## **ASD1 Laboratoire 1 : Complexité**

# Fonction 1 : chercherPosition

Estimation théorique de la complexité :

Cette fonction est une recherche linéaire et est donc dépendante des données au sein du conteneur. *Afin de simplifier l’estimation, nous partons du principe que chaque élément n’apparaît qu’une seule fois dans le conteneur et que nous ne cherchons pas en dehors de celui-ci. Le principe d’équiprobabilité est donc respecté.*

* **Meilleur des cas :** On trouve en première position ce que l’on cherche => **O(1)**
* **Pire des cas :** Ce que l’on cherche n’existe pas dans le conteneur => **O(n)**
* **Cas moyen :** Nous avons 1/n chance de trouver l’élément à la place 1, 2, 3, …, n. Nous avons donc la formule : 1/n \* (1+2+3+…+n) => 1/n \* (n(n+1))/2 => (n+1) / 2 => **O(n)**

Tableau de mesures :

Représente le nombre d’itérations faites pour N éléments de manière théorique, puis empirique.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Nb éléments** | 500 | 2500 | 4500 | 6500 | 8500 | 10500 |
| **Théorique** | 500 | 2500 | 4500 | 6500 | 8500 | 10500 |
| **Empirique** | 249 | 1243 | 2183 | 3216 | 4111 | 5068 |

Nous pouvons remarquer qu’en pratique, avec cet échantillon de test, nous arrivons à une complexité de **O(n/2)**.

Graphique :

# **Fonction 2 : trier**

Estimation théorique de la complexité :

Cette fonction de tri n’est pas dépendante des données. Nous avons deux boucles imbriquées, la première fait N itérations, puis la seconde en fait N – 1. Nous avons donc n\*(n-1) => n2 – n => **O(n2)**.

Tableau de mesures :

Représente le nombre de comparaison faites pour N éléments de manière théorique, puis empirique.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Nb éléments** | 5 | 25 | 125 | 625 | 3’125 | 15’625 |
| **Théorique** | 25 | 625 | 15’625 | 390’625 | 9'765’625 | 244'140’625 |
| **Empirique** | 20 | 600 | 15’500 | 390’000 | 9’762’500 | 244’125’000 |

Nous pouvons remarquer qu’en pratique, avec cet échantillon de test, nous arrivons à une complexité de **O(n2 - n)**.

Graphique :

# **Fonction 3 : chercherSiContient**

Estimation théorique de la complexité :

Dans cette fonction de recherche dans une liste triée, le nombre d’itérations dépendra de la position de l’élément recherché dans la liste ainsi que du nombre d’éléments se trouvant dans cette liste. Nous pouvons en tirés les complexités suivantes :

* **Meilleur des cas :** L’élément recherché est exactement au milieu du conteneur => **O(1)**
* **Pire des cas :** Ce que l’on cherche n’existe pas dans le conteneur => O(log2(n)) **=> O(log(n))**
* **Cas moyen :** … => O(log2(n)) **=> O(log(n))**

Tableau de mesures :

Représente le nombre d’itérations éxécutées pour N éléments de manière théorique, puis empirique.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Nb éléments** | 1’000 | 5’000 | 10’000 | 500’00 | 100’000 | 1’000’000 |
| **Théorique** | 9 | 12 | 13 | 15 | 16 | 19 |
| **Empirique** | 8 | 11 | 12 | 14 | 15 | 18 |

Nous pouvons remarquer qu’en pratique, avec cet échantillon de test, nous arrivons à une complexité de **O(log2(n) - 1)** et donc **O(log(n))**

Graphique :

# **Fonction 4 : f**

Estimation théorique de la complexité :

C'est une fonction récursive qui s'appelle 3 fois et effectue 2 additions à chaque appelle. En développant sous forme d'arbre, on peut voir que l'arbre à une profondeur de N et on se retrouve donc avec une complexité d'**O(3n)**.

Tableau de mesures :

Représente le nombre de comparaison faites pour des valeurs de N de manière théorique, puis empirique.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **N** | 5 | 6 | 7 | 8 | 9 | 10 |
| **Théorique** | 243 | 729 | 2’187 | 6’561 | 19’683 | 59’049 |
| **Empirique** | 242 | 728 | 2’186 | 6’560 | 19’682 | 59’048 |

Nous pouvons remarquer qu’en pratique, avec cet échantillon de test, nous arrivons à une complexité de **O(3n - 1)**.

Graphique :

# **Fonction 5 : g**

Estimation théorique de la complexité :

Dans cette fonction, nous considérons le nombre d’addition afin d’en évaluer la capacité. Nous avons deux boucles imbriquées, la première allant de 0 à n-1, ce qui équivaut à une complexité de O(n), la seconde boucle va de n en redescendant par saut de deux, la complexité de cette boucle est donc de O(log(n)). Pour avoir la complexité de la fonction entière, nous devons donc multiplier ces deux valeurs étant donné que les boucles sont imbriquées et nous obtenons donc la complexité suivante : **O(n\*log(n))**

Tableau de mesures :

Représente le nombre d’additions faites pour des valeurs de N de manière théorique, puis empirique.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Nb éléments** | 16 | 64 | 128 | 512 | 1000 | 5000 |
| **Théorique** | 64 | 384 | 896 | 4608 | 10000 | 65000 |
| **Empirique** | 64 | 384 | 896 | 4608 | 10000 | 65000 |

Nous pouvons observer que les valeurs théoriques et empiriques sont identiques.

Graphique :

*Les valeurs théoriques sont exactement les mêmes que les empiriques*

# **Fonction 6 : random**

Estimation théorique de la complexité :

La fonction « push\_back »[[1]](#footnote-1) possède une complexité constante, **O(1)**. Nous avons une boucle qui execute la fonction « push\_back » N fois. Nous avons donc N \* 1 => **O(n).**

Tableau de mesures :

Représente le temps d’exécution en milliseconde pour N éléments ajouter dans un vecteur vide. Les éléments ajoutés ont une valeur comprise entre 1 et 100.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Nb éléments** | 1000 | 5000 | 10’000 | 20’000 | 40’000 | 100’000 |
| **Temps d’exécution en milliseconde** | 0.049 | 0.225 | 0.410 | 2.102 | 4.186 | 43.265 |

Graphique :

# **Fonction 7 : random2**

Estimation théorique de la complexité :

La fonction « insert[[2]](#footnote-2) » de la classe « vector » à une complexité linéaire par rapport au nombre d’élément ajouté + le nombre d’élément qui doivent être déplacer (donc qui suivent l’élément ajouté). Nous avons une boucle qui fait N itérations avec pour chacun un appel à la fonction « insert ». Nous avons donc : n \* (1 + n - 1) => n \* n => **O(n2)**.

Tableau de mesures :

Représente le temps d’exécution en milliseconde pour N éléments ajouter dans un vecteur vide. Les éléments ajoutés ont une valeur comprise entre 1 et 100.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Nb éléments** | 1000 | 5000 | 10’000 | 20’000 | 40’000 | 100’000 |
| **Temps d’exécution en milliseconde** | 0.122 | 1.077 | 3.558 | 14.269 | 68.829 | 485.012 |

Graphique :

1. <http://www.cplusplus.com/reference/vector/vector/push_back/> [↑](#footnote-ref-1)
2. <http://www.cplusplus.com/reference/vector/vector/insert/> [↑](#footnote-ref-2)