Collège de Candolle

ÉPREUVE REGROUPÉE

Discipline: INFORMATIQUE **Date:** 13/12/2024
Cours: 2 INDP 06 08 09 **Durée:** 105 minutes

Maître: H GARCIA
Matériel autorisé: aucun (aide-mémoire inclus à la fin de l'épreuve)

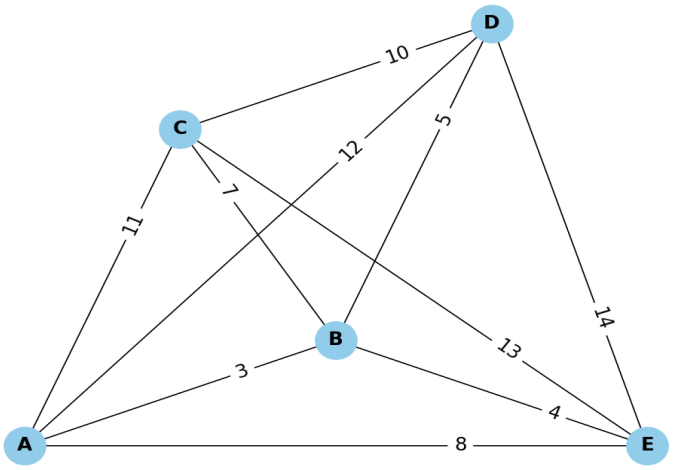
Nom et prénom: **Total: 50 points**

Ex 1: *(5 pts)* Voyageur de commerce glouton

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

Un voyageur de commerce nommé Edsger et habitant dans la ville E souhaite partir de chez lui (ville E), passer par chaque autre ville une fois et revenir chez lui, dans la ville E.

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



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

La ville la plus proche qui ne soit pas répétée.

b. (2pts) Suis cet algorithme pour déterminer l'itinéraire que devrait suivre Edsger:

i. Itinéraire glouton: EBACDE

ii. Distance totale: 42

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

i. Si oui, justification:

ii. Sinon, itinéraire et distance meilleurs: EBDCAE (38)

Ex 2: *(3 pts)* stratégies algorithmiques

Donner le nom des trois stratégies algorithmiques vues en classe:

a. (1pt) Cette stratégie consiste à explorer toutes les possibilités d'un problème avant de donner une réponse:

Force Brute

b. (1pt) Cette stratégie fait un choix **optimal** à chaque étape en écartant toutes les autres possibilités. Ceci conduit à une résolution beaucoup plus rapide mais elle peut parfois ne pas être idéale:

Algorithme Glouton

c. (1pt) Cette stratégie réduit la complexité d'un algorithme en divisant son entrée en plusieurs parties puis en résolvant chaque sous-partie avant de combiner leurs solutions pour trouver une solution globale:

Diviser pour régner

Ex 3: *(8 pts)* Algorithmes de tri...

Voici une liste désordonnée de nombres: **[1, 8, 0, 5, 3]**

a. (4pts) Choisir un algorithme de tri simple (parmi sélection, insertion ou bulles)
Écrire chaque étape dans la résolution de cet algorithme, c'est-à-dire la liste des nombres après chaque échange effectué dans l'algorithme choisi. De plus, souligner les nombres triés par l'algorithme au fur et à mesure de l'avancement du tri.

- Algorithme choisi:
- Etapes de l'algorithme pour trier la liste:

Tri par Sélection	Tri par Insertion	Tri à Bulles
[**0**,8,**1**,5,3]	[**0**,1,8,5,3]	[1,**0**,**8**,5,3] [**0**,**1**,5,3,8]
[0,**1**,**8**,5,3]	[0,1,**5**,8,3]	[1,0,**5**,**8**,3] [0,1,**3**,**5**,8]
[0,1,**3**,5,**8**]	[0,1,**3**,5,8]	[1,0,5,**3**,**8**]

b. (4pts) Faire de même avec un algorithme avancé (tri par fusion ou tri par pivot)
Chaque étape peut contenir plusieurs listes.

Nouvelle liste désordonnée de nombres: **[6, 8, 0, 3, 5, 2, 9, 4]**

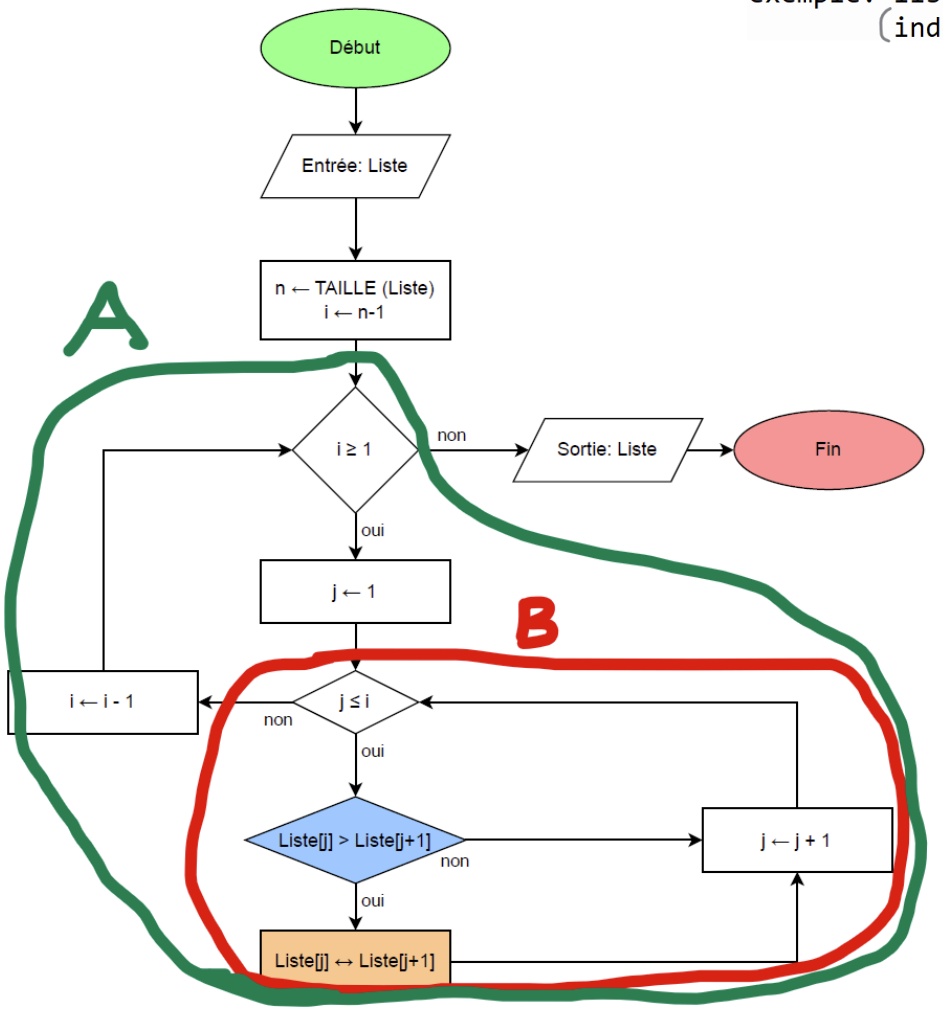
- Algorithme choisi:
- Etapes de l'algorithme pour trier la liste:

Tri par Fusion		Tri par Pivot
[6,8,0,3,5,2,9,4]	[6,8] [0,3] [2,5] [4,9]	[0,3,2] **4** [6,8,5,9]
[6,8,0,3] [5,2,9,4]	[0,3,6,8] [2,4,5,9]	[0] **2** [3] [4] [6,8,5] **9**
[6,8] [0,3] [5,2] [9,4]	[0,2,3,4,5,6,8,9]	[0] [2] [3] [4] **5** [6,8][9]
[6] [8] [0] [3] [5] [2] [9] [4]		[0] [2] [3] [4] [5] [6] **8** [9]

Ex 4: *(7 pts)* Logigramme

Voici le logigramme d'un algorithme de tri (les index des éléments de la liste vont de 1 à n):

```
exemple: liste = [5, 4, 8, 0]
                 (indexes: 1  2  3  4)
```



a. (1pt) De quel algorithme s'agit-il?

Bulle

b. (2pts) Que fait chaque instruction de la boucle B (entourée à l'intérieur) ? Résumer en une phrase le travail effectué par cette boucle.

A comparer chaque paire et faire les échanges si la paire est désordonnée (au sein d'une bulle)

c. (2pts) Faire de même pour la boucle A (entourée à l'extérieur).

à comptabiliser où commence la dernière paire qu'il faut comparer au sein d'une bulle. Autrement dit, i+1 est la fin de la liste non triée, au-delà, la liste est déjà triée.

d. Avec la liste exemple (**`[5, 4, 8, 0]`**), combien de fois va se répéter...

i. (1pt) la boucle A (extérieure)? 3 fois

ii. (1pt) la boucle B (intérieure)? 3+2+1=6

Ex 5: *(8 pts)* Pseudocode

Voici le pseudo code d'un algorithme:

```
ENTREE système
ENTREE somme
rendu = [] (liste vide)
TANT_QUE (somme > 0)
    {
     monnaie = système[dernier]   (monnaie de plus haute valeur)
     SI (monnaie > somme)
     ALORS
           {
            ENLEVER monnaie DE système
           }
     SINON
           {
            AJOUTER monnaie A rendu
            somme = somme - monnaie
           }
    }
SORTIE rendu
```

a. (1pt) A quoi sert cet algorithme?

A rendre une somme de monnaies avec le moins de pièces possible.

b. (1pt) A quelle stratégie algorithmique appartient-il ?

Glouton

c. (1pt) Entourer la boucle de cet algorithme en couleur.

d. Avec le système **[1, 2, 5]** et la somme **6**, combien de fois sera exécutée...

i. (1pt) la ligne surlignée (retrait d'une monnaie de la liste)

2 fois

ii. (1pt) le bloc correspondant au `TANT_QUE (TANT QUE (somme > 0))`

4 fois

e. Expliquer brièvement ce que fait l'algorithme dans les deux situations suivantes:

i. (1pt) système **[1, 2, 5]** et la somme **0**

rendu = [ ]

La boucle ne s'exécute pas

ii. (2pts) système **[1, 2, 5]** et la somme **0,5**

rendu = [ ]

La boucle s'exécute 3 fois pour enlever les monnaies du système

Ex 6: *(8 pts)* code Python

Voici un programme Python:

```
1  def cherche(liste, valeur):
2      trouve = False
3      for element in liste:
4          if element == valeur:
5              trouve = True
6      return trouve
7
8  l = [1, 4, 7, 11, 13, 42, 57]
9  v = 4
10 print(cherche(l, v))
```

a. (1pt) Que fait ce programme?

Cherche un élément (valeur) dans une liste

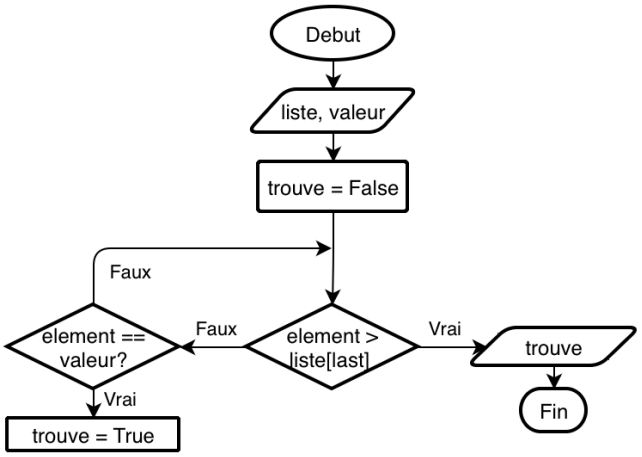
b. (1pt) Qu'affiche ce programme quand on l'exécute?

True

c. (2pts) Combien de fois la boucle des lignes 3 à 5 va-t-elle se répéter ?

7 fois

d. (4pts) Traduire la fonction **cherche( )** en logigramme ou en pseudo code ci-dessous:



PSEUDOCODE

```
ENTREE liste
ENTREE valeur
trouve ← false
POUR chaque element DANS liste
{
   SI (element == valeur)
   ALORS
      {
         trouve ← True
      }
}
SORTIE trouve
```

Ex 7: *(5 pts)* récursivité

a. (2pts) Décrire en quelques mots ce qu'est la récursivité en informatique:

Technique de programmation où une fonction s'appelle elle-même pour résoudre un problème.

b. (2pts) Dans la fonction python récursive ci-dessous, indique à quelle ligne on trouve...

i. la condition de sortie de la récursion 5

ii. l'appel récursif 9

iii. la définition de la fonction 1

iv. le programme principal 12

c. (1pts) Qu'affiche ce programme quand on l'exécute?

3
2
1
0

programme python récursif:

```
1  def compte_a_rebours(n):
2      """
3      affiche un compte à rebours
4      """
5      if n == 0:
6          print(n)
7      else:
8          print(n)
9          compte_a_rebours(n-1)
10
11
12 compte_a_rebours(3)
```

Ex 8: *(6 pts)* Voici une fonction Python récursive:

```
1  def tri_pivot(liste):
2      # si la liste est vide ou contient un seul élément
3      if len(liste) <= 1:
4          return liste
5      else:
6          # pivot: dernier élément de la liste
7          pivot = liste[-1]
8
9          gauche = [x for x in liste if x < pivot]
10         milieu = [x for x in liste if x == pivot]
11         droite = [x for x in liste if x > pivot]
12
13         # Appliquer tri_pivot aux sous-listes et les recombiner
14         return tri_pivot(gauche) + milieu + tri_pivot(droite)
```

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

Diviser pour régner -> Tri pivot

b. (2pts) A quelle ligne se fait l'appel récursif ? Justifier!

14

c. (2pts) A quelle ligne s'effectue la combinaison des solutions aux sous-parties de liste à trier? Justifier!

14

FIN DE L'ÉPREUVE (aide-mémoire suit)

Aide-mémoire informatique

2ème année, 1er 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.

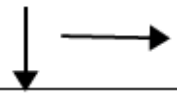
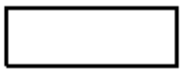
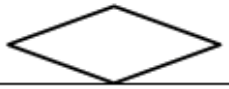
Glouton

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	Débute 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: % //
var % div vaut 0 si var est divisible par div
ex: 5 ** 2 → 25 5 // 2 → 2 5 % 2 → 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.
ex: x += 1 est comme x = x + 1

Listes :
liste: `L = [-1, 3.2, "abc"]`
liste vide : `[]`
multiplication: `[0]*3 -> [0, 0, 0]`
élément d'une liste: `L[1] = 5`
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)`
longueur d'une liste L :
`len(L)`
somme des valeurs d'une liste:
`sum(liste)`
slice: `L[i:j]` (i inclus ! 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 :
1ère 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 f à deux paramètres:
```
def f(x, y):
    reponse = x**2 + y**2
    return reponse
```

appel de la fonction: f()
variable = f(3, 5)
print(f(2, 6))
etc.