# TD n°6 - Complexité

### Exercice 1

Pour les fonctions suivantes, calculer le nombre d'opérations élémentaires qu'elles effectuent. Donner également la complexité asymptotique.

Dans cet exercice on considèrera qu'un printf fait autant d'opérations élémentaires qu'il a de variables à afficher.

```c
int f1(int n){
  for (int i=0; i<n; i+=1){
    printf("%d\n", i);
  }
  for (int j=0; j<n; j+=1){
    printf("%d\n", j);
  }
}
```

```c
int f2(int n){
  for (int i=0; i<n; i+=1){
    for (int j=0; j<n; j+=1){
      printf("%d %d\n", i, j);
    }
  }
}
```

```c
int f3(int n){
  for (int i=0; i<n; i+=1){
    for (int j=i+1; j<n; j+=1){
      printf("%d %d\n", i, j);
    }
  }
}
```

```c
int f4(int n){
  for (int i=2; i<=n-3; i+=1){
    for (int j=i-2; j<i+3; j+=1){
      printf("%d %d\n", i, j);
    }
  }
}
```

### Exercice 2

On veut savoir si un tableau de $n$ éléments contient des doublons.

On admet que résoudre ce problème pour un tableau non trié se fait au mieux avec une complexité en $O(n^2)$. La complexité de la résolution est en $O(n)$ si le tableau est trié. On admet qu'on peut trier une liste avec une complexité en $O(n\log(n))$.

1. Pour détecter les doublons, est-il rentable de commencer par trier le tableau ? Justifiez proprement.
2. Écrire en C la recherche de doublons dans un tableau trié, puis calculez sa complexité.
3. Écrire en C la recherche de doublons dans un tableau non trié, puis calculez sa complexité.
4. Montrer la terminaison et la correction des deux versions de l'algorithme écrites.

### Exercice 3

Pour les complexités, on donnera un ordre de grandeur, avec les notations de Landau, et non pas un décompte précis des opérations élémentaires.

1. Écrire une fonction C `bool egal(int* tab1, int* tab2, int n1, int n2)` qui vérifie si deux tableaux sont égaux, c'est à dire qu'ils ont même taille et même contenu. Donner sa complexité dans le meilleur et dans le pire cas, en temps et en espace.
2. Écrire en C une fonction `int minimum(int* tab, int n)` qui renvoie le plus petit élément d'un tableau. Donner sa complexité dans le meilleur et dans le pire cas, en temps et en espace.
3. Écrire en C une fonction `bool recherche(int* tab, int n, int x)` qui cherche un élément **`x`** dans un tableau **`tab`**. Donner sa complexité dans le meilleur et dans le pire cas, en temps et en espace.

### Exercice 4

Calculer proprement la complexité des fonction récursives suivantes :

```c
int f1(int n){
  if (n==0){return 1;}
  else{return 1+f1(n/2);}
}
```

```c
int f2(int n){
  if (n==0||n==1){return n;}
  else {return f2(n-1);}
}
```

```c
bool f3(int n, int m){
  if (m==1) {return false;}
  else if (n==1){return true;}
  else if (n>m){return f3(n-1,m);}
  else {return f3(n,m-1);}
}
```

```c
int f4(int n){
  if (n==0 || n==1){return n;}
  else {return f4(n-1)+f4(n-2);}
}
```