# **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 trie 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 :

Cette fonction de trie 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 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(3^N).

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 :

Cette fonction de trie 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 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** | 0,048808 | 0,225212 | 0,409617 | 2,10165 | 4,18602 | 43,2651 |

Graphique :

# **Fonction 7 : random2**

Estimation théorique de la complexité :

La fonction « insert » 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 on 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)