**Objectifs ASD**

* Se familiariser avec le calcul des complexités.
* Estimer empiriquement la complexité d’une fonction :
  + En comptant des opérations
  + En mesurant le temps d’exécution.
* Comparer graphiquement les complexités théoriques calculées avec celles mesurées empiriquement.
* Identifier le meilleur cas, le pire cas et le cas moyen quand le temps d’exécution dépend de données d’entrée.

**Objectifs C++**

* Réviser la classe vector de la STL.
* Réviser les itérateurs en C++.
* Comprendre les informations qu’on trouve dans la documentation de la STL concernant la complexité des méthodes utilisées.

## Fonction n° 1, Recherche la position d'une valeur dans un vector.

Pour la complexité, on considère le nombre d'itérations.

Tester pour diverses valeurs de val, présentes ou non dans le vector.

@param v vector dans lequel on cherche

@param val valeur à chercher

@return la position de la valeur dans le vector si trouvé, -1 sinon

**size\_t chercherPosition( const vector<int>& v , int val ) {**

**for(size\_t i = 0; i < v.size(); ++i) {**

**if(v.at(i) == val)**

**return i;**

**}**

**return -1;**

**}**

Calcul théorique de la complexité :

Pire des cas :

* La valeur rechercher ne se trouve pas dans le tableau. Nous allons donc le parcourir en entier. Le nombre d’itération sera égal au nombre de case du tableau

Meilleur cas :

* La valeur se trouve à la première position du tableau, le nombre d’itération sera donc égal à 1.

Cas moyen :

Les constantes étant négligeable, la complexité (théorique) de cet algorithme est en O(n).

Tableau de résultat

|  |  |  |
| --- | --- | --- |
| Taille tableau | Ité. Mesurées | Ité. Calculées |
| 10 | 4 | 5 |
| 100 | 49 | 50 |
| 1000 | 498 | 500 |
| 10000 | 5073 | 5000 |
| 100000 | 50100 | 50000 |

Nous pouvons constater que les deux droites sont pratiquement confondues. Nous avons choisi de faire une moyenne sur 1'000 tests ce qui permet une grande précision.

## Fonction n°4, f

Pour la complexité, on considère le nombre d'additions

@param n

@return

**unsigned long long f( unsigned n ) {**

**if( n == 0 ) return 1;**

**return f(n-1) + f(n-1) + f(n-1);**

**}**

La fonction ci-dessus représente un algorithme récursif. Ce dernier va donc s’appeler lui-même en décrémentant la valeur d’appel (n) jusqu’à arriver à n = 0. Ce n’est pas la valeur de retour qui nous intéresse mais le nombre d’addition. Ces dernières sont au nombre de 2 pour chaque « liste » d’appel. Il faudra multiplier à cela le nombre de 3 pour les trois appels recursifs, à la puissance n-1. De plus, le nombre d’addition est parfaitement défini en fonction du nombre n, il ne sert donc a rien de tester plusieurs fois l’algorithme pour une même valeur. Finalement voici le tableau avec les valeurs mesurées et calculées. Elles sont identiques c’est pourquoi nous n’en avons représenté qu’une série dans le graphique-.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N | Somme mesurée | Valeur ajoutée | 3 puissance n-1 (p) | p\*2 | Somme calculée |
| 0 | 0 | 0 |  | 0 | 0 |
| 1 | 2 | 2 | 1 | 2 | 2 |
| 2 | 8 | 6 | 3 | 6 | 8 |
| 3 | 26 | 18 | 9 | 18 | 26 |
| 4 | 80 | 54 | 27 | 54 | 80 |
| 5 | 242 | 162 | 81 | 162 | 242 |
| 6 | 728 | 486 | 243 | 486 | 728 |
| 7 | 2186 | 1458 | 729 | 1458 | 2186 |
| 8 | 6560 | 4374 | 2187 | 4374 | 6560 |
| 9 | 19682 | 13122 | 6561 | 13122 | 19682 |
| 10 | 59048 | 39366 | 19683 | 39366 | 59048 |
| 11 | 177146 | 118098 | 59049 | 118098 | 177146 |
| 12 | 531440 | 354294 | 177147 | 354294 | 531440 |
| 13 | 1594322 | 1062882 | 531441 | 1062882 | 1594322 |
| 14 | 4782968 | 3188646 | 1594323 | 3188646 | 4782968 |
| Mesuré | | | Calculé | | |

La valeur à ajouter à la somme se calculant par , l’algorithme s’apparente à une somme partielle du type

BREF A TERMINER !

**Consignes A enlever**

Ce laboratoire se déroule sur une période de deux semaines. Vous devez rendre un rapport et le code qui vous a servi à mesurer ou estimer les complexités.

Différentes fonctions C++ vous sont fournies dans le fichier .cpp en annexe.

Pour chacune de ces fonctions, estimez **théoriquement** leur complexité. Lorsque ces fonctions utilisent une méthode de la STL, utilisez sa documentation (via cplusplus.com par exemple) pour en connaître la complexité.

Pour chacune de ces fonctions, les commentaires de l’en-tête précisent également sur quelle base en estimer **empiriquement** la complexité. Soit en comptant un nombre d’opérations, soit en mesurant un temps d’exécution.

Pour l’estimation empirique de la complexité, créez un programme de test qui appelle les différentes fonctions avec différentes tailles de jeux de données (N) et différents paramètres suivant les fonctions.