# Correction DS

# Exercice1_DS_2014

- Considérons la fonction traiterChaine présentée ci-dessous:

```
void traiterChaine (char* str) {//str est une chaine de caractères
     int i = 0 ;
     int j ;
     int len = strlen (str) ;//strlen retourne la longueur de la chaine str

     while (i < len - 1) {
           if (str [i] == '/' && str [i + 1] == '/') {
              for (j = i + 1, j < len - 2 ; j++)
                  str [j] = str [j + 1] ;
              str [len - 1] = ' ';
           } else
                ++i ;
     }
}
```

# Exercice1_DS_2014

- 1. Expliquer ce que fait la fonction traiterChaine().

✓La fonction traiterChaine() permet d'éliminer toute redondance de '/'.

exp: A//B -> A/B

- 2. Soit str une chaine de longueur n, donner le contenu de str qui maximise le nombre d'opérations exécutées par ce programme.

✓La complexité au pire est obtenu si str contient n '/' : str="/////...//"

# Exercice1_DS_2014

- 3. En déduire la complexité de ce programme au pire des cas à un O près.

```
while (i < len - 1) {
      if (str [i] == '/' && str [i + 1] == '/') {
         for (j = i + 1, j < len - 2 ; j++)
             str [j] = str [j + 1] ;
         str [len - 1] = ' ';
      } else
            ++i ;
}
```

O(n)

Au pire O(n)

Au pire O($n^2$)

# Exercice1_DS_2014

- 3. En déduire la complexité de ce programme au meilleur des cas à un O près.

```
while (i < len - 1) {
      if (str [i] == '/' && str [i + 1] == '/') {
         for (j = i + 1, j < len - 2 ; j++)
             str [j] = str [j + 1] ;
         str [len - 1] = ' ';
      } else
            ++i ;
}
```

O(n)

Au meilleur (pas de '//'), le test est toujours négatif -> boucle for non exécuté

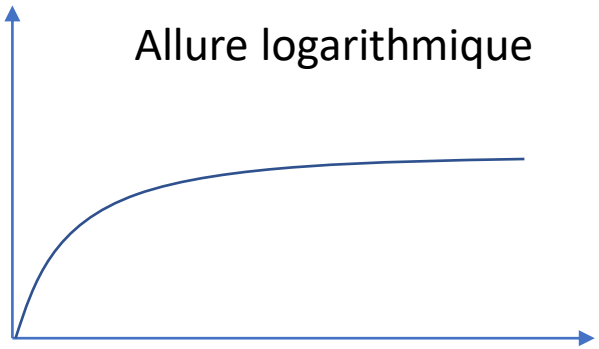
Au meilleur O(n)

# Exercice2_DS_2014

```
void P1(int n) {
int s = 0;

WHILE (n > 0){
      n = n/3;
      s = s + 1 ;
}
```

- La variation de complexité en terme de données:
  - N=9=$3^2$ -> 3 opérations
  - N=27= $3^3$ -> 4 opérations
  - N=81= $3^4$ -> 5 opérations
  - N=243= $3^5$ -> 6 opérations
- O($log_3(n)$)



# Exercice2_DS_2014

```
void P2(int n) {
int s = 0;

FOR(i = 0;i< n; i++) {
     FOR(j= i;j<= n*n;j++){
          s = s + 1;
     }
}
```

O($n$)

O($n^2$)

- O($n^3$)

```
void P3(int n) {
int s = 0;

FOR(i = 0;i< n;i++) {
     FOR(j= 0;j< i*i; j++){
          FOR(k= 0;k<j;k++){
               s = s + 1;
          }
     }
}
```

O($n$)

O($n^2$)

O($n^2$)

- O($n^5$)

# Exercice

- 1. Expliquer ce que fait la fonction Algo().

✓Il s'agit d'un algorithme de tri croissant par insertion:

Insérer l' élément dans sa bonne position dans la partie triée

- 2. En déduire la complexité de ce programme au pire et au meilleur des cas à un O près.

✓Au Pire: O($n^2$)

✓Au meilleur: O($n$)

```
Algo(int A[],int n)
{
int j,i,elt;
for (j=2 ;j<n ;j++)
        {
        elt=A[j] ;  i=j-1 ;
        while(i>0 && A[i]>elt)
        {
                A[i+1]=A[i] ;
                i-- ;
        }
        A[i+1]=elt ;
}
```