Exercice 1 :

1) Somme_Multiple_3ou5(n:entier):entier

```
Début
        i<-1
        somme<-0
        tant que (i<=n) faire
                si i mod 3 =0 ou i mod 5 =0
                        somme<-somme+i
                i<-i+1
        fin tant que
        retourner somme
Fin
```

2) A chaque itération : 2 divisions ; 2 additions . Au total 4 opérations arithmétiques par itération. L'algorithme fera n itérations au total. Donc l'algorithme fera au plus 4n opérations arithmétiques pour faire la somme des entiers multiples de 3 ou 5 inférieurs à n.

Pour améliorer la compléxité, on peut essayer d' écrire le problème sous forme mathématique : somme de tous les multiples de 3 inférieur à n d'y ajouter la somme de tous les multiples de 5 et d'y retrancher tous les multiples de 15. On a :

$$\text{Somme}= [n/3]\Sigma_{i=1}\, 3i + [n/5]\Sigma_{i=1}\, 5i - [n/15]\Sigma\, 15i$$

$$\text{Somme}= 3\,[n/3]\Sigma_{i=1}\, i + 5\,[n/5]\Sigma_{i=1}\, i - 15\,[n/15]\Sigma\, i$$

L'algorithme aura donc un nombre constant d'opérations quelle que soit la valeur de n.

Exercice 2 :

1) 1.1 Algo A : 1 affectation par itération. L'algorithme fait n/2 itérations. 1 affectation pour l'initialisation. Au total on a, n/2 +1 affectations.

1.2 Algo B : 2 affectations par itération. L'algorithme fait n/2 itérations. Il y a 2 affectations pour l'initialisation. Au total on a, 2* n/2 +2=n+2 affectations.

1.3 Algo C : 1 affectation pour l'initialisation.

1 affectation par itération de la boucle j. La boucle sur j est répétée n fois. Au total on a, n affectations.

2 affectations + les n affectations de la boucle sur j par itération de la boucle sur i. La boucle sur i est répétée n fois. Au total pour la boucle sur i, cela fait n*(n+2).

Le total fait donc n*(n+2)+1 affectations.

1.4 Algo D : 1 affectation pour l'initialisation.

2 affectations + les affectations de la boucle sur j par itération de la boucle sur i. LA boucle sur i est répétée n fois.

La boucle j fera 1 affectation par itération. Cette boucle sera éxécutée n-i fois. Ce nombre dépend de la valeur dans la boucle i. Elle sera donc éxécutée n-2 fois au premier passage puis n-3fois,...1fois. Au total cela fera donc (n-2)+(n-3)+...+1=((n-2)(n-1))/2 affectations.

On arrive à un total de 1+2n+((n-2)(n-1))/2 affectations.

1.5 Algo E : 1affectation pour l'initialisation.

1 affectationpar itération de la boucle. Cette boucle sera éxécutée min(m ,n) fois.

On arrive à un total de 1+min(m,n) affectations.

1.6 Algo F : 2 affectations pour l'initialisation.

1 affectation par itération de la boucle. Cette boucle sera éxécutée max(m,n) fois.

On arrive à un total de 1+max(m,n) affectations.

1.7 Algo G: 2 affectations pour l'initialisation.

1 affectation par itération de la boucle. Cette boucle sera éxécutée (m+n) fois.

On arrive à un total de 2+(m+n) affectations.

1.8 Algo H: 2 affectations pour l'initialisation.
1 affectation par itération de la boucle. Cette boucle sera éxécutée (m*n) fois.
On arrive à un total de 2+(m*n) affectations.
2) Les algorithmes A,B,E,F,G sont des algorithmes linéaires. Les algorithmes C,D,H sont des algorithmes quadratiques.

Exercice 3 :

1) L'algorithme calcule 2^n-1. Si on calcule les premières itérations on obtient les valeurs suivantes pour r : 1,3,7,15,...

2) Cet algorithme calcule aussi 2^n-1.

Exercice 4 :

1) Minimum(t:tableau):entier

```
Début
	i<-0
	ind_min<-0
	tant que (i<t.n) faire
		si (t[i]<t[ind_min])
			ind_min<-1
		fin si
		i<-i+1
	fin tant que
	retourner t[ind_min]
Fin
```

2) A chaque itération il y a une comparaison entre les éléments du tableau. L'algorithme fait t.n itérations. Au total il y a donc t.n comparaisons.

3) Pour modifier l'algorithme, plusieurs façon de le faire :
- -soit on change juste le < par un >
- -soit on utilise l'algorithme Minimum sur le tableau -t au lieu du tableau t.

4) Un algorithme qui recherche à la fois le minimum et le maximum du tableau pourrait ressembler à :

```
Min_Max(t:tableau):2 entiers
Début
	min<-Minimum(t)
	max<-Maximum(t)
	retourner min,max
Fin
```

5) L'idée d'un algorithme qui effectue 3n/2 comparaisons pour trouver à la fois le minimum et le maximum est de regrouper les éléments par paire. Pour une paire donnée, on va donc comparer la plus petite des 2 valeurs à la valeur minimale connue jusqu'à présent et on fera pareil entre la plus grande des 2 valeurs et la valeur maximale connue. Ce qui revient à dire que pour chaque paire, on fait une comparaison entre les éléments de la paire pour connaitre la plus grande et la plus petite des 2 valeurs, ensuite on fait les comparaisons avec le minimum et le maximum. Ce qui au total fait 3 comparaisons par paire d'éléments et il y au total n/2 paires d'éléments d'où le résultat.
6) 2_plus_grandes_valeurs(t:tableau):2 entiers

```
Début
        i<-0
        ind_max1<-0
        tant que (i<t.n) faire
                si (t[i]<t[ind_max1])
                        ind_max1<-i
                fin si
                i<-i+1
        fin tant que
        si ind_max1=0 alors
                ind_max2=1
        sinon
                ind_max2=0
        fin si
        i<-0
        tant que (i<t.n et i!=ind_max1) faire
                si (t[i]<t[ind_max2])
                        ind_max2<-i
                fin si
                i<-i+1
        fin tant que
        retourner t[ind_max1] et t[ind_max2]
Fin
```

Cet algorithme coûte en nombre de comparaisons :
-première boucle : n comparaisons
-deuxième boucle : n-1 comparaisons
Au total, cela donne 2n-1 comparaisons.

7) Il s’agit ici de modéliser la recherche de la plus grande valeur comme lors d’un tournoi de tennis par exemple. Voici un exemple sur 8 entiers, pour trouver l’élément maximum. Pour trouver le deuxième plus grand il est nécessairement parmi les éléments marqués en rouge sur la figure ci-dessous, c’est-à-dire tous les éléments qui ont été comparés à la plus grande valeur. Dans notre cas, il va donc falloir en plus des 7 comparaisons représentées sur la figure, comparer 3 et 5 puis 5 et 7. Ce qui fera bien 9 comparaisons en tout.