**Exercices algorithme**

**Exercice 1:**

Écrire un module permettant de faire le tri d'un tableau T de type TAB et de taille n, avec la méthode de tri par sélection.

Algorithm TriSelection

Var

T=tab[n] :int

i,j,posmin,Tmp :int

Begin

For i in range(n-1)

Posmin=i

For j in range(i+1,n)

If T[j]<T[posmin] then

Posmin=j

end

If posmin<> T[i]

Tmp=T[i]

T[i]=T[posmin]

T[posmin]=Tmp

end

End

End TriSelection

**Exercice 2 :**

Def sontInvOuOpp(a,b)

If (a+b) then

Retourn 0

else (a\*b) then

Retourn 1

End

End sontInvOuOpp

Def existeInvOuOppConsecutifs(T :tab)

N=len(t)

For i in range(n-1)

if (sontInvOuOpp(t[i],t[i+1]==1) then

return true

else

return false

end

end

def existeInvOuOpp(T : tab)

For i in range(n-1)

For j in range(n-1)

if (sontInvOuOpp(t[i],t[j]==1) then

return true

else

return false

end

end

def nbInvOuOpp(T :tab)

nb=0

For i in range(n-1)

For j in range(n-1)

if (sontInvOuOpp(t[i],t[j]==1) then

return nb=+1

return nb

**Exercice 3 :**

**Algorithme Tri\_Selection**  
**local**:   m, i , j , **n**, temp Î  Entiers naturels  
**Entrée** : Tab Î  Tableau d'Entiers naturels de **1** à **n** éléments  
**Sortie** : Tab Î  Tableau d'Entiers naturels de **1** à **n** éléments

**début**  
**pour** i **de** 1**jusquà n-1faire***// recommence une sous-suite*  
  m ¬i ; *// i est l'indice de l'élément frontière Tab[ i ]*  
  **pour** j **de** i+1 **jusquà n faire***// liste non-triée : (****a****i+1,****a****2, ... ,****a****n)*  
   **si** Tab[ j ] <  Tab[ m ] **alors** *//****a****j est le nouveau minimum partiel*  
     m ¬j ;  
     temp ¬ Tab[ m ] ;  
     Tab[ m ] ¬ Tab[ i ] ;  
     Tab[ i ]  ¬ temp *//on échange les positions de****a****i et de****a****j*  
m ¬i   
   **Fsi**  
**fpour**  
**fpour**  
**Fin Tri\_Selection**

**procédure** tri\_insertion(**tableau** T)

**pour** i **de** 1 **à** taille(T) - 1

*# mémoriser T[i] dans x*

x ← T[i]

*# décaler les éléments T[0]..T[i-1] qui sont plus grands que x, en partant de T[i-1]*

j ← i

**tant que** j > 0 **et** T[j - 1] > x

T[j] ← T[j - 1]

j ← j - 1

# placer x dans le "trou" laissé par le décalage

T[j] ← x

PROCEDURE tri\_bulle ( TABLEAU a[1:n])

passage ← 0

REPETER

permut ← FAUX

POUR i VARIANT DE 1 A n - 1 - passage FAIRE

SI a[i] > a[i+1] ALORS

echanger a[i] ET a[i+1]

permut ← VRAI

FIN SI

FIN POUR

passage ← passage + 1

TANT QUE permut = VRAI

-----perd de temp et de stockage de données

Cet algorithme n'est pas adapté aux grands ensembles de données car sa complexité temporelle moyenne et dans le pire des cas est assez élevée.

**Exercice 4 :**

**Recherche du maximum**

Def Maximum(A)

max ← A[1]

pour i ← 2 `a n faire

si max ¡ A[i] alors

max ← A[i]

renvoyer max

2/

n – 1

**Exercice 5 :**

**Recherche du deuxième plus grand  élément**

Deuxieme-Plus-Grand (A)

rangmax ← 1

pour i ← 2 a n faire

si A[rang max] <> A[i] alors

rangmax ←i

si rangmax <> 1 alors

rangsecond ← 1

sinon rang second ← 2

pour i ← 2 `a n faire si i <> rang max et A[rang second] <>A[i] alors

rangsecond ← i

renvoyer A[rang second]

2/ 2n−3

**Exercice 6 :**

**Recherche du maximum et du minimum**

Maximum-et-Minimum(A)

max ← A[1]

pour i ← 2 a n faire

si max ¡ A[i] alors

max ← A[i]

min ← A[1]

pour i ← 2 a n faire

si min in A[i] alors

min ← A[i]

renvoyer max et min

2/ 2n − 2

**Exercice 7 :**

long pgcd (long a, long b) {

long r;

while (b > 0) {

r = a % b;

a = b;

b = r;

}

return a;

}

2/ , *rn*