heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

1 INTRODUCTION

L'objectif de ce laboratoire est de nous familiariser avec les notions de complexité, en évaluant de façon théorique et empirique la complexité de fonctions C++ données.

Pour avoir une mesure correcte des temps d'exécution des fonctions, nous avons fait une moyenne sur 10 tests différents à chaque fois.

2 FONCTIONS

2.1 CHERCHERPOSITION

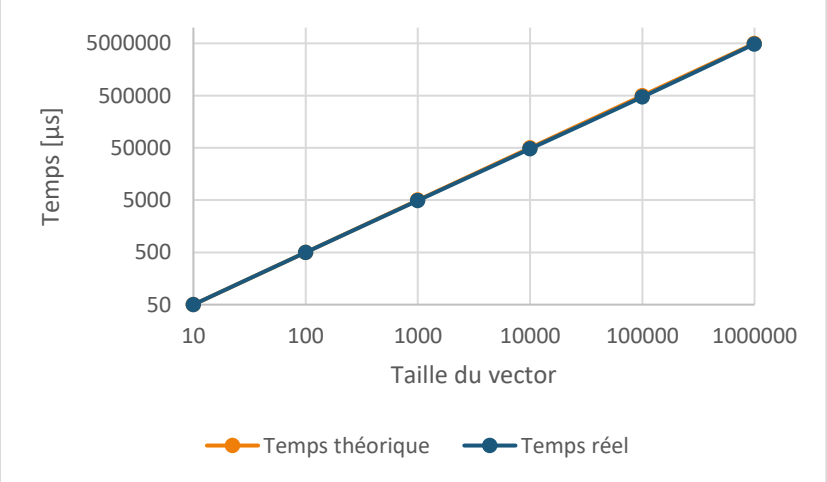
La complexité est en **O(n)**, la fonction est composée uniquement d'une boucle *for* qui parcourt le *vector* jusqu'à trouver la bonne valeur. Cette complexité correspond au pire cas ainsi qu'au cas moyen.

Cette complexité varie selon les données reçues par la fonction. Le meilleur de cas est lorsque la valeur recherchée est à l'entrée du *vector*, ce qui donne une complexité en **O(1)**.

Taille du vector	**10^2**	**10^3**	**10^4**	**10^5**	**10^6**	**10^7**
Temps théorique [µs]	50	500	5000	50000	500000	5000000
Temps réel [µs]	50	499	4909	47938	469997	4862062

Comme nous pouvons le constater, lorsque nous multiplions la taille du tableau d'un facteur 10, le temps augmente proportionnellement par 10. La complexité réelle se rapproche donc de la complexité théorique.

Sur le graphique ci-contre, nous remarquons que les courbes se confondent car il y a très peu de différence entre la complexité théorique et réelle.



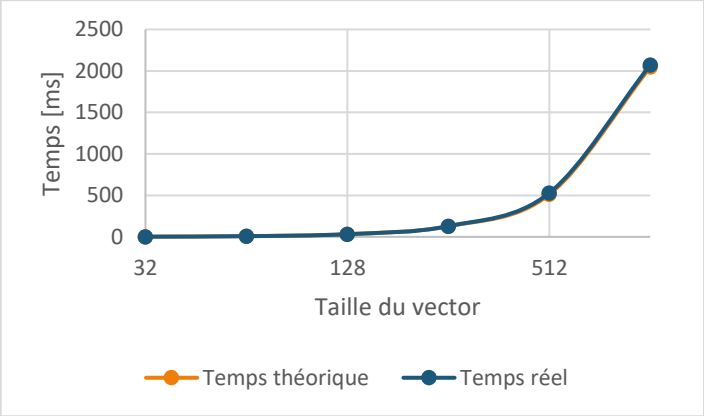
2.2 TRIER

Complexité en ***O(n²)*** car il s'agit de deux boucles *for* imbriquées entre elles. Dans le meilleur des cas, le tableau est déjà trié, la complexité est donc en ***O(1)***. Le cas moyen correspond au pire cas, soit ***O(n²)***.

Taille du vector	**32**	**64**	**128**	**256**	**512**	**1024**
Temps théorique [ms]	2	8	32	128	512	2048
Temps réel [ms]	2.09	8.16	32.48	128.82	529.84	2072.60

Comme nous pouvons le constater, lorsque nous multiplions la taille du tableau d'un facteur 2, le temps augmente proportionnellement par 2^2. La complexité réelle se rapproche donc de la complexité théorique.

Sur le graphique ci-contre, nous remarquons que les courbes se confondent car il y a très peu de différence entre la complexité théorique et réelle.



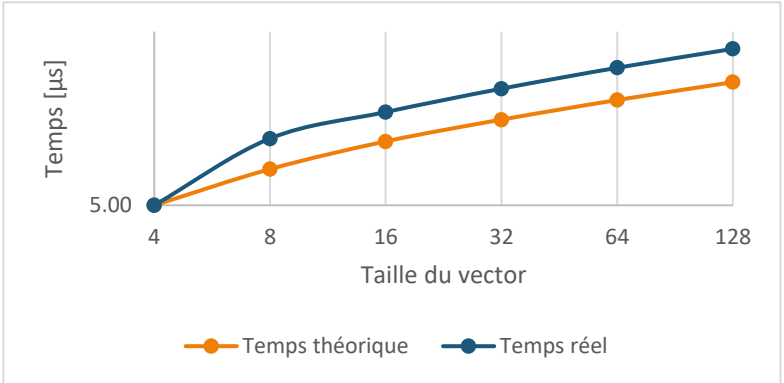
2.3 CHERCHERSICONTIENT

La fonction *chercherSiContient* est une recherche dichotomique et sa complexité est en ***O(log(n))***. Si la valeur recherchée est au milieu du tableau, la complexité sera en ***O(1)***. La complexité du cas moyen est la même que pour le pire cas.

Taille du vector	**4**	**8**	**16**	**32**	**64**	**128**
Temps théorique [µs]	5	6	6.9	7.7	8.5	9.3
Temps réel [µs]	5	7	8	9	10	11

Nous avons décidé de travailler avec des tableaux de petites tailles pour analyser cette fonction car, en effet, le tableau doit être trié avant le calcul, ce qui peut prendre un temps considérable.

Sur le graphique, nous constatons que le temps d'exécution croît rapidement au début pour ensuite se stabiliser.

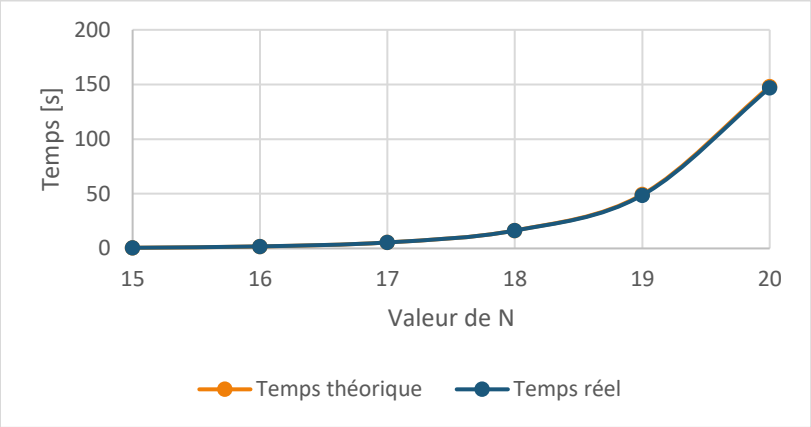


2.4 F

La complexité théorique est en ***O(3ⁿ)***. En effet, pour chaque appel de la fonction avec un entier strictement positif, la fonction sera appelée récursivement 3 fois et ainsi de suite. La complexité est indépendante des données pour cette fonction.

N	15	16	17	18	19	20
Temps théorique [s]	0.61	1.83	5.49	16.47	49.41	148.23
Temps réel [s]	0.61	1.88	5.49	16.36	48.71	146.93

Les mesures de temps obtenues sont représentatives de la complexité de la fonction. En passant de ***n*** à ***n+1***, le temps d'exécution est triplé.



Nous pouvons constater sur le graphique que les temps théoriques et réels se confondent car ceux-ci sont très proches.

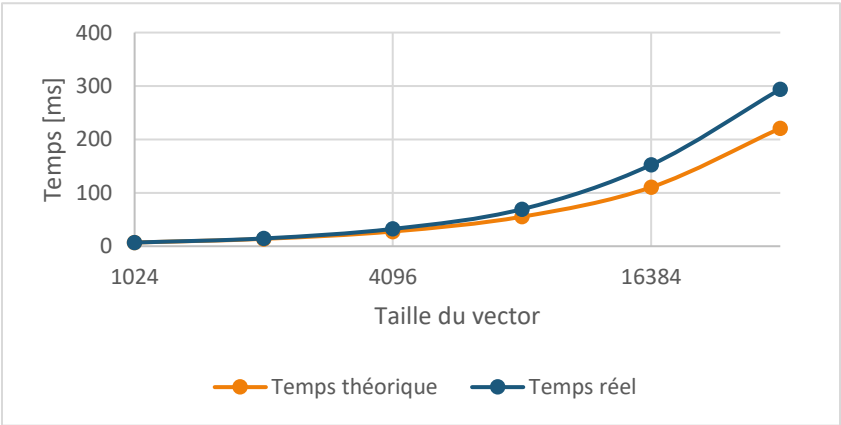
2.5 G

La complexité théorique est en ***O(n*log(n))***. En effet, la fonction est composé d'une boucle *for* ayant une complexité en ***O(log(n))*** (le compteur est divisé par 2 à chaque fin de boucle), imbriquée dans une boucle *for* avec une complexité en ***O(n)***.

Taille du vector	1024	2048	4096	8192	16384	32768
Temps théorique [µs]	6.9	13.8	27.6	55.2	110.4	220.8
Temps réel [µs]	6.9	14.56	32.42	69.35	152.34	319.95

Pour cette fonction, nous constatons une différence entre le temps théorique et réel.

Ceci dit, nous voyons tout de même sur le graphique ci-contre que les complexités théorique et réel ont une courbe semblable.

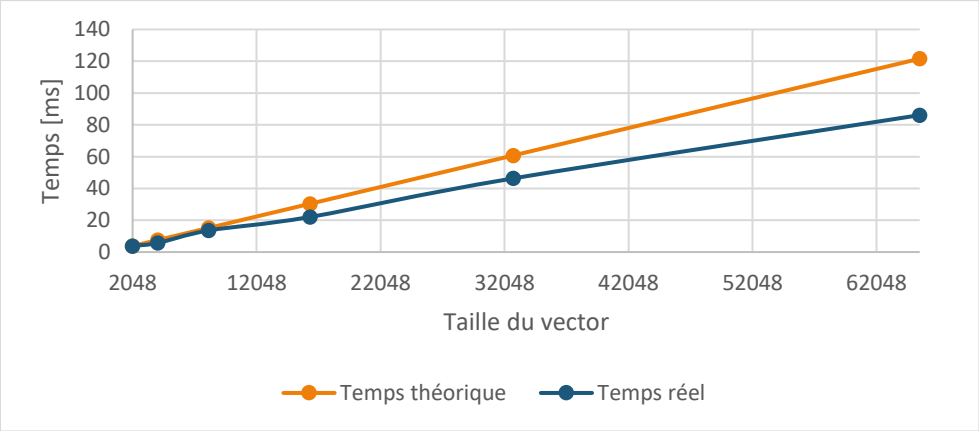


2.6 RANDOM

La fonction est composée d'une simple boucle *for* parcourant en vector en lui assignant des valeurs aléatoires. Sa complexité est donc en ***O(n)*** et celle-ci est constante car indépendante du type de données.

Taille du vector	**2048**	**4096**	**8192**	**16384**	**32768**	**65536**
Temps théorique [ms]	3.8	7.6	15.2	30.4	60.8	121.6
Temps réel [ms]	3.8	5.75	13.58	22.1	46.36	86.1

Nous constatons que le temps réel d'exécution est légèrement inférieur au temps théorique. En travaillant avec des tableaux plus grands cette différence ne serait pas apparue.

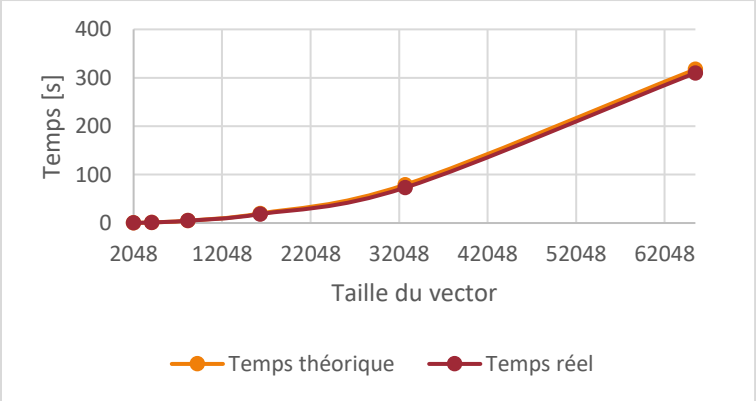


2.7 RANDOM2

La fonction est similaire à la précédente, aussi une seule boucle *for*, mais l'utilisation du *vector::insert* modifie fortement la complexité de celle-ci. En effet, à l'instar de *vector::push_back*, *vector::insert* va parcourir en entier le *vector* avant d'y insérer la valeur désirée. La complexité est donc en ***O(n²)***, soit 2 boucles *for* imbriquées, celle-ci est constante car indépendante du type de données.

Taille du vector	**2048**	**4096**	**8192**	**16384**	**32768**	**65536**
Temps théorique [s]	0.31	1.24	4.96	19.84	79.36	317.44
Temps réel [s]	0.31	1.19	4.81	18.45	73.32	310

Comme pour la fonction *random*, nous constatons que le temps réel est légèrement plus rapide que le temps théorique



3 CONCLUSION

Ce laboratoire nous a permis de nous familiariser avec la complexité par la pratique. Nous avons ainsi pu constater l'impact *considérable* que peux avoir tel ou tel choix de fonction lors de la réalisation d'un logiciel.

L'exemple le plus probant est, pour nous, la différence entre l'utilisation de *vector::push_back* et *vector::insert*. Celui-ci modifie une complexité linéaire en une complexité exponentielle et le temps d'exécution se voit quadruplé lors le volume de données est doublé.