IFT339 - Exercices sur la complexité O et les listes

Exercice 1.

(a) Pour s'échauffer tranquillement, considérez le code suivant.

```
std::vector<int> vec(10);
vec[0] = 0;
int x = vec[0];
x = 10;
std::cout << vec[0] << std::endl;
```

Dites quelle sera la sortie affichée et expliquez pourquoi.

(b) Si l'intention était de manipuler x tout en modifiant le contenu du vecteur, il serait peut-être plus utile d'utiliser `int& x = vec[0];`. Ceci est problématique. Considérez le code suivant.

```
std::vector<int> vec(10);
vec[0] = 0;
int& x = vec[0];

for (int i = 0; i < 10000; ++i)
    vec.push_back(i);
x = 10;
//...
```

Ceci cause parfois un crash quand je l'exécute (et parfois non, en tout cas, il y a un problème!). Expliquez ce qui ne va pas avec ce code.

(c) Proposez une modification à l'implémentation du vector pour éviter ce problème.

Exercice 2.
Considérez une alternative à l'implémentation vector vue en classe. On stocke tab un tableau de tableaux, et la classe a une variable membre subsize qui définit le nombre d'éléments à stocker dans chaque sous-tableau. Donc, tab[0] stocke les subsize premiers éléments, tab[1] les subsize suivants, et ainsi de suite.

```
template <typename TYPE>
class vector{
private:
    TYPE** tab;
    size_t nbelem;
    size_t subsize; //taille des sous-tableaux
    size_t nbtabs;  //taille de tab
    //...autres?
}
```

On veut supporter les fonctions operator[ ], push_back, pop_back. Imaginez comment les implémenter, sans nécessairement le faire. Est-ce que les variables membres suggérées sont suffisantes? Justifiez votre réponse.

Donnez ensuite la complexité en temps de ces fonctions, en fonction de $n = nbelem$ et $m = subsize$. Justifiez vos réponses.

Exercice 3.
Donnez la notation O des fonctions suivantes.

a) $50 + 250n + 3n^3$

b) $0.00001n^2 + 500000n$

c) $2n^2 - 10000n$

d) 10^{100}

e) $2^n + n^{10}$

f) $2n \cdot \log n + 100n$

g) $2n \cdot \log n \cdot \log n + 100n^3$

h) $50n + 2m^2 + 10m$

i) $50n^2 + 10nm + 4n \log m + m^2$

Exercice 4.

Pour chacun des segments de code suivants, donnez leur complexité en temps avec la notation O, en fonction de n (et de m si présent).
a)

```
int* tab = new int[n];
for (int i = 0; i < n; i++)
{
    tab[i] = i;
}
for (int i = n - 2; i >= 0; --i)
{
    tab[i] += tab[i + 1];
}
delete [] tab;
```

b)

```
vector< deque<int> > tab(10);
for (int i = 0; i < tab.size(); ++i)
{
    for (int j = 0; j < n; ++j){
        tab[i].push_front(i + j);
    }
}
```

c) En utilisant fouilleDicho tel que vu en classe.

```
int* tab = new int[n];
for (int i = 0; i < n; i++)
{
    tab[i] = 2 * i;
}

for (int i = 0; i < n; i++)
{
    if (fouilleDicho(tab, n, i))
        cout<<"OUI"<<endl;
    else
        cout<<"NON"<<endl;
}
delete [] tab;
```

d)

```
void init_tab(int* t, int n)
{
    for (int i = 0; i < n; i++)
        t[i] = 0;
}

int** tabs = new int*[n];
for (int i = 0; i < n; i++)
{
    tabs[i] = new int[m];
}

for (int i = 0; i < n; i++)
{
    initTab(tabs[i], m);
}

for (int i = 0; i < n; ++i)
    delete [] tabs[i];
delete [] tabs;
```

e)

```
vector< vector<double> > v(n);
for (int i = 0; i < v.size(); ++i){
    for (int j = 0; j < m; ++j){
        v[i].push_back( rand() );    //temps O(1)
    }
}
for (int i = 0; i < v.size(); ++i){
    std::sort(v[i].begin(), v[i].end());
}
```

Exercice 5.
Écrivez un code ou pseudo-code pour faire la recherche d'un élément dans un tableau circulaire **trié** en temps $O(\log n)$ (voir le devoir 2). Vous avez accès aux pointeurs debut_cap, fin_cap, debut_elem, fin_elem. Vous pouvez utiliser tous les concepts vus en classe.