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 :

- $2^{10}n = O(n+2)$
- $n^2 + 2n = O(n^2)$
- $n^2 + 10n + 1 = O(n^2 + 6n)$
- $n^4 = O(n^2 + n^4)$
- $n^4 + 5n^4 + n + 1 = O(n4)$
- $log(n)^{k1} = O(n^{k2})$ pour toutes constantes positives non nulles k_1 et k_2
- $nlog(n)^2 = O(n^2)$

(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 $f1 = O(g)$ et $f2 = O(g)$, alors $f1 + f2 = O(g)$
- Si $f1 = O(g1)$ et $f2 = O(g2)$, alors $f1f2 = O(g1g2)$
- 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 fibonnacci(n) de complexité en temps $O(n)$ et de complexité en espace $O(n)$ dans le pire des cas.