Collège de Candolle

CORRIGE - ÉPREUVE REGROUPÉE

Discipline: INFORMATIQUE **Date:** 16/12/2022
Cours: 2INDP 01 à 10 **Durée:** 105 minutes

Maître: Chiou/Barros/Garcia/Schmidt **Total: 44 points**

Ex 1: On souhaite trier la suite de nombres suivante: [4, 1, 2, 5, 3] **(3 pt)**

a. Observez les deux premières étapes de l'algorithme de tri ci-dessous et donnez son nom.

Réponse: Il s'agit d'un algorithme de tri par **bulles**.

b. Est-ce que l'algorithme de tri est arrivé à son terme? Explicitez votre réponse.

Réponse : **Non**, car seuls les deux derniers éléments sont certainement bien placés après les 2 premières étapes. Il faut encore procéder à **2 étapes** pour finaliser l'algorithme. On sera alors certain que tous les éléments sont ordonnés.

Ex 2: Voici toutes les étapes d'un tri de la liste suivante: [1, 3, 2] **(2 pt)**

[1, 3, 2] / [1, 2, 3] / [2, 3, 1] / [2, 1, 3] / [3, 1, 2] / [3, 2, 1] -> Solution: [1, 2, 3]

Quelle **stratégie algorithmique** a-t-on utilisée pour trouver le résultat ? (justifiez votre réponse)

Réponse : Il s'agit de la stratégie "**Force brute**". En effet, on voit qu'on a fabriqué **toutes les combinaisons possibles**. Après cela, on a cherché la (les) solution et il n'y a en a qu'une: [1, 2, 3].

Ex 3: Triez la liste suivante avec l'algorithme de **tri par pivot** (appelé aussi quicksort): **(~5 pts)**

[0, 4, 2, 5, 6, 3, 1]

On choisit le dernier élément comme pivot pour chaque étape.

[0, 4, 2, 5, 6, 3, 1]
[0] **[1]** **[4, 2, 5, 6, 3]**
[0] **[1]** **[2]** **[3]** **[4, 5, 6]**
[0] **[1]** **[2]** **[3]** **[4, 5]** **[6]**
[0] **[1]** **[2]** **[3]** **[4]** **[5]** **[6]**
[0, 1, 2, 3, 4, 5, 6]

Ex 4: On considère le problème de rendu de monnaie optimal où l'on doit rendre 10€.
On dispose du système de monnaies imaginaire suivant : 1€, 5€ et 8€.

a. Quel est le rendu de monnaie qu'on obtient avec un algorithme glouton ?
1 x 8€, 2 x 1€ (glouton = choix de la plus grande pièce possible à chaque étape)

b. Est-ce la solution optimale ? (oui / non)
Si oui, expliquez votre réponse. Sinon, donnez la solution optimale.
Non, la solution optimale est: **2 x 5€** (2 pièces est plus optimal que 3)

Ex 5: Un Sherpa doit traverser la montagne pour vendre des marchandises dans le village voisin. Il ne peut pas transporter plus de 20 kg dans son sac à dos. **(~6 pts)**

a. Quels objets doit-il mettre dans son sac s'il applique un algorithme glouton dans la résolution de ce problème ?

objets	1. A	2. D	3. E			totaux
masses	8	6	5			19
valeurs	10	9	7			26

b. Est-ce la solution optimale ? (~~oui~~ / **non**)
non, voici une meilleure solution:

objets	1. A	2. C	3. D	4. F		totaux
masses	8	3	6	2		19
valeurs	10	5	9	6		30

Ex 6: Le programme suivant doit lister tous les multiples de 5 inférieurs à 100. **(1 pt)**
Corrigez le code pour qu'il le fasse correctement. Pour cela, entourez une erreur et écrivez clairement sa correction.

```
diviseur = 5

for i in range(100):

    if i / diviseur == 0:     remplacé par: % (modulo)

        print(i, 'est un multiple de ', diviseur)
```

Ex 7: On souhaite que le programme suivant affiche un décompte de 10 jusqu'à 0. **(2 pt)**
Corrigez le code pour qu'il le fasse correctement.

```
1  #Compte à rebours
2  def compte_a_rebours(nombre) :
3      while True :          <-- nombre>=0
4          print(nombre)
5          nombre = nombre - 1
6
7  compte_a_rebours(10)
```

Ex 10: Établissez sous forme de pseudo-code ou logigramme (à choix) un algorithme permettant de trouver **la valeur la plus grande** d'une liste désordonnée de nombres. **(~6 pts)**

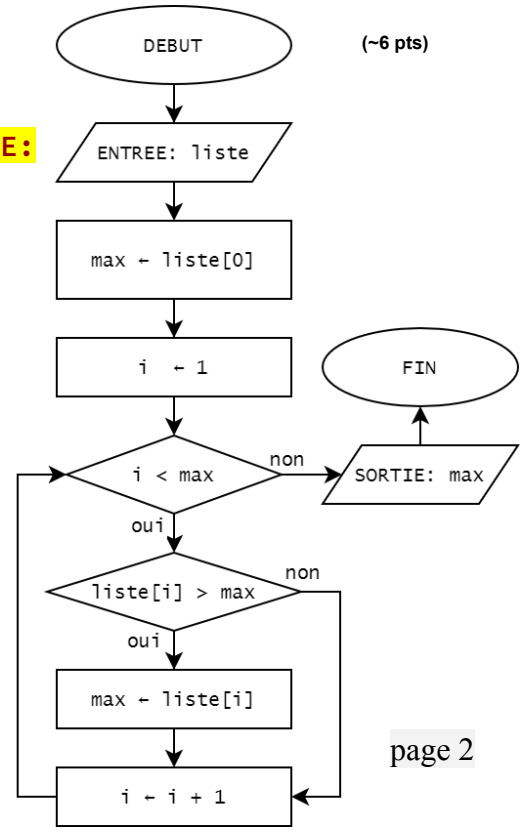
PSEUDO-CODE:

```
ENTREE: liste

max ← liste[0]
POUR i DE 1 à LONGUEUR(liste) {
    SI liste[i] > max
    ALORS max ← liste[i]

SORTIE: max
```

LOGIGRAMME:



Ex 8: Complétez les parties manquantes (en jaune) du programme ci-dessous : **(~6 pts)**

```
# déclarer une liste vide L

L = []

# répéter 10 fois

for i in range(10):

# Demander deux nombre N1 et N2

    N1 = float(input("Donnez un nombre décimal : "))

    N2 = float(input("Donnez un autre nombre décimal : "))

    # identifier le grand nombre et le petit nombre :
    # d'abord, si ce sont les mêmes nombres, l'exécution s'arrête,

    if N1 == N2 :
        break

    # puis, si les nombres sont différents, on classe les nombres.
    elif N1 < N2 :
        grand_Nombre = N2
        petit_Nombre = N1

    else :
        grand_Nombre = N1
        petit_Nombre = N2

    # Calculer la différence entre ces deux nombres
    difference = grand_Nombre - petit_Nombre

    # ajouter la différence dans la liste L

    L.append(difference)

# fin de répéter

# Calculer la somme des éléments de L.
total = sum(L)

# Calculer la moyenne des éléments de L:

moyenne = total / len(L)

# fin du programme
```

Ex 9: Voici le pseudo-code d'un algorithme nommé "binary search". (~10 pts)

a. Quelle est la stratégie employée dans cet algorithme? (1 pts)
réponse: **Diviser pour régner**

b. Écrivez la fonction "binary_search" en Python correspondant à cet algorithme. (~6 pts)

```
def binary_search(liste, valeur):
  # Écris ton code ci-dessous:
  # 1 ligne/cm, indentation = 1cm
  # lignes vides avant return: ok
  min = 0
  max = len(liste)-1
  trouve = False
                while min<=max and trouve==False:
    milieu = (min+max)//2
    if liste[milieu] > valeur:
      max = milieu - 1
                    elif liste[milieu] < valeur:
      min = milieu + 1
    else:
      trouve = True
  return trouve
```

c. Indiquez clairement dans le pseudo-code les modifications permettant de retourner la position (au lieu de **VRAI**) où a été trouvée la **valeur** dans la **liste** ou "-1" (au lieu de **FAUX**) si la **valeur** n'est pas dans la **liste** . (3 pts)

```
DONNEES_ENTREES: liste, valeur

min ← 0
max ← longueur(liste)
trouve ← -1
TANT_QUE (min < max) ET (trouve EST_EGAL_A -1) {
  milieu ← ENTIER( (min+max)/2 )
  SI liste[milieu] EST_EGAL_A valeur
  ALORS trouve ← milieu
  SINON {
      SI liste[milieu] < valeur
      ALORS min ← milieu+1
      SINON max ← milieu-1
      }
}

DONNEE_SORTIE: trouve
```

Ex 10: Cet exercice est corrigé après l'ex. 7.