

UNIVERSITÉ DE SHERBROOKE  
DÉPARTEMENT D'INFORMATIQUE

**IFT 339**

**Série d'exercices - Thème #2 : : Complexité algorithmique**

**Exercice 1 :**

Calculer le nombre maximum d'instructions élémentaires nécessaires pour chaque opération du type Entier, en fonction de la taille de l'entrée  $n$  (nombre de bits de la représentation).

(Correction en classe)

**Exercice 2 :**

Démontrer les relations  $O$  suivantes :

- $2^{10}n = O(n + 2)$
- $n^2 + 2n = O(n^2)$
- $n^2 + 10n + 1 = O(n^2 + 6n)$
- $n^4 = O(n^2 + n^4)$
- $n^4 + 5n^4 + n + 1 = O(n^4)$
- $\log(n)^{k_1} = O(n^{k_2})$  pour toutes constantes positives non nulles  $k_1$  et  $k_2$
- $n \log(n)^2 = O(n^2)$

(Correction en classe)

**Exercice 3 :**

Démontrer que :

- $n^2 + 1 \neq O(n + 2)$

(Correction en classe)

**Exercice 4 :**

Démontrer que :

- Si  $f = O(g)$  et  $g = O(h)$ , alors  $f = O(h)$
- Si  $f_1 = O(g)$  et  $f_2 = O(g)$ , alors  $f_1 + f_2 = O(g)$
- Si  $f_1 = O(g_1)$  et  $f_2 = O(g_2)$ , alors  $f_1 f_2 = O(g_1 g_2)$
- Si  $f = O(g)$ , alors  $hf = O(hg)$

(Correction en classe)

**Exercice 5 :**

Est-il impossible d'avoir  $f \neq O(g)$  et  $g \neq O(f)$  ? Si oui, donner un exemple de deux fonctions qui satisfont cette propriété.

(Correction en classe)

**Exercice 6 :**

Donner la complexité en temps dans le pire des cas des algorithmes ci-dessous :

```
def recherche_lin(tab, x):
    pos = 0
    while (pos < len(tab) and tab[pos] != x):
        pos += 1
    if (pos < len(tab)):
        return pos
    else:
        return -1
```

```
def tri_insertion(tab):
    for i in range(len(tab)):
        elem = tab[i]
        j = i
        while(j >= 0 and tab[j-1] > elem):
            tab[j] = tab[j-1]
            j -= 1
        tab[j] = elem
    return tab
```

```

def tri_selection(tab):
    for i in range(len(tab)):
        pos_min = i
        for j in range(i+1, len(tab)):
            if(tab[j] < tab[pos_min]):
                pos_min = j
        tmp = tab[i]
        tab[i] = tab[pos_min]
        tab[pos_min] = tmp
    return tab

```

```

def fibonacci(n):
    if(n==0):
        return 0
    elif(n==1):
        return 1
    else:
        return fibonacci(n-1) + return fibonacci(n-2)

```

### Exercice 7 :

Décrire un algorithme pour le calcul de la fonction fibonacci(n) de complexité en temps  $O(n)$  et de complexité en espace  $O(n)$  dans le pire des cas.

### Exercice 8 :

Pour chacun des codes suivants, donner la complexité en temps avec la notation  $O$  ?

```

void tri_bulles(vector V){
    for(int i = V.size()-1 ; i > 0; i--){
        for(int j = 0 ; j < i; i++){
            if(V[j+1] < V[j])
                swap(T[j], T[j+1])
        }
    }
}

```

```

void fonction1(int n){
    for(int i = 0; i < n ; i++){
        int t = 1;
        while (t <= n){
            t = t * 2;
            cout << t;
        }
    }
}

```

```

int fonction2(int n){
    if(n == 0)
        return 1;
    else=
        return fonction2(n/2) * n;
}

```

```

void fonction3(int n){
    for(int i = 0; i < n ; i++){
        int k = 0;
        while(k <= log(i)){
            cout<<k;
            k+= 1;
        }
    }
}

```

### Exercice 9 :

Donner un code pour faire la recherche d'un élément dans un tableau d'entier circulaire trié en temps  $O(\log n)$

```

void trouver(Array A, int x){
    ....
}

```