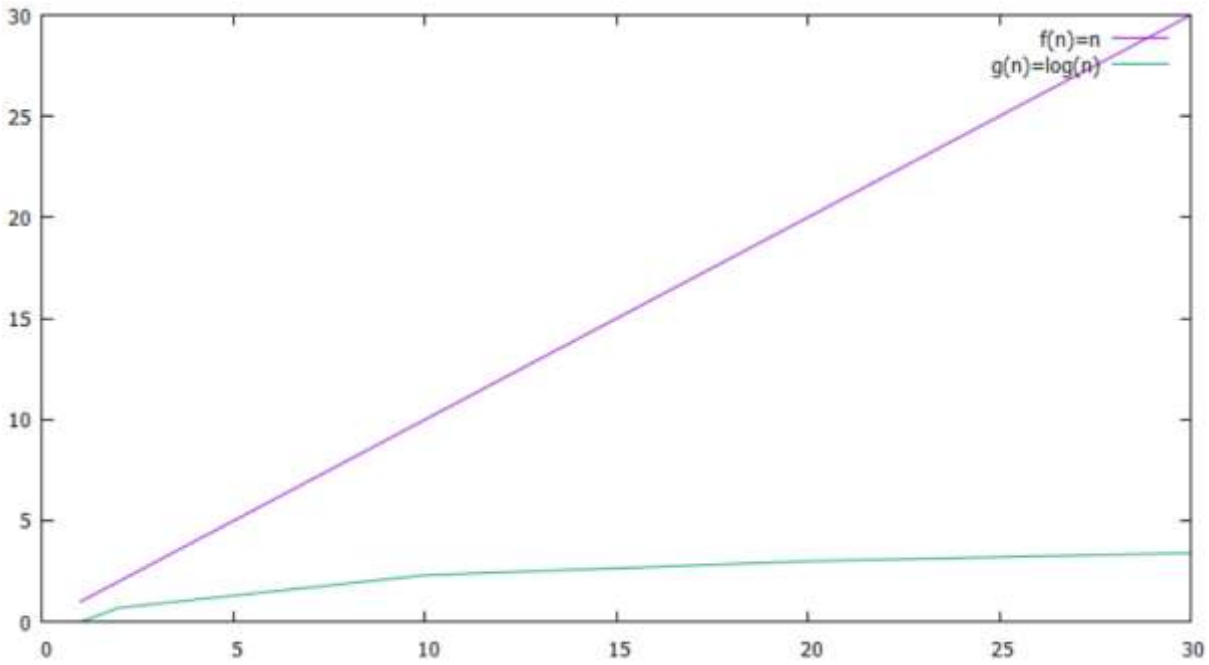
TP1 ACT Ordres de grandeur asymptotiques

Question 1.1.2 :

Illustration des courbes obtenues :



On observe que la courbe f(n)=n évolue plus rapidement que celle de g(n)=log(n), ce qui est normale étant donné que par comparaison des fonctions on a :

F(n) = n > g(n) = log(n)

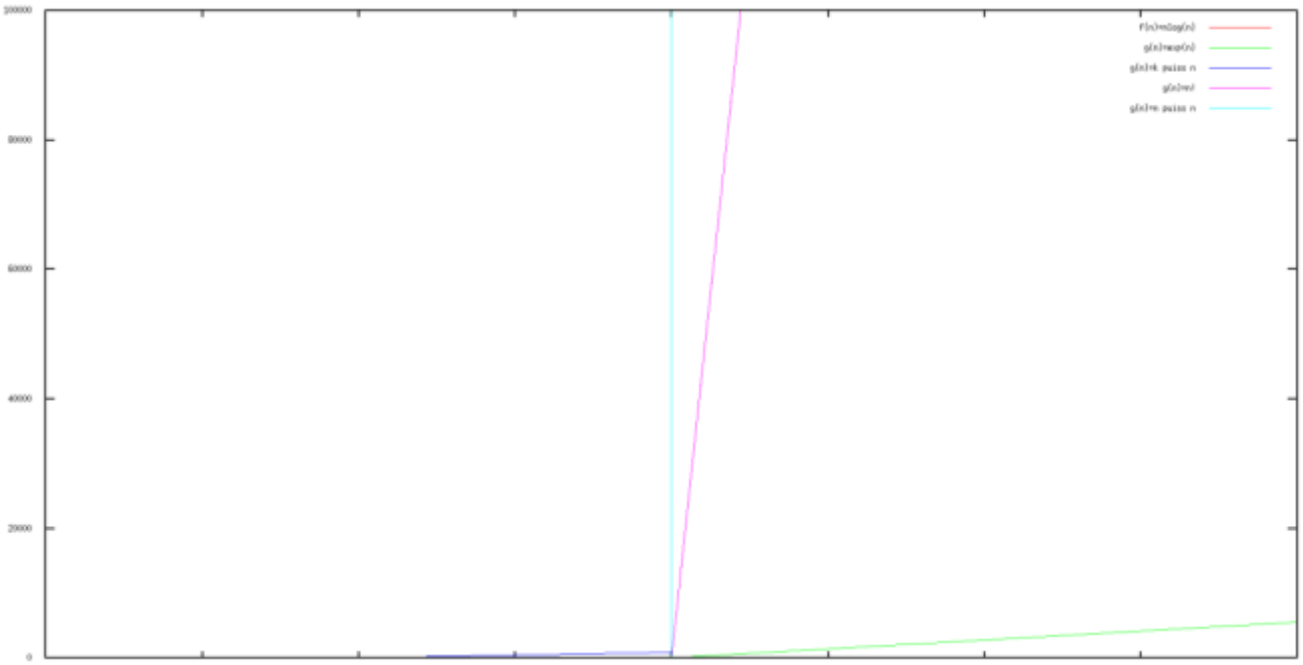
Démonstration par les limites :

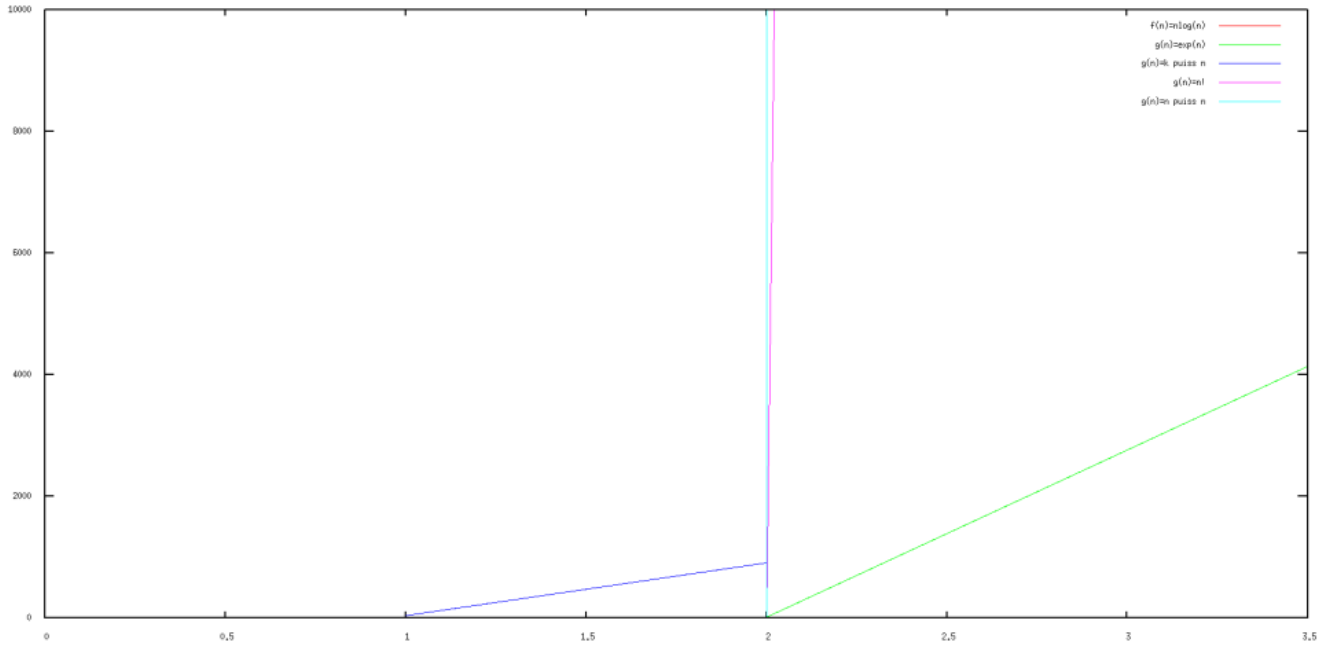
$\lim_{+\infty} \frac{n}{Log(n)} = +\infty$ étant donné que n > log(n). Ce qui nous amène à notre assertion précédente.

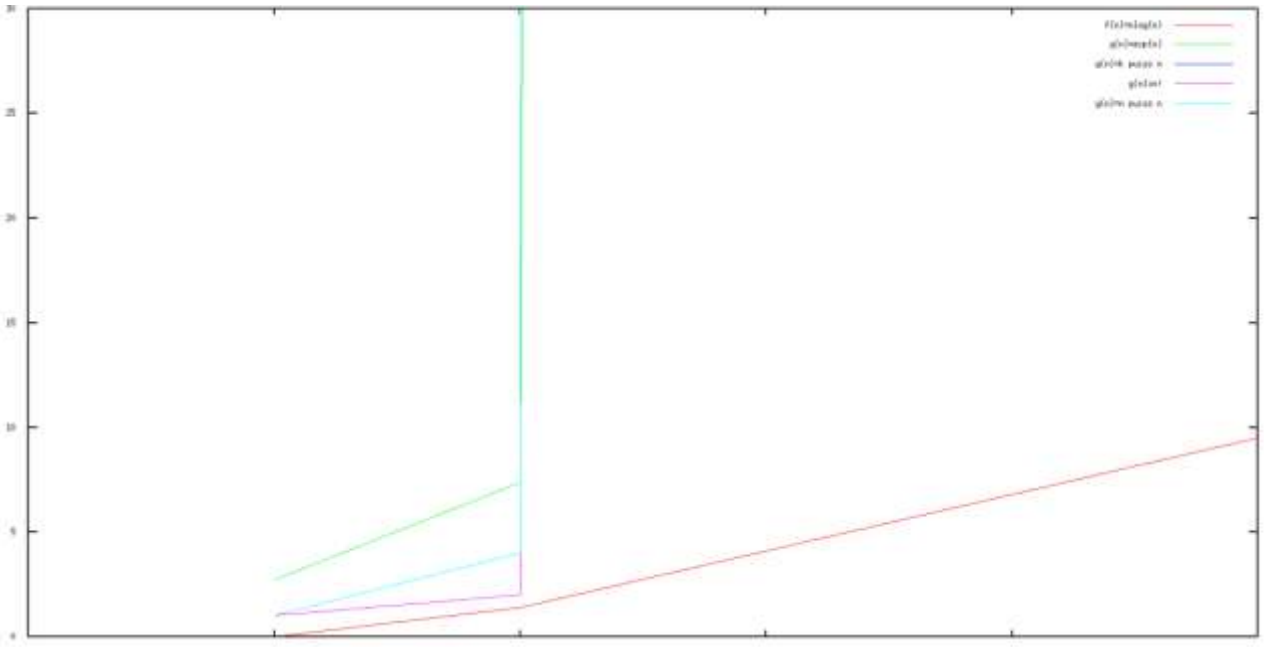
Comparaison des fonctions de références :

Question 1.2.4 :

Graphique des comparaisons :

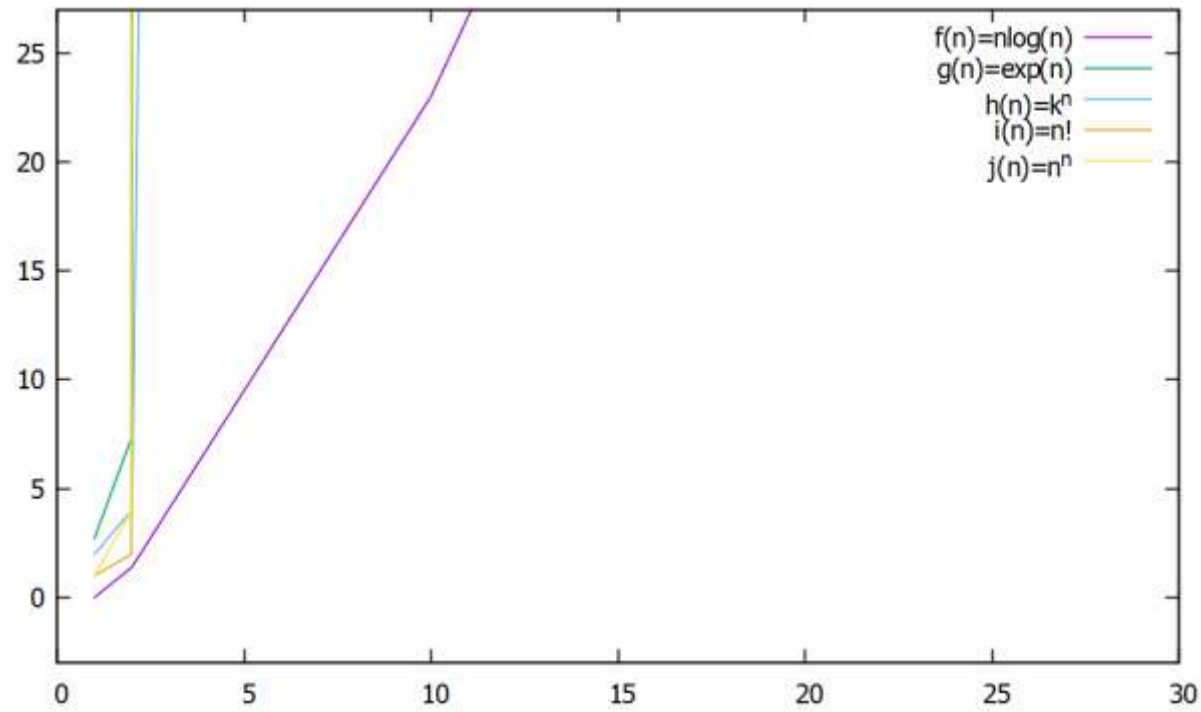






Sur ces différentes images nous avons joué avec l'échelle des X et des Y de sorte à bien montrer les différences entre les fonctions sur les 2 axes. Car avec une échelle fixe, il nous a été impossible de classer ces fonctions en se basant sur l'ordre des courbes car elles se chevauchaient surtout sur l'axe des X.

Voici l'image obtenue sans variation de l'échelle :

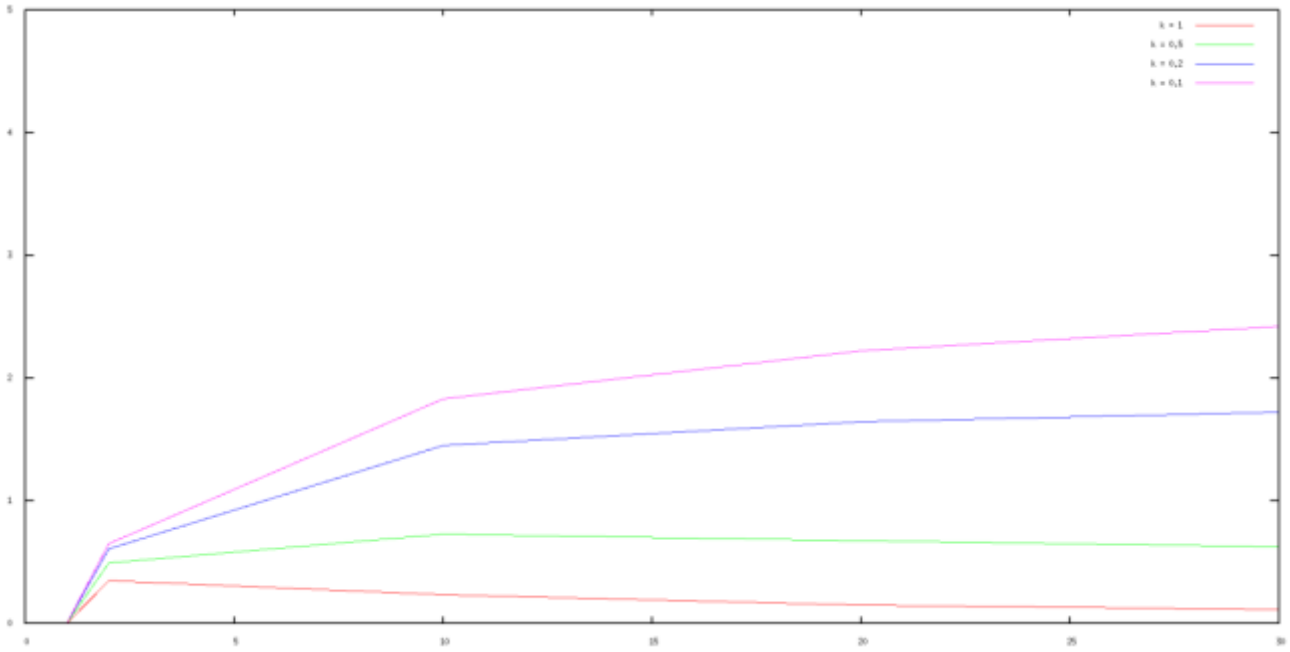


On remarque bien que les 4 fonctions représentées par leur couleur se chevauchent et qu'il est absolument impossible de les classer.

Limites graphiques

Question 1.3.1 : n^{ξ} vs log(n)

Graphique illustrant le rapport des deux fonctions :

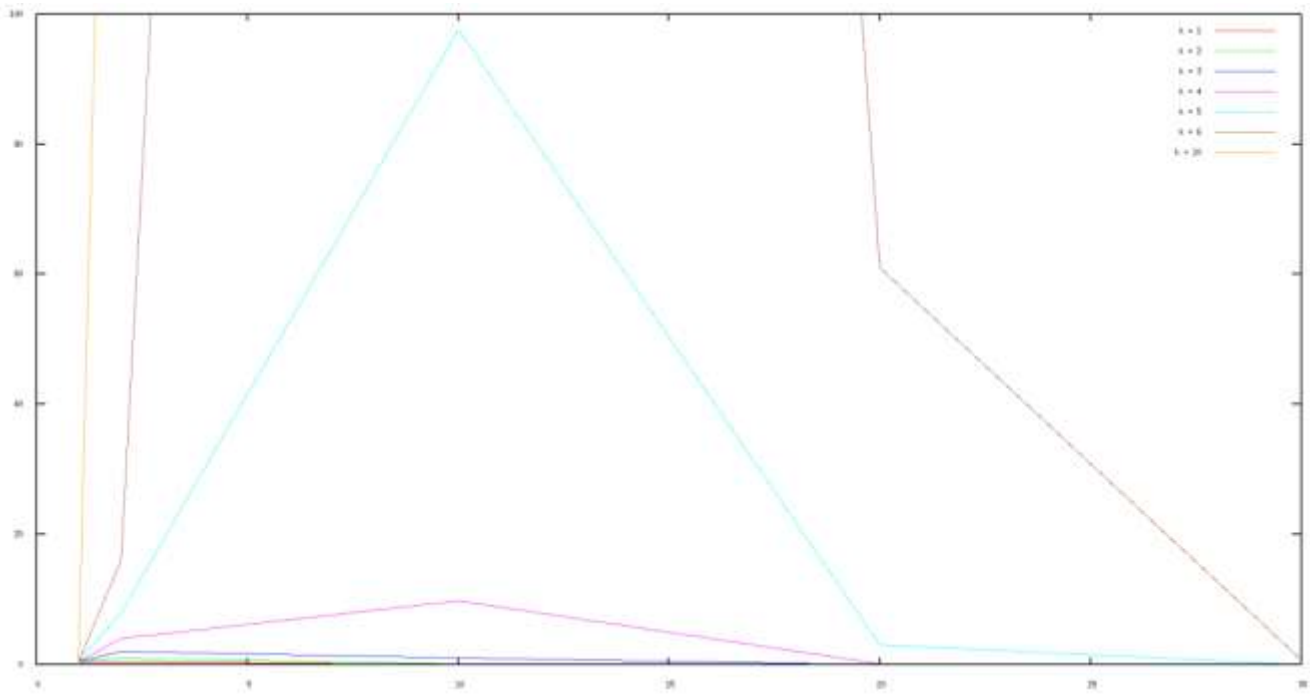


Sur ce graphique on observe que plus la valeur de epsilon est grande plus le rapport des fonctions est petit ce qui implique que :

Plus epsilon est petit, plus la limite du rapport des fonctions tend vers 0.

Question 1.3.2 : 2^n vs n^k

Graphique illustrant le rapport des deux fonctions :



Pour ce graphique nous avons chois les valeurs de k suivantes :

K = 1, k = 2, k = 3, k = 4, k = 5, K = 6, k = 7, k = 8, k = 9, k = 10.

On remarque que plus la valeur de K est grande, plus le rapport des fonctions est grand. Néanmoins arrivé à une certaine valeur de n, la valeur de K peut être considérée comme négligeable car comme on le voit sur le graphique toutes les courbes tendent vers 0.

Mesures sur structures de données

Question 2.0 :

Pour mesurer le temps d'exécution d'une fonction de manière expérimentale, il suffit de mesurer le temps avant lancement de la fonction, d'exécuter la fonction puis de remesurer le temps après exécution de la fonction. Et enfin de faire la soustraction des 2 temps mesurer ce qui au final nous donne le temps d'exécution de la fonction.

Question 2.1.1 :

Fonction recherche() de la Classe Tableau :

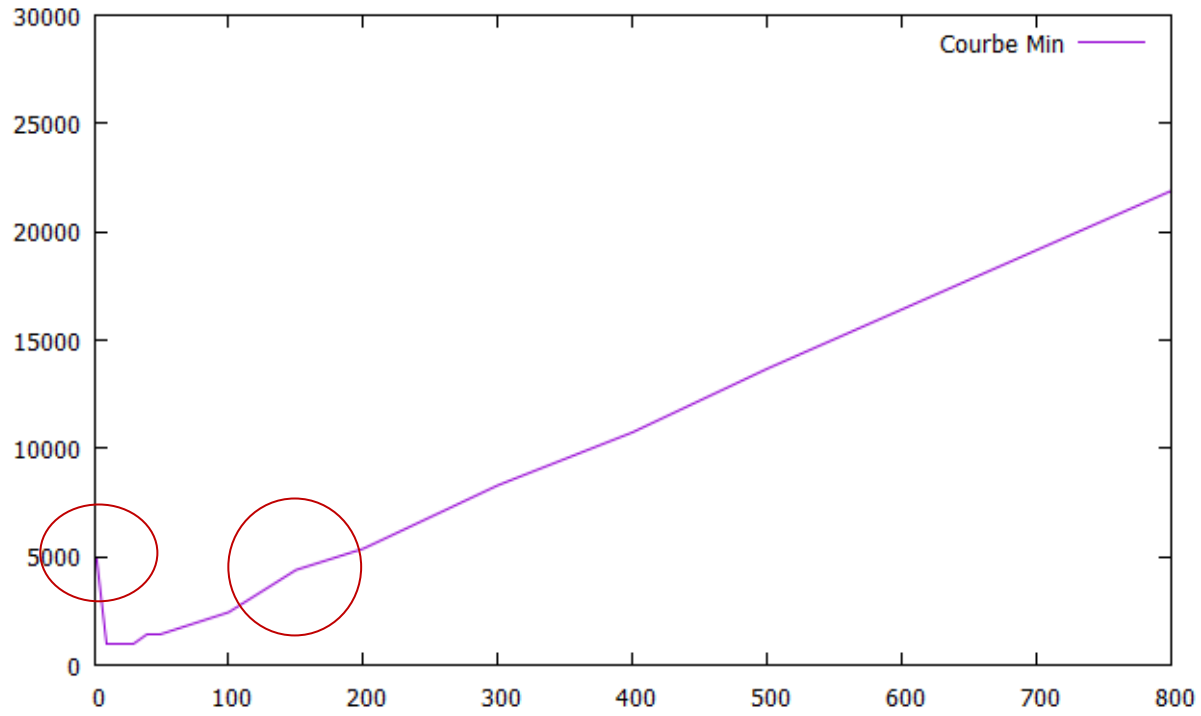
```
	@Override
public boolean recherche(int e) {
	for(int j=0;j<this.tab.length;j++){
		if (this.tab[j] == e) {
			return true;
		}
	}
	return false;
}
```

On a une boucle for qui parcours chaque élément du tableau une fois par pas de 1. En supposant que la taille du tableau est n la complexité serait de l'ordre de n.

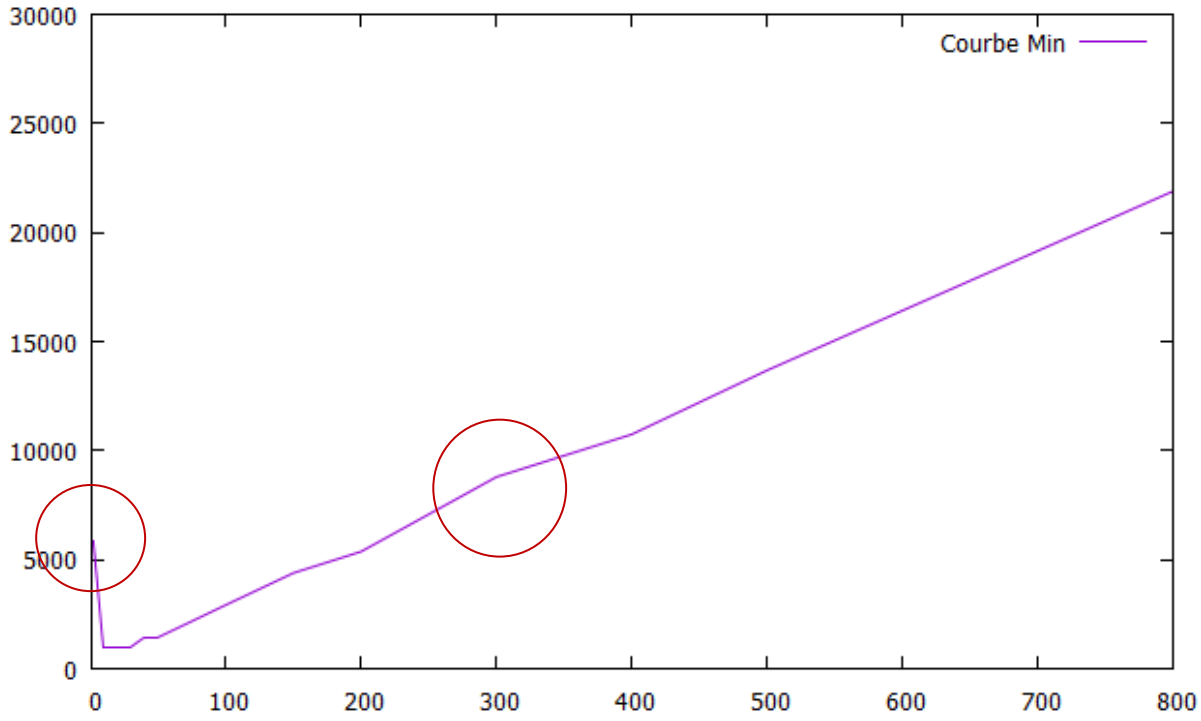
➔Complexité en O(n)

Question 2.1.2 :

Exécution 1 :



Exécution 2 :



Les zones entourées en rouge sur les graphiques révèlent une légère différence au niveau de l'allure de la courbe.

La forme de la courbe est normale, cela est due au fait que plus les nombres à traiter sont grand plus le temps de calcul est long ce qui explique que la courbe augmente de niveau constamment.

Question 2.1.3 :

On constate que les courbes d'exécution sont différentes, ceci est dû à beaucoup de paramètres mais essentiellement des performances de la machine (dépend du nombre de processus en cours).

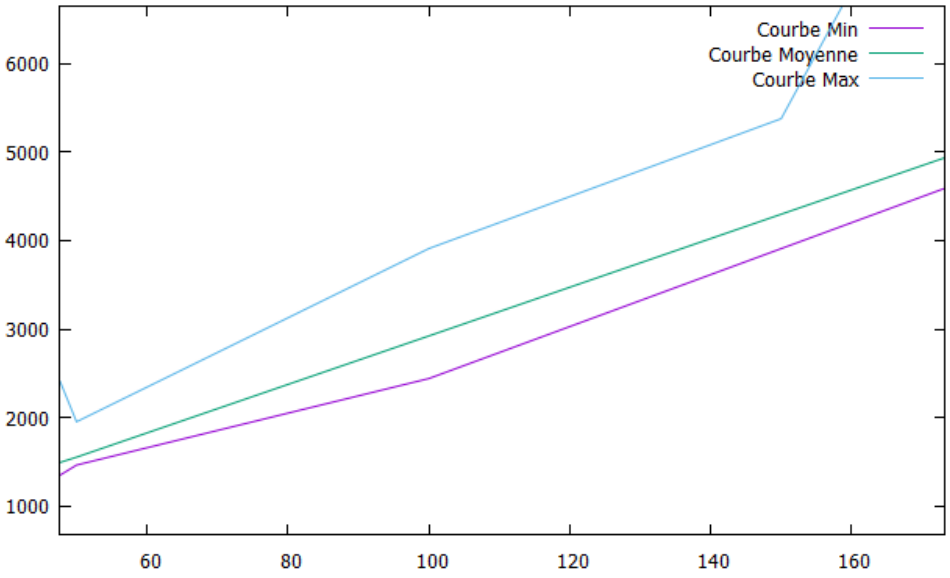
Question 2.1.4 :

Pour y remédier, on pourrait fait une moyenne des temps d'exécution sur plusieurs lancements de sorte à réduire les variations et avoir des résultats constants pour accroitre la précision de la mesure.

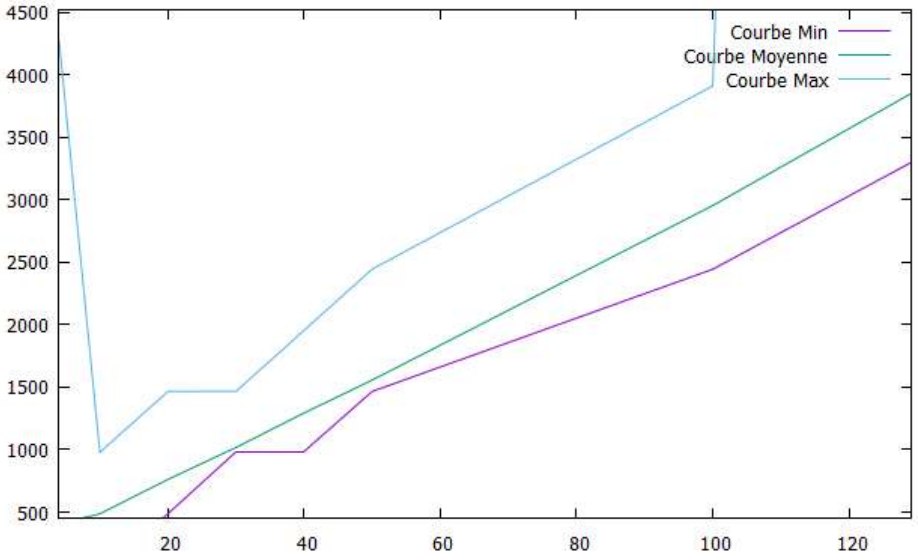
Question 2.1.5 :

Graphique des courbes du minimum, maximum et de la moyenne

Exécution 1 :



Exécution 2 :



Question 2.1.6 :

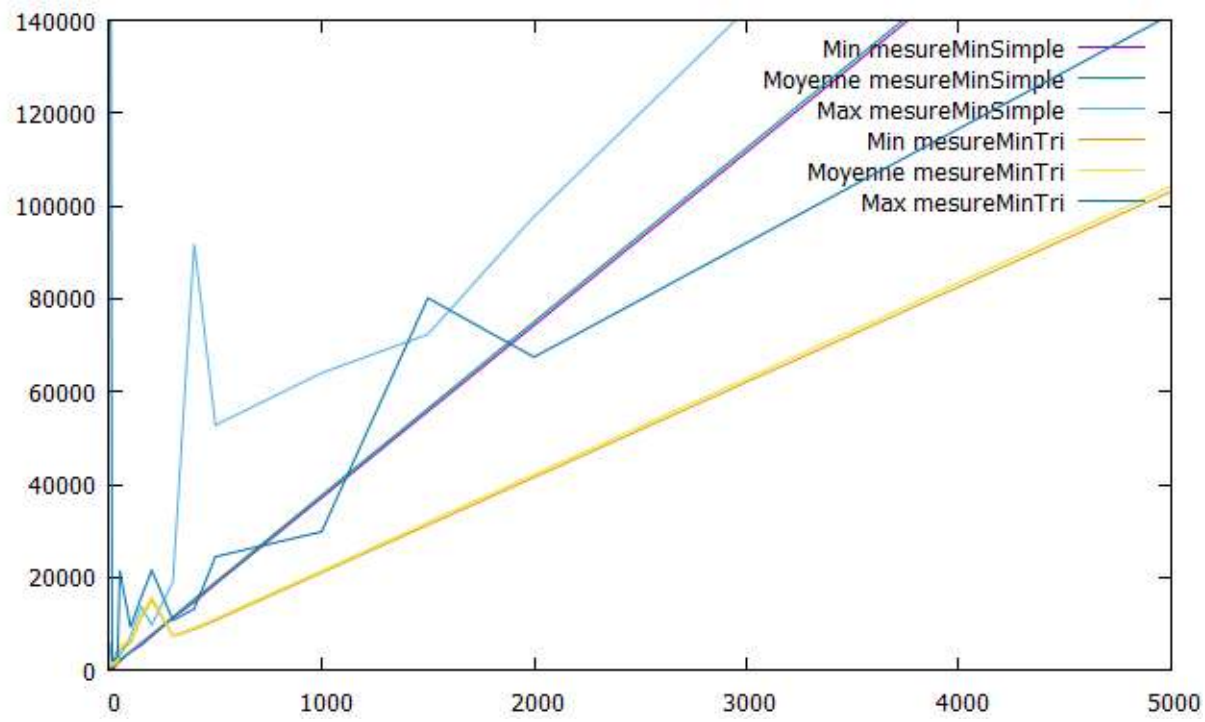
On a bien le résultat attendu c'est-à-dire une droite qui croit constamment et qui représente la courbe de la fonction f(n) = n ce qui est la complexité mentionnée en 2.1.1.

On remarque que sur 2 exécutions la courbe de la moyenne garde la même allure ce qui est ce que l'on cherchait à obtenir.

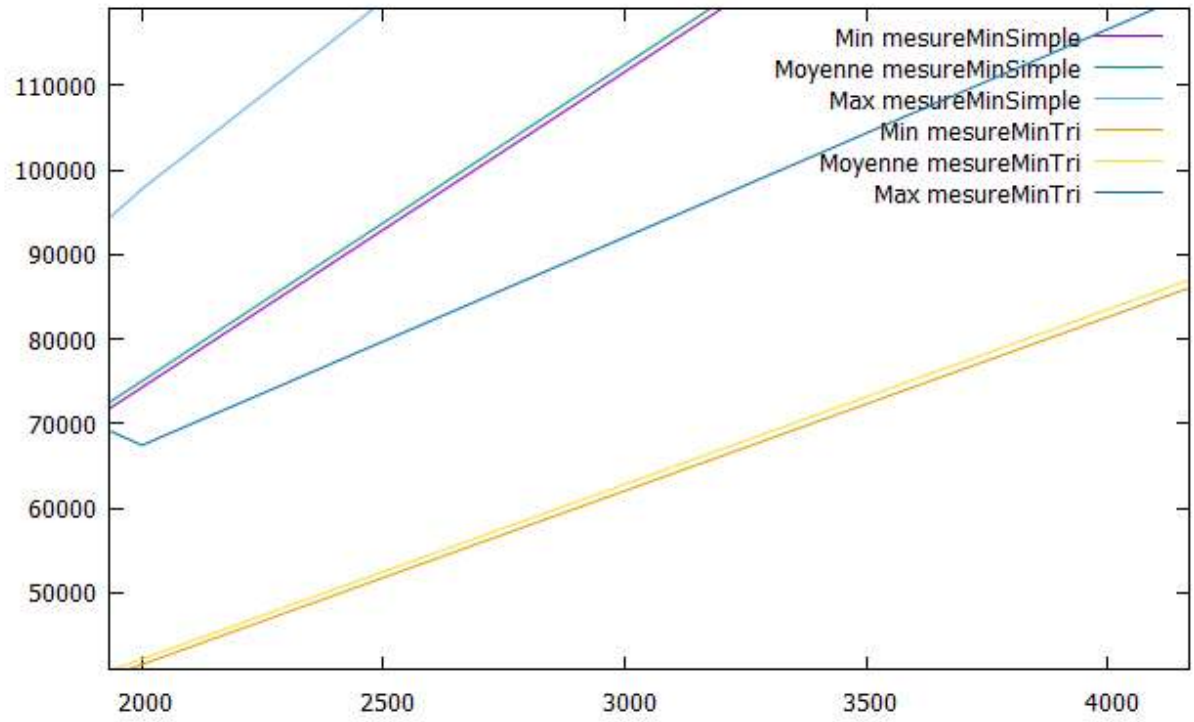
Un minimum () de performances

Question 2.2.2 :

Graphique des méthodes mesureMinSimple et mesureMinTri :



Zoom sur les courbes :



Sur ce graphique on peut observer que les courbes concernant la méthode mesureMinimumTri est sont en dessous des courbes de la méthode mesureMinimumSimple.

Question 2.2.3 :

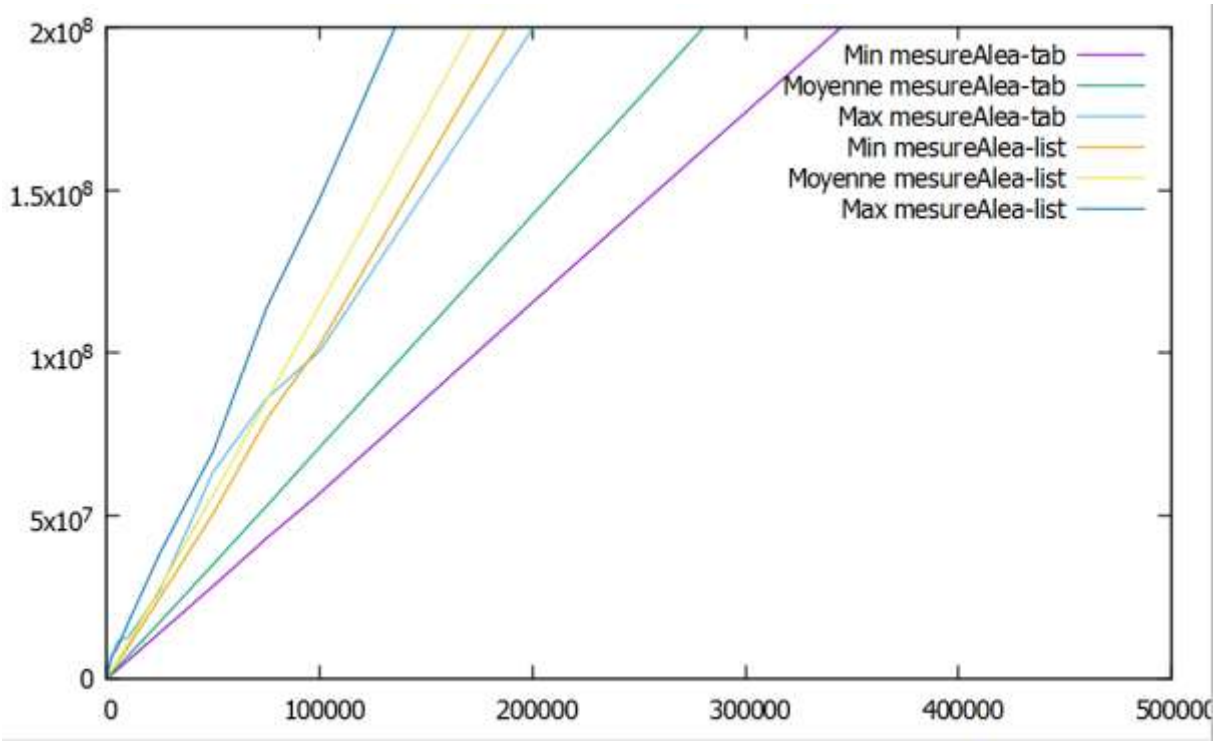
On peut en conclure que la méthode mesureMinTri est plus rapide que la méthode mesureMinSimple.

Ceci est due au fait que la recherche dans un tableau trié est en log(n)cf recherche dichotomique.

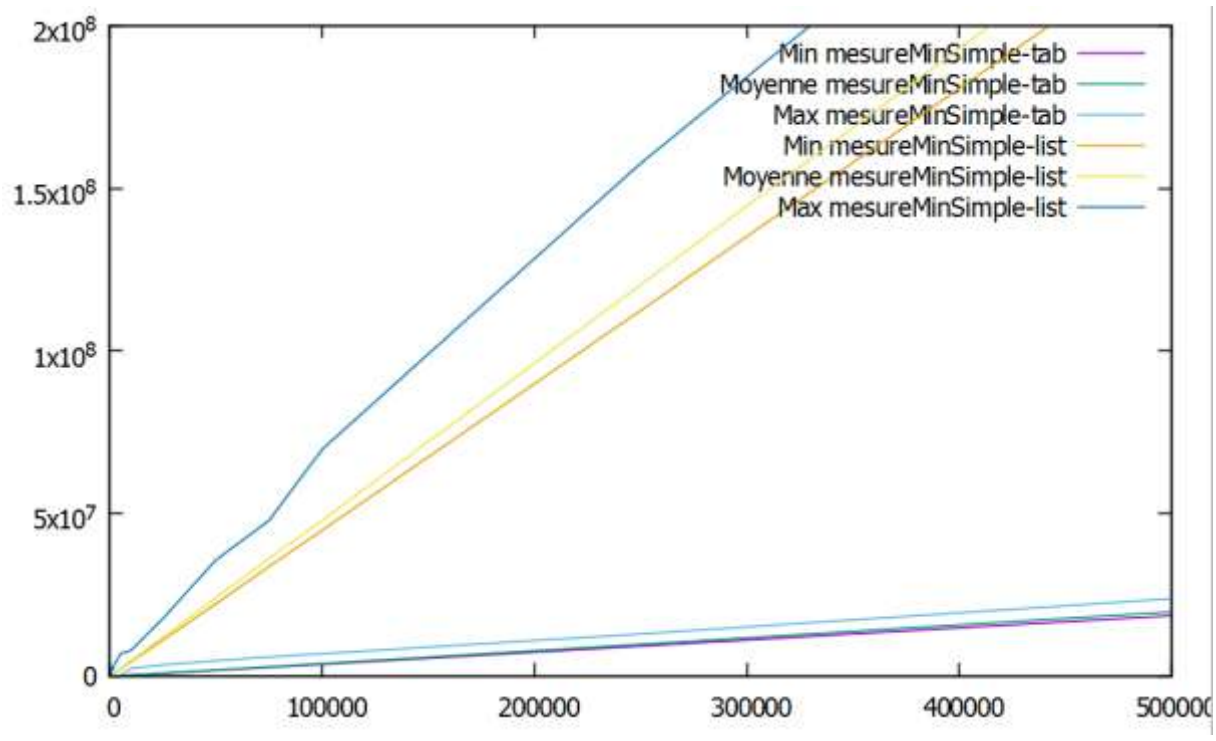
Les structures de base pour le meilleur et surtout pour le pire

Question 2.3.2 :

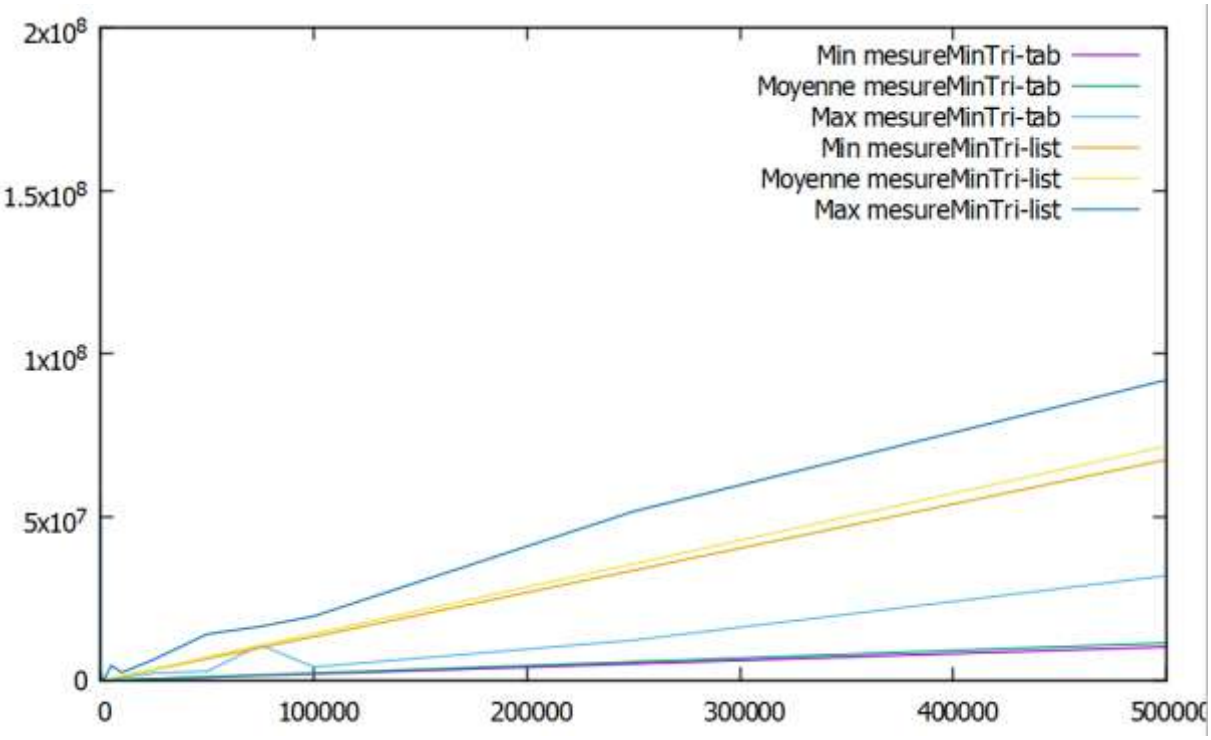
Fonction alea() -- complexité en O(n)



Fonction minSimple() -- complexité en O(n)

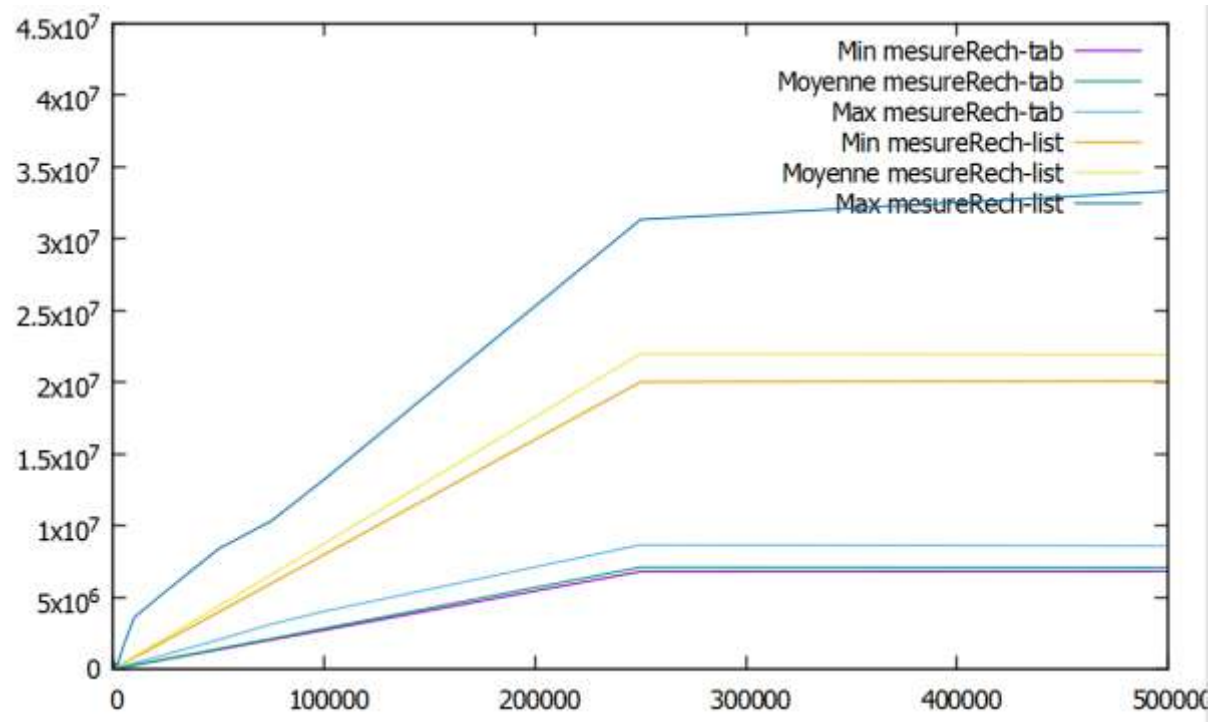


Fonction minTri()-- complexité en O(nlog(n))



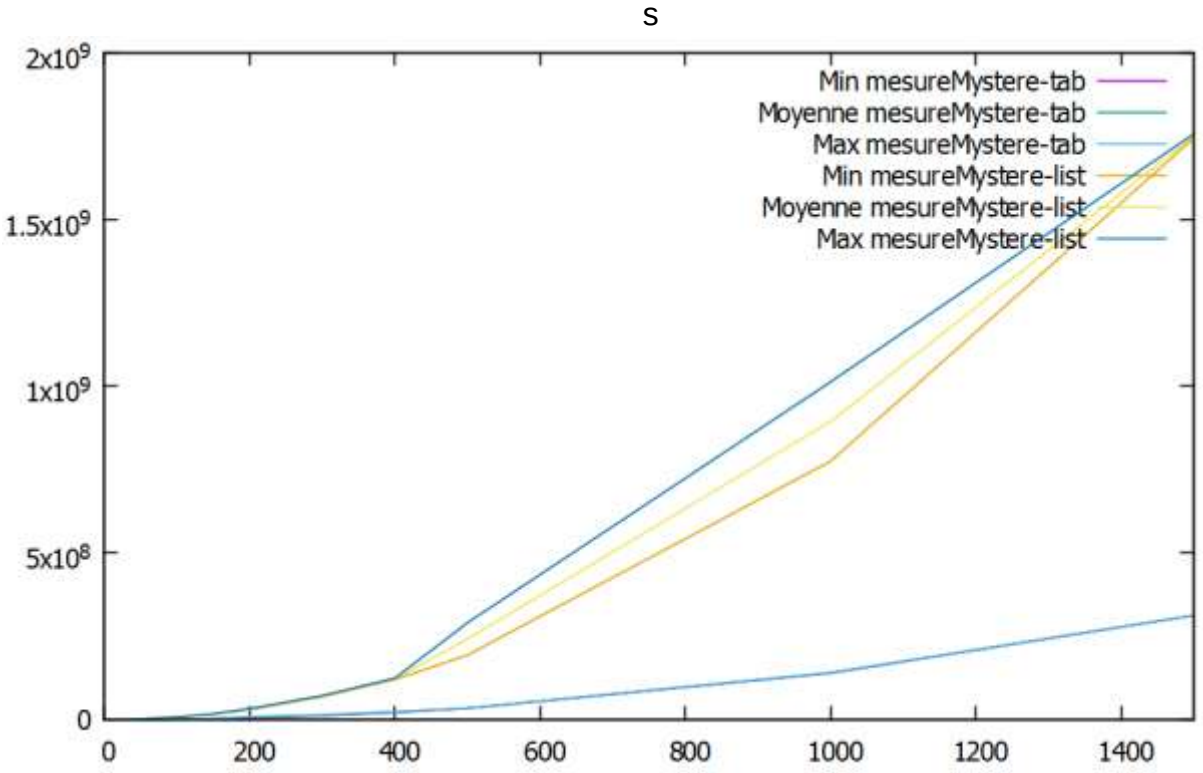
Sur ces graphiques (mesureMinSimple et mesureMinTri), on remarque que l'exécution sur les tableaux est plus rapide pour la fonction mesureMinTri ce que l'on a affirmé dans la partie précédente. De plus on remarque que pour les listes, l'exécution est nettement plus rapide pour la fonction mesureMinTri.

Fonction recherche()-- complexité en O(n)



Fonction mystere() -- complexité en O(2n² + nlog(n))

Note : pour cette fonction nous avons utilisé un tableau avec des valeurs plus petite et de taille plus petite car l'exécution était beaucoup trop longue sur les données fournies.



Pour cette fonction on remarque encore une fois que l'exécution sur les tableaux est beaucoup plus rapide car la courbe d'exécution semble stable pour les tableaux alors que celle des listes croît très rapidement.

D'après ces graphiques, on peut en déduire que les tableaux sont plus efficaces que les listes, en effet pour chacune des fonctions citées précédemment on remarque que l'exécution est toujours plus rapide pour les tableaux.