EXERCICE 1

1) *Algorithme naïf pour recherche le plus grand et le deuxième plus grand*

```
Max1max2()
        {
          Mx1 = T[0] ;
        For (i=1 ; i<n ; i++)
              If (T[i] >max1)
                {
                   Max1= T[i];
                  Indmax1=I ;
                }
        If (indmax1 != 0)
                  Max2= T[0];
        Else
                  Max2= T[1];

        For (i=1 ; i<n ; i++)
          If( i!= indmax1)
              If (T[i] >max2)
                    Max2= T[i];
        Return (max1 , max2) ;
       }
```

Complexité (en nombre de comparaisons) : Premier max : n-1 ; Second max : n-2.
Complexité = O(2n-3)

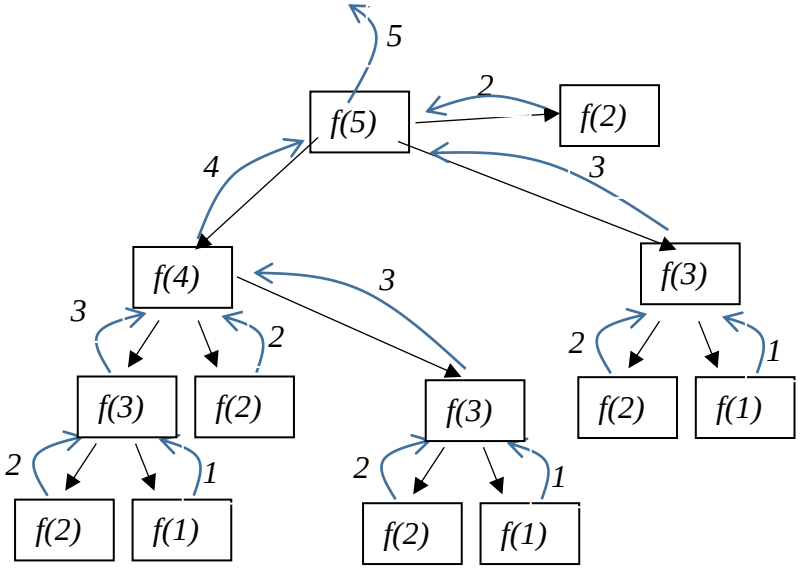
2.a) algorithme de tri :

```
void tri(int()T ; int n)
{
        int i = 0;
        while ( i < n - 1 )
                if ( T[i] <= T[i + 1])
                      i++;
                else
                {
                     Swap(T[i] , T[i + 1] ;
                     if (i > 0)  i-- ;
                     else  i++ ;
                }
}
```

2.b) Complexité :
Au meilleur cas, le tableau est trié, on fait n-1 comparaisons et aucun swap.
Au pire cas, les éléments du tableau sont dans l'ordre décroissant ; pour chaque valeur de l'indice i, on effectue i swaps et 2*i comparaisons ; le nombre de swaps est alors 0+1+2+…+n-1 soit n(n+1)/2.
Le nombre de comparaisons est n(n+1) ; l'algorithme est en O(n^2).

Exercice 2

1) *n=3 f renvoie 3*
   n=4 f renvoie 3
   n=5 f renvoie 5
2)



3) *Cet algo calcule pour un entier n le plus grand nombre premier p inferieur ou égal à n.*
   - *Incovenient : refait des calculs plusieurs fois (engendrés par les appels recusifs).*
   - *Alternative : utilisation d'une table où stocker ces résultats.*
4) *T(n)=(T(n-1)+ T(T(n-1))+ T(T(T(n-1)))+…+T(2))*(1+ T(n-1))*
5) *Une version itérative de cet algorithme en O(n$\sqrt{n}$).*

```
Int f(int n)
{
 While ! (EstPremier(i))
        i--
return i
}
```

```
Booléen EstPremier (entier n)
Entier a ← 1 ;
Si (n mod 2 = 0 ) alors retourner false
Répéter
      a ← a+2 ;
Jusqu'à  (n mod a = 0)  or (a² > n);
Retourner (n mod a ≠ 0) ;
```

Exercice 3

Première procédure

On utilise 3 tableaux :
Lettre [26] contenant les lettres majuscules de l'alphabet ;
Texte [n] contenant n lettres (texte d'entrée) ;
Freq[26] contenant les fréquences des lettres majuscules dans le texte d'entrée.

```
Algorithme
{
for (j=0 ; j<26 ; j++)
     Freq [j] = 0 ;   // initialisation
for (j=0 ; j<26 ; j++)
      for (i=0 ; i<n ; i++)
           If (Texte[i] ==Lettre[j]) Freq [j]++ ;   // ces deux boucles pourraient être inversés.
Return Freq[] ;
}
```

Complexité O(26n).

Deuxième procédure

La deuxième méthode utilise une fonction position qui retourne la position d'une lettre dans l'alphabet (un entier compris entre 0 et 25)

```
Algorithme
{
for (i=0 ; i<26 ; i++)
     Freq [i] = 0 ;   // initialization
for (i=0 ; i<n ; i++)
     Freq[position[Texte[i]]]++ ;
Return Freq[] ;
}
```

Complexité O(n).