On utilise les congruences pour cribler les entiers de **1 à n = 15k (*k* ∈ $\mathbb{N}^*$)** par l'application de la fonction G relative au crible de Goldbach. Les entiers non congrus à 2n[P_i] restituent les nombres premiers q appartenant à [n ; 2n] dont le crible G dénombre.

L'originalité de ce criblage modulo 15, réside principalement en utilisant en partie le principe d'Ératosthène et en ce qu'il crible les entiers **≡ 2n[P_i]** de **1 à n** en les marquant d'un **0** : par famille, dont le nom sera préfixé par fam par la suite, de même ces entiers congrus ou pas : **seront** représentés par des **1** ou des **0**. Pour ce faire on utilise les restes R_i de la division de 2n par P_i ; dont la fonction serra d'indexer la position de départ de ces nombres premiers P_i qui criblent de 1 à n.
Je suis parti d'une condition nécessaire : tout nombre premier q supérieur à 30 est de la forme : 30k+1, 30k+7, 30k+11, 30k+13, 30k+17, 30k+19, 30k+23 ou 30k+29
et le crible les répartit en 8 familles :
fam_1 ={q premiers > 30, q = 1 [30]}
fam_7 ={q premiers > 30, q = 7 [30]}
fam_11={q premiers > 30, q = 11 [30]}
fam_13={q premiers > 30, q = 13 [30]}
fam_17={q premiers > 30, q = 17 [30]}
fam_19={q premiers > 30, q = 19 [30]}
fam_23={q premiers > 30, q = 23 [30]}
fam_29={q premiers > 30, q = 29 [30]}.

Petit rappel :

[« Condition nécessaire et Rappel : arithmétique modulaire dans les nombres réels **R**:

Soient a,P,q,r ∈ **R** , a = Pq+r ; d'où on peut écrire a ≡ r [P] or : a – r = Pq ; donc P divise a – r .
On dit aussi que ''a et r sont congrus modulo P''. »]

*<u>On calcule les congruences de ces entiers **A** non nuls et **2n**, de **1 à n** avec</u>* **<u>le crible G</u>**:

Si **A et 2n** partage le même reste R_i de **2n** par P_i, **A** est ≡ R_i[P_i] tous ces **A** seront représenté par **0**.

Principe de base : on calcule le reste R de 2n par $P_i \leq \sqrt{2n}$; R_i est donc un entier **A** appartenant à **[1 ; n].** Les Entiers avant criblage, seront représentés par des **1**.
On part de ce reste R_i de **2n** par P_i où l'on aura remplacé **1** par **0**, puis on remplace les **1** par **0 :** par pas de P_i de R_i à **n** ces entiers appartenant à **[1 ; n]**.

On aura donc marqué **0** tous les entiers congrus à **2n** modulo $P_i \leq \sqrt{2n}$; de **1 à n**.
*Tous les **1** sont donc non congrus à **2n** modulo[P_i]. Ils restituent par conséquent les nombres premiers **q** appartenant **à[n ; 2n].***

<u>Par famille en progression arithmétique de raison 30</u> : on calcule le reste R_i de la division de **2n** par $P_i \leq \sqrt{2n}$; puis on calcule : **J = R_i + k*P_i** ; si **j %30 == Fam** ; on calcule l'index : **j//30 = idx** et on crible les entiers congrus à R_i [P_i]par pas de P_i , de **idx** à **n//30** : en remplaçant le **1** par **0**.
Fam est une des 8 familles, de premier terme : **{1,7,11,13,17,19,23,29}**

La limite **n=15k + a** est fixée et en fonction de la forme de **n**, on fixe la famille **= Fam.**

Cette limite progresse **modulo 15**. On va donc parcourir l'ensemble des nombres pairs **2n** pour **n ≥ 150**. La conjecture étant vérifiée de **6 à 300**.

Pour démontrer la conjecture on va utiliser le principe de fonctionnement du **crible G** dans les congruences et cette particularité : qui décale les congruences d'un rang sur les entiers suivants lorsque **n** augmente de **15**.
Ce qui revient à décaler la congruence d'un entier, **sur son successeur** modulo 30.

Une seule Fam est suffisante pour vérifier la conjecture en fonction de la forme de **n**.
Pour ce faire, on utilise les deux cribles : Goldbach et Ératosthène pour la même limite **n** de **1 à n** et la même Fam fixée, conditionnée par la forme de **n**.

1_) **On suppose que la conjecture est fausse**, on va prouver le contraire. Pour ce faire on va montrer principalement que le fonctionnement du crible G, en duo avec le crible d'Ératosthène conduit à une contradiction, prouvant que l'infirmation de la conjecture est fausse !

On va détailler les différentes parties du criblage, une condition suffisante est le criblage des congruences des entiers de **1 à n** principalement ; entiers criblé aussi préalablement, par le crible Ératosthène.

La **fonction G** du crible de Goldbach, fait ressortir une particularité. Propriété que l'on va utiliser pour résoudre la conjecture, avec cette fonction qui crible les congruences de ces entiers.

On fixe la **famille 7[30]**, limite n=15k + a ; a=7. 2n =30k +14. 7+37 = 44
(« cette Fam est donc suffisante pour 15k +7, sachant que l'on pourrait disposer des deux autres Fam 1[30] et 13[30] qui sont complémentaires. **»)**

Étape 1_) : (« *selon le principe d'Ératosthène dans les congruences*. »)

lorsque l'on crible **modulo 15** les entiers de **7 à n** en progression arithmétique de **raison 30** . Ces entiers de « *Goldbach* » sont représentés par des [11111111…] de **7→n//30** que l'on **crible avec** la fonction **G :** cette dernière remplace le **1** par **0** si cet entier est **≡ 2n[P_i]** puis par pas de **$P_i \leqslant \sqrt{2n}$** on remplace **1** par **0**, en partant de l'index déterminé par le programme du **crible G** et tel que défini ci-dessus**.**

Ces entiers sont les entiers d'Ératosthène de **7→n//30 contenant les nombres premiers P : [,1,] et leurs multiples [,0,]** .

Ce qui donne en exemple ci-dessous, cette image de **7 à n // 30** extraite du document, où la fonction G remplace le **1** par **0** si l'entier est congru à **R_i** modulo **[P_i]** par pas de **$P_i \leqslant \sqrt{2n}$;** relative à la ligne **n°1 étape 1_)**

n° 1 : [**0**, **1**, **1**, **0**, **0**, **1**, **1**, **1**, **1**, **0**, **0**, **0**, **1**, **0**, **0**, **0**, **1**, **1**, **0**, **1**, **0**, **0**, **0**, **1**, **1**, **0**, **0**, **1**, **1**, **1**] 15k + a fonction G

étape 2_)
la ligne n°2 sont les entiers de la même famille avec la même limite. Mais criblé par le **crible E** d'Ératosthène de **7→n//30**

n°2 : [**1**, **1**, **1**, **1**, **1**, **1**, **0**, **0**, **0**, **1**, **1**, **1**, **1**, **1**, **0**, **1**, **1**, **0**, **1**, **1**, **1**, **0**, **0**, **0**, **1**, **1**, **1**, **0**, **0**, **1**] 15k + 7 : fonction E.

La fonction **E** remplace le **1** par **0** par pas de **$P_i \leqslant \sqrt{n}$** ; si c'est un multiple de **P_i** (« On connaît ce principe, pour le **crible É** avec **Pi ⩽ √n** et **non racine de 2n** »). (« L'ordre entre n°1 et n°2 n'a aucune importance. »)

n° 1 : [**0**, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1] 15k + 7 fonction G
n°2 : [**1**, **1**, **1**, **1**, **1**, **1**, 0, 0, 0, **1**, **1**, **1**, **1**, **1**, 0, **1**, **1**, 0, **1**, **1**, **1**, 0, 0, 0, **1**, **1**, **1**, 0, 0, **1**] 15k + 7 : fonction E
n°3: [**1**, **1**, **1**, **1**, **1**, **0, 0, 0**, **1**, **1**, **1**, **1**, **1**, **0**, **1**, **1**, **0**, **1**, **1**, **1**, **0**, **0**, **0**, **1**, **1**, **1**, **0, 0,** **1**] 15k + 7 ; 8 couples **P + q = 2n**

étape 3_) :
la ligne n°2 d'Ératosthène criblée, **va être** à nouveau criblé **mais** par la fonction G, ou plus simplement : il suffit de mettre en rouge chaque élément d'Ératosthène correspondant au **0** de la ligne **n°1** des entiers congrus à R_i [P_i] par la fonction G..
(« **ex :** on superpose la ligne **n°1** sur la ligne **n°2** ; *ce qui donne* **n°3** criblée par la fonction **G.** **»)**

Dans les entiers de Goldbach ligne **n° 1**, ces **0** et **2n** sont **≡ [P_i]**. Donc en aucun cas ces entiers congrus, ne peuvent restituer des nombres premiers **q** appartenant à **[2n;n]**

Les **1** dans **n°1,** sont non congrus à 2n[P_i], ils représentent donc la restitution des nombres premiers **q** appartenant à **[n ;2n]** les **1** représentent aussi les nombres premiers **P** dans l'ensemble **n° 2** d'Ératosthène de **[7 à n] ;** ainsi que les **1** dans **n°3** ie : après le 3^{éme} criblage par la fonction **G.** **Où ces 1** vont donc représenter les couples **(p+q) = 2n** pour la limite fixée dans cette Fam.

Ce qui répond en partie à la conjecture pour cette limite **n = 907 :** les **1** de la ligne **n°3** après le troisième criblage, décomposent ***2n = 1814*** en somme de deux premiers **P+q .**

Or que se passe-t-il si **n** augmente de **15** soit : **15(k+1) + 7** ? les congruences se décalent d'un rang sur l'entier suivant !

Les entiers congrus à $\mathbf{R_i}$ **[$\mathbf{P_i}$]** lors du criblage de **15k +7** augmentent de 30, donc **après l'application** de la fonction G, ligne 4 : les congruences sont décalées d'un rang vers la droite de **7 à n**, pour l'ensemble de ces entiers de Goldbach criblés ; ces entiers congrus ou pas **sont ceux du criblage précédent, augmentés de 30**. Que la fonction G a vérifiée ligne **n°4** et plus généralement lignes **$\mathbf{n°_n}$ lorsque n → ∞.**

Un entier non congru à **2n** modulo $\mathbf{P_i}$ qui restitue **q** dans **[2n ;n]** ; **q** reste premier lorsque **2n** augmente de **30** d'où son complémentaire 2n – q augmente aussi de 30 par obligation ; provoquant ainsi ce décalage d'un rang des congruences sur les entiers successeurs appartenant à **[7 ; n].**

Lorsque la fonction G pour **15(k+1) + 7** recalcule les congruences de ces entiers de Goldbach, avec les nouveaux restes $\mathbf{R_i}$; ces entiers qui n'étaient **pas congrus à 2n modulo $\mathbf{P_i}$** et qui augmentent de 30 : **ne le seront pas d'avantage pour 2n + 30** ; Le contraire serait absurde, cela changerait la propriété des entiers appartenant à [n ;2n].
Ce qui permet dans un premier temps, de prédire pour **15(k+1) +7** les entiers successeurs augmentés de 30 qui sont congrus ou pas à **(2n + 30)[$\mathbf{P_i}$], conséquence de ce décalage d'un rang des congruences.**
Prédiction faisable sur plusieurs criblages successifs sans risque d'erreur.

Il vient que quel que soit **n** et la **fam** fixés, cela provoque pour les **1** et **0** ce décalage d'un rang par cette application de la fonction G, relatif **à 15(k+1) +7** donc à **30(k+1) +14**. Ce que nous montre l'illustration ci-dessous de façon formelle et que l'on peut vérifier :

On réitère pour les deux autres ligne **5** et **6 :**
2ème image : où on constate que l'on va répliquer ligne **n°4** à la droite du **0,** l'image précédente de **n = 15k +7** des entiers criblés et qui ont par obligation augmenté de 30 **mais qui donne bien la même image** ; diminué du dernier élément, sur la ligne **n°4** et comme inconnue le ou les premiers éléments ligne **n°4,** si on prédit plusieurs criblage successifs**,** 15(k+1) à 15(k+4) par exemple.

n° 4 : [0, **0**, **1**, **1**, 0, 0, 1, 1, 1, **1**, 0, 0, 0, **1**, 0, 0, 0, 1, 1, 1, 0, **1**, 0, 0, 0, **1**, **1**, 0, 0, 1, **1**] 15(k+1) + 7
n°5 : [**1**, **1**, **1**, **1**, **1**, **1**, 0, 0, 0, **1**, **1**, **1**, **1**, **1**, 0, **1**, **1**, 0, **1**, **1**, **1**, 0, 0, 0, **1**, **1**, **1**, 0, 0, **1**] c'est la même image que la précédente
n°6: [1, **1**, **1**, **1**, 1, 1, 0, 0, 0, **1**, 1, 1, 1, **1**, 0, 1, 1, 0, **1**, 1, **1**, 0, 0, 0, **1**, **1**, 1, 0, 0, **1**] 15(k+1) + 7 on a répliqué l'image **n°3** avec les congruences décalées d'un rang, relatif au criblage des entiers **pour n = 15k +7.** Avec comme conséquence **: une variation négligeable** du nombre de couples **p+q =2n. 9** dans ce cas précis.

(« On comprend le principe de fonctionnement de ces deux fonctions G et E, où on a vérifié formellement **que l'on a reproduit** l'image **3** des entiers qui ont vérifié la conjecture pour **n=15k +7**, **avec les congruences** décalée **d'un rang,** car 2n a augmenter de 30 par conséquent les congruences des entiers précédents, se décalent sur leurs successeurs augmentés de 30.»)

De part ce constat et ce décalage d'un rang des congruences pour 15(k+1) +7.

Supposons que la conjecture soit fausse quel que soit un entiers n = 15(k+1) + a et quel que soit Fam fixée:

Lors du troisième criblage ligne **n°6** pour **n = 15(k+1) +7** : Tous les **1** d'Ératosthène doivent devenir des **0** donc des entiers **≡ 2n[$\mathbf{P_i}$]** par l'application de la **fonction G**.!
Sans avoir besoin d'utiliser le crible G, pour le vérifier : ligne **n°6** ce décalage d'un rang doit mettre le **0** sur chaque **1** de la ligne **5** d'Ératosthène...sinon cela contredit cette supposition !

D'où au minimum : une condition obligatoire est nécessaire, il ne faut pas de **1** consécutifs ou encore pas de **1** précédant un **0**. Afin que le décalage d'un rang des **1** de Goldbach ne se superposent pas sur les **1** consécutif d'Ératosthène. De la même façon, il ne faut pas de **1** consécutifs ou précédent un **0,** qui viendrait se décaler sur un **1** d'Ératosthène, **car en** se décalant, cela libère de par ce fait la congruence du **1** qui était marqué **0.**
En vertu de ces deux cribles : il s'agit d'un couple de premiers **(p+q) = (30(k+1)+14)** ce qui contredirait la supposition.

Or : il y a des **1** consécutifs ainsi que des entiers non congrus à **2n[$\mathbf{P_i}$]** en progression arithmétique de raison 30, de **7 à n**
Qui se vérifie avec les images précédentes des criblages successifs relatif à ces deux fonctions E et G :**15k , 15(k-1) , 15(k-2) ...etc** d'où la supposition est fausse.!
Ce qui contredit cette première hypothèse, mais aussi de par le fait qu'à chaque nouveau criblage lorsque **n** augmente de **15**, **on repart du début** avec tous les nombres premiers consécutifs, ainsi que les entiers non congrus à **2n[P_i]** ie : congruences consécutives décalées d'un rang. C'est-à-dire des nombres premiers d'Ératosthène non congrus 2n[P_i] consécutifs…. !

On peut penser que ce n'est pas suffisant. On pourrait supposer, qu'à partir d'une certaine limite **n** il n'y a plus de **1** consécutifs de **1 à n**, ainsi que des entiers non congrus à **2n [P_i]**. ???
Ce qui est absurde le crible recommence chaque fois du début, avec ses premiers consécutifs.!
Par conséquent c'est une condition insuffisante qui ne permet pas de remettre en cause la contradiction ci-dessus.

De plus, il y aura toujours un **1** précédant un **0** ou successeur d'un **0 ;** ainsi que le décalage des congruences qui s'ensuit invaliderait l'hypothèse dans les premières valeurs de 7 à n
L'estimation du TNP avec la fonction : **n log n**, ***ne permet absolument pas de l'affirmer, de façon formelle suite à l'explication qui précède !***

Mais admettons : que l'on suppose que ce n'est pas formellement suffisant et sans le prouver.
Cette première hypothèse contredite est une Conséquence de ce qui suit.

Lorsque **n** progresse **[15],** où les entiers de Goldbach **congrus ou pas 2n[P_i]** augmentent de 30 et ce quel que soit **n ≥ 150** et quel que soit la famille choisie **fam = {1,7,11,13,17,19,23,29} , ce décalage d'un rang des congruences** sur les successeurs : est une conséquence de l'application de la fonction du **crible G** avec son principe de fonctionnement dans les congruences et non pas l'inverse..!
Propriété qui permet de garder la propriété des entiers appartenant à [n ; 2n], soit multiple de P_i ou nombre premier q, conformément au TFA. (« *Théorème Fondamental de l'Arithmétique* »).

Mais une deuxième contradiction peut être montrée :

Lors du criblage de **15(k+1) + 7**, il faut dans un premier temps marquer d'un **0 tous les 1** de Goldbach : **entiers qui n'étaient pas** congrus à **30k [P_i] marqués 1** précédemment **et ce** : **avec les nouveaux** restes **R_i** de **(30(k+1)+ 14)** par **P_i**.

Afin qu'ils soient **≡ (30(k+1)+14)[Pi]** ; donc **≠ 1** et par conséquent cela ne peut pas restituer **des nombres premiers q** appartenant **à [(30(k+1)+14);(15(k+1)+7)]** dans l'exemple illustré ci-dessus.

Mais : il faut aussi marquer tous les **0** du criblage précédent de **n = 15k +7**, c'est-à-dire **:** les entiers de Goldbach **≡ R_i [P_i],** donc qui étaient **≡ 2n[P_i] avec les R_i de ce criblage précédent**...
Sinon ils donneraient un **1**: **≢((30k+1)+14)[Pi]** ; c'est – à – dire q appartenant à [n ; 2n] ; quel que soit **n** et la **fam** fixés, ce qui serait contraire à cette supposition de conjecture fausse.

Par conséquent : il vient qu'ils sont marqués avec les même **P_i et** leur nouveau **R_i**…? **Ce qui est impossible** !

En première impossibilité : les **R_i** du criblage précédent qui ont remplacé les **1** en **0 ont changés** ....! **La congruence** aussi par obligation. **Ce qui contredit** la supposition !

En deuxième impossibilité : il y a le fait qu'étant donné qu'il s'agit des mêmes P_i avec leur nouveau **R_i** qui criblent ces congruences, ces derniers ne sont pas assez nombreux pour remplacer modulo 30 tous les **1** en **0 par pas de P_i ;** *il en serait de même dans Ératosthène ce qui serait absurde.*

Avec le décalage des congruences d'un rang qui s'en ai suivi... ; les **P_i** qui criblent n'ont pas changés ! Peut-être 1 de plus en fonction de racine de 2n qui évolue de façon plus que négligeable, mais qui ne peut avoir aucune incidence sur le criblage.

On a vérifié que la ligne des entiers d'Ératosthène ne se décale pas lorsque **n** augmente de**15** ; par contre cette ligne augmente d'un entier lorsque **n** augmente de **30**...Donc pour 15(k+2) +7
On prouve par-là que la supposition de l'infirmation de cette conjecture est fausse, inversement l'affirmation de la conjecture est donc vraie.

Une fonction euristique **n / (log n)²** donne une estimation minimale de solutions vérifiant la conjecture, si cette dernière est fausse il existe **n** tel que cette fonction est nulle = 0 , donc fausse.

La fonction heuristique **n / (log n)² n'est donc pas nulle suite à cette résolution.**

(« L'annexe jointe en deuxième document montre l'effet de la fonction de Goldbach de façon simple ; il faudrait pour que la conjecture soit fausse utiliser les **R_i des 4** criblages précédents ce qui est absurde, la division de 2n par Pi ne donne qu'un reste **R_i** par **P_i** et non plusieurs...! »)

On a une relation entre les nombres premiers **q** et **p** qui dépendent de leur congruence...c'est à dire qu'un nombre premier **q** à pour antécédent un entier [7; n] ; ≢2n[Pi] quel que soit **n[15]**

Il en va de même qu'il est impossible que le crible G ne remplace aucun ***,1,*** *par* ***,0,*** *le nombre d'entiers non congrus à 2n[P_i] qui restituent les premiers q[n : 2n]est < au nombres de premiers* ***P****[1 ;n].*

C'est-à-dire que : $\frac{N}{(log.2N)} < \frac{N}{\log N}$.

Théorème : *tout nombre pair 2n ≥ 6 peut s'écrire comme la somme de deux nombres premiers (p+q)*

Corollaire : tout nombre pair 2n ≥ 180 peut s'écrire comme la somme de deux nombres premiers (p+q) appartenant à une famille en progression arithmétique de raison 30 et de premier terme 1 ou P premier avec P appartenant à [7 ; 29].

Lefeu gilbert le 23 /03/2019 .

Salutations

En information complémentaire : *sont donnés les différents indexes (idx) de départ des deux fonctions G et E pour : 15k = 900 et fam 7 ;*

Afin de comprendre la différence entre les deux fonctions qui criblent et le principe de base du fonctionnement

Voici les indexes de départ de P_i du crible G ; pour n= 900 + a, a = 7 et fam = 7 : relatif aux j%30 == 7 « du premier exemple ci-dessous. »

Goldbach :

P_i = 7 , idx = 4 puis par pas de 7

P_i = 11 , idx = 10 puis par pas de 11 **→n//30**

P_i = 13 , idx = 0 puis par pas de 13 **→n//30**

P_i = 17 , idx = 3 puis par pas de 17 **→n//30**

P_i = 19 , idx = 14 puis par pas de 19

P_i = 23 , idx = 15 puis par pas de 23 départ.

P_i = 29 , idx = 9 puis par pas de 29

P_i = 31 , idx = 22 puis par pas de 31

P_i = 37 , idx = 9 puis par pas de 37 et en dernier P_i = 41 , idx > 914 d'où il ne peut cribler. $P_i \leq \sqrt{1814} = 42,....$

Ératosthène :

dans Ératosthène 7*31 = j ; j%30 ==fam, partent de idx **= 7,** puis par pas de 7 et de **31→n//30**

dans Ératosthène 11*17 = j ; j%30 == fam, partent de idx **= 6** puis par pas de 11 et de 17

dans Ératosthène 13*19 = j ; j%30 == fam, partent de idx **= 8** puis par pas de 13 et de19

dans Ératosthène 23*29 = j ; j%30 == fam, partent de idx **= 22** puis par pas de 23 et de 29

chaque j est donc un produit ≡fam[30] contrairement à Goldbach.

31 → *> racine de 907 ne marquera rien , ainsi que 23 et 29 qui ne marquent que l'idx de*

Dans Goldbach **:** $j = R_i + k*P_i$

Si j%30==fam ; alors :

j//30 = début d'index = idx

*P_i part de idx, par pas de P_i où on remplace le **1** par **0** →n // 30 . Puis on réitère avec P_i et R_i suivant.*

****

Annexe 1 :

La fonction 2 du théorème de Goldbach est une conséquence directe du TNP: (*logarithme naturel)*

π(G) : la fonction de compte du nombre de nombres premiers q[N ; 2N] , pour G un entier 2N = 30k ($k \in \mathbb{N}^*$)

π(G) vaut $\sim \lim_{N\to+\infty} \frac{N}{(log 2N)}$

Le TNP dit que $\pi(n) = \frac{N}{(log N)} + o\left(\frac{N}{\log N}\right)$, donc le nombre de nombres premiers dans $]N,2N]$ vaut

$$\pi(2n) - \pi(n) = N\left(\frac{2}{log(2N)} - \frac{1}{\log N}\right) + o\left(\frac{N}{\log N}\right)$$

$$= N \times \frac{2\log N - \log(2N)}{\log(2N)} + o\left(\frac{N}{\log N}\right)$$

$$= N \times \frac{2\log N - \log(2N)}{\log(2N)} + o\left(\frac{N}{\log N}\right)$$

$$= \frac{N}{(log 2N)} + o\left(\frac{N}{\log N}\right)$$

Le nombre de couples de nombres premiers (p,q) pour n[1 ; 29] vaut environ lorsque $\lim_{n\to+\infty}$:

$$\frac{\left(\frac{15k+a}{Log(15k+a)}\right)}{\mathrm{Ln}\left(2 * \left(\frac{15k+a}{Log\,(15k+a)}\right)\right)}$$

En fonction de la forme de **n** ; on appliquera un coefficient multiplicateur.

On utilise ces coefficients **en fonction des familles = fam** de nombres premiers criblées :

Pour **n=15k +a, a :**{ 1,7,11,13,17,19,23,29} coef = 0,375 ; ***ce qui donne 3 fam sur 8*** ≡ **a**[30] qui peuvent être criblées

Pour **n=15k +a, a :**{ 10,20} coef = 0,5 ; ***ce qui donne 4 fam sur 8*** ≡ **a**[30] qui peuvent être criblées

Pour **n= 15k +a, a :**{ 3,6,9,12} coef = 0,75 ; ***ce qui donne 6 fam sur 8*** ≡ **a**[30] qui peuvent être criblées

Pour n=15k +a , a = 0 pas de coefficient : ***les 8 fam peuvent être criblées***

On peut aussi appliquer plus simplement la fonction : $\lim_{n\to+\infty} \frac{15k+a}{(log(15k+a))^2}$

****

Annexe 2 :

Les deux programmes en Python :
Pour changer de Famille à cribler, on change le dernier paramètre : **7** ; par {**1.11.13.17.19.23.29**}

```
 # On crible
   E_Crible(premiers, n, 7)
```

----------***Ératosthène n =15k ou 15k + n en fonction de la famille choisie*** ne pas se tromper de fam !----

```
from itertools import product
from time import time


def candidate_range(n):
    cur = 5
    incr = 2
    while cur < n+1:
        yield cur
        cur += incr
        incr ^= 6  # or incr = 6-incr, or however


def eratostene(n):
    n = int(n**0.5)
    prime_list = [2, 3]
    sieve_list = [True] * (n+1)
    for each_number in candidate_range(n):
        if sieve_list[each_number]:
            prime_list.append(each_number)
            for multiple in range(each_number*each_number, n+1, each_number):
                sieve_list[multiple] = False
    #print(prime_list[3:])
    return prime_list[3:]


def demander_N():
    n = input("Donnez N: ")
    n = int(n.strip().replace(" ", ""))
    #n = int(30 * round(float(n)/30))
    #print(f"On prend N = {n}  (30 * {int(n/30)})")
    return n


def lprint(text="", liste=None):
    if len(liste) < 250:
        print(text + str(liste))


def E_Crible(premiers, n, fam):
    start_crible = time()

    # On génère un tableau de N/30 cases rempli de 1
    crible = n//30*[1]
    lencrible = len(crible)
```

```
    GM = [7,11,13,17,19,23,29,31]
    # On calcule les produits: j = a * b

    for a in premiers:
        for b in GM:
            j = a * b
            if j%30 == fam:
                index = j // 30  # Je calcule l'index,
        # On crible directement à partir de l'index
        for idx in range(index, lencrible, a):  # index qui est réutilisé ici...
            crible[idx] = 0
        #print(j)

    total = sum(crible)
    lprint("crible:", crible)
    print(f"Nombre premiers criblés famille {fam} : {total} ----- {int((time()-start_crible)*100)/100}")


def main():
    # On demande N a l'utilisateur
    n = demander_N()

    # On récupère les premiers de 7 à √N
    premiers = eratostene(n)
    #lprint("premiers:", premiers)
    #print(f"nombres premiers entre 7 et n: {len(premiers)}")

    start_time = time()
    # On crible
    E_Crible(premiers, n, 7)    # pour changer de fam changer le dernier paramètre (1,11,….29)
    temps = time()-start_time
    print(f"--- Temps total: {int(temps*100)/100} sec ---")

main()
```

**********Goldbach *: n =15k ou 15k + n en fonction de la famille choisie,* ne pas se tromper !******

Pour Goldbach : crible _G.T.Y_mod30 :

Attention de ne pas se tromper :

Exemple :

Je veux cribler n = 15k +1 ; soit 2n = 30k+2
30k +2 = soit 1+31 donc fam 1 ; ou 13+19 : donc fam 13 donne q =19[30] *et inversement*.

Pour n = 15k chaque fam criblera les entiers non congrus à 2n[P_i] q complémentaire = 30k
Fam 17, donne q = 13[30] *et inversement*. Fam 11, donne q = 19[30] *et inversement*…etc

---

```
from time import time
from os import system


def candidate_range(n):
    cur = 5
```

```
    incr = 2
    while cur < n+1:
        yield cur
        cur += incr
        incr ^= 6  # or incr = 6-incr, or however


def eratostene(n):
    n = int((2*n)**0.5)
    prime_list = [2, 3]
    sieve_list = [True] * (n+1)
    for each_number in candidate_range(n):
        if sieve_list[each_number]:
            prime_list.append(each_number)
            for multiple in range(each_number*each_number, n+1, each_number):
                sieve_list[multiple] = False
    #print(prime_list[3:])
    return prime_list[3:]


def demander_N():
    n = input("Donnez N: ")
    n = int(n.strip().replace(" ", ""))
    #n = int(30 * round(float(n)/30))
    #print(f"On prend N = {n}  (30 * {int(n/30)})")
    return n


def lprint(text="", liste=None):
    if len(liste) < 200:
        print(text + str(liste))


def GCrible(premiers, n, fam):
    start_crible = time()

    # On génère un tableau de N/30 cases rempli de 1
    crible = n//30*[1]
    lencrible = len(crible)

    # On calcule les restes: ri = 2*n/pi
    nbpremiers = len(premiers)
    n2 = 2*n

    for i, premier in enumerate(premiers):
        reste = n2 % premier
            # tant que ri % 30 != fam on fait ri += 2pi
        if reste % 2 == 0:
            reste += premier
        pi2 = 2*premier
```

```
        while reste % 30 != fam:
            reste += pi2
        # Ensuite on divise ri par 30 pour obtenir l'indexe
        reste //= 30
        # On crible directement avec l'index
        for index in range(reste, lencrible, premier):
            crible[index] = 0

    total = sum(crible)
    lprint("crible:", crible)
    print(f"Nombres non congru 2n[pi] {1} à {n} famille {fam} premiers de {n} à {n2}: {total}
----- {int((time()-start_crible)*100)/100}")


def main():
    # On demande N a l'utilisateur
    n = demander_N()

    # On récupère les premiers entre 7 et  √ 2N
    premiers = eratostene(n)
    #lprint("premiers:", premiers)
    #print(f"nombres premiers entre 7 et {int((2*n)**0.5)}: {len(premiers)}")

    start_time = time()
    # On crible
    GCrible(premiers, n, 7)  # pour changer de fam changer le dernier paramètre (1,11,….29)
    temps = time()-start_time
    print(f"--- Temps total: {int(temps*100)/100} sec ---")


main()
system("pause")
```

****

Annexe 3 :
ces deux programme en C++ utilisé avec code:bloc est l'Application Windows 64bite

Eratosthène :

```
// -*- compile-command: "/usr/bin/g++ -g goldbache.cc" -*-
#include <vector>
#include <iostream>
#include <cmath>
#include <stdlib.h>
#include <time.h>
using namespace std;
// fill Erathosthene sieve crible for searching primes up to 2*crible.size()*32+1
// crible is a (packed) bit array, crible[i] is true if 2*i+1 is a prime
```

```
// crible must be set to true at startup
void fill_crible(vector<unsigned> & crible,unsigned p){
  crible.resize((p-1)/64+1);
  unsigned cs=crible.size();
  unsigned lastnum=64*cs;
  unsigned lastsieve=int(std::sqrt(double(lastnum)));
  unsigned primesieved=1;
  crible[0] = 0xfffffffe; // 1 is not prime and not sieved (2 is not sieved)
  for (unsigned i=1;i<cs;++i)
    crible[i]=0xffffffff;
  for (;primesieved<=lastsieve;primesieved+=2){
    // find next prime
    unsigned pos=primesieved/2;
    for (;pos<cs;pos++){
      if (crible[pos/32] & (1 << (pos %32)))
              break;
    }
    // set mutiples of (2*pos+1) to false
    primesieved=2*pos+1;
    unsigned n=3*primesieved;
    for (;n<lastnum;n+=2*primesieved){
      pos=(n-1)/2;
      crible[(pos/32)] &= ~(1<<(pos %32));
    }
  }
}
unsigned nextprime(vector<unsigned> & crible,unsigned p){
  // assumes crible has been filled
  ++p;
  if (p%2==0)
    ++p;
  unsigned pos=(p-1)/2,cs=crible.size()*32;
  if (2*cs+1<=p)
    return -1;
  for (;pos<cs;++pos){
    if (crible[pos/32] & (1<<(pos%32))){
      pos=2*pos+1;
      // if (pos!=nextprime(int(p)).val) CERR << "error " << p << endl;
      return pos;
    }
  }
  return -1;
}

typedef unsigned long long ulonglong;

size_t ECrible(const vector<ulonglong> & premiers,ulonglong n,int fam){
  int cl=clock();
  size_t lencrible=n/30,nbpremiers=premiers.size();
  vector<bool> crible(lencrible,true);
  // ulonglong n2=2*n;
  vector<ulonglong> indices(nbpremiers);
  for (size_t i=0;i<nbpremiers;++i){
    ulonglong p=premiers[i];
    ulonglong produit;
    int GM[]={7,11,13,17,19,23,29,31};
    for (size_t j=0;j<sizeof(GM)/sizeof(int);j++){
      produit = p*GM[j];
      if (produit %30==fam){
              produit /= 30;
              break;
      }
    }
    indices[i]=produit;
  }
  ulonglong nslices=lencrible/3000000,currentslice=0;
  if (nslices==0) nslices=1;
  for (;currentslice<nslices;++currentslice){
    size_t slicelimit=currentslice+1;
    slicelimit=slicelimit==nslices?lencrible:(currentslice+1)*(lencrible/nslices);
    for (size_t i=0;i<nbpremiers;++i){
      ulonglong p=premiers[i];
      size_t index;
      for (index=indices[i];index<slicelimit;index+=p)
              crible[index]=0;
      indices[i]=index;
```

```
    }
  }
  size_t total=0;
  for (size_t index=0;index<lencrible;++index)
    total += int(crible[index]);
  cout << "Nombre premiers criblés famille " << fam << " plus petits que "<< n <<": " << total << " time " << (clock()-cl)*1e-6<< endl;
  return total;
}

int main(int argc,char ** argv){
  vector<unsigned> crible;
  ulonglong N;
  int fam=1;
  if (argc>1){
    N=atoll(argv[1]);
    if (argc>2)
      fam=atoi(argv[2]);
  }
  else {
    cout << "Syntaxe " << argv[0] << " N fam. Donnez N puis fam: " ;
    cin >> N;
    cin >> fam;
  }
  double sqrtN=unsigned(std::sqrt(double(N)));
  fill_crible(crible,sqrtN);
  vector<ulonglong> premiers;
  for (ulonglong p=7;p<=sqrtN;){
    premiers.push_back(p);
    p=nextprime(crible,p);
    if (p==unsigned(-1))
      break;
  }
  ECrible(premiers,N,fam);
  cin>>N;
}
```

Goldbach :

```
// -*- compile-command: "/usr/bin/g++ -g goldbachs.cc" -*-
#include <vector>
#include <iostream>
#include <cmath>
#include <stdlib.h>
#include <time.h>
using namespace std;
// fill Erathosthene sieve crible for searching primes up to 2*crible.size()*32+1
// crible is a (packed) bit array, crible[i] is true if 2*i+1 is a prime
// crible must be set to true at startup
void fill_crible(vector<unsigned> & crible,unsigned p){
  crible.resize((p-1)/64+1);
  unsigned cs=crible.size();
  unsigned lastnum=64*cs;
  unsigned lastsieve=int(std::sqrt(double(lastnum)));
  unsigned primesieved=1;
  crible[0] = 0xfffffffe; // 1 is not prime and not sieved (2 is not sieved)
  for (unsigned i=1;i<cs;++i)
    crible[i]=0xffffffff;
  for (;primesieved<=lastsieve;primesieved+=2){
    // find next prime
    unsigned pos=primesieved/2;
    for (;pos<cs;pos++){
      if (crible[pos/32] & (1 << (pos %32)))
              break;
    }
    // set mutiples of (2*pos+1) to false
    primesieved=2*pos+1;
    unsigned n=3*primesieved;
```

```
    for (;n<lastnum;n+=2*primesieved){
      pos=(n-1)/2;
      crible[(pos/32)] &= ~(1<<(pos %32));
    }
  }
}
unsigned nextprime(vector<unsigned> & crible,unsigned p){
  // assumes crible has been filled
  ++p;
  if (p%2==0)
    ++p;
  unsigned pos=(p-1)/2,cs=crible.size()*32;
  if (2*cs+1<=p)
    return -1;
  for (;pos<cs;++pos){
    if (crible[pos/32] & (1<<(pos%32))){
      pos=2*pos+1;
      // if (pos!=nextprime(int(p)).val) CERR << "error " << p << endl;
      return pos;
    }
  }
  return -1;
}

typedef unsigned long long ulonglong;

size_t GCrible(const vector<ulonglong> & premiers,ulonglong n,int fam){
  int cl=clock();
  size_t lencrible=n/30,nbpremiers=premiers.size();
  vector<bool> crible(lencrible,true);
  ulonglong n2=2*n;
  vector<ulonglong> indices(nbpremiers);
  for (size_t i=0;i<nbpremiers;++i){
    ulonglong p=premiers[i];
    ulonglong reste=n2 % p;
    if (reste %2==0)
      reste += p;
    ulonglong pi2=2*p;
    while (reste %30!=fam)
      reste += pi2;
    reste /= 30;
    indices[i]=reste;
  }
  ulonglong nslices=lencrible/3000000,currentslice=0;
  if (nslices==0) nslices=1;
  for (;currentslice<nslices;++currentslice){
    size_t slicelimit=currentslice+1;
    slicelimit=slicelimit==nslices?lencrible:(currentslice+1)*(lencrible/nslices);
    for (size_t i=0;i<nbpremiers;++i){
      ulonglong p=premiers[i];
      size_t index;
      for (index=indices[i];index<slicelimit;index+=p)
                crible[index]=0;
      indices[i]=index;
    }
  }
  size_t total=0;
  for (size_t index=0;index<lencrible;++index)
    total += int(crible[index]);
  cout << "Nombre premiers criblés famille " << fam << " entre "<< n << " et " << n2 <<": " << total << " time " << (clock()-cl)*1e-6<< endl;
  return total;
}

int main(int argc,char ** argv){
  vector<unsigned> crible;
  ulonglong N;
  int fam=1;
  if (argc>1){
    N=atoll(argv[1]);
    if (argc>2)
      fam=atoi(argv[2]);
  }
  else {
    cout << "Syntaxe " << argv[0] << " N fam. Donnez N puis fam: " ;
    cin >> N;
    cin >> fam;
```

```
  }
  double sqrt2N=unsigned(std::sqrt(2*double(N)));
  fill_crible(crible,sqrt2N);
  vector<ulonglong> premiers;
  for (ulonglong p=7;p<=sqrt2N;){
    premiers.push_back(p);
    p=nextprime(crible,p);
    if (p==unsigned(-1))
      break;
  }
  GCrible(premiers,N,fam);
  cin>>N;
}
```

Lefeu gilbert le 25/03/2019.