**Exercice 1 (2.5 points). Soit l’algorithme de tri suivant :**

Trier faux

Tant que (non Trier) faire

DTQ

Trier vrai

Pour i 0 à n-2 faire

DP

Si T[i]  T[i+1] alors

DSI

tmp  T[i]

T[i]  T[i+1]

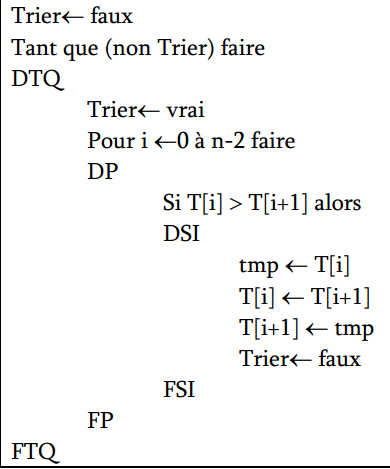
T[i+1]  tmp

Trier faux

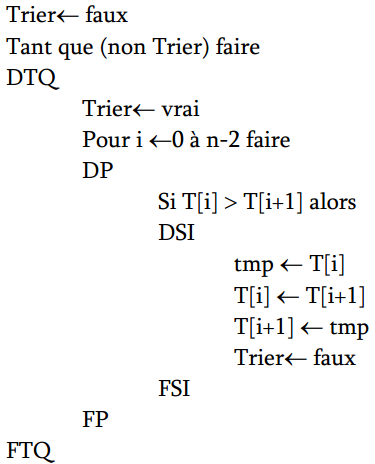
FSI

FP

FTQ



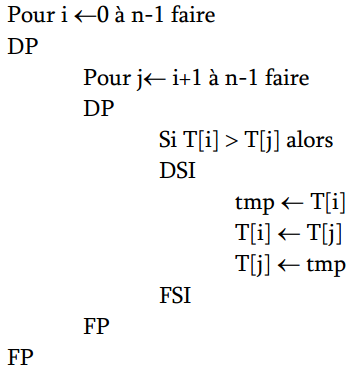
**Exercice 1 (2.5 points). Soit l’algorithme de tri suivant :**



1. Donner le nom de cet algorithme de tri. ………………..……………………………………………

2. Calculer sa complexité

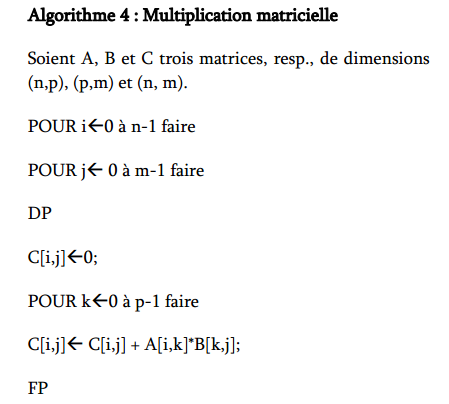
Exercice 1 (2.5 points). Soit l’algorithme de tri suivant :

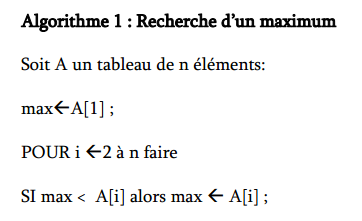


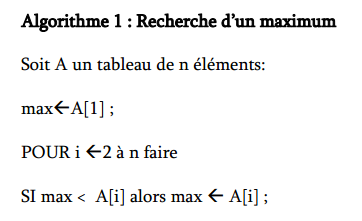
1. Donner le nom de cet algorithme de tri. ………………..……………………………………………

2. Calculer sa complexité.

**Exercice 3 : Calculer la complexité au pire des algorithmes suivants :**







**EXERCICE 3 :**

Un polynôme peut être représenté par une LLC.

1. Dire comment en définissant la structure de données du maillon ainsi que le modèle LLC.

2. Ecrire les algorithmes ci-dessous, ensuite calculer leur complexité.

a. calcul du polynôme en un point x donné.

b. dérivé d'un polynôme.

c. Intégral d’un polynôme

d. somme de deux polynômes.

e. produit de deux polynômes.

**EXERCICE 6 :**

Ecrire un algorithme qui permet d'afficher les éléments d'une LLC dans l'ordre inverse en utilisant une pile.

EXERCICE 4 :

On veut implémenter une file d'attente où chaque élément (X) contient un nombre variable d'entiers (Nx).

1. Définir les structures de données nécessaires

2. Quelle est la meilleure présentation pour implémenter cette file. Donner son modèle.

EXERCICE 7 : Evaluation d’expression arithmétique

Une expression arithmétique est une combinaison des opérandes (les nombres) et des

opérateurs ((, ), +, -, \* ; /, %). Elle peut être présentée sous trois formes :

- Infixée : <expG> <opérateur> <expD>

- Postfixée : <expG> <expD> <opérateur>

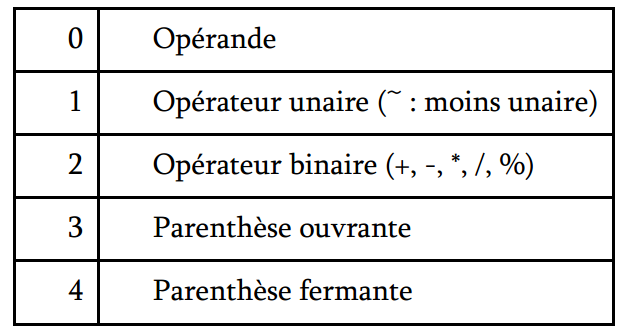
- Préfixée : <opérateur> <expG> <expD>

1. Ecrire cette expression en postfixé et préfixé : 5 \* (((9+8)\*(4\*6))+7)

2. Ecrire la procédure qui permet de tester si une expression infixée (en chaînes de

caractères) est bien écrite et de retourner l’expression dans un tableau dont les

éléments sont de type : (Valeur, Type).



3. Ecrire l’algorithme qui permet de transformer une expression infixé en une expression postfixé

4. Ecrire l’algorithme qui permet de transformer une expression infixé en une expression préfixé

5. Pour chaque présentation, donner l’algorithme d’évaluation correspondant en utilisant la fonction Calcul (<opd1>op<opd2>) qui permet de retourner le résultat de l’expression <opd1> op <opd2>

**Soient les modules suivants :**

1. Som (n) calcule la somme des n premiers entiers naturels

2. Som (a, b) calcule la somme des entiers a et b

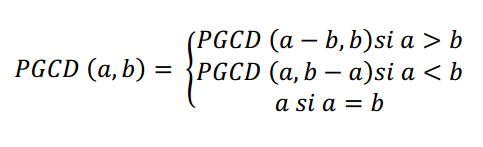
3. Prod (a, b) calcule le produit de entiers a et b

4. Puiss (a, b) calcule a puissance b

5. Quotient (a, b) calcule le quotient de a par b

6. Reste (a, b) calcule le reste de division de a par b

7. PGCD (a, b) calcule le Plus Grand Commun Diviseur entre a et b



8. RechMax (Tab, IndDeb, IndFin) recherche l’indice de l’élément le plus grand de Tab de taille n

9. RechMin (Tab, IndDeb, IndFin) recherche l’indice de l’élément le plus petit de Tab de taille n

10. Rech (Tab, IndDeb, indFin, x) : recherche l’indice de l’élément contenant la valeur x dans Tb (Tab

est un tableau de taille n non trié)

11. RechDichotomique (Tab, IndDeb, IndFin, x) : recherche par dichotomie la valeur x dans Tab (Tab

est un tableau de taille n trié par ordre décroissant)

12. Tri\_Fusion (Tab, n) : trie le tableau par la méthode de fusion.

13. Som (Tab, n) : calcule la somme des éléments de Tab de taille n.

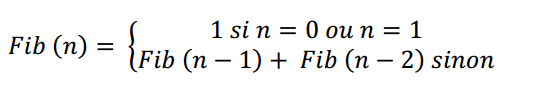
14. Prod (Tab, n) : calcule le produit des éléments de Tab de taille n.

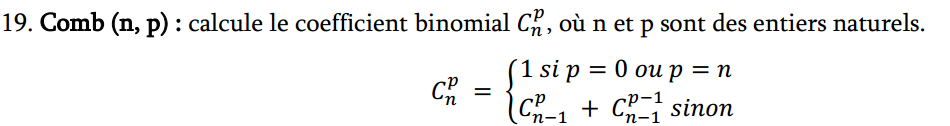
15. Moyenne (Tab, n) : calcule la moyenne des éléments de Tab de taille n.

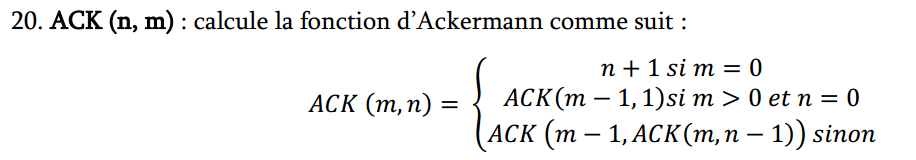
16. InverserLLC (L, LI) : inverse la liste L (L est une LLC unidirectionnelle)

17. RechLLC (L, x) : recherche un élément dans la liste L (L est une LLC unidirectionnelle

18. Fib (n) : calcule la suite Fibonnacci comme suit :







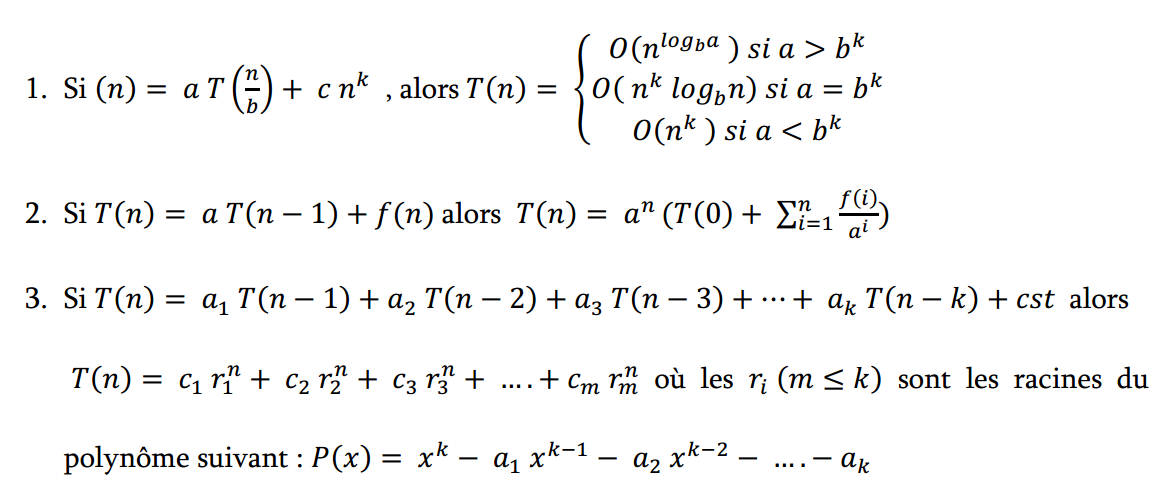
Pour chaque module:

- Écrire l’algorithme récursif correspondant

- Calculer le temps d’exécution, ensuite, déduire la complexité de l’algorithme (si possible)

- Décursiver l’algorithme

**Indication sur les résolutions des récurrences**



**EXERCICE 2 :**

Soit L une LLC non ordonnée où chaque maillon contient les informations d’un livre:

- L’identifiant du livre (entier positif)

- Le titre

- L’auteur

- L’année d’édition (entre 1500 et 2050)

- Langue d’édition (A : Arabe, F : Français, E : Anglais, D : Allemand, S : Espagnol,

I : Italien)

- Liste de mots clés (table de 10 éléments de type chaîne [25])

1. Définir la structure de données du maillon ainsi que le modèle LLC

2. Trier la liste selon l’ordre croissant de l’identifiant.

3. Eclater la liste en deux LLCs L1 et L2 tel que L1 contient les livres arabes et français et

L2 contient les autres livres.

4. Calculer la complexité des algorithmes précédents.

**EXERCICE 1 :**

Soit L une LLC unidirectionnelle ordonnée contenant des valeurs entières.

1. Développer les modules suivants qui permettent de :

a. Construire la liste à partir de n données lues.

b. Imprimer la liste

c. Calculer la longueur de la liste.

d. Rechercher le maillon contenant la valeur « v » dans la liste

e. Rechercher la valeur du maillon de position « pos ».

f. Rechercher l'élément qui a le plus grand nombre d'occurrences.

g. Insérer une valeur « v » dans la liste.

h. Supprimer tous les maillons contenant la valeur « v ».

i. Supprimer le maillon de position « pos ».

j. Interclasser cette liste avec une autre liste ordonnée.

k. Détruire la liste

2. Calculer la complexité des modules ci-dessus, en fonction de la taille de la liste (soit

n).

3. Refaire la question 1 dans le cas où la liste ordonnée L est

a. bidirectionnelle,

b. circulaire et

c. bidirectionnelle circulaire.

**Exercice 04 Calculer la complexité de l’algorithme suivant :**

i ←1

Tant que (i < n) faire

DTQ

j← 1

Tant que (j < 2\*n) faire

j ← j\*2

i ← i+1

FTQ

**Partie 3 : Les Piles**

Exercice 01 : Ecrire un algorithme qui permette d'afficher les éléments d'une LLC dans l'ordre

inverse en utilisant une pile.

**Exercice 02 :** Evaluation d’expression postefixé

1. Représenter les expressions suivantes sous forme polonaise postfixée :

a+b , (a+b)/d , ((c+d) +(d-e))+5, -(a+b)+(5+b)c, -(((a+b)+(c-d))/5)+a5

2. Donner l'algorithme d'évaluation avec l'utilisation d'une pile.

Exercice 03 : Soit à analyser des expressions mathématiques qui utilisent les symboles (, { et pour

l'ouverture des sous expressions et les symboles ), } et pour leur fermeture respective. Chaque

symbole de fermeture doit être associé à son symbole d'ouverture. Utiliser le modèle de pile pour

écrire l'algorithme qui effectue une telle vérification.

**Partie 2 : Les Files d’attentes**

1. Implémenter une file d'attente d'entiers en utilisant un tableau File(-1 : 100), où File(-1) est

utilisée pour indiquer la tête, File(0) pour indiquer la queue et File(1:100) contient les éléments

de la file d'attente. Comment initialiser la file vide ? Traduire les opérations du modèle.

2. Comment implémenter une file d'attente où chaque élément contient un nombre variable

d'entiers.

3. Une file d'attente avec priorité est une file d'attente dans laquelle l'opération de défilement

récupère l'élément le plus prioritaire. Définir le modèle et l'implémenter.

**Exercice 03 :** Un polynôme peut être représenté par une Llc. Dire comment. Ecrire les

algorithmes suivants :

1. calcul du polynôme en un point x donné.

2. dérivé d'un polynôme.

3. somme de deux polynômes.

4. produit de deux polynômes.