Exercice 1 : Questions de cours (04 points)

1) **Toutes les métaheuristiques sont itératives, paramétrables, polynomiales, approchées……..1**
2) **Une méthode constructive : ACO ; une méthode transformative : AG , PSO , BCO. ……..1**
3) **L'ajustement des paramètres sert à améliorer l'efficacité de l'algorithme en termes de qualité de solution et temps d'exécution. ……..0.5**
4) **……..1.5**

	Algorithme génétique	Programmation génétique
Une solution	Une chaine (souvent binaire)	Un programme (en lisp)
Structure d'une solution	Linéaire	Arborescente
Longueur d'une solution	Fixe	Variable

Exercice 2 (08 points)

1) **a) Sélection par roulette ……..1.5**

Individu	f(x)	Mise à l'échelle de f f(x)+6	P(x)
(1,0)	2	8	1/3
(5,1)	9	15	5/8
(2,3)	-5	1	1/24
		24	

b) Sélection par rang ……..1.5

Individu	f(x)	Rang de x	P(x)
(1,0)	2	2	1/3
(5,1)	9	3	1/2
(2,3)	-5	1	1/6
		6	

2) **Une solution est un couple d'entiers compris entre -30 et +30, or $2^4<30<2^5$; il faut alors 5 bits +1 bit du signe pour représenter chaque composante, il faut donc 12 bits pour représenter un individu. ……1.5**

3) **a) Appliquer cet opérateur aux individus (5,1) et (2,3) : ……..1.5**

(5,1) = 000101000001 — **Croisement** → **000110000011 = (6,3)**
(2,3) = 000010000011 → **000001000001 = (1,1)**

b) Algorithme : ……..1.5

```
Void croisement() {
for (i=0 ; i < pop_size-1 ; i+=2) {
    r = rand() ;
    if (r <= pc )
       {  ind = int (rand()*12 );
          for ( j = ind ; j< n ; j++)
             { Temp = Pop[i][j] ;
               Pop[i][j] = Pop[i+1][j] ;
               Pop[i+1][j] = temp ; }}}
```

// les conversions binaire-décimal peuvent se faire avant le croisement ou bien avant l'évaluation.
Complexité = O(n*pop_size/2) ……..0.5

Exercice 3 (08 points)

1) **Nombre de solutions candidates : (n-1) !/2……..1**
2) **Solutions candidates et leurs longueurs respectives : ……..1.5**

x	Longueur(x)
x1=1234	5+1+6+4=16
x2=1243	5+3+6+2=16
x3=1324	2+1+3+4=10

3) Algorithme retournant la longueur d'une solution :

```
int longueur(int i) {
int s=0 ;
for(j=0 ; j<n-1 ; j++)
        s+=d[pop[i][j]] [pop[i][j+1]]
return s+ d[pop[i][k-1]] [pop[i][0]]  }
```

……..1.5

Complexité O(n) ……..0.5

4) Matrice Tau……..1.5

Tau[1][2]= ρ*τ0 +1/longueur(x1) + 1/ longueur(x2) = 2*0.8+1/16+1/16 = 1.725
Tau[1][3]= ρ*τ0 +1/longueur(x3) + 1/ longueur(x2) = 2*0.8+1/10+1/16 = 1.7625
Tau[1][4]= ρ*τ0 +1/longueur(x1) + 1/ longueur(x3) = 2*0.8+1/10+1/16 = 1.7625
Tau[2][3]= ρ*τ0 +1/longueur(x1) + 1/ longueur(x3) = 2*0.8+1/10+1/16 = 1.7625
Tau[2][4]= ρ*τ0 +1/longueur(x2) + 1/ longueur(x3) = 2*0.8+1/10+1/16 = 1.7625
Tau[3][4]= ρ*τ0 +1/longueur(x1) + 1/ longueur(x2) = 2*0.8+1/16+1/16 = 1.725

0	**1.725**	**1.7625**	**1.7625**
1.725	**0**	**1.7625**	**1.7625**
1.7625	**1.7625**	**0**	**1.725**
1.7625	**1.7625**	**1.725**	**0**

5) Algorithme de mise à jour de la matrice Tau : ……..1.5

```
Void MAJDelta(int n ; int[] X) {
    L=longueur(x) ;
    for(i=0 ; i<n-2 ; i++)
       Delta[x[i]][ x[i+1]] += Q/L ;
    Delta[x[n-1]][ x[0]] += Q/L ;
Void MAJTAU(int n ; float[] Tau) {
    for(i=0 ; i<n-1 ; i++)
        for(j=0 ; j<n-1 ; j++)
            Tau[i][j] = ro*Tau[i][j]+Delta[i][j] ;
```

Complexité $O(n^2)$ ……..0.5