Table des matières

1 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 de tout algorithme en se basant seulement sur leur temps d'exécution.

1.1 Sujet

Comme dit dans la courte introduction, le but de ce stage fut de trouver la complexité des algorithmes juste en connaissant leur temps d'exécution. Le projet s'est donc décomposé en deux phases. Durant la première, le but fut de retrouver la complexité d'algorithmes bien connus (tri à bulles par exemple) afin de vérifier que la méthode utilisée donnait des résultats corrects. Ensuite, le programme est testé sur des codes étudiants posté sur caseine [1]. Pour déterminer la complexité, la méthode utilise est la régression linéaire multiple.

2 Principes

2.0.1 Explications

Le but final de ce programme est de déterminer la complexité des algorithmes testés. Pour ce faire, le programme utilise la régression linéaire multiple. Ici, on se restreint à 6 classes de compléxités ($log(n), n, nlog(n), n^2, n^2log(n), n^3$).
La régression linéaire obtenue est $H_\theta(n)$. Pour réaliser cette régression linéaire, l'algorithme est lancé pour différentes tailles de données (i.e : n), et le temps d'exécution est mesuré à chaque fois. Ensuite, le programme construit le tableau avec chaque valeur de features correspondante à l'entrée. Une fois la régression linéaire construite, il y à deux approches possible afin de déterminer la complexité de l'algorithme :
La première consiste à regarder chaque θ_i de $H_\theta(n)$ et de prendre le plus grand.
La seconde consiste à décomposer la régression linéaire en plusieurs régressions ($H^i_\theta(n)$) et de calculer l'erreur résiduelle de chaqu'une d'entre elles et de prendre l'erreur la plus faible pour déterminer la complexité.

2.0.2 Notations utilisées

n : différentes tailles du tableau
Régression linéaire : $H_\theta(n) = \theta_0 + \theta_1 log(n) + \theta_2 n + \theta_3 nlog(n) + \theta_4 n^2 + \theta_5 n^2 log(n) + \theta_6 n^3$
Régression linéaire permettant de determiner la complexité
$H^1_\theta(n) = \theta_0 + \theta_1 log(n)$
$H^2_\theta(n) = \theta_0 + \theta_1 n$
$H^4_\theta(n) = \theta_0 + \theta_1 n^2$
$H^6_\theta(n) = \theta_0 + \theta_1 n^3$

Remarque : Dans cette première partie, on s'interesse seulement a déterminé la complexité entre $O(log(n))$, $O(n)$, $O(n^2)$ et $O(n^3)$.

2.0.3 Protocole

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

1. http ://caseine.org/

- 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 1 – Exemple de tableau utilisé pour la régression

- On réalise la régression linéaire $H_\theta(n)$.
- Afin de determiné la complexité, on regarde le plus grand coefficiant θ_i, et si on ne peux pas conclure, on découpe la régression linéaire en plusieur régression (i.e $H^i_\theta(n)$).
- On calcul l'erreur résiduelle de chaqu'une des régressions afin de trouver le minimal pour determiner la compléxité.
- On trace le graphique contenant toutes les régressions linéaire ainsi que l'éxecution de l'algorithme.
- On extrait un tableau contenant l'erreur résiduelle pour chaque $H^i_\theta(n)$ et le temps d'éxecution total.

2.1 Proof of concept

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

2.1.1 Tri à bulles simple $O(n^2)$

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

Le graphique obtenu ressemble donc à ça :

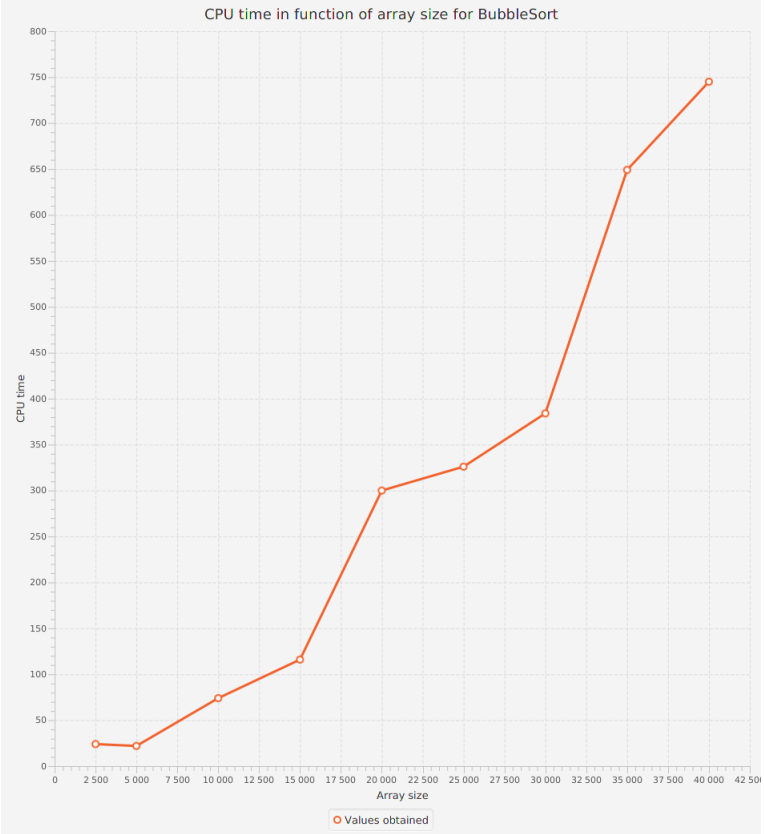

FIGURE 2 – Graphique regroupant le temps CPU en fonction de la taille du tableau

Ensuite, le programme calcul l'équation de la régression linéaire obtenu.
On obtient le resultat suivant :

$$\mathbf{H(n) = 257498.66 - 35867.89log(n) + 54.14n - 5.30nlog(n) + (7.10 * 10^{-4})n^2} \\ \mathbf{-(5.52 * 10^{-5})n^2log(n) + (1.13 * 10^{-10})n^3}$$

Or, 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 trouver une autre méthode afin de determiner la complexité à partir de la régression linéaire.

Afin de déterminer la compléxité plus précisément, on calcul les régressions linéaires $H_\theta^i(n)$.
Et pour chaqu'une de ces regressions linéaires, on calcul la somme des carrés de l'erreur résiduelle.
Logiquement, l'erreur résiduelle obtenu la plus faible correspond à la régression de la complexité de l'algorithme.
On obtient alors les résultats suivants :

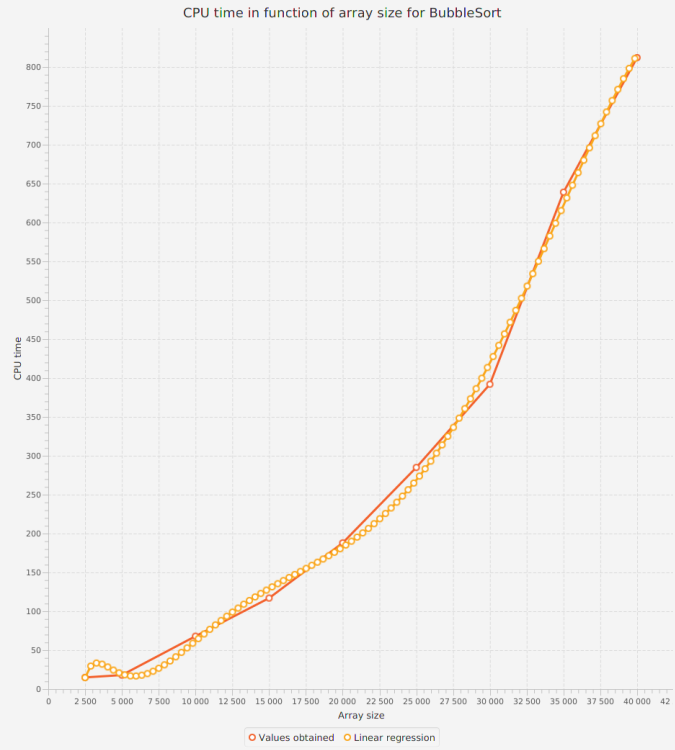

FIGURE 3 – Régression linéaire

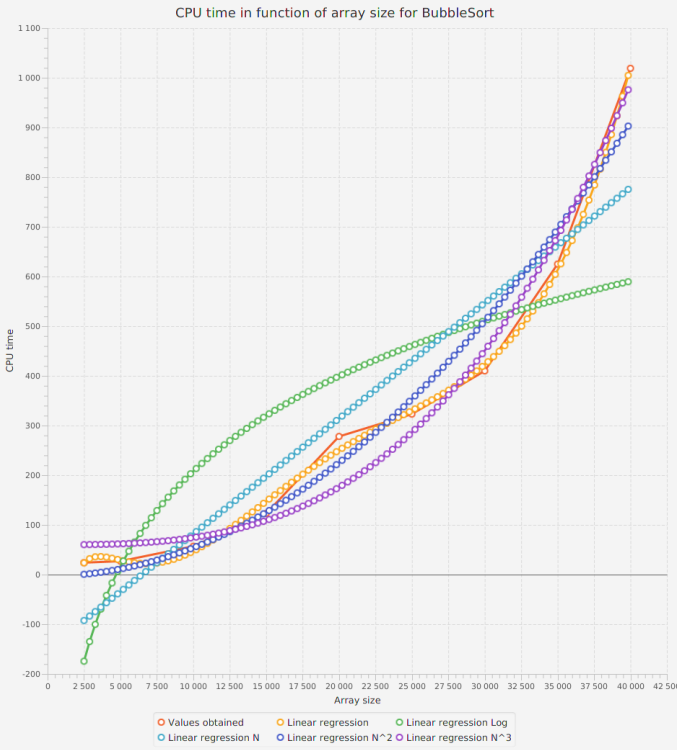

FIGURE 4 – Toutes les régressions linéaire

Régression linéaire	$H_\theta(n)$	$H^1_\theta(n)$	$H^2_\theta(n)$	$H^4_\theta(n)$	$H^6_\theta(n)$	Temps total
Erreur résiduelle :	0.06	29280.90	9225.85	**146.96**	1772.99	$2.5s$.

FIGURE 5 – Tableau récapitulatif des erreurs résiduelles

$H^1_\theta(n)$	$H^2_\theta(n)$	$H^4_\theta(n)$	$H^6_\theta(n)$
19824%	6177%	**0**%	1106%

FIGURE 6 – Indicateur sur les erreurs résiduelles

Nombre d'essais	% de bon résultats
10	100%
50	94%
100	99%

FIGURE 7 – Taux de réussite du programme

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 compléxité en $\mathbf{O(n^2)}$ comme voulu.

2.1.2 Tri à bulles parasité $O(k_1 n + k_2 nlog(n) + n^2)$

De la même façon que l'exemple précédent, sauf que cette fois, on parasite le tri à bulles en rajoutant un tri fusion au milieu (sur une copie du tableau) et des parcours de tableau. Ceci a pour but d'avoir une compléxité du type $O(k_1 n + k_2 nlog(n) + n^2)$ afin de voir si la méthode utilisée donne bien une compléxité en $O(n^2)$.
Ceci à pour but de montrer la robustesse de l'algorithme.
On applique ensuite la même méthode pour determiné 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 :

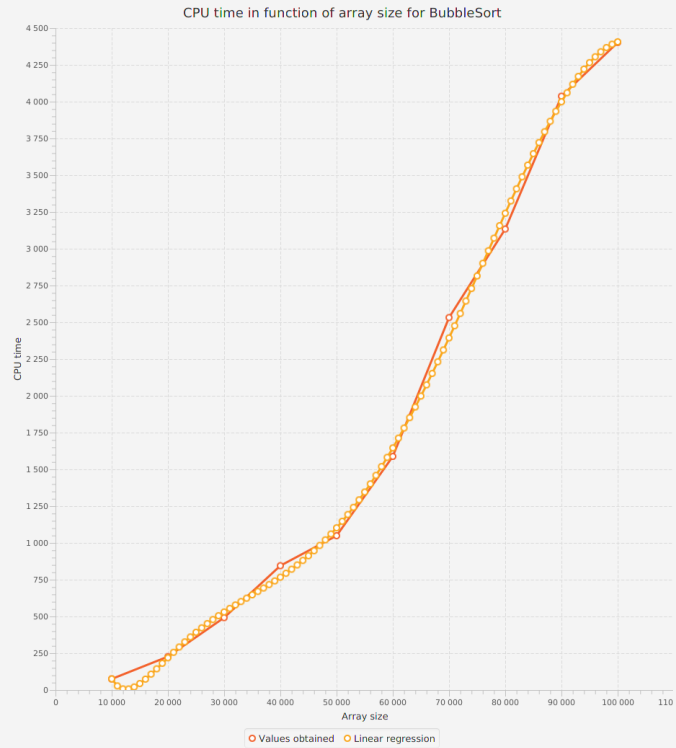

FIGURE 8 – Régression linéaire

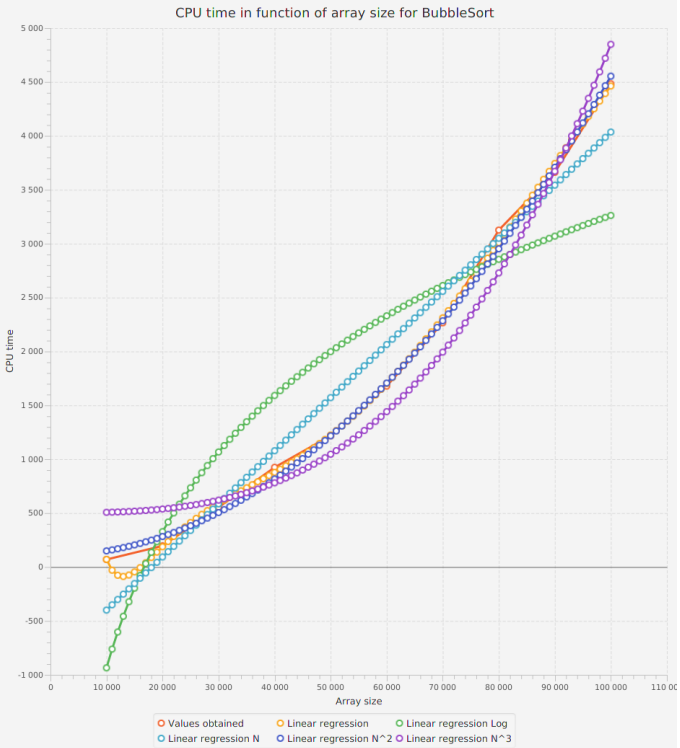

FIGURE 9 – Toutes les régressions linéaire

Avec la regression linéaire suivante : $\mathbf{H(n) = 59257.87 - 7300.68O(log(n))) - 1.52O(n)) + 0.33O(nlog(n)))}$
$\mathbf{-3.51 * 10^{-4}O(n^2)) + 3.00 * 10^{-5}O(n^2log(n))) - 1.49 * 10^{-10}O(n^3))}$

Et les erreurs résiduelles suivantes :

k_1	k_2	$H_\theta(n)$	$H^1_\theta(n)$	$H^2_\theta(n)$	$H^4_\theta(n)$	$H^6_\theta(n)$	Temps total	% de bon résultats
10	10	$1.07 * 10^{-4}$	21309.57	5001.23	**77.23**	3509.56	2.34s.	96%
10	100	0.01	51398.92	17751.08	**786.113**	1565.41	3.78s.	81%
10	1000	5.69	488242.69	51767.94	**47055.61**	236880.03	15.98s.	15%
100	10	0.06	31106.71	8491.66	**0.030**	3646.38	2.82s.	94%
100	100	0.144	54525.30	10107.00	**1210.93**	14289.97	4.13s.	76%
100	1000	6.36	702891.28	72563.16	**67464.92**	334756.27	19.44s.	10%
1000	10	0.06	30226.83	8314.84	**4.46**	2965.85	2.91s.	92%
1000	100	0.006	4301.05	5483.6	**3066.28**	18301.45	4.256s.	70%
1000	1000	1.34	878114.43	130920.98	**39558.52**	295243.44	18.88s.	8%

FIGURE 10 – Tableau récapitulatif des erreurs résiduelles

k_1	k_2	$H^1_\theta(n)$	$H^2_\theta(n)$	$H^4_\theta(n)$	$H^6_\theta(n)$
10	10	27491%	6375%	**0**%	4444%
10	100	6438%	2158%	**0**%	30033%
10	1000	937%	10%	**0**%	403%
100	10	$1.03 * 10^8$%	$2.83 * 10^7$%	**0**%	$1.21 * 10^7$%
100	100	4406%	735%	**0**%	1080%
100	1000	941%	7%	**0**%	396%
1000	10	$6.77 * 10^5$%	$1.86 * 10^5$%	**0**%	66379%
1000	100	40%	78%	**0**%	496%
1000	1000	2119%	230%	**0**%	646%

FIGURE 11 – indicateur sur les erreurs résiduelles

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 compléxité en $\mathbf{O(n^2)}$ comme voulu. Cependant, on peut voir que si l'on parasite énormement le code de base, on n'obtient difficilement plus de 10% de bons résultats.

2.1.3 Power plant $O(n)$

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

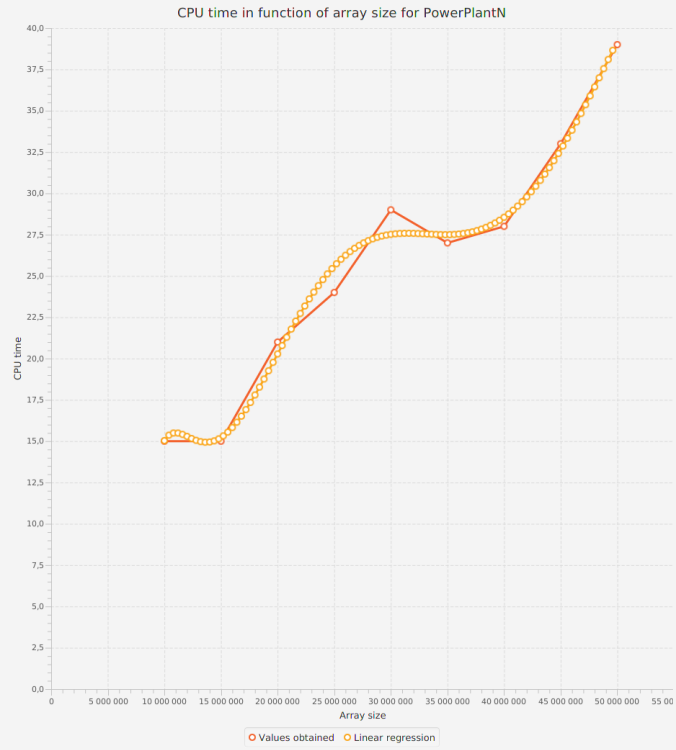

FIGURE 12 – Régression linéaire

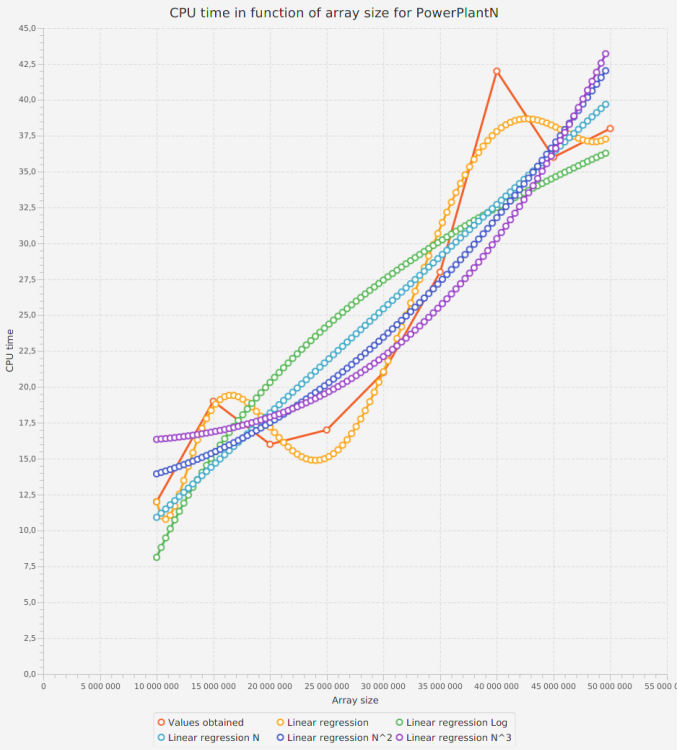

FIGURE 13 – Toutes les régressions linéaire

Et les erreurs résiduelles suivantes :

$H_\theta(n)$	$H^1_\theta(n)$	$H^2_\theta(n)$	$H^4_\theta(n)$	$H^6_\theta(n)$	Temps total
$1.28 * 10^{-4}$	15.0	**1.18**	3.77	18.86	$0.2s.$

FIGURE 14 – Tableau récapitulatif des erreurs résiduelles

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

FIGURE 15 – Indicateur sur les erreurs résiduelles

Nombre d'essais	% de bon résultats
10	90%
50	56%
100	66%

FIGURE 16 – Taux de réussite du programme

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 compléxité en $\mathbf{O(n)}$ comme voulu.

2.1.4 Produit matriciel $O(n^3)$

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

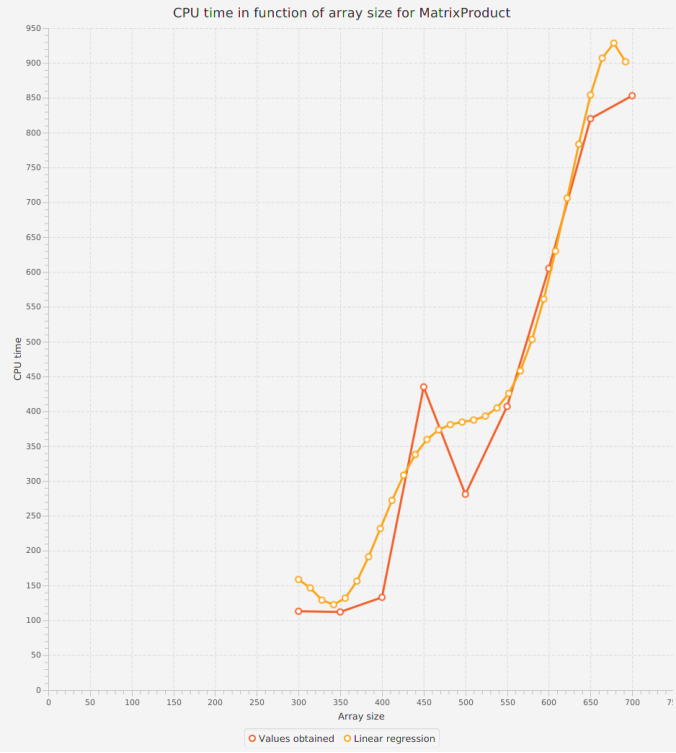

FIGURE 17 – Régression linéaire

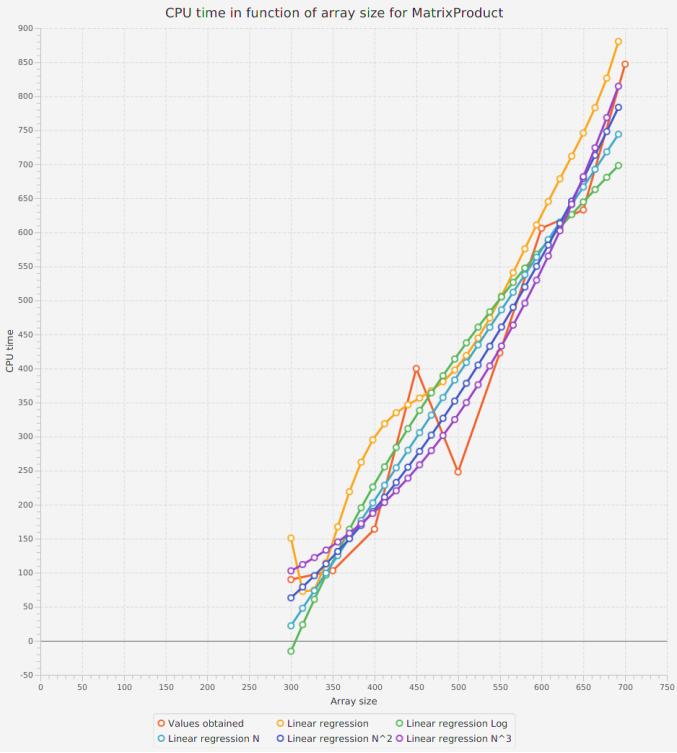

FIGURE 18 – Toutes les régressions linéaire

$H_\theta(n)$	$H^1_\theta(n)$	$H^2_\theta(n)$	$H^4_\theta(n)$	$H^6_\theta(n)$	Temps total
3867.37	89729.89	62900.34	45130.21	**37722.21**	$3.7s.$

FIGURE 19 – Tableau récapitulatif des erreurs résiduelles

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

FIGURE 20 – Indicateur sur les erreurs résiduelles

Nombre d'essais	% de bon résultats
10	80%
50	74%
100	71%

FIGURE 21 – Taux de réussite 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 compléxité en $\mathbf{O(n^3)}$ comme voulu.

2.1.5 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).

2.2 Résultats expérimentaux sur code étudiant

3 Machine learning

Suite au calcul de la somme des carrés de l'erreur résiduelle, on obtient le bon résultat avec une complexité en $O(n^2)$ ici pour le tri à bulles. Cependant, il reste un petit problème a résoudre, comment distinguer une complexité en $O(n)$ et une en $O(nlog(n))$ ainsi que une complexité en $O(n^2)$ et une en $O(n^2log(n))$ car celles-ci sont très proche l'une de l'autre ce qui complique le calcul de l'erreur résiduelle.

3.1 Etude du sujet

Intro

3.2 Problème posé

Bla

3.3 Solution proposée

Bla

Transition

3.4 Réalisation