Introduction

Pour ce devoir, nous avons choisi de faire notre étude de performance sur des algorithmes qui déterminent la primalité d'un nombre. Pour ce faire, nous avons choisi un algorithme déterministe (algorithme de divisions successives) et deux algorithmes probabilistes (algorithme de *Miller-Rabin* et algorithme de *Solovay-Strassen*).

Les algorithmes probabilistes sont des algorithmes qui vérifient certaines propriétés qui sont propres à tous les nombres premiers pour déterminer si le nombre testé peut être considéré comme "*pseudopremier*". Il est donc possible qu'un nombre composé satisfasse également les propriétés testées et que le test le considère comme pseudopremier. Toutefois, certains algorithmes permettent de refaire le test à plusieurs reprises avec des paramètres différents dans l'optique de réduire la probabilité d'erreur. C'est le cas de Miller-Rabin et Solovay-Strassen.

Avec un nombre assez grand d'itérations d'un test probabiliste, on peut atteindre un niveau de certitude assez grand pour considérer le nombre comme "pratiquement premier" (dépendamment du contexte d'utilisation). Par exemple, avec 40 itérations de Miller-Rabin, il y a une probabilité de 2^{-80} de dire qu'un nombre *n* est pseudopremier alors qu'il est composé, c'est-à-dire approximativement $8.27\,x\,10^{-25}$. Pour mettre cette probabilité en perspective, imaginons la situation très optimiste où un test déterministe retourne son résultat booléen (un seul bit) sur un processeur et où le résultat n'a besoin d'être conservé que pour un seul cycle. Si le processeur a une vitesse de *2.4 GHz* et 291 millions de transistors, la probabilité qu'un rayon cosmique (une particule avec beaucoup d'énergie qui se déplace à travers l'Univers à grande vitesse) frappe le bon transistor au bon moment pour changer la valeur de retour de *false* à *true* est d'approximativement $1.8\,x\,10^{-24}$. Il y a donc plus de chances qu'un facteur physique influence le test que d'obtenir une erreur par un test probabiliste.

La méthodologie pour un test probabiliste sur *n* est la suivante :

1. Choisir aléatoirement un nombre *a*
2. Tester une propriété valide pour tous nombres premiers par rapport à *a* et *n*. Si la propriété n'est pas satisfaite, on indique que *n* est composé.
3. Répéter l'étape 1 jusqu'à ce que le nombre d'itérations désiré soit atteint

Description des algorithmes

Divisions successives

L'algorithme de divisions successives est l'algorithme de test de primalité le plus simple à comprendre. Cet algorithme consiste à diviser successivement la valeur par tous les nombres entre 2 et sa racine carrée. Il est inutile de dépasser sa racine carrée puisque, à chaque fois que nous testons pour un facteur *k*, nous obtenons la réponse pour le facteur $\frac{n}{k}$ en même temps.

La complexité algorithmique de cet algorithme dans le pire cas est $O(\sqrt{n})$.

Miller-Rabin

Gary L. Miller a développé un algorithme de test de primalité qui est déterministe (à condition que l'hypothèse de Riemann étendue soit vraie). Michael O. Rabin a proposé une version pour le rendre probabiliste. Nous nommerons cette version *algorithme de Miller-Rabin*.

Pour une certaine base $a \in]1,\ n-1[$, on commence avec le *petit théorème de Fermat* pour *n* premier :
$a^{n-1} = 1\ (mod\ n)$
On prend ensuite la racine unitaire dans le corps fini Z/Zn (si *n* premier) pour vérifier que chaque racine est soit 1 ou $-1\ (mod\ n)$.
Donc, si $n-1\ =\ 2^s d\ (d\ impair\ et\ n\ premier)$, on a *soit* :

- $a^d\ =\ 1\ (mod\ n)$
  c'est-à-dire : si nous obtenons 1, nous savons que nous obtiendrons 1 par la suite pour tous les *r* également.

ou

- $\exists r \in [0, s-1]\ |\ a^{2^r d}\ =\ -1\ (mod\ n)$
  c'est-à-dire : s'il y a un -1 pour un certain *r*, nous obtiendrons 1 par la suite pour tous les nombres > *r*.

L'algorithme utilise la contraposée :
n est composé si :

- $a^d \neq 1\ (mod\ n)$

et

- $a^{2^r d} \neq -1\ (mod\ n),\ \forall r \in [0, s-1]$

Il peut être démontré que, pour chaque nombre *n* composé impair, au moins $\frac{3}{4}$ des bases *a* choisies seront *témoins* de la composition de *n* (c'est-à-dire que nous avons une probabilité de $\frac{1}{4}$ de tomber sur une base qui fera passer une itération du test alors que *n* est en fait composé). La probabilité d'erreur est donc 4^{-k}, où *k* est le nombre d'itérations de Miller-Rabin.

L'algorithme a une complexité algorithmique au pire cas de $O(log^3 n)$.

Solovay-Strassen

Robert M. Solovay et Volker Strassen ont développé un algorithme probabiliste qui se base sur le critère d'Euler :

$a^{\frac{n-1}{2}} = (\frac{a}{n}),$ *où* n *est premier et* $(\frac{a}{n})$ *représente le symbole de Legendre*

L'idée est donc de prendre un entier $a \in]1,\ n-1]$ et de vérifier que le critère d'Euler est maintenu.

Le symbole de Legendre $(\frac{a}{n})$ donne 1 si $\exists x \mid x^2 = a\ (mod\ n)$ (***a*** est alors dit résidu quadratique), 0 si $a = 0\ (mod\ n)$, -1 sinon (si ***a*** n'est pas un résidu quadratique).

Le symbole de Jacobi est une généralisation du symbole de Legendre pour un *n* impair quelconque, en prenant le produit des symboles de Legendre avec chacun des facteurs premiers de *n* comme parties inférieures des symboles.

Pour l'explication derrière le critère d'Euler, on commence avec le petit théorème de Fermat :

Supposons que $a \neq 0\ (mod\ n)$ *et* n *premier*

$a^{n-1} = 1\ (mod\ n)$

On peut réécrire :

$a^{n-1} - 1 = (a^{\frac{n-1}{2}} - 1)(a^{\frac{n-1}{2}} + 1) = 0\ (mod\ n)$

Si a est un résidu quadratique, alors $\exists x \mid x^2 = a\ (mod\ n)$.

Alors on peut réécrire $a^{\frac{n-1}{2}} = (x^2)^{\frac{n-1}{2}} = x^{n-1} = 1\ (mod\ n)$ (petit théorème de Fermat)

Puisque les entiers Z/Zn forment un corps fini, $(a^{\frac{n-1}{2}} - 1)(a^{\frac{n-1}{2}} + 1) = 0\ (mod\ n)$ implique qu'un ou l'autre facteur est congruent à 0 (mod n). Donc, si ***a*** n'est pas résidu quadratique, alors $a^{\frac{n-1}{2}} \neq 1\ (mod\ n)$, alors le premier facteur ne peut pas être nul. C'est alors le deuxième facteur qui est 0 (mod n). Donc $a^{\frac{n-1}{2}} = -1\ (mod\ n)$ pour un ***a*** non-résidu quadratique.

Il est possible de démontrer que la probabilité d'erreur pour chaque itération de ***a*** est de $\frac{1}{2}$, ce qui donne une probabilité d'erreur de 2^{-k}, où *k* est le nombre d'itérations de Solovay-Strassen.

L'algorithme a une complexité algorithmique au pire cas de $O(log^3 n)$.

Hypothèses

Nous avons l'hypothèse que durant les trois tests notre ordinateur aura sensiblement la même charge de travail extérieure. De plus, tous les nombres que nous utilisons sont impairs, puisque nous travaillons avec de gros chiffres et les nombres pairs sont triviaux.

Les nombres sont générés avec une distribution uniforme de bits entre 512 et 1024 bits pour les algorithmes probabilistes et entre 10 et 40 bits pour l'algorithme déterministe. Nous utilisons un générateur de nombre du système d'exploitation pour avoir une bonne entropie.

Nous imposons comme contrainte pour les algorithmes probabilistes d'avoir une probabilité d'erreur $\leq 4^{-40}$.

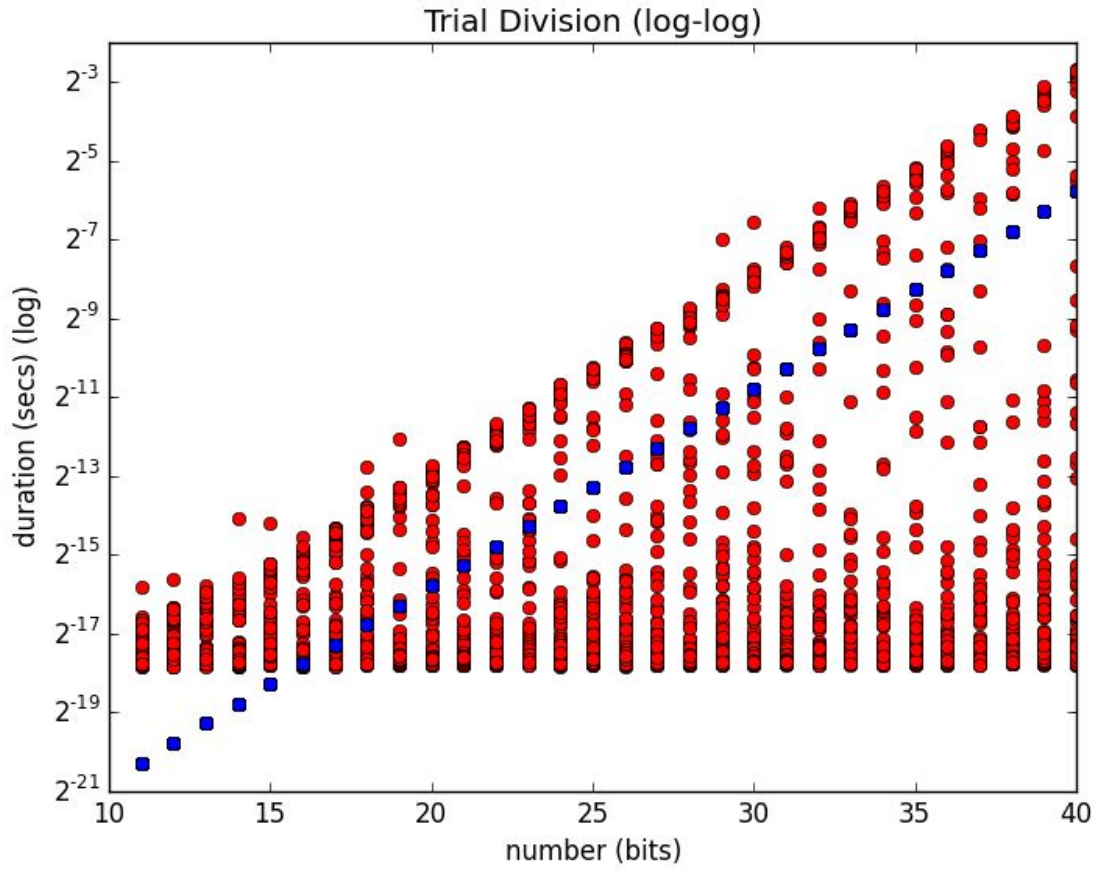
Comparaison

Pour commencer, nous allons comparer nos résultats expérimentaux avec les réultats théoriques.
Les graphiques qui suivent comportent des points rouges pour les données expérimentales et des points bleus pour les données théoriques. Les axes sont sur une échelle logarithmique (base 2) pour pouvoir afficher des nombres d'une telle amplitude. Cela veut donc dire que les courbes n'ont pas une allure habituelle.
Nous utilisons des échantillons de 3000 nombres pour chaque test.

Nous avons quelques données aberrantes qui, nous pensons, ont été causées par d'autres processus sur l'ordinateur.

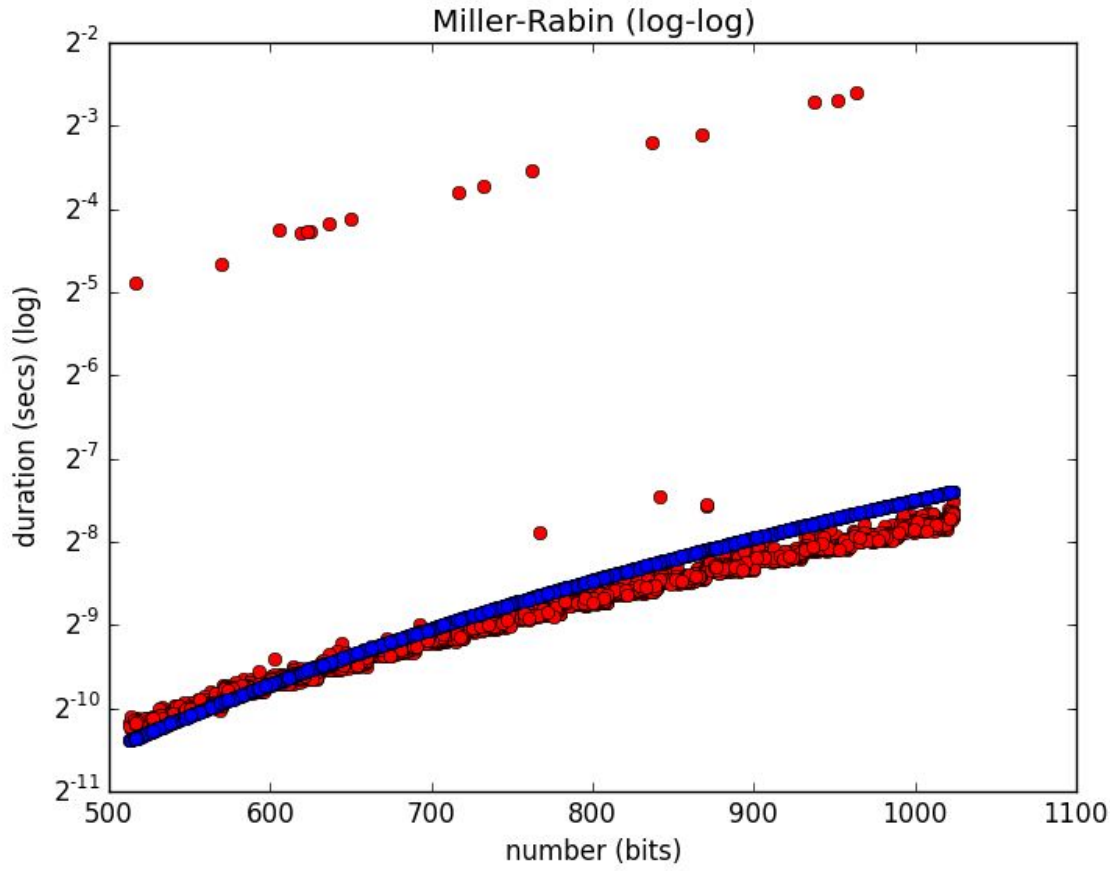
Divisions successives

L'algorithme de divisions successives n'atteint pas aussi souvent le pire cas (il doit uniquement tester $\sqrt{n}$ facteurs lorsque *n* est premier ou un carré parfait). C'est pour cette raison qu'il y a plusieurs points qui ne suivent pas la courbe théorique du pire cas. L'allure de la courbe théorique est très semblable à la courbe expérimentale.



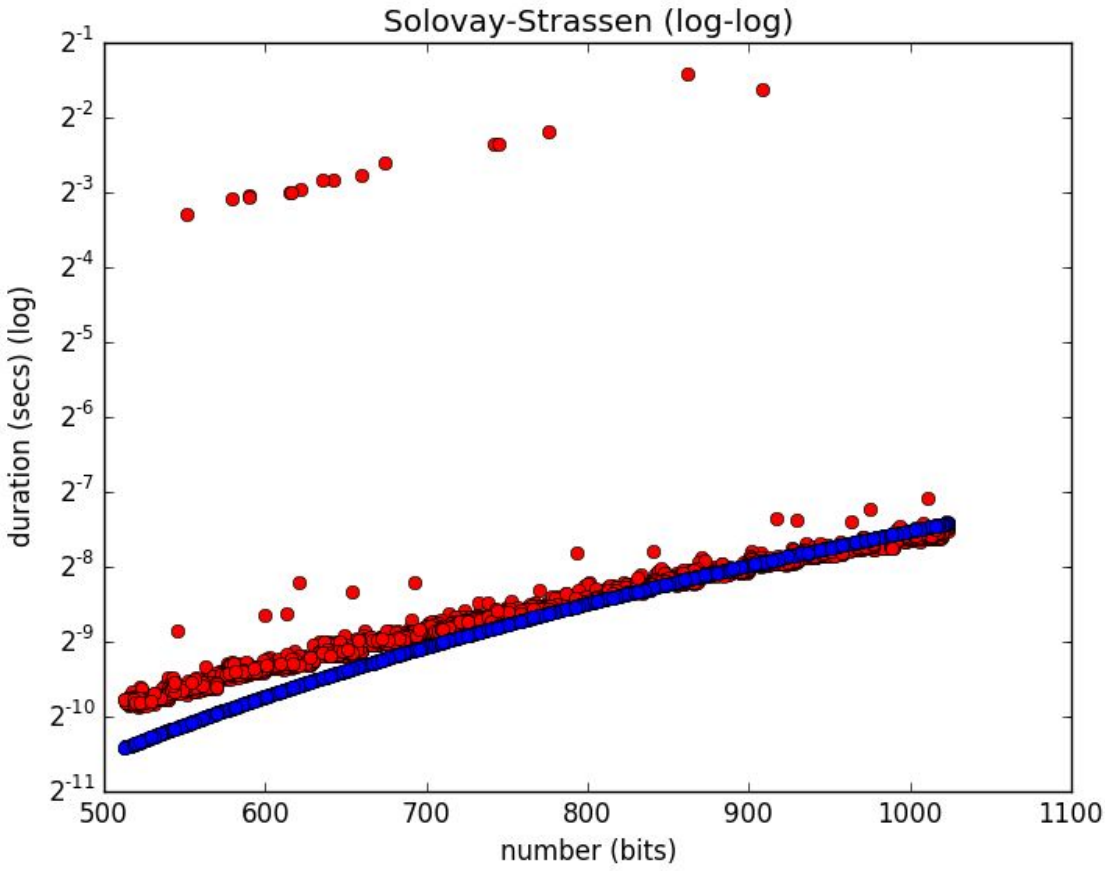
Miller-Rabin

L'allure de la courbe théorique est, ici aussi, très semblable à la courbe expérimentale. Les deux semblent suivre $O(\log^3 n)$.



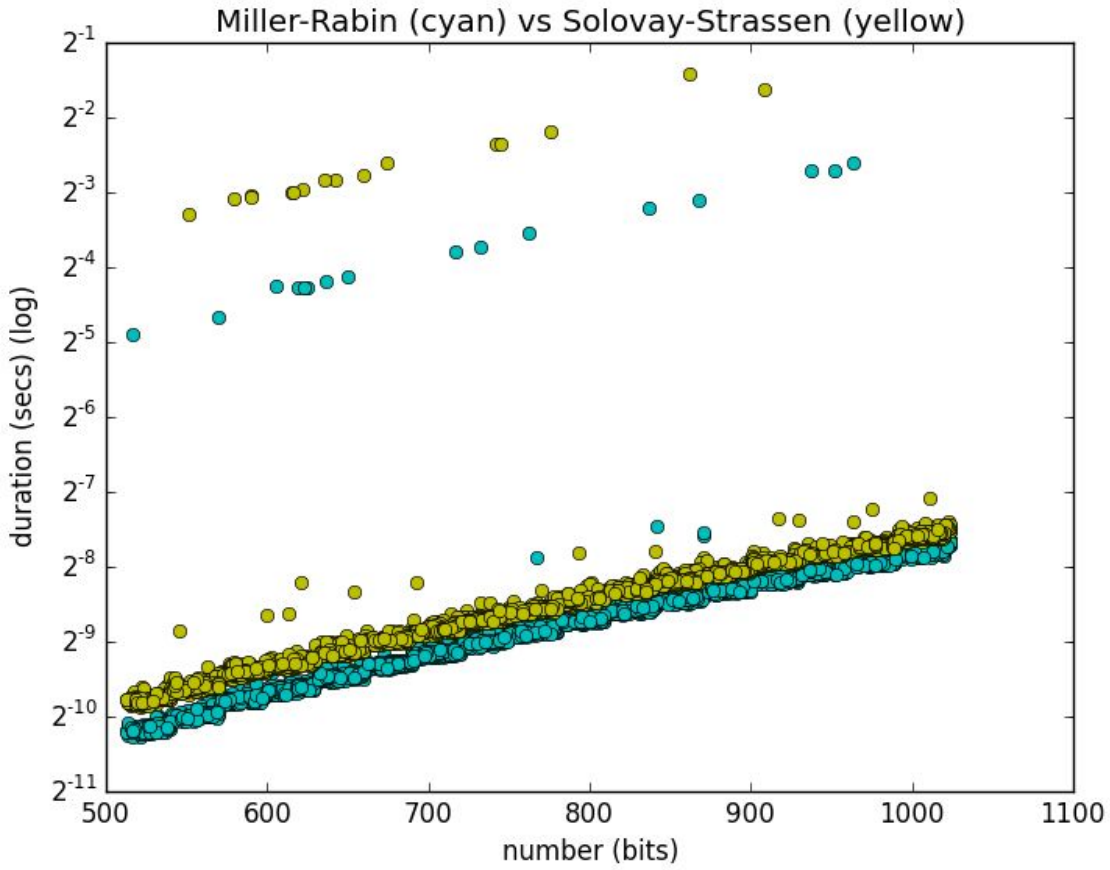
Solovay-Strassen

La courbe théorique est semblable à la courbe expérimentale. Par contre, la comparaison est un peu moins exacte que celle de Miller-Rabin. La courbe des données expérimentales semblent tout de même suivre $O(\log^3 n)$.



Comparaison Miller-Rabin vs Solovay-Strassen

Ces deux courbes sont très proches. Notre implémentation de Miller-Rabin est sensiblement plus rapide que notre implémentation de Solovay-Strassen.



Outils utilisés

Voici les particularités de l'ordinateur que nous avons utilisé pour faire nos tests :

- CPU : Intel(R) Core(TM) i7-2760QM CPU @ 2.40GHz
- RAM : 8 GB
- OS : Linux 4.1.2-2-ARCH

Nous executons le code dans la machine virtuelle standard de Python 3.4.3. Nous avons utilisé *MathPlotLib* pour générer automatiquement nos graphiques. Nous avons utilisé la bibliothèque *timeit* de Python pour calculer le temps d'execution de nos algorithmes. Nous avons utilisé le générateur de nombres pseudo-aléatoires de Python pour produire nos données de tests. Ce générateur de nombres utilise le générateur de nombres pseudo-aléatoires du système d'exploitation pour s'assurer d'avoir beaucoup d'entropie.

Conclusion

La conclusion la plus intéressante ici était de réaliser à quel point les algorithmes probabilistes sont plus rapides que les algorithmes déterministes. Nous ne pouvions tout simplement pas faire les tests avec l'algorithme par division avec plus de 50 bits, car cela prenait trop de temps. En comparaison, les algorithmes probabilistes travaillaient avec des nombres qui avaient jusqu'à 1024 bits (soit un facteur d'environ 1.6 x 10^293 comparé aux nombres de 50 bits).