POLYTECH GRENOBLE

G-SCOP

Rapport de stage

Analyseur de complexité algorithmique

Auteur :
Benjamin BESNIER

Tuteur :
Hadrien CAMBAZARD
Référents :
Nadia BRAUNER
Matej STEHLIK

20 juillet 2018

Remerciments

Avant tout développement sur cette expérience professionnelle, il apparaît opportun de commencer ce rapport de stage par des remerciements, à ceux qui m'ont beaucoup appris au cours de ce stage, et même à ceux qui ont eu la gentillesse de faire de ce stage un moment très profitable.
Aussi, je remercie Hadrien CAMBAZARD, mon maître de stage qui m'a formé et accompagné tout au long de cette expérience professionnelle avec beaucoup de patience et de pédagogie. Enfin, je remercie également Nadia BRAUNER ainsi que Matej STEHLIK pour les conseils qu'ils ont pu me prodiguer au cours de ces douze semaines.

Résumé

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus. Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi. Proin porttitor, orci nec nonummy molestie, enim est eleifend mi, non fermentum diam nisl sit amet erat. Duis semper. Duis arcu massa, scelerisque vitae, consequat in, pretium a, enim. Pellentesque congue. Ut in risus volutpat libero pharetra tempor. Cras vestibulum bibendum augue. Praesent egestas leo in pede. Praesent blandit odio eu enim. Pellentesque sed dui ut augue blandit sodales. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae ; Aliquam nibh. Mauris ac mauris sed pede pellentesque fermentum. Maecenas adipiscing ante non diam sodales hendrerit. Ut velit mauris, egestas sed, gravida nec, ornare ut, mi. Aenean ut orci vel massa suscipit pulvinar. Nulla sollicitudin. Fusce varius, ligula non tempus aliquam, nunc turpis ullamcorper nibh, in tempus sapien eros vitae ligula. Pellentesque rhoncus nunc et augue. Integer id felis. Curabitur aliquet pellentesque diam. Integer quis metus vitae elit lobortis egestas. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Morbi vel erat non mauris convallis vehicula. Nulla et sapien. Integer tortor tellus, aliquam faucibus, convallis id, congue eu, quam. Mauris ullamcorper felis vitae erat. Proin feugiat, augue non elementum posuere, metus purus iaculis lectus, et tristique ligula justo vitae magna. Aliquam convallis sollicitudin purus. Praesent aliquam, enim at fermentum mollis, ligula massa adipiscing nisl, ac euismod nibh nisl eu lectus. Fusce vulputate sem at sapien. Vivamus leo. Aliquam euismod libero eu enim. Nulla nec felis sed leo placerat imperdiet. Aenean suscipit nulla in justo. Suspendisse cursus rutrum augue. Nulla tincidunt tincidunt mi. Curabitur iaculis, lorem vel rhoncus faucibus, felis magna fermentum augue, et ultricies lacus lorem varius purus. Curabitur eu amet.

Mots-clés : Performance, complexité, algorithmes.

Table des matières

Chapitre 1

Présentation de G-Scop

Chapitre 2

Présentation du projet

L'étude de la complexité algorithmique tient une place prépondérante lorsqu'on s'intéresse à des programmes devant traité un grand nombre de données. En effet, entre deux programmes ayant la même finalité, il se peut que la différence de temps d'exécution soit comptée en jours. C'est pourquoi, il est important de choisir un algorithme optimal pour chaque programme. Mais bien que la complexité de certains programmes soit bien connue (telle que les tris de tableau), certaines complexités sont plus compliquées à prédire. Le but de ce stage fut de trouver une façon de connaître la complexité temporelle d'algorithmes en se basant seulement sur leurs temps d'exécution.

2.1 Sujet

Comme dit dans la courte introduction, le but de ce stage fut d'essayer de retrouver la complexité des algorithmes en se basant sur leur temps d'exécution. Il n'existe pas, à l'heure actuelle de méthode permettant de connaître la classe de complexité. Dans ce stage le but fut donc de se concentrer sur une méthode d'approche et d'en tirer des conclusions afin de voir si la méthode choisie est une méthode d'approche fonctionnelle ou non. La méthode d'approche choisit, est de procéder par régression linéaire sur le temps d'exécution.

2.1.1 Explications

Le problème est de savoir si on peut déterminer automatiquement la complexité temporelle au pire cas d'un algorithme en disposant de son code source et de la possibilité d'effectuer des tests numériques dans un temps limité.

Dans ce travail, on va se restreindre à 6 classes de complexité :

$$log(n), n, nlog(n), n^2, n^2log(n), n^3$$

On propose d'exploiter les temps d'exécution sur des instances de différentes tailles et de procéder par régression linéaire pour identifier la complexité qui semble refléter au mieux l'exécution de l'algorithme. Deux approches sont évaluées dans la suite de ce document.
La première consiste à faire une unique régression $H_\theta(n)$ multi-variable prenant la forme suivante :

$$H_\theta(n) = \theta_0 + \theta_1 log(n) + \theta_2 n + \theta_3 nlog(n) + \theta_4 n^2 + \theta_5 n^2 log(n) + \theta_6 n^3$$

On examine ensuite les coefficients θ_i de la régression pour tenter d'isoler le ou les termes le plus significatifs qui pourraient indiquer la complexité réelle de l'algorithme.
La deuxième consiste à effectuer une régression par classe de complexité donc quatre régressions distinctes :

$$H_\theta^1(n) = \theta_0 + \theta_1 log(n)$$
$$H_\theta^2(n) = \theta_0 + \theta_1 n$$
$$H_\theta^4(n) = \theta_0 + \theta_1 n^2$$
$$H_\theta^6(n) = \theta_0 + \theta_1 n^3$$

Dans cette deuxième méthode, on se passe volontairement de $O(nlog(n))$ et $O(n^2log(n))$ car leurs temps d'exécution sont très proches de $O(n)$ (resp. $O(n^2)$) ce qui les rends très difficiles a distinguer avec cette méthode.

On examine ensuite l'erreur obtenue (la somme des carrés des écarts aux points connus) pour retenir comme complexité réelle de l'algorithme la régression H^k minimisant cette erreur, car on propose de retenir comme complexité la régression donnant l'erreur minimum puisque l'erreur quantifie l'adéquation de la fonction aux temps d'exécution observés.

Une fois, ces deux approches réalisées, et si on obtient des résultats concluant, on va s'intéresser au temps d'exécution minimal nécessaire afin d'obtenir des résultats concluant à nouveau.

On formule les hypothèses suivantes sur les méthodes d'approches :

Hypothèse 1 Le coefficient θ_i le plus grand de $H_\theta(n)$ correspond à la complexité temporelle de l'algorithme.

Hypothèse 2 La régression $H_\theta^k(n)$ ayant l'erreur la plus faible correspond à la complexité temporelle de l'algorithme.

2.1.2 Protocole

- Dans un premier temps on exécute l'algorithme pour toutes les tailles de tableau en mesurant le temps d'exécution à chaque fois

- Une fois le temps mesuré, on construit le tableau des features.

$log(n)$	n	$nlog(n)$	n^2	$n^2log(n)$	n^3	Temps CPU
9.21	10000	92103.40	$1.0 * 10^8$	$9.0 * 10^8$	$1.0 * 10^{12}$	$56ms$
9.90	20000	198068.75	$4.0 * 10^8$	$4.0 * 10^9$	$8.0 * 10^{12}$	$187ms$
10.31	30000	309268.58	$9.0 * 10^8$	$9.0 * 10^9$	$2.7 * 10^{13}$	$491ms$
10.60	40000	423865.39	$1.6 * 10^8$	$2.0 * 10^{10}$	$6.4 * 10^{13}$	$884ms$
10.82	50000	540988.91	$2.5 * 10^9$	$3.0 * 10^{10}$	$1.3 * 10^{14}$	$1482ms$
11.00	60000	660126.00	$3.6 * 10^9$	$4.0 * 10^{10}$	$2.2 * 10^{14}$	$1612ms$
11.16	70000	780937.54	$4.9 * 10^9$	$5.0 * 10^{10}$	$3.4 * 10^{14}$	$2139ms$
11.29	80000	903182.55	$6.4 * 10^9$	$7 * 10^{10}$	$5.1 * 10^{14}$	$2962ms$
11.41	90000	1026680.85	$8.1 * 10^9$	$9.0 * 10^{10}$	$7.3 * 10^{14}$	$3708ms$
11.51	100000	1151292.55	$1.0 * 10^{10}$	$1.0 * 10^{11}$	$1.0 * 10^{15}$	$4608ms$

FIGURE 2.1 – Exemple de tableau utilisé pour la régression

Les deux approches sont alors réalisées en parallèles, on distinguera a posteriori laquelle est la plus pertinante (si tenté, qu'il y en ai une).

- Dans la première approche, avec la régression $H_\theta(n)$, on va s'intéresser à ses coefficients θ_i afin de trouver le terme permettant de nous donner un indice sur la complexité et ainsi valider l'hypothèse 1.

- Dans la deuxième approche, on effectue une régression par classe de complexité puis on calcule l'erreur obtenue pour chacune d'entre elle. On retient ensuite la régression $H_\theta^k(n)$ ayant l'erreur minimal pour essayer de conclure et ainsi valider l'hypothèse 2.

- On trace le graphique contenant toutes les régressions linéaire ainsi que l'exécution de l'algorithme.

- On extrait un tableau contenant l'erreur résiduelle pour chaque $H_\theta^i(n)$ et le temps d'exécution total.

- Une fois un résultat obtenu, on va chercher à réduire au maximum le temps d'exécution, tout en gardant un taux de bon résultat convenable. Pour réduire ce temps, on va réduire au maximum les tailles des entrées.

2.2 Proof of concept

Dans cette partie, on s'intéresse à des algorithmes bien connu afin de déterminer si notre méthode fonctionne ou non.

2.2.1 Tri à bulles simple $O(n^2)$

Tailles de tableau utilisées : 2500, 5000, 10000, 15000, 20000, 25000, 30000, 35000, 40000
Complexité théorique : $O(n^2)$

Lorsqu'on exécute l'algorithme avec des entrées différentes et que l'on mesure le temps d'exécution, on obtient le graphique suivant :

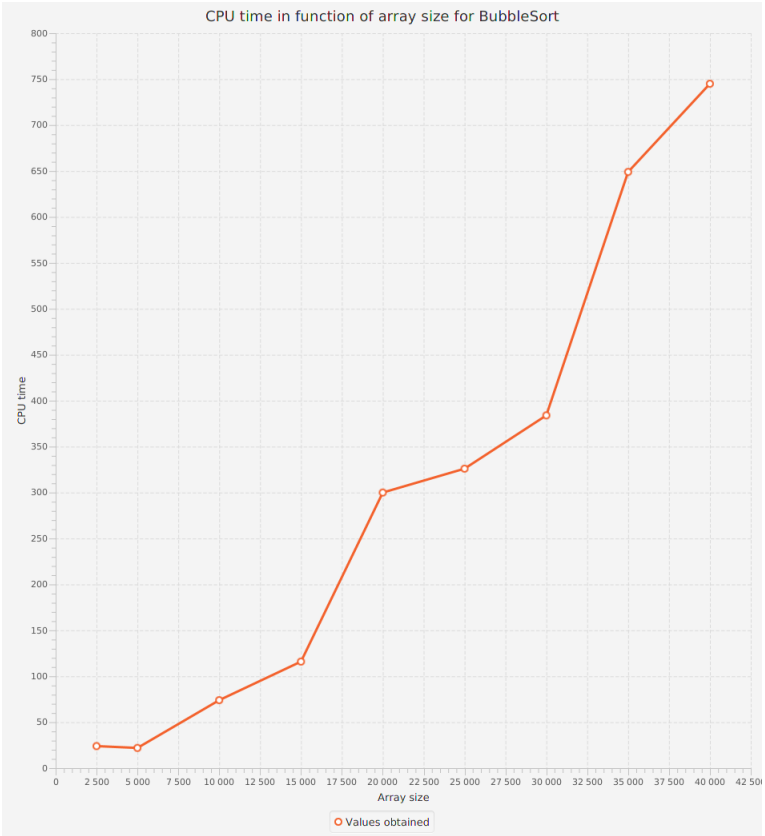

FIGURE 2.2 – Le temps d'exécution de l'algorithme en ms en fonction de la taille du tableau

Le programme procède ensuite à la régression.
On obtient la régression suivant :
$$\mathbf{H(n) = 257498.66 - 35867.89log(n) + 54.14n - 5.30nlog(n) + (7.10 * 10^{-4})n^2} \\ \mathbf{-(5.52 * 10^{-5})n^2log(n) + (1.13 * 10^{-10})n^3}$$

On rappel **l'hypothèse 1** étant : " Le coefficiant θ_i le plus grand de $H_\theta(n)$ correspond à la complexité temporelle de l'algorithme."

Or, dans ce cas, si on choisit de prendre le plus grand θ afin de déterminer la complexité, ici, on obtient une complexité en $O(n)$ ce qui est incorrecte, on doit donc utiliser la deuxième méthode qui consiste a réaliser une régression linéaire par classe de complexité $(H^i_\theta(n))$ et de calculer l'erreur.

On obtient alors les résultats suivants :

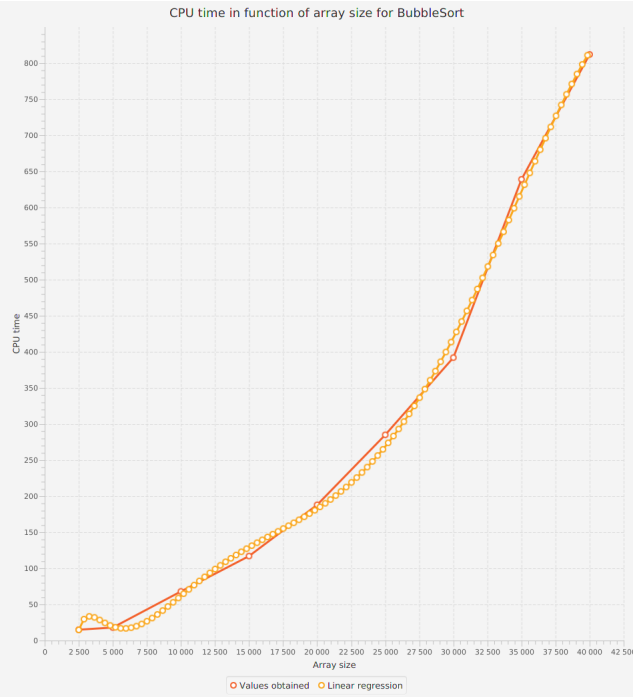

FIGURE 2.3 – Régression linéaire en ms en fonction de la taille des données (n)

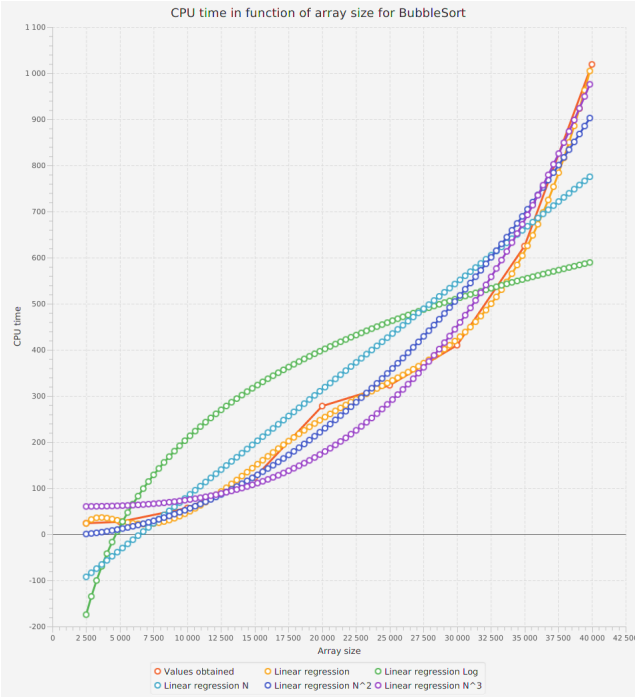

FIGURE 2.4 – Toutes les régressions linéaire en ms en fonction de la taille des données (n)

On obtient les erreurs suivantes :

Régression linéaire	$H_\theta(n)$	$H^1_\theta(n)$	$H^2_\theta(n)$	$H^4_\theta(n)$	$H^6_\theta(n)$
Erreur résiduelle :	0.06	29280.90	9225.85	**146.96**	1772.99
Indicateur sur les erreurs :	#	19824%	6177%	**0**%	1106%

FIGURE 2.5 – Tableau récapitulatif des erreurs résiduelles

L'indicateur de la seconde ligne est calculé comme ceci :

$$\frac{H^k_\theta - H^{min}_\theta}{H^{min}_\theta}$$

Où H^{min}_θ correspond au $H^k_\theta(n)$ ayant la valeur d'erreur la plus petite.
De plus, ici l'indicateur pour $H_\theta(n)$ est # car on ne compare que les $H^k_\theta(n)$ entre eux.

On rappel **l'hypothèse 2** : "La régression $H^k_\theta(n)$ ayant l'erreur la plus faible correspond à la complexité temporelle de l'algorithme."

On observe donc que l'erreur résiduelle la plus faible est obtenue pour $H_\theta(n)$, car celle-ci possède bien plus de variables que les autres, donc elle peut vraiment s'ajuster à la courbe de référence.
Cependant, on observe que l'erreur résiduelle la plus faible étant pour la régression de la forme $\mathbf{H^4_\theta(n) = \theta_0 + \theta_1 n^2}$, on obtient bien une complexité en $\mathbf{O(n^2)}$ comme voulu et **l'hypothèse 2** est vérifiée.

On cherche ensuite à déterminer la véracité (i.e le programme donne la bonne complexité) de nos résultats, ainsi que le temps moyen d'exécution. Pour ce faire on lance plusieurs fois le programme avec

les mêmes entrées et on regarde le nombre de fois ou le H_θ^{min} correspond à celui attendu.

% de bon résultats
99%

FIGURE 2.6 – Taux de réussite

On essaie maintenant de réduire au maximum la taille des entrées (tout en gardant le même nombre d'entrées) afin de réduire le temps d'exécution tout en gardant un taux de bon résultat acceptable.

Taille maximal	Temps d'exécution moyen	% de bon résultats
10000	$0.2s$	90%
25000	$1.1s.$	90%
40000	$2.5s.$	98%
50000	$4.8s.$	66%
100000	$28.2s.$	88%

FIGURE 2.7 – Taux de réussite et temps d'exécution moyen du programme

Pour le tri à bulle, on peut donc réduire la taille maximal du tableau d'entrée à 10000, ce qui permet de réellement réduire le temps d'exécution tout en gardant un taux de bon résultat convenable.

2.2.2 Tri à bulles parasité $O(k_1n + k_2nlog(n) + n^2)$

De la même façon que l'exemple précédent, sauf que cette fois, on parasite le tri à bulles en rajoutant plusieurs tri fusion au milieu (sur une copie du tableau) et des parcours de tableau. Ceci a pour but d'avoir une complexité du type $O(k_1n + k_2nlog(n) + n^2)$ afin de voir si la méthode utilisée donne bien une complexité en $O(n^2)$. On fait ensuite varier les k_1 et k_2.
Ceci a pour but de montrer la robustesse de l'algorithme si l'algorithme a tester n'est pas un algorithme à complexité "simple".
On applique ensuite la même méthode pour déterminé la complexité.
Tailles de tableau utilisées : 2500, 5000, 10000, 15000, 20000, 25000, 30000, 35000, 40000
Complexité théorique : $O(n^2)$

On obtient les graphiques suivants :

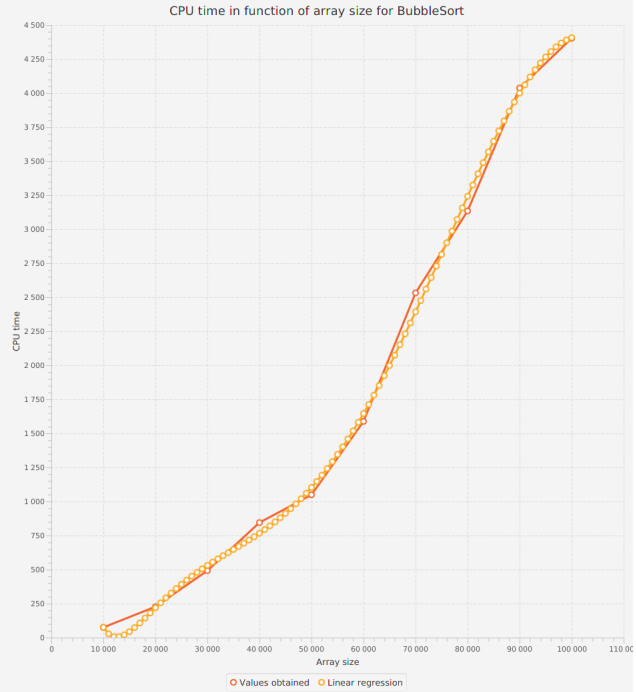

FIGURE 2.8 – Régression linéaire en ms en fonction de la taille des données (n)
Avec $k1 = 10$ et $k2 = 10$

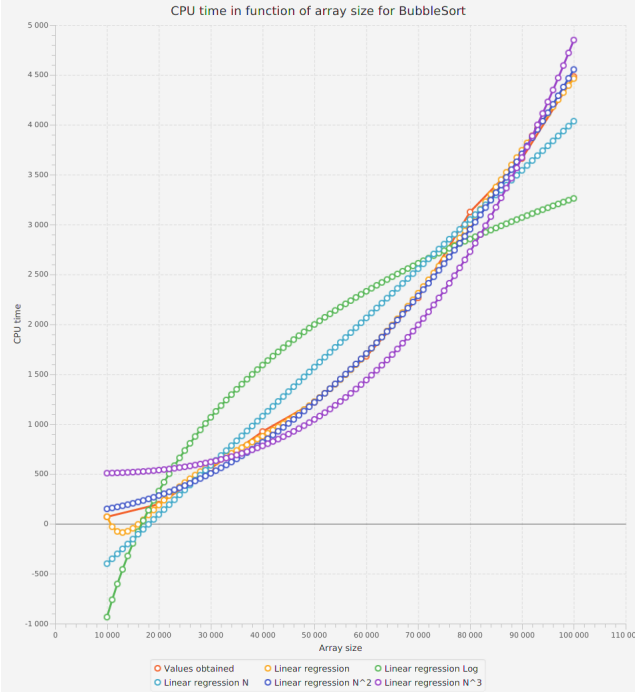

FIGURE 2.9 – Toutes les régressions linéaire en ms en fonction de la taille des données (n)
Avec $k1 = 10$ et $k2 = 10$

on obtient :

$$\mathbf{H(n) = -122433.44 + 21017.43log(n) - 106.00n + 11.96nlog(n)}$$
$$\mathbf{-0.0054n^2 + 4.72 * 10^{-4}n^2log(n) - 3.55 * 10^{-9}n^3}$$

Si on essaie de démontrer **l'hypothèse 1**, on prend alors θ_1, or, si on prend celui-ci, on obtient une complexité en $O(log(n))$ ce qui ne correspond pas à la complexité attendue en $O(n^2)$.

Et les erreurs résiduelles suivantes :

k_1	k_2	$H_\theta^1(n)$	$H_\theta^2(n)$	$H_\theta^4(n)$	$H_\theta^6(n)$
10	10	27491%	6375%	**0**%	4444%
10	100	6438%	2158%	**0**%	30033%
100	10	$1.03 * 10^8$%	$2.83 * 10^7$%	**0**%	$1.21 * 10^7$%
100	100	4406%	735%	**0**%	1080%
1000	10	$6.77 * 10^5$%	$1.86 * 10^5$%	**0**%	66379%
1000	100	40%	78%	**0**%	496%

FIGURE 2.10 – Indicateur sur les erreurs résiduelles

k_1	k_2	Temps exécution moyen	% de bon résultats
10	10	$2.34s.$	96%
10	100	$3.78s.$	81%
100	10	$2.82s.$	94%
100	100	$4.13s.$	76%
1000	10	$2.91s.$	92%
1000	100	$4.256s.$	70%
10	1000	$15.98s.$	15%
100	1000	$19.44s.$	10%
1000	1000	$18.88s.$	8%

FIGURE 2.11 – Taux de réussite et temps d'exécution moyen du programme

L'erreur résiduelle la plus faible étant pour la régression de la forme $\mathbf{H_\theta^4(n) = \theta_0 + \theta_1 n^2}$, on obtient bien une complexité en $\mathbf{O(n^2)}$ comme voulu et **l'hypothèse 2** est validée. Cependant, on peut voir que si l'on parasite énormément le code de base, on n'obtient difficilement plus de 10% de bons résultats ainsi que des temps d'exécution bien trop long, ceci peut être explique par le fait que lorsque k_1 et k_2 sont suffisamment grand, $k_1 n$ (resp. $k_2 nlog(n)$) est du même ordre que n^2 ce qui fausse les résultats.

Par exemple, si on garde k_1 et k_2 valant 1000, mais qu'on augmente la taille maximal des entrées, en utilisant les tailles d'entrées suivantes :
25000, 50000, 100000, 150000, 200000, 250000, 300000, 350000, 400000

On obtient alors le résultat suivant :

k_1	k_2	Temps exécution moyen	% de bon résultats
1000	1000	$450.2s.$	75%

FIGURE 2.12 – Taux de réussite et temps d'exécution moyen du programme

On observe que si l'on augemente les tailles des entrées, on retrouve un taux de résultat convenable, cependant le temps d'exécution augment grandement.

2.2.3 Power plant $O(n)$

Tailles de tableau utilisées : 10000000, 15000000, 20000000, 25000000, 30000000, 35000000, 40000000, 45000000, 50000000
Complexité théorique : $O(n)$

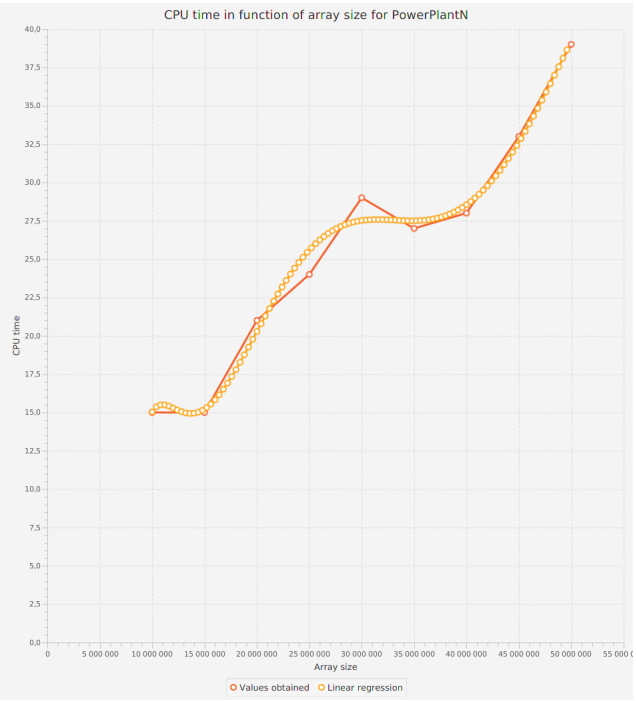

FIGURE 2.13 – Régression linéaire en ms en fonction de la taille des données (n)

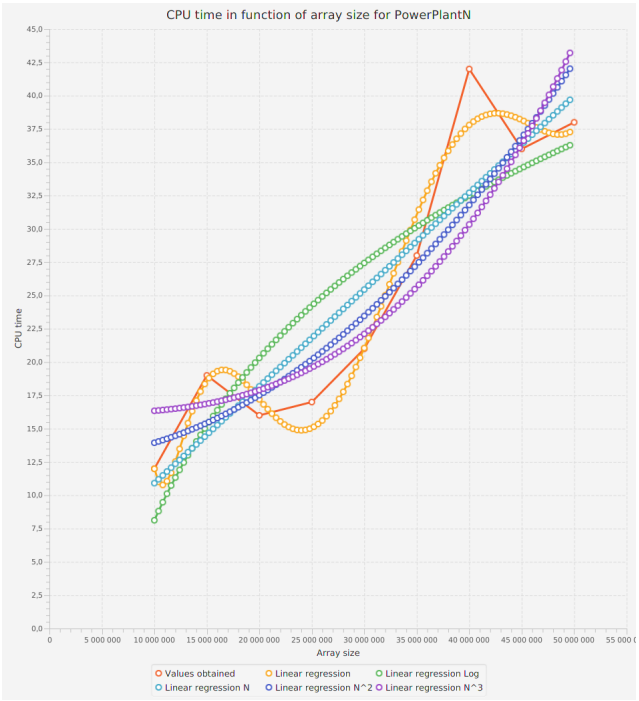

FIGURE 2.14 – Toutes les régressions linéaire en ms en fonction de la taille des données (n)

$$\mathbf{H(n) = 920333.51 - 67505.55log(n) + 0.27n - 0.016nlog(n) + 6.18 * 10^{-9}n^2 - 3.23 * 10^{-10}n^2log(n) + 1.31 * 10^{-18}n^3}$$

Si on essaie de démontrer **l'hypothèse 1**, on prend alors θ_1, or si on prend celui-ci, on obtient une complexité en $O(log(n))$ ce qui ne correspond pas à la complexité attendue en $O(n)$.

Les erreurs résiduelles suivantes :

$H^1_\theta(n)$	$H^2_\theta(n)$	$H^4_\theta(n)$	$H^6_\theta(n)$
1171%	**0%**	219%	1498%

FIGURE 2.15 – Indicateur sur les erreurs résiduelles

L'erreur résiduelle la plus faible étant pour la régression de la forme $\mathbf{H^2_\theta(n) = \theta_0 + \theta_1 n}$, on obtient bien une complexité en $\mathbf{O(n)}$ comme voulu et **l'hypothèse 2** est validée.

% de bon résultats	Temps d'éxécution moyen
66%	$0.2s.$

FIGURE 2.16 – Taux de réussite et temps d'exécution moyen du programme

On test ensuite la taille maximal limite afin d'avoir des résultats convainquant.

Taille maximal	Temps d'exécution total	% de bon résultats
50000	$> 0.1s.$	10%
100000	$> 0.1s.$	20%
1000000	$> 0.1s.$	30%
5000000	$> 0.1s.$	42%
10000000	$> 0.1s.$	58%
50000000	$0.2s.$	70%

FIGURE 2.17 – Taux de réussite et temps d'exécution du programme en fonction de la taille maximal des entrées

2.2.4 Produit matriciel $O(n^3)$

Tailles de tableau utilisées : 300, 350, 400, 450, 500, 550, 600, 650, 700
Complexité théorique : $O(n^3)$

$$\mathbf{H(n) = -8.23 * 10^7 + 2.98 * 10^7 log(n) - 2245344.62n + 400100.61nlog(n)}$$
$$\mathbf{-3572.69n^2 + 439.46n^2log(n) - 0.10n^3}$$

Si on essaie de démontrer **l'hypothèse 1**, on prend alors θ_1, or si on prend celui-ci, on obtient une complexité en $O(log(n))$ ce qui ne correspond pas à la complexité attendue en $O(n^3)$.

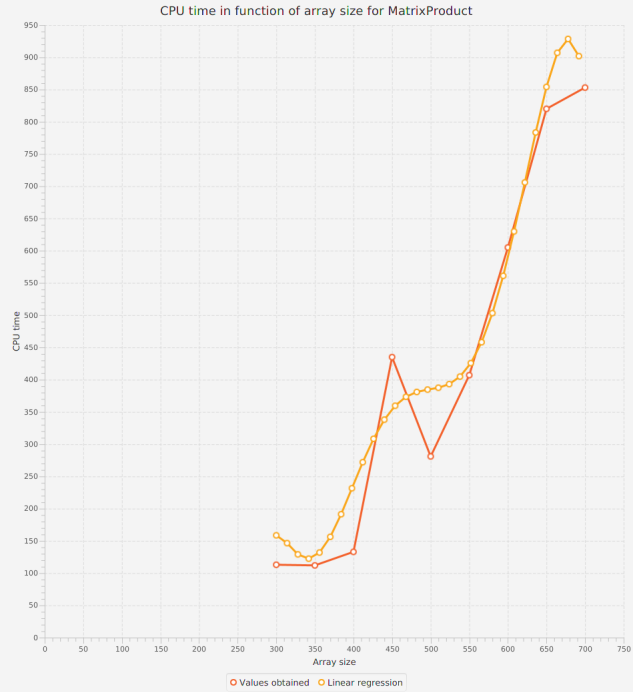

FIGURE 2.18 – Régressions linéaire en ms en fonction de la taille des données (n)

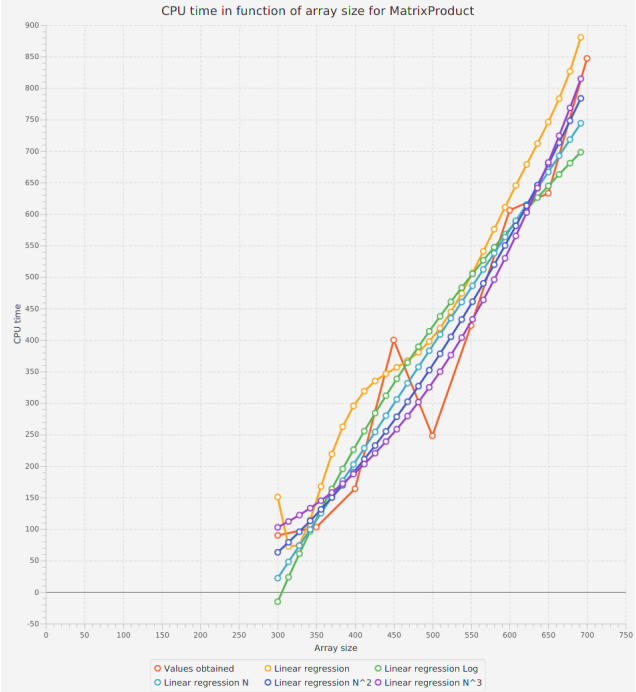

FIGURE 2.19 – Toutes les régressions linéaire en ms en fonction de la taille des données (n)

$H_\theta^1(n)$	$H_\theta^2(n)$	$H_\theta^4(n)$	$H_\theta^6(n)$
137%	66.7%	19.6%	**0**%

FIGURE 2.20 – Indicateur sur les erreurs résiduelles

% de bon résultats	Temps moyen d'exécution
71%	$3.7s.$

FIGURE 2.21 – Taux de réussite et temps d'exécution moyen du programme

L'erreur résiduelle la plus faible étant pour la régression de la forme $\mathbf{H_\theta^6(n) = \theta_0 + \theta_1 n^3}$, on obtient bien une complexité en $\mathbf{O(n^3)}$ comme voulu et donc **l'hypothèse 2** est validée.

On test ensuite la taille maximal limite afin d'avoir des résultats convainquant.

Taille maximal	Temps d'exécution total	% de bon résultats
100	$< 0.1s$.	74%
500	**1.5s**.	**81**%
700	$3.7s$.	71%
1100	$11s$.	85%

FIGURE 2.22 – Taux de réussite et temps d'exécution du programme en fonction de la taille maximal des entrées

2.2.5 Conclusion

Suite à ces différentes expérimentations sur des algorithmes, les résultats permettant la conclusion suivante. Pour les programmes à complexité "simple" (i.e $O(n), O(n^2), O(n^3)$) le programme donne un taux de bonnes réponses satisfaisantes. Cependant, si le l'algorithme testé possède une somme de différentes complexité, le taux de bonne réponse chute drastiquement (moins de 10% dans certains cas).

Annexes

Annexe 1

Intro

1 Partie 1

Bla

1.1 Sous-partie 1

Bla

1.2 Sous-partie 2

Bla

1.3 Sous-partie 3

Bla

2 Partie 2

Bla

2.1 Sous-partie 1

Bla

2.2 Sous-partie 2

Bla

2.3 Sous-partie 3

Bla

Bibliographie

[1] Auteur Ailleurs. Titre3. <http://www.url2.org/>, 2014. [Online ; accessed 16-January-2014].

[2] Auteur Autre. Titre2. <http://www.url1.org/>, 2014. [Online ; accessed 16-January-2014].

[3] Auteur Elle. Titre5. <http://www.url4.org/>, 2014. [Online ; accessed 16-January-2014].

[4] Auteur Livre1. *Titre Livre1*. Editeur1, 2014.

[5] Auteur Lui. Titre4. <http://www.url3.org/>, 2014. [Online ; accessed 16-January-2014].

[6] Auteur Untel. Titre1. <http://www.url0.org/>, 2014. [Online ; accessed 16-January-2014].