liste = [2, 4, 1, 10, 5, 4]
19 resultat = algo_tri(liste)
20 print(resultat)
```

Répondez aux questions suivantes :

- a. Comment s'appelle l'algorithme de tri implémenté dans la fonction `algo_tri` ? Donnez son nom et décrivez son principe. (2pt.)

Tri fusion (merge sort). Son principe repose sur la stratégie "diviser pour régner" : on divise récursivement la liste en deux sous-listes jusqu'à obtenir des listes de taille 1 (déjà triées), puis on fusionne ces sous-listes triées en comparant leurs éléments un par un pour reconstruire une liste triée complète.

- b. À quelle(s) ligne(s) se produit l'appel récursif ? (2pt.)

lignes 5 et 6

- c. Quelle est la première liste 'gauche' obtenue après le premier appel récursif ? Et quelle est la liste 'gauche' après le deuxième appel récursif ? (2pt.)

`[2, 4, 1] (liste[:3]) // [2] ([1])`

d. Expliquez pourquoi la condition de sortie est `len(liste) <= 1`

(1pt.)

Une liste vide ou de taille 1 est déjà triée par définition, donc on la renvoie directement sans division ni fusion supplémentaire, évitant une récursion infinie.

Ex 2: Étant donné le programme Python suivant :

(6 pts.)

```
1 def algo_tri(liste):
2     n = len(liste)
3     for i in range(n - 1):
4         ind_min = i
5         for j in range(i + 1, n):
6             if liste[j] < liste[ind_min]:
7                 ind_min = j
8         liste[i], liste[ind_min] = liste[ind_min], liste[i]
9         print("Après l'itération ", i + 1, ":", liste)
10    return liste
11
12 # Exemple d'utilisation
13 liste = [7, 3, 9, 1, 5]
14 liste_triee = algo_tri(liste)
15 print("Liste triée:", liste_triee)
```

Affiche ce qui suit après son exécution :

```
Après l'itération 1: [1, 3, 9, 7, 5]
Après l'itération 2: [1, 3, 9, 7, 5]
Après l'itération 3: [1, 3, 5, 7, 9]
Après l'itération 4: [1, 3, 5, 7, 9]
Liste triée: [1, 3, 5, 7, 9]
```

a. (2 pts.) De quel algorithme de tri s'agit-il ? Donnez son nom et décrivez son principe.

Tri par sélection

Cet algorithme parcourt la liste de gauche à droite, trouve à chaque itération l'élément minimal dans la sous-liste non triée restante, et l'échange avec le premier élément de cette sous-liste pour le placer à sa position finale

b. (2 pts.) Pourquoi le programme ne s'est-il pas arrêté après l'exécution de l'itération 3, alors que la liste était déjà triée ?

Car l'algorithme effectue toujours n-1 itérations complètes

c. (2 pts.) Quel est le rôle de la variable `ind_min` dans l'algorithme ?

La variable `ind_min` stocke l'indice de l'élément minimal trouvé dans la sous-liste non triée à partir de la position `i`.

Ex 3: Voici un programme python :

(7 pts.)

```
1 pieces = [1, 6, 10, 24, 48]
2 rendu = []
3
4 somme = int(input("Somme:"))
5 while len(pieces) > 0:
6     monnaie = pieces[-1]
7     while monnaie <= somme:
8         rendu.append(monnaie)
9         somme -= monnaie
10    pieces.pop()
11 print(rendu)
```

Dans le cas ou le programme affiche une erreur, justifiez:

- a. Qu'affiche ce programme si l'utilisateur tape 12 à l'exécution de la ligne 4 ?

(2 pts.)

[10, 1, 1]

- b. Qu'affiche ce programme si l'utilisateur tape 6.5 à l'exécution de la ligne 4 ?

(2 pts.)

Error

- c. Quel est le nom de la stratégie utilisée par cet algorithme ?

(1 pts.)

L'algorithme utilise la stratégie gloutonne.

- d. Si on introduit ce programme dans une machine, on aimerait que la machine rende un nombre minimum de pièces. Est-ce que les solutions proposées par ce programme sont toujours optimales ? Justifier.

(2 pts.)

Non. Il choisit toujours la plus grande pièce possible à chaque étape sans revenir en arrière.

Ex 4:

(2 pts.)

Quelle est la stratégie utilisée par l'algorithme Quicksort (ou tri par pivot) ?

(1 pts.)

Diviser pour régner

Nommez un autre algorithme qui utilise la même stratégie.

(1 pts.)

Mergesort (Tri Fusion), Quicksort (Tri rapide), binary search (recherche binaire)

Ex 5: Dichotomie

(10 pts.)

- a. Pour chacune des propositions, indique si elle est vraie ou fausse en justifiant. (4 pts.)

Vrai/Faux	Proposition
VRAIE	Pour utiliser la recherche dichotomique sur une liste, elle doit être triée. Justification: La recherche dichotomique nécessite une liste triée pour comparer l'élément du milieu avec la cible et éliminer efficacement une moitié de la liste à chaque étape.
FAUX	L'approche dichotomique fonctionne moins bien que l'approche brutale (brute-force) avec de grandes listes. Justification: L'approche dichotomique est bien plus efficace que brute-force sur de grandes listes ($O(\log n)$ vs $O(n)$), car elle divise l'espace de recherche par 2 à chaque itération.

- b. Si je pense à un jour de l'année et à chacune de vos questions je réponds "plus", "moins" ou "exact". Combien vous faudra-t-il d'essais **au maximum** pour trouver la réponse en utilisant une approche dichotomique? **Justifiez votre réponse.** (3 pts.)

Il faut 9 essais maximum. Avec 365 jours, $\log_2(365) = 9$, car $2^8 = 256 < 365$ et $2^9 = 512 > 365$ et . Ainsi 9 questions suffisent pour identifier n'importe quel jour.

- c. En détaillant les étapes, utilisez la recherche dichotomique pour trouver l'indice de la valeur **23** dans la liste suivante: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31] (3 pts.)

- Étape 1 : Milieu = 5, liste[5] = 13 < 23 → recherche droite [17, 19, 23, 29, 31] (indices 6-10)
- Étape 2 : Milieu = 8, liste = 23 = 23 → indice trouvé : 8

Ex 6: Voici un programme:

(8 pts.)

```
1 def rec(liste, element):
2     gauche = 0
3     droite = len(liste) - 1
4
5     while gauche <= droite:
6         milieu = (gauche + droite) // 2
7         if liste[milieu] == element:
8             return milieu
9         elif liste[milieu] < element:
10            gauche = milieu + 1
11        else:
12            droite = milieu - 1
13    return -1
14
15
16 liste = [1, 3, 5, 7, 9, 11, 13]
17 indice = rec(liste, 7)
18 print("L'élément est trouvé à l'index : ", indice)
19
```

- a. (2 pts.) Quelle stratégie algorithmique emploie cet algorithme sous forme de code Python ?
Justifier!

recherche dichotomique (ou recherche binaire) -> Diviser pour régner

Il travaille sur une liste **triée** (ici : [1, 3, 5, 7, 9, 11, 13])
À chaque étape, il calcule le **milieu** de l'intervalle de recherche
Il compare l'élément du milieu avec la valeur cherchée
Selon le résultat, il **élimine la moitié** de l'intervalle (gauche ou droite)
Il répète jusqu'à trouver l'élément ou réduire l'intervalle à vide

- b. (2 pts.) La fonction **rec** utilise-t-elle la récursivité ?
Justifier!

NON

il n'y a aucun appel à **rec(...)** à l'intérieur de la fonction elle-même

- c. (2 pts.) Dans quel(s) cas la fonction **rec** retournerait-elle la valeur **-1** ?
Justifier!

Lorsque l'élément recherché n'est pas présent dans la liste

- d. (2 pts.) Que va afficher le programme précédent lorsqu'on l'exécute ?
Justifier!

L'élément est trouvé à l'index : 3

La liste est : [1, 3, 5, 7, 9, 11, 13] → indices : 0, 1, 2, 3, 4, 5, 6
On cherche l'élément 7 :
milieu = (0+6)//2 = 3 → liste[3] = 7 → **égalité** ! → La fonction retourne immédiatement milieu, c'est-à-dire 3

Option A

Voici une liste d'objets qu'un commerçant souhaite emporter dans sa valise pour les vendre. Il espère gagner un maximum d'argent mais le poids est limité à 30 kg pour prendre l'avion.

objets	A	B	C	D	E	F	G	H
poids (kg)	6	5	10	12	15	14	9	8
valeur (fr)	42	30	50	48	65	60	32	28

Utiliser l'algorithme glouton pour composer le sac de ce voyageur.

a. Proposer un critère pour choisir "le meilleur" objet à chaque étape. (1 pt.)

À chaque étape, l'algorithme glouton choisit l'objet qui a le plus grand rapport valeur/poids parmi ceux qui restent et qui rentrent encore dans le sac (poids restant $\geq$ poids de l'objet)

b. Compléter le tableau suivant en indiquant dans l'ordre les objets pris par l'algorithme.

Attention, il se pourrait qu'il y ait plus de colonnes que nécessaire. (2 pts.)

objets	A	B	C	G			totaux
poids (kg)	6	5	10	9			30
valeur (fr)	42	30	50	32			154

c. La solution glouton du b est-elle optimale ? Choisir ci-dessous et compléter. (2 pts.)

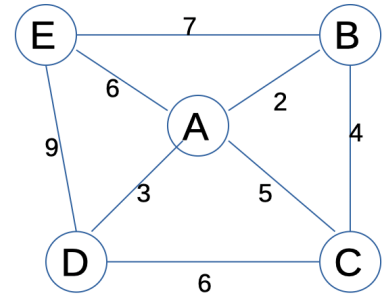
- oui, car l'ensemble $\{A, B, C, G\}$ de valeur 154 fr est meilleur que toutes les autres combinaisons d'objets respectant la contrainte de 30 kg.

Option B

Voici un graphe représentant 5 villes et les distances qui les séparent.

Un voyageur de commerce souhaite partir de la ville A pour aller dans la ville E.

Pour l'aider à trouver l'itinéraire le plus court, on te demande d'utiliser un algorithme glouton.



- a. (1 pt.) Quel serait le critère glouton pour choisir, à chaque étape, la prochaine ville à visiter?

À chaque étape, choisir la route la plus courte partant de la ville où l'on se trouve et allant vers une ville encore non visitée (en visant finalement E)

- b. (2 pts.) Appliquer cet algorithme pour déterminer l'itinéraire que devrait suivre le voyageur:

- i. Itinéraire glouton:

$A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$

- ii. Distance totale:

$2 + 4 + 6 + 9 = 21$

- c. (2 pts.) Est-ce que cet itinéraire est le vraiment le meilleur possible?

- i. Si oui, justification:

- ii. Sinon, itinéraire et distance meilleurs:

$A \rightarrow D \rightarrow C \rightarrow B \rightarrow E$, distance totale = 20

$A \rightarrow E$, distance totale = 6

Ex 8: programmation récursive simple

(8 pts.)

Voici un programme Python:

```
1 def f(n):
2     if n == 0:
3         print("Partez!")
4     else:
5         print(n)
6         f(n-1)
7
8 f(5)
```

a. Comment s'appelle la fonction récursive?

(1 pt.)

f

b. Quelle est la condition de sortie de la fonction récursive?

(1 pt.)

n == 0

c. Qu'affiche ce programme?

(2 pts.)

5
4
3
2
1
Partez!

d. Combien de fois s'exécute la fonction récursive?

(1 pt.)

6 fois

e. Combien de fois s'exécute la fonction si on l'appelle avec n = 0 : f(0) ?

(1 pt.)

1 fois: Elle entre directement dans le cas n == 0

f. Qu'est-ce qui va changer dans l'exécution du programme si on supprime la ligne 4 (**else:**) et qu'on désindente les deux lignes suivantes pour qu'elles soient alignées avec le **if** ?

(1 pt.)

Boucle infinie (récursion infinie) et erreur RecursionError

g. Est-il possible de réécrire ce programme sans utiliser de récursion ? Justifier sans réécrire un programme en Python.

(1 pt.)

On peut le remplacer par une boucle for ou while qui parcourt les nombres de 5 à 0 (inclus), en affichant chaque nombre, et enfin "Partez!" à la fin.

Ex 9: (Bonus)

(4 pts.)

Indiquer dans l'ordre tout ce qui sera affiché dans la console à la fin de l'exécution de chaque programme Python ci-dessous. Justifier avec ce tableau de valeurs des variables.

programme	tableau de valeurs des variables	Affichage
<pre> L=[4,2,9,5] Lg=[] Ld=[] p=L[0] for m in L : if m<=p: Lg.append(m) else : Ld.append(m) print('Lg=',Lg) print('Ld=',Ld) </pre>	<pre> L=[4,2,9,5] Lg=[] Ld=[] p=4 m=4 L=[4,2,9,5] Lg[4] Ld[] p=4 m=4 L=[4,2,9,5] Lg[4,2] Ld[] p=4 m=2 L=[4,2,9,5] Lg[4,2] Ld[9] p=4 m=9 L=[4,2,9,5] Lg[4,2] Ld[9,5] p=4 m=5 </pre>	<pre> Lg= [4, 2] Ld= [9, 5] </pre>

Aide-mémoire informatique

2^{ème} année, 1^{er} semestre

Algorithmes

Stratégies:

Force brute

Consiste à calculer toutes les possibilités d'un problème puis à trouver la (les) solution(s) optimale(s). On trouve toujours la meilleure solution mais selon l'algorithme et la taille de l'entrée, il est possible que le temps de calcul soit rédhibitoire.

Glouton

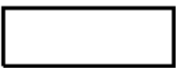
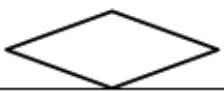
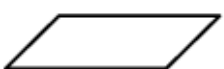
On choisit, à chaque étape de l'exécution de l'algorithme, une possibilité qu'on estime optimale (localement). Il faut donc un critère pour faire ce choix. On laisse tomber toutes les autres et on gagne ainsi en efficacité.

Par contre, on n'est pas sûr de trouver la solution optimale (globalement).

Diviser pour régner

On divise un problème complexe en plusieurs sous-problèmes moins complexes. Une fois que les sous-problèmes sont suffisamment petits, on les résout. On finit par combiner les sous-solutions afin d'obtenir la solution globale au problème.

Rappel pour les logigrammes:

Symbole	Nom	Rôle
	Flèche	Indique la séquence du programme
	Terminaison	Début ou termine un programme
	Processus	Exécute une tâche
	Décision	Génère une décision (oui, non, etc.)
	Entrée/Sortie	Accepte une entrée ou produit une sortie

Programmation (Python)

Calculs :

opérations de base: + - * / **

reste et quotient de division euclidienne : % //

Rappel : $x \% y$ vaut 0 si x est divisible par y

exemples : $5 ** 2 \rightarrow 25$ $5 // 2 \rightarrow 2$ $5 \% 2 \rightarrow 1$

Variables et affectations :

types: int, float, str, bool, list ...

int(), float() pour **convertir** en int ou en float

affectation simple : `ma_variable = -5`

permutation de variables: `a, b = b, a`

affectations combinées: `+=`, `-=`, `*=`, etc.

exemple : `x += 1` est comme `x = x + 1`

Listes :

liste: `L = [-1, 3.2, "abc"]`

liste vide : `[]`

multiplication: `[0]*3 -> [0, 0, 0]`

définir un élément d'une liste :

`L[1] = 5` , `L[1]` doit déjà exister,
l'indice commence à 0.

ajouter un élément à une liste :

`L.append(-2.5)`

supprimer élément à la **position i** :

`del L[i]` ou `L.pop(i)`

supprimer élément de **valeur v** dans L :

`L.remove(v)`

supprimer le dernier élément dans L :

`L.remove()` ou `L.pop()`

longueur d'une liste L :

`len(L)`

somme des valeurs d'une liste:

`sum(liste)`

slice: `L[i:j]` (de i inclus à j exclu)

`L[i: ]` (de i inclus à la fin)

`L[ :j]` (du début à j exclu)

Chaînes de caractères :

chaîne: `txt="bonjour"` ou `txt='bonjour'`

concaténation: `txt + " est un texte"`

élément, slice d'une chaîne: `texte[i]`

comme listes mais non modifiable

Entrées/sorties :

sortie: `print(valeur, ... , end='\n')`

valeur de type quelconque, end optionnel

entrée: `texte = input("message")`

Attention: retourne toujours un str

Commentaires :

sur une ligne : `# texte`

plusieurs lignes : `"""texte"""` ou `"""texte"""`

Structures de contrôle :

1^{ère} ligne finit par `:` puis indentation des lignes du bloc

tests simples: `==`, `!=`, `<`, `<=`, `>`, `>=`, `in`

tests combinés: `and`, `or`, `not`, `a < b < c`

conditionnelle

if condition1:

 instructions

elif condition2:

 # elif = contraction de else if

 instructions

else:

 instructions

boucle while (boucle non bornée) :

while condition:

 instructions

boucle for (boucle bornée) :

for i in liste: # boucle parcourt la liste

 instructions

for i in range(5): # pour i de 0 à 4

 instructions

range(i,j,k): de i (inclus) à j (exclu) par pas de k

sauter, quitter une boucle : `continue`, `break`

Fonctions:

définition de la fonction - exemple d'une fonction f à deux paramètres :

def f(x, y):

 reponse = $x**2 + y**2$

 return reponse

appel de la fonction- exemples avec la fonction f :

variable = f(3, 5)

print(f(2, 6))

etc.