Exercice 5 :

1) Il y a deux sous-tableau de somme maximale : 1,4,-5,5,-1,1,4 et 5,-1,1,4 dont la somme vaut 9.

2) Un algorithme qui calcule la somme des éléments d'un tableau entre les indices g et d est :

```
sous_somme(t:tableau,g:entier,d:entier):entier
Début
	i<-g
	somme<-0
	tant que (i<=d) faire
		somme<-somme+t[i]
		i<-i+1
	fin tant que
	retourner somme
Fin
```

3) Un algorithme qui retourne la somme maximale parmi tous les couples possibles est le suivant :

```
somme_max(t:tableau):entier
Début
	max<-0
	pour l de 1 à n faire
		pour g de 1 à n-l+1 faire
			somme<-sous_somme(t,g,g+l-1)
			si (max<somme)
				max<-somme
			fin si
		fin pour
	fin pour
	retourner max
Fin
```

Exercice 6 :

On suppose que tous les tableaux représentant des polynômessont de taille n+1.

```
1) somme_polynome(p:tableau,q:tableau):tableau
Début
	Variables locales : i:entier R:tableau

	pour i de 0 à n faire
		R[i]<-p[i]+q[i]
	fin pour
	retourner R
Fin
```

```
2) derive_polynome(p:tableau):tableau
Début
	Variables locales : i:entier R:tableau
	pour i de 1 à n faire
		R[i-1]<-p[i]*i
	fin pour
```

```
        R[n]<-0
        retourner R
Fin
```

3) Soit $p = \sum_{k=0}^{n} a_k X^k$ et $q = \sum_{k=0}^{n} b_k X^k$ alors leur produit vaut $r = \sum_{k=0}^{n+n} (\sum_{i=0}^{k} a_i b_{k-i}) x^k$
L'algorithme se déduit directement de la formule

```
produit_polynome(p:tableau,q:tableau):tableau
Début
        Variables locales : i:entier R:tableau
        pour k de 0 à 2n faire
                r[k]<-0
                pour i de 0 à k faire
                        r[k]<-r[k]+p[i]*q[k-i]
                fin pour
        fin pour
        retourner R
Fin
```

Exercice 7 :

1) Si on prend l'entier 4563, l'algorithme doit nous renvoyer 4. Il faut faire ici des divisions successives par 10 jusqu'à obtenir un reste plus petit que 10 et compter le nombre de divisions que l'on fait. La taille sera le nombre de divisions effectuées+1.

```
taille_entier(n:entier):entier
Début
        Variables locales : r,taille:entier
        taille<-0
        r<-n
        tant que (r>10) faire
                r=r/10
                taille<-taille+1
        fin tant que
        retourner taille+1
Fin
```

```
taille_entier(n:entier):entier
Début
        Variables locales : r,taille:entier
        si (n<10)
                retourner 1
        sinon
                retourner taille_entier(n/10)+1
        fin si
Fin
```

2) L'addition de deux entiers est faite en additionnant, les chiffres des unités entre eux et ensuite des dizaines, puis des centaines, etc. Le nombre d'opérations élémentaires dépend donc de la taille des 2 entiers et plus précisément de la taille du plus grand des 2 entiers.

Exercice 8 :

1° Un exemple de tableau unimodal est : 1 3 5 7 10 6 4 2
Les éléments compris entre la position 0 dans le tableau et la position 4 sont croissants puis de la position 5 à 7 les éléments sont en ordre décroissant.
Dans l'algorithme suivant, t est le tableau, n est sa taille. La variable croissant vaut 1 quand le tableau est croissant et 0 sinon.

```
Verif_unimodal(t:tableau,n:entier):booléen
Début
        Variables locales : croissant:entier
        croissant<-1
        pour i de 0 à n-2 faire
                si croissant=1 et t[i]<=t[i+1]
                        croissant=0
                si croissant=0 et t[i]>=t[i+1]
                        retourner faux
        fin pour
        retourner vrai
Fin
```

2) Pour atteindre une complexité en O(log n) il faut couper le tableau pour n'explorer qu'une partie du tableau, comme on le fait avec une recherche dichotomique. Ici le principe va être le suivant : — pour un tableau a un seul élément, l'élément maximal est cet élément. — Dans un tableau unimodal dont les éléments sont compris entre les indices g et d. On va calculer l'indice m, au milieu de ce sous-tableau. Si t[m] < t[m + 1], alors on va chercher l'élément maximal entre les indices m + 1 et d, sinon on va chercher entre les indices g et m. L'algorithme s'écrit donc :

```
Max_unimodal(t:tableau,g:entier,d:entier):booléen
Début
        si(g>d)
                m<-(g+d)/2
                si (t[m]<t[m+1])
                        retourner Max_unimodal(t,m+1,d)
                sinon
                        retourner Max_unimodal(t,g,m)
                fin si
        sinon
                retourner t[g]
        fin si
Fin
```

3) Une synthèse pour la recherche du maximum et la recherche d'un élément dans différents tableaux de taille n.

Recherche d'un élément : tableau quelconque->O(n) tableau unimodal->O(n) tableau trié->O(log n)
Recherche du maximum : tableau qeulconque->O(n) tableau unimodal->O(log n) tableau trié->O(1)

Exercice 9 :

1) Supposons que nous ayons un tableau T en entrée et que nous voulions créer un tableau S en sortie qui contient les éléments de T sans les doublons. L'algorithme consiste à parcourir le tableau T et pour chaque élément on va vérifier s'il est déjà dans le tableau S, si oui on passe à l'élément

suivant sinon on insère l'élément dans le tableau S et on passe à l'élément suivant dans le tableau T. Ce qui conduit à une complexité de $O(n^2)$.
2) Si le tableau est trié, c'est plus facile car les doublons se suivent dans le tableau, il suffit de parcourir le tableau T, et de rajouter les éléments de T dans S que si le dernier élément inséré dans S est différent de l'élement de T à insérer. On a une complexité en $O(n)$.

3) Il est donc plus efficace de trier le tableau avec un tri en $O(n \log n)$ puis d'appliquer l'algorithme de la question 2 que d'appliquer l'algorithme de la question 1.