Préambule :

Pour toute limite $N \geq 4$ *fixée, on peut considérer la conjecture de Goldbach comme un algorithme de décomposition d'un entier naturel positif* **pair** $= 2N \geq 8$ *en somme de deux nombres premiers* $p' + q$.

Propriété des congruences *petit rappel,* $2n - A = B$ *:*

« On va utiliser les congruences pour définir les entiers A *de* 1 *à* n *tel que* $A \not\equiv 2n[P]$ *, afin de pas s'occuper des nombres premiers* q *de* n *à* $2n$ *, qui ne sont que les complémentaires de ces entiers* $A \not\equiv 2n[P]$ *par raport à* $2n$ *. En inversant le vecteur* B *des nombres* q *de* n *à* $2n$*, sur le vecteur* A *de* 1 *à* n *, tout nombre premier* q *a donc pour complémentaire :* $A \not\equiv 2n[P]$*, qu'il soit premier ou pas. »*

Il existe y et y' tel que : $2n = P*y + R$ *et* $A = P*y' + R \Rightarrow 2n - A = P*(y - y')$*; donc* P *divise* $2n - A = B$*. Inversement si* y *n'existe pas , alors* P *ne divise pas la différence* $2n - A = B \Rightarrow q$ *qui est donc un nombre premier* $\in [n;2n]$ *ayant pour antécédent* $A \not\equiv 2n[P]$*.*
Chaque entier A, peut être congru à 2n (modulo P) de façon unique, à l'ordre près de ses facteurs (TFA) *, avec* $P \leqslant \sqrt{2n}$.

[« Suivant ce qui suit : *tout nombres premier > 5 serra donc de la forme* $30k + i$.
Démontrons que tout nombre premier supérieur à 30 appartient à l'une de ces 8 familles (i) avec $(i) \in (1,7,11,13,17,19,23,29)$.
Soit p *un nombre premier supérieur à 30. Il n'existe donc pas de décomposition en nombre premier de* p.
30 étant le produit de 2, 3 et 5. Tout nombre premier supérieur à 30 n'est pas divisible par 2, 3, ou 5 et s'écrit donc de la forme $2^i.3^j.5^k + pr$ *avec* pr *non divisible par 2, 3 et 5 et inférieur à 30.*
Donc pr =1 ou $\in$ *[7 ; 29] étant l'un des nombres premiers > 1, inférieurs à 30 et différents de 2, 3, ou 5.*
Il en resulte que tout nombres premier > 5, est bien de la forme 30k + i ainsi que leur multiples .»]

(« *Plus généralement , on peut dire que pour toute Limite N ayant criblée les entiers positifs* $A \leq N$*, afin dénombrer le nombre de décompositions de* $2N$ *à l'aide des trois fonctions croissantes du TNP (ci-après).*
Alors on peut utiliser le résultat du criblage ≤ à cette limite N*, pour dénombrer le nombre de décompositions du nombre pair suivant:* $2n + 2 = p'+q$*, cela est* ***dû à la propriété récurrente de l'algorithme*** *de Goldbach,(«décalage d'un rang du champ des congruences lorsque* N *augmente de* 1*.») qui crée un système dynamique et va s'étendre sur plusieurs limites* N *successives lorsque* $N \rightarrow \infty$ *. ie:)* $\Rightarrow$ *les entiers* $2N$ *successifs, d'où l'impossibilité de supposer que* $2N + 2 \neq (p' + q)$.
Mais surtout cet algorithme ***est incompatible avec celui d'Ératosthène****, les indexes de départ des nombres premiers* $P \leq \sqrt{2N}$ *qui criblent, sont différents du crible Ératosthène, ainsi que les restes* R *de* $2N$ *par* P *qui changent à chaque limite* $N+1$ *criblée.*
Ce qui rend impossible une conjecture fausse, le contraire indiquerait, que le reste R *de* $2N$ *par* P*, correspond à l'indexe de chaque* p'*, pour tout* $p' \leq N$*, d'où il en résulterait que tous les :* $p' \equiv 2N\ [P]$ *ce qui est clairement faux.*

Car en effet pour cela, il faudrait utiliser tous les restes R précédents des $2N\text{-}_{2,}\text{-}_{4,}\text{-}_{6\ldots}\text{-}_{n}$ ***Par P, des limites N précédentes criblées, ce qui est impossible !***

On peut réitérer indéfiniment cette limite N, qui augmentera **le champ des congruences** $< N$ *permettant de repousser de plus en plus loin ces limites* N *successives, donc la décomposition des 2N successifs, où il est impossible de supposer que* $2N+2...+2 \neq p' + q$ *.*

On verra aussi, sans perte de généralité, que pour toute limite $n > 3\ 000\ 000$ ***criblée*** *; il existe toujours un nombre premier* $p' \not\equiv 2n[P]$ *de 1 à* $((\sqrt{n})/30)$*, ie : une soltution, tel que :* $2n = p'+q$ *. Ce que l'on peut vérifier avec le programme C++ en fin de document pour la limite* $n \leq 2*10^{19}$*. »)*

[« *En résumé pour tous les entiers naturels positifs contenant l'ensemble des nombres premiers > 5, de la forme* $30k + i$ ***(avec*** $i \in [1,7,11,13,17,19,23,29]$***)****:*
Il est évident que tout nombre premiers $p' \leq N$ *congru à* $2N$ *modulo* $P_1, P_2, P_3 \ldots P_n$ *lors d'une limite* $N=15k$*,* $k \geq 1$ *précédente criblée et vérifiée, ne peut plus être congrus modulo* $P_1, P_2, P_3 \ldots P_n$ *, pour la limite suivante* $N=15(k+1)+ i$ *. Car en effet, les nombres* p' ***qui étaient congrus à 2n modulo P ,*** *se trouvent ainsi libérés de leur congruence pour 2n + 2 modulo P, du simple fait que les restes* R *vont changer; alors que le nombre de nombre premiers* q *entre ces deux intervalles successifs* $[n+1 ; 2n+2]$ *serra le même à une exception près , ainsi que les nombres premiers* P_n *qui criblent.*
Ce qui nous garanti toujours, une densité minimum de solutions > 0*, qui vérifiront* $2n+2 = p'+ q$*,* ***ou par famille*** $2n + 30 = p'+ q$.
Or : Si on suppose fausse la conjecture, il faut que pour la limite suivante $N=15(k+1)+i$ *les nombres* p' *qui étaient congrus* $P_1, P_2, P_3 \ldots P_n$ *le soient à nouveau par ces mêmes nombres premiers* P_n *qui criblent («avec* $P_n \leq$ *racine de 2N») .*

Mais alors suivant cette supposition, ceux qui n'étaient pas congrus modulo P_n *ne peuvent pas non plus être encore, non congrus modulo* P_n *! Or , le nombre de premiers* q *entre ces deux intervalles successifs* $[n+1 ; 2n+2]$ *est déjà définis et*

vérifié ... on aura donc la même quantité à une exception près; «ce qui est facile à vérifier et rend contradictoire cette supposition».
Cela ne serait plus le cas si tous les nombres ***p'*** *qui étaient congrus ou pas modulo* P_n *, deviennent congrus modulo* P_n *par miracle et en utilisant les restes* ***R*** *des limites* ***N*** *précédentes criblées , ce qui est impossible. Le nombre de premiers* P_n *qui criblent serra aussi le même à une exception près.. ! »]*

Explication *:*

Pour cela on utilise deux algorithmes ***« avec des indexes de départ différents et incompatibles en générale »****, qui vont cribler les entiers naturels* ***A*** *positifs de* ***1 à N*** *avec :*

1_) *Celui d'Ératosthène en criblant les entiers A premiers* ***p',*** *de* ***1 à N*** *avec* $P \leq \sqrt{N}$***, P premier.*** *Où* ***P crible*** *à partir de ses indexes définis par l'algorithme, où cet algorithme est incompatible avec les indexes de départ de Goldbach «voir programme» ci-dessous. Puis avec :*

2_) *Celui de Goldbach qui va* ***recribler*** *mais avec* $P \leq \sqrt{2N}$ *à partir* ***de ses indexes, différents*** *de ceux d'Ératosthène; les entiers naturels* $A \not\equiv 2N\ [P]$*, de* ***1 à N,*** *ce qui* ***crée un champ de congruences avec une densité non nulle*** $\Rightarrow$ $q \in [N;2N]$ **.**
Par conséquent si $A = p'$*, tel que* $A \not\equiv 2N\ [P] \Rightarrow q$ *premier, on obtient obligatoirement la décomposition de* $2N = p'+q$**.**

3_) Pour l'ensemble des entiers naturels $A \leqslant N$, positifs impairs, (*«représenté dans le programme de l'algorithme par des ,1,»)* , avec **A** de raison **2 de 1 à N**, fixons la limite **N = 9** qui va vérifier la conjecture , prenons un nombre premier $P \leqslant \sqrt{2N}$, que l'on va utiliser avec le reste **R** de la division Euclidienne de **2N par P** pour calculer les $A \not\equiv 2N\ [P]$.
L'algorithme de Goldbach, va donc utiliser les congruences, pour construire un axe des ordonnées dans le sens ↓: de l'amont vers l'avale.
(« Indiquer le nombre de nombres premiers $A \not\equiv 2N[P] \Leftrightarrow q \in [N ; 2N]$ pour toute limite N fixée, ainsi que le nombre de décompositions de $2N = (p'+q)$. Il est clair que tout nombre premier **q** a pour antécédent $A \not\equiv 2N[P]$ **et** si : $A = p'$ tel que $A \not\equiv 2N[P]$ il est évident que **q** premier dépendra par conséquent de **p' ,** il en résultera $p'+ q = 2N$ **et on ne peut pas dire que p' et q sont indépendant l'un de l'autre »)**

On va donc pour tout entier ou limite $A \leqslant N$ calculer le reste **R** de **2N** par **P** pour tout les nombres $P \leq \sqrt{2N}$, afin de vérifier si **A** représenté par **[,1,]** est congru ou pas à **2N** modulo **P**. Si $A \equiv 2N[P] = 0$**, sinon 1 .**

Chaque entier **[,1,]** tel que $1 \equiv 2N[P]$, initialisera un (rayon ou diagonale) d'entiers **A** congrus à 2N modulo P, *[que l'on peut nommer un champ de congruences]*, sur **l'axe des ordonnées** dans ce sens ↓, qui seront marquées **[,0,]** ; voir illustration ci-dessous .

Il vient par obligation : si $1 \equiv 2N[P]$ il vient que lorsque N augmente de 1, 2N augmente de 2 , donc $(1+2) \equiv 2N+2$ **[P],** avec **P = 3.** (« *Exemple : 1* $\equiv$ *28 [P] d'où (1+2)* $\equiv$ *30 [P] et 28 – 1* $\Leftrightarrow$ *30 – 3* $\neq$ *q premier, le contraire contredirait le TFA et le TNP.* »)

Inversement : si **[,1,]** tel que $1 \not\equiv 2N[P]$, il initialisera une diagonale d'entiers **A** non congrus à 2N modulo P, qui seront marquées **1**. Donc : (« *Exemple :* $1 \not\equiv 30\ [P]$ *d'où* $(1+2) \not\equiv 32\ [P]$ *, 30 – 1* $\Leftrightarrow$ *32 – 3 = q premier.* ») .
Conséquence directe de l'égalité qui s'ensuit, on obtient le décalage d'un rang des congruences, lorsque la limite N augmente de 1, pour la décomposition de 2n + 2, en somme de deux premiers :
$(2N - A) \Leftrightarrow (2N +2) - (A + 2)$, équivalent à $(A+2) \not\equiv (2N+2)\ [P]$ *ou l'inverse* $(A+2) \equiv (2N+2)\ [P]$**.**

Chaque diagonale parcourt l'ensemble des entiers **A** impairs de **1 à N**, pour toutes limites **N** fixée , ***qui viendront croiser*** l'axe des abscisses d'Ératosthène criblé de **1 à N** . ***Pour toute limite N fixée*** :
Quelque soit l'entier **2N** > 4 qui vérifie la conjecture, tel que $2N=p'+q$; alors la conjecture est aussi vérifiée pour **2N+2** .

C'est une conséquence directe du décalage récurrent des congruences, il existe pour la suite **N** qui a été criblée et donné le nombre de décompositions de $2N = p'+ q$ **,** ainsi que le nombre d'entiers $A \not\equiv 2N[P]$ qui ***précèdent*** $A+2$ ***premier p'*** et de part le fait de ce décalage d'un rang des congruences, qui s'ensuit ; on obtient le nombre de solutions pour la limite **N+1** suivante , **c'est à dire** le nombre de décompositions $p'+ q = 2N + 2$, à une unité près.

« *Équivalent au décalage d'un rang des nombres premiers* ***q de n à 2n*** *, mais en sens inverse pour être complémentaire par rapport à* ***2n.*** **Ce qui implique :** *si la conjecture est fausse pour* ***2n +2*** *, il faut que tous ces nombres* ***q*** *disparaissent pour ne pas se décler d'un rang… ?* **»**

Ce nombre de décompositions d'un entier **2N,** est par conséquent, toujours vérifié lors de la limite précédente **N–1** criblée, dû à cette propriété récurrente : décalage d'un rang des congruences correspondant à l'égalité suivante :

(2N – A) ⇔ (2N +2) – (A + 2).

De la même manière, que l'on connaît le nombre de ***(A≢ 2n[P]) ⇔ q ∈[N,2N],*** *qui a été vérifié et ne peut varier au maximum que de* ***1*** pour la limite suivante **N + 1** ; ce qui implique le même nombre de premiers ***q ∈[(N+1) , (2N+2)]*** à une unité près. ***Ce qui serait donc impossible***, si la conjecture était fausse, **car :**

1:) elle est une conséquence directe du TNP et de la répartition des nombres premiers et en :

2:) il ne pourrait y avoir qu'un seul nombre premiers ***q ∈[N,2N]*** *dans le pire des cas,* ***c'est à dire :***

(« U*ne aberration telle: qu'il faudrait que tout reste* ***R de 2N par P = p' ≤ N ,*** *d'où* ***p' ≡ 2N[P] puis que la répartition des nombres premiers p' , se fassent modulo P comme les produits (P*p) avec le*** *principe du crible Ératosthène, on marquerait tous les nombres premiers* ***p' congrus à 2N (mod P)****; ce qui est absolument faux .!*

Ou encore, que le postulat de Bertrand se réalise; c'est à dire qu'il existe une limite ***n > 14*** *où entre* ***n et 2n*** *il existe effectivement un seul ou deux nombres premiers* ***q, ie;*** *un ou deux entiers* ***(A≢ 2n[P]) ≤ n*** *.*

Ce qui est impossible et serait contraire au TNP, car pour la limite ***n – 1*** *on connaît déjà le résultat de ce nombre de*

A≢ 2n[P] *⇔ aux nombres premiers* ***q*** *ayant été vérifiés, qui est environ équivalent à* $\frac{n}{(\log 2n)}$ *conséquence du TNP»)*

Ou encore, aucun **A ≢2N[P]** qui ***précèdent*** **A+2** ***premier p' ,*** donnant obligatoirement un couple **p'+ q = 2N+2** lors du décalage d'un rang des congruences **.**

(«On peut itérer ce processus indéfiniment pour toute limite **N** fixé, illustration page 10. ..

En définitive, lorsque **N** → ∞, le rapport du nombre de nombres premiers **p'≤ N** , tel que **p' ≢2N[P]** vaut environ **1/5** dans cet exemple**. »)**

On peut d'aileurs affirmer que pour toute limite N ≥ 3 000 000 criblée, entre 1 et ((√N)/30) , il existe toujour un nompres premier p' ≢2N[P] qui donne une solution pour l'entier 2N ...!(*voir progamme C++ en fin de document, page 19.)*

1.) le programmes Goldbach : sert à initialiser les diagonales d'entiers **A**, congrues ou non à 2N modulo P sur l'axe des ordonnées ↓ de Goldbach , dans le sens inverse du plan ci-dessous .

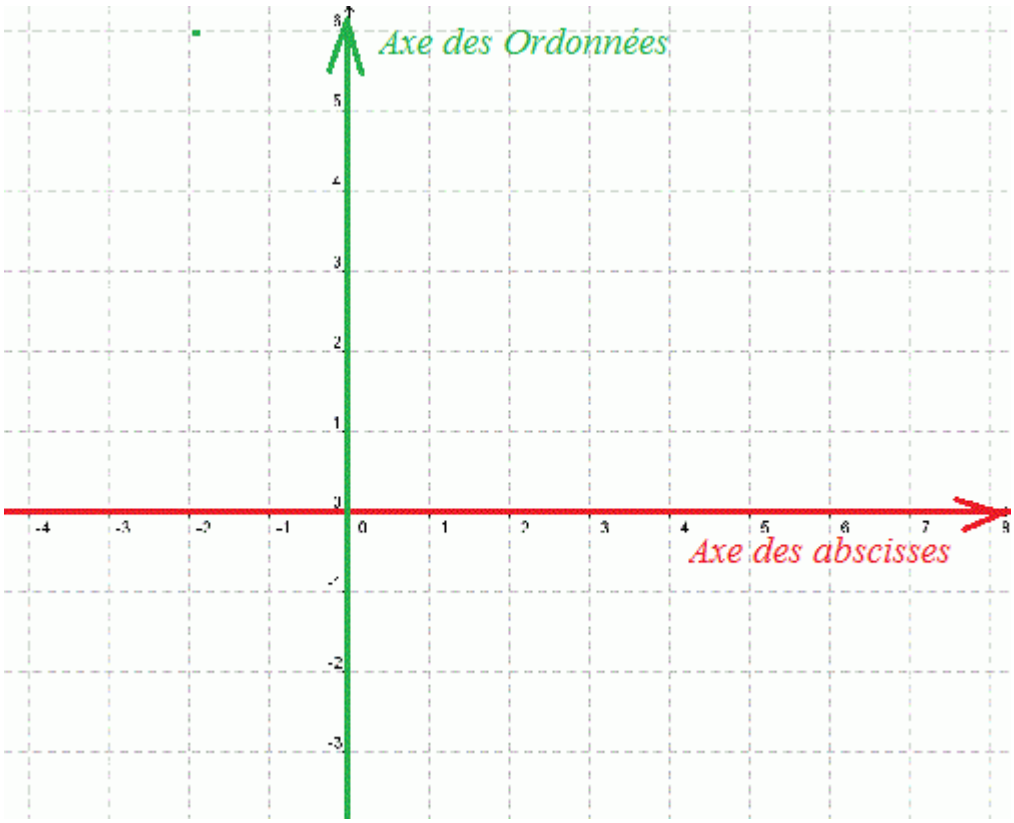
2.) Cet axe vient croiser l'axe des abscisses → *représentant* les nombres premiers **p'** d'Ératosthène de **1** à **N** qui ont été criblés. Chaque rang des ordonnées correspond à chaque rang des abscisses :

Par exemple : le point 4 d'ordonnée vient croiser le point d'abscisse 4 lorsque 2N = 32, on a 3≡0[3] et 32 ≡2 [3] ce qui donne une diagonale initialisée par **3 non congru à 32 [P] «** *les restes R de 32 et 3 par 3 sont différents ».*

D'où l'axes des ordonnées de Goldbach initialisé par A = 3, représentera une diagonale non congrue à 2N [P], qui viendra croiser l'axe des abscisses Ératosthène afin de correspondre à l'égalité lorsque **2N** augmente de 2: ***« 32 – 3 = 29 et 34 – 5 = 29 » Cette égalité est donc récurrente, lorsque 2N augmente de 2 ; [2N – A ⇔ (2N+2) – (A+2)]***

3.) Autrement dit pour tout limite N criblée, lorsque la limite N augmente de 1, 2N augmente de 2, l'axe des ordonnées « *va descendre d'un rang* » sur l'axe des abscisses, ce qui décale d'un rang la diagonale des congruences sur cet axe d'abscisses, initialisée par 3 qui est non congrus à 2N+2 [P]; qui en venant se décaler d'un rang, croisera 5 sur l'axe des abscisses d'Ératosthène ; où A+2 = 5 est donc le successeur de 3 lorsque 2N augmente de 2, ce qui donnera comme complémentaire de 5 par rapport à 2N + 2 , **la même différence d**, le contraire est donc impossible suivant l'égalité évidente ci-dessus : ***[2N – A = (2N+2) – (A+2)]*** .

L'axe des ordonnées, représente les diagonales de congruences des entiers A impairs de : **[1.3.5.7.9.11....etc.. N]**



4.) Il n'est pas réaliste de supposer la conjecture fausse inférieur à une limite K, qui serait le point où, le nombre de solutions 2N = p' + q commenceraient à décroître si cette supposition serait réalisable au point, 2N + 2, + 4 + 6…..+**X**.

Conséquence immédiate : le nombre de nombres premiers **q** par rapport à 1, appartenant à **[N , 2N]** chuterait pour tendre vers **1** au point **X ,** si les entiers **A sont congrus à 2N+X [P],** dans l'hypothèse d'une conjecture fausse, ce qui serait contraire aux fonctions du **TNP** qui en donnent une estimation, ainsi que les fonctions ayant été calculées à ce jour**.**

Ce que l'on va voir avec cet algorithme : les congruences criblées par Goldbach , sont des diagonales de congruences, qui sont initialisées à la base par le chiffre [1], ce nombre de diagonales congrues ou pas à 2N[P], tend vers l'infini,

Si le reste de **2N par P = 1**, on aura une diagonale d'entiers $A \equiv 2N[P]$ qui seront marqués **[,0,]** ;
dans le cas contraire, si dans la division Euclidienne de **2N par P** , le reste **R≠1**, on initialise une diagonale de $A \not\equiv 2N[P]$ qui restent marqués **[,1,]**. Illustré ci-dessous.

5.) L'algorithme de Goldbach représente toutes ces diagonales de congruences, lorsque l'on crible les entiers de **1 à N;** les congruences permettent de donner le nombre de solutions, qui vérifient **2N = p'+ q**

6.) Il existera par conséquent une infinité de diagonales , soit congrues , soit non congrues, à **2N[P]** lorsque 2N tend vers l'infini , qui viendront se décaler d'un rang sur l'axe des abscisses Ératosthène pour chaque limite N + 1 ⇒ 2N+2.

Ce décalage récurrent d'un rang des congruences pour toute limite **N** criblée, permet de donner le nombre de solutions qui décomposent 2N + 2 = p' + q et de vérifier aussi par là même, le nombre de solution pour plusieurs entiers pairs consécutifs : 2N+2 ,+4 ,+6...etc 2N +X , avec X < au nombre de rang de la limite N criblée.

*7.) Par conséquent, même si **A n'est pas** un nombre premier , **mais si** il précède **A+2** premier **p'** et qu'il est d'ordonnée **non congru à 2N [P] :***
*Lorsque 2N augmente de 2 , soit 2N+2, on crée une nouvelle diagonale, l'axe d'ordonnées descend d'un rang , avec l'axe d'abscisses ce qui provoque le décalage d'un rang de la diagonale d'ordonnée sur cet axe d'abscisse et qui a pour antécédant **(A) ≢ (2N) [P].***
*D'où cette diagonale non congrue à 2N (mod P) et avec **(A+2) ≢ (2N+2) [P] ⇒ ((2N+2) – (A+2) = q)** vérifiera donc la conjecture pour **2N+2** ...!*

8.) *Donc : p*our supposer la conjecture fausse , il faut qu'aucune diagonale de $A \not\equiv 2N[P]$ avec ***A*** un nombre premier **p'≤ N** , ne vienne croiser ***p'*** un nombre premier sur l'axe des abscisses .
Ce qui est impossible pour une raison simple : il faudrait que le nombre P (≤ √N) ou (≤ √2N) qui crible dans les deux algorithmes Ératosthène et Goldbach , partent du même index ; ce qui est clairement impossible.
L'index de départ du nombre P ≤ √N qui crible dans Ératosthène part d'un index calculé par le produit de P*p ou de P^2 , conformément à l'algorithme et de son programme que le lecteur connaît.
Alors que dans Goldbach le départ du nombre P ≤ √2N, part d'un index calculé par le reste R de 2N par P , qui est différent par obligation et donne une égalité récurrente lorsque la limite N augmente de 1.

Ce n'est en définitive , qu'une conséquence directe du TNP et de ses deux fonctions (voir ci dessous).

Autrement , dit il ne faut plus à partir d'une limite N = K de [,1,] non congrus à 2N modulo P.
(« Pour ce faire, il faudrait utiliser les restes R des limites précédente 2N ***– 2, – 4 ..etc,*** *ce qui est impossible. »)*

Lorsque 2N augmente de 2, *il faut aussi qu'aucune diagonale de $A \not\equiv 2N[P]$ avec* ***A*** *premier ou pas , du point* ***7.)*** *ci-dessus relatif à 2N,* ***ne précède*** *une diagonale avec* ***A+2 = p'*** *un nombre premier* ***, qui viendrait valider la conjecture pour 2N+2, du fait de ce décalage récurrent d'un rang qui s'ensuit.!***
Il en serait ainsi de toutes les limites N – 1 , – 2 ,– 3 … etc.. précédentes, où aucune diagonale ne doit être non congrue à 2N (mod P) ; d'où, la fonction du TNP serait fausse , voir ci-après.

Ou encore que les restes de 2N par P soit toujours = 1 , qui donnerait que des 0 sur l'axe des ordonnées de Goldbach (" *mais c'est impossible, car cela entraîne la disparition des nombres premiers* ***$q \in [N,2N]$***") !

9.) Ces deux axes , *« ne peuvent dans l'immédiat prouver » la conjecture de façon rigoureuse,* ***mais affirme que pour toute limite N criblée on connaît le nombre de solutions pour la limite N+1, donc le nombre de décompositions pour l'entier*** **2N +2** , ce qui invalide la supposition **que 2N+2 ≠ p' + q .**
Mais : *Il est absolument formel de prévoir, à partir d'une limite N criblée et du nombre de solutions qui décomposent un entier pair* ***2N = p'+ q ;*** *le nombre* ***de solutions*** *qui vérifient les entiers suivants :* ***2N +2, + 4, + 6….+ X*** *, avec X inférieur au nombre de rangs de la limite* ***N qui vient d'être criblée.***
(« Illustré fin de page 4 à 5 pour N = 300 , 390 , 1140. »)

10.) **Ce que l'on sait** : l'axe des ordonnées qui initialise chaque ("diagonale") représentées par les entiers $A \not\equiv 2N[P]$ vaut ***N / Ln 2N*** *équivalent au nombre de nombres premiers* ***$q \in [N;2N]$ où N = n, qui est comme la conjecture de Goldbach, une conséquence directe du TNP.***

****

La fonction 2 du théorème de Goldbach est une conséquence directe du TNP: (log = *logarithme naturel)*
G(n): la fonction de compte du nombre de nombres ***$A \not\equiv 2n[P] \Leftrightarrow q \in [n;2n]$***

Corollaire du TNP : **G(n)** vaut $\lim_{n \to +\infty} \frac{n}{(\log 2n)}$

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

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

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

$$= n \text{ x } \frac{2\log n - \log(2n)}{\log(2n)\log n} + o(\frac{n}{\log n})$$

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

Alors que celui des abscisses Ératosthène, relatif aux nombres premiers ***p'*** vaut vaut $\frac{n}{\log n}$

*L*a probabilité d'avoir une diagonale de **A** non congrue à ***2n (mod P)*** qui vérifie la conjecture est donc :

de $\frac{1}{(\log(2n) * \log n)}$ *Autrement dit : On peut admettre que le nombre de couples p+q qui décomposent l'entier* ***2n*** *«avec l'entier* ***n*** *qui sert donc de limite pour l'algorithme en dénombrant le nombre de* ***(A=p') ≢ 2n[P]*** *», qui vaut ≈* $\frac{n}{((\log 2n)(\log n))}$

Par exemple : Si on fixe la limite n = 91 , 2n = 182 , on aurait au minimum pour les trois Familles concernées :

$\frac{91}{(\log 182 * \log 91)}$ **= 3, 87653..** *couples p+q = 182 au lieu d'un réel de* ***6*** *couples, pour ces trois Familles*

En utilisant la fonction asymptotique du TNP modifiée, comme expliquée dans ***la note ci-dessous*** **.**
On aurait environ pour ces 3 Familles :

5,45... couples p'+ q = 182. ***[****« On a 20,17..* ***p'*** *< 91 . Soit (****20,17 / 8 * 3****) / (****ln 182 /3,75****) =* ***5,45****04..»****]***

Si la conjecture était fausse *, on aurait* ***12*** *nombres premiers* ***q*** *entre* ***n*** *et* ***2n*** *au lieu de* ***18*** *et donc inférieur à la fonction du TNP qui en donne un équivalent de* ***17,4865****…. pour cette limite* ***2n****.*

Alors que l'on sait, lors de la limite précédente (2N – 2), ce nombre de premiers ***q*** *était identique à une unité près, ce qui serait donc absurde et faux, si la conjecture était supposée fausse.*

(«Voila .! Ce qu'il en est pour mon algorithme de Goldbach à l'heure actuelle. Équivalent à Éra tosthène de ***n*** *à* ***2N****»)*

La seule certitude, c'est que l'on peut montrer à partir d'une décomposition de Goldbach et des diagonales qui re-coupent l'axe des abscisses, jusque où la conjecture est vérifiée sans avoir besoins de tester ou de cribler pour la limite suivante N+1, le nombre de solutions qui vérifie **2N + 2** .
Tout en sachant que cette limite se repousse systématiquement pour tendre vers l'infini , ainsi que le nombre de nombres premiers vérifiés lors de la limite précédente [N – 1 , 2N – 2] et où ce nombre de premiers ***q ∈ [N,2N] ne peut varier que de 1 par famille entre ces deux intervalles : q ∈ [N+1 , 2N+2].***

On peut en conclure que cette conjecture ne peut être fausse, sans contredire sa propriété récurrente et ces fonctions croissantes du TNP ; ce qui serait impossible dans le cas contraire et on arrive à une estimation ~ 1/5 du nombre de nombres premiers p' criblés , tel que **p' ≢ 2N[P] ⇒ p'+ q = 2N**.

Note relative à ces deux fonctions du TNP caractérisées par ces deux algorithmes :

On sait que l'estimation asymptotique du nombre de nombres premiers **p' ≤ N** vaut environ ***N sur ln de N.***
On sait que le nombre d'entiers naturels des 8 familles de la forme **30k + i** pour une limite **N** fixée, vaut **N / 3,75.**
On connaît aussi que l'estimation asymptotique du nombre de **A ≢ 2N[P] ≤ N** ⇔ nombres premiers **q ∈ [N ; 2N]** vaut environ ***N sur ln de 2N.***

Il est clair que sans perte de généralité, on peut arranger ces fonctions du TNP de la manière suivante :

$\frac{n}{\log n}$ est identique à $\left(\frac{n/3,75}{(\log n/3,75)}\right)$ en ne travaillant que dans l'ensemble des entiers la forme **(30k + i)**

Exemple : pour **N=100**, l'estimation vaut ~ $\frac{100}{\log 100}$ **= 21,71..** ⇔ $\frac{100/3,75}{(\log 100/3,75)}$ **= 21,71…**

D'où : On peut utiliser la deuxième fonction du TNP $\frac{n}{(\log 2n)}$ pour avoir estimation asymptotique du nombre de couples **p'+ q = 2N .**

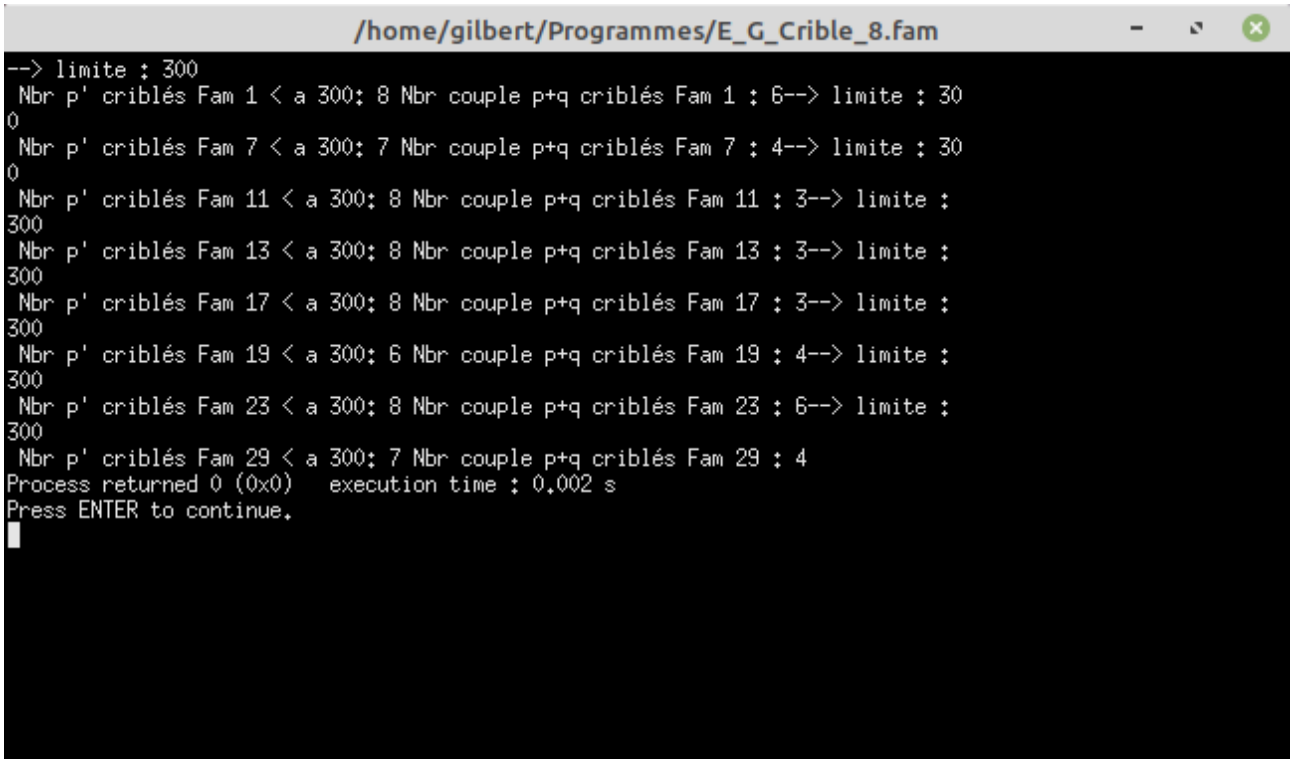
***Relatif à l'algorithme de Goldbach** , qui va re-cribler les nombres **p'** du crible d'Ératosthène pour cette même limite **N** fixée, sans perte de généralité*

De la façon suivante et Pour une limite N = 300 fixée : Il suffit de prendre le résultat asimptotique :

$\frac{300/3,75}{(\log 300/3,75)}$ **= 52,59… que l'on divise** par le **Log naturel de 2n /3,75.**

*Ce qui donne pour cet exemple et pour un réel de **60 p'≤ 300** :* $\frac{52,59}{(\log 600/3,75)}$ **= 30,83..** couples **(p'+q=2N)** pour un réel de **33 .**

Ci joint pour ces 8 Fam, illustration du résultat du crible Goldbach / Ératosthène limite fixée ≤ N.

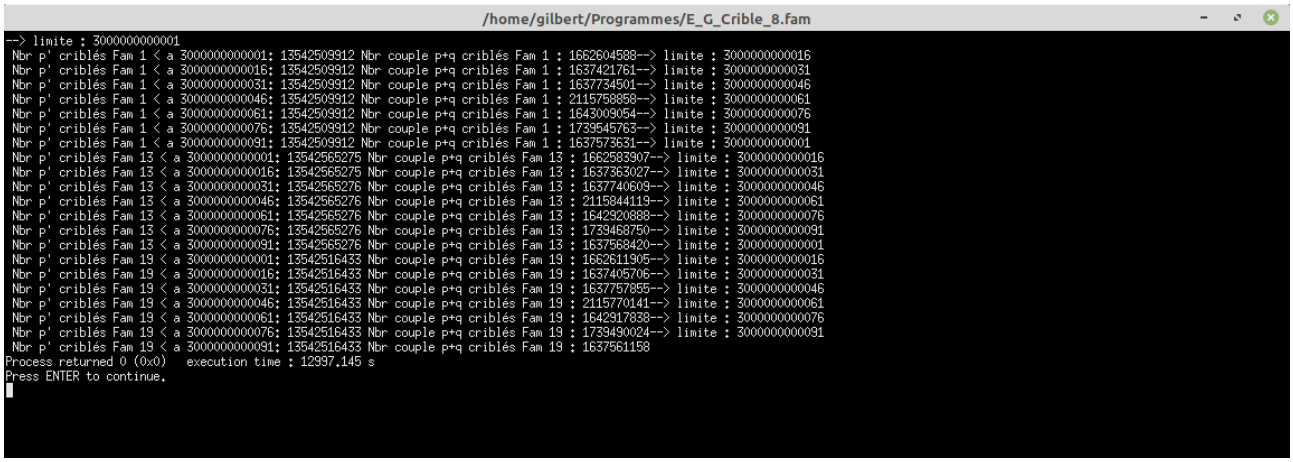


Autre exemple et pour une limite N = 12 001 , où il n'y a que trois familles de couples de nombres premiers qui seront criblées La Fan 1, la Fam 13 et La fam 19 : Ce qui donne **12001/3,75 = 3200** entiers **A** de la forme **30k+i.**
Puis ***3200 / (ln 12001 3,75)** , donne **1277** nombre **p'≤ N**. Pour ces trois familles , environ: 1277 /* 8 * 3.

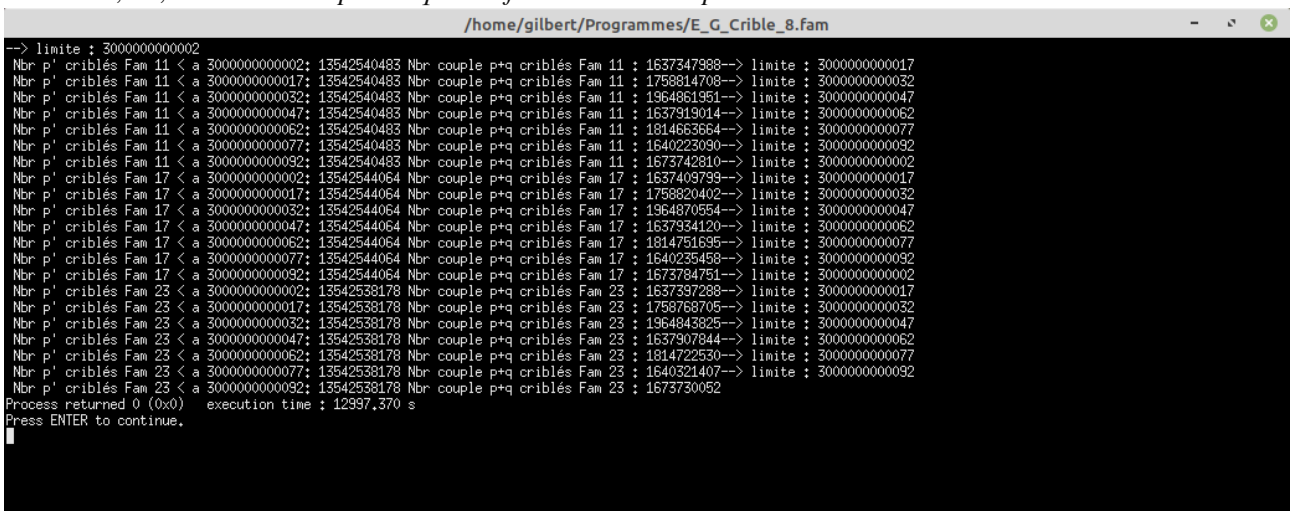
En criblant avec l'algorithme de Goldbach, on obtient aux réel pour ces **3** familles ***216** couples p'+ q = 24 002 . Pour une estimation asymptotique de **178** couples* ***car :***
***1)** on a (**1277 /** (ln 24002 / 3,75)) = 474,79..*
***2)** et pour ces trois Fam : (**474,79 /** 8) * **3** =178,04 couples qui décomposent 24002 en somme de deux premiers.*

*«Ceci dit: c'est une fonction qui reste à affiner, car si l'écart entre le nombre réel de couples (p'+q) est supérieur à l'estimation asymptotique pour de petite limite **N**, on risque d'avoir le contraire pour de Grande limite **N** … Car la différence entre le nombre de P' criblés et le nombre de couple (p+q) **est oscillatoire** lorsque **N**→∞, comme on peut le vérifier sur l'illustration ci-dessous .»*

Limite du criblage ≤ **N** de raison 15, de *3000 000 000 001* à *3000 000 000 106* et les **3 Fam** de nombres premiers **p'** concernés pour cette décomposition de 2N ; **Fam 1, Fam 13 et Fam 19.**
Résultat du nombre de décompositions **p'+ q = 2N** pour les 7 entiers **2N** consécutifs de raison 30 =
6 000 000 000 002 ; **6 0….032** ; **6 0….062** ; **6 0….092** ; **6 0….122** ; **6 0….152** et **6 0….182** ;



ou encore avec 2N+2 de raison 30 , limite N = 3000 000 000 002 de raison 15 et les trois autres Familles qui correspondent à ces entiers 2N+2 ...
Fam: 11 , 23, et 17. « *le temps indiqué est faux ...car le temps mis est d'environ 2 heures* »



Avec la limite N ci-dessous = 6 600 000 000 010 ; 2n = 13 200 000 000 020
et ses 4 Fam (1;7;13;19) ; estimation asymptotique pour les 4 Fam :

6600000000010 / 3,75 = 1760000000002 ***entiers 30k+i***
1760000000002 / (ln 6600 000 000 010 / 3,75) = **223 591 696 785** ***premiers p' ≤ N***

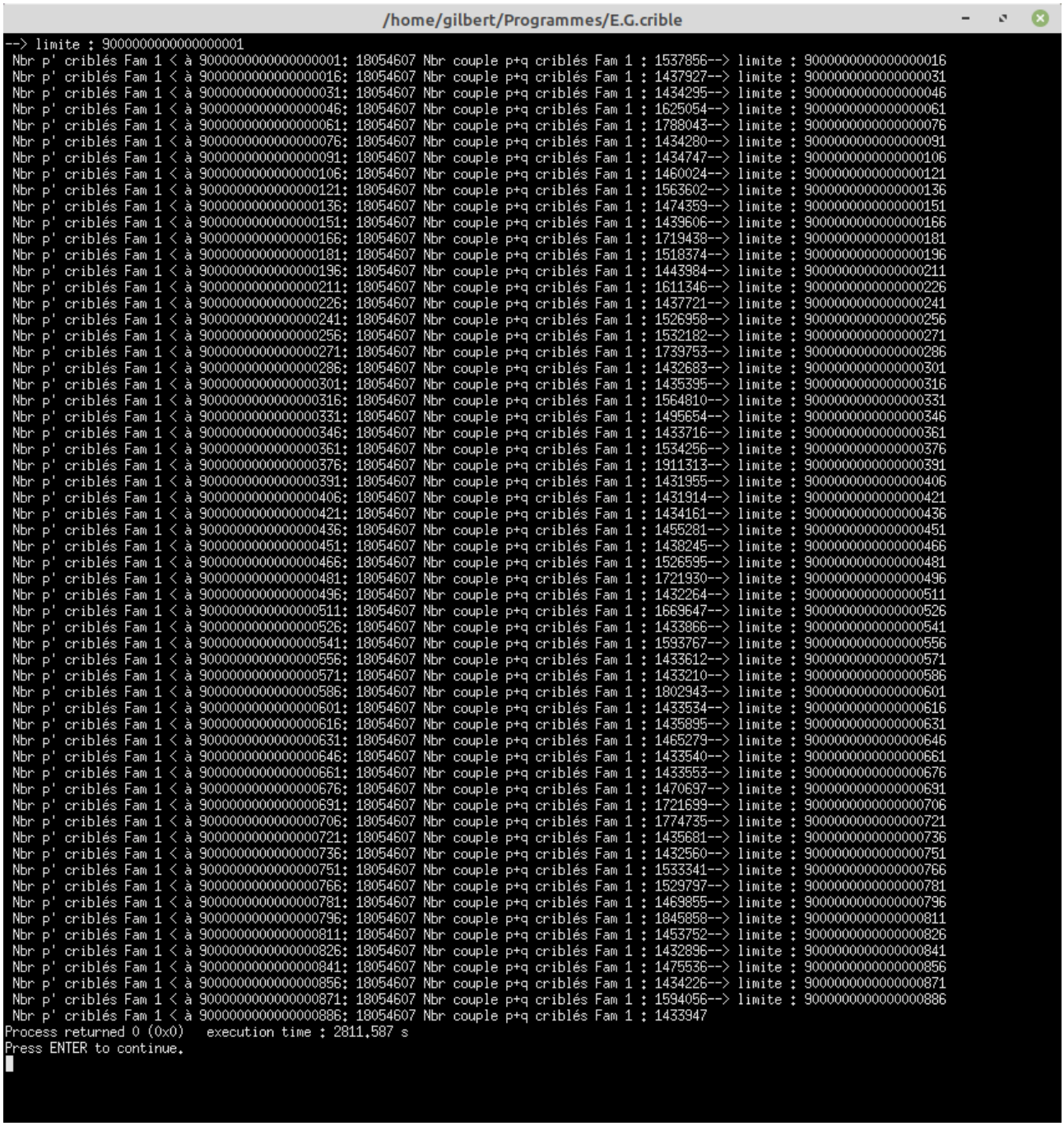
soit environ pour ces 4 Fam: 223591696785 / 8 * 4 = 111 795 848 392,5

puis : 111 795 848 392,5 / (ln 13 200 000 000 020 / *3,75)* =***138 76 771 028*** *(P' + q = 2N)*

Ou encore :
223591696785 / (ln 13 200 000 000 020 / *3,75)* = *((27753542057 /* 8) * 4) Vaut ~ ***13 876 771 028***

couples p'+ q ; ou par Famille vaut ~ : 3 469 192 757 couples (p'+ q) = 2N
On peut aussi vérifier ces formules :
En utilisant simplement la limite **N ≥ 3 000 000 criblée, entre 1 et ((√N)/30) avec le résultat du programme comme ci-dessous :**

criblage des p' ≤ (√n)/30 . Dans l'algorithme : le limite n , progresse modulo 15 *« il y a 3 fam i pour n =15k+1, soit 2n = 30k+2 ; exemple avec la Fam 30k+1sont complémentaire q est aussi de la même Fam.»*



/home/gilbert/Programmes/E_G_Crible_8.fam

```
famille : 1 limite : 6600000000010
Nombre premiers criblés famille 1 plus petits que 6600000000010: 28967518720 time 840.754
Nombre couples p+q=2N criblés famille 1 : 3411256593 time 876.828
famille : 1 limite : 6600000000025
Nombre premiers criblés famille 1 plus petits que 6600000000025: 28967518720 time 830.226
Nombre couples p+q=2N criblés famille 1 : 3410685839 time 5179.47
famille : 1 limite : 6600000000040
Nombre premiers criblés famille 1 plus petits que 6600000000040: 28967518721 time 5127.24
Nombre couples p+q=2N criblés famille 1 : 3637444519 time 5170.46
famille : 1 limite : 6600000000055
Nombre premiers criblés famille 1 plus petits que 6600000000055: 28967518721 time 5119.43
Nombre couples p+q=2N criblés famille 1 : 4550506175 time 5171.94
famille : 1 limite : 6600000000070
Nombre premiers criblés famille 1 plus petits que 6600000000070: 28967518721 time 9418.66
Nombre couples p+q=2N criblés famille 1 : 3453976619 time 9464.75
famille : 7 limite : 6600000000010
Nombre premiers criblés famille 7 plus petits que 6600000000010: 28967578247 time 9415
Nombre couples p+q=2N criblés famille 7 : 3411253438 time 9463.73
famille : 7 limite : 6600000000025
Nombre premiers criblés famille 7 plus petits que 6600000000025: 28967578247 time 9417.21
Nombre couples p+q=2N criblés famille 7 : 3410707544 time 13763.8
famille : 7 limite : 6600000000040
Nombre premiers criblés famille 7 plus petits que 6600000000040: 28967578247 time 13710.6
Nombre couples p+q=2N criblés famille 7 : 3637421453 time 13766.3
famille : 7 limite : 6600000000055
Nombre premiers criblés famille 7 plus petits que 6600000000055: 28967578247 time 13717
Nombre couples p+q=2N criblés famille 7 : 4550474625 time 13766.1
famille : 7 limite : 6600000000070
Nombre premiers criblés famille 7 plus petits que 6600000000070: 28967578247 time 18011.9
Nombre couples p+q=2N criblés famille 7 : 3453890955 time 18055.1
famille : 13 limite : 6600000000010
Nombre premiers criblés famille 13 plus petits que 6600000000010: 28967565166 time 18000
Nombre couples p+q=2N criblés famille 13 : 3411240918 time 18056.1
famille : 13 limite : 6600000000025
Nombre premiers criblés famille 13 plus petits que 6600000000025: 28967565166 time 18010.7
Nombre couples p+q=2N criblés famille 13 : 3410706294 time 22348
famille : 13 limite : 6600000000040
Nombre premiers criblés famille 13 plus petits que 6600000000040: 28967565166 time 22311.5
Nombre couples p+q=2N criblés famille 13 : 3637301251 time 22356.4
famille : 13 limite : 6600000000055
Nombre premiers criblés famille 13 plus petits que 6600000000055: 28967565166 time 22297.1
Nombre couples p+q=2N criblés famille 13 : 4550423761 time 22354.6
famille : 13 limite : 6600000000070
Nombre premiers criblés famille 13 plus petits que 6600000000070: 28967565166 time 26598.1
Nombre couples p+q=2N criblés famille 13 : 3453932549 time 26639.2
famille : 19 limite : 6600000000010
Nombre premiers criblés famille 19 plus petits que 6600000000010: 28967477637 time 26597.7
Nombre couples p+q=2N criblés famille 19 : 3411186846 time 26646.6
famille : 19 limite : 6600000000025
Nombre premiers criblés famille 19 plus petits que 6600000000025: 28967477637 time 26604.8
Nombre couples p+q=2N criblés famille 19 : 3410694760 time 30942.2
famille : 19 limite : 6600000000040
Nombre premiers criblés famille 19 plus petits que 6600000000040: 28967477637 time 30894.6
Nombre couples p+q=2N criblés famille 19 : 3637331827 time 30943.8
famille : 19 limite : 6600000000055
Nombre premiers criblés famille 19 plus petits que 6600000000055: 28967477637 time 30895.5
Nombre couples p+q=2N criblés famille 19 : 4550384284 time 30940.3
famille : 19 limite : 6600000000070
Nombre premiers criblés famille 19 plus petits que 6600000000070: 28967477637 time 35188.6
Nombre couples p+q=2N criblés famille 19 : 3453881974 time 35237.1

Process returned 0 (0x0)   execution time : 51359.822 s
Press ENTER to continue.
```

ℒ 𝒢 .

Exemple : on utilisera, **n** ou **N**

Secteur des diagonales d'entiers impairs, initialisée sur l'axe des ordonnées de **Goldbach** : par les congruence =[1,] ou [0,]
où à chaque augmentation de 1, de la limite **n** criblée , les diagonales de congruences se décalent d'un rang sur l'axe des abscisses qui descend d'un rang.

[0, 1, 0, 0, 1, 1, 0, 0, 1, 0
[1, 3, 5, 7 ,9 , 11, 13, 15, 17, 19 ,

illustration : *on va juste décaler la diagonale des congruences d'un rang lorsque N augmente de 1, donc N+2*

Donnez N: 20
*[3, 5] <….***nombre P ≤ √2n**
1 *<…..reste r de 2n par P*
0
*crible: [0, **1**, 0, 0, **1**, **1**, 0, 0, **1**, 0] de 1 à 19 diagonales de Goldbach initialisée sur l'axe des ordonnées ?*
*[1, **3**, 5, 7 ,9 , **11**, 13, 15, **17**, 19] , axe d'abscisses d'Ératosthène*

Nombres non congru 2n[P] 1 à 20 , premiers q de 20 à 40: 4 , 3 couples (p+q) = 40

Donnez N: 21
[3, 5] P
0
2
*crible: [**1**,[0, **1**, 0, 0, **1**, **1**, 0, 0, **1** , 0]* ⇔ ***au décalage inverse d'un rang des nombres q de 39 à 21***
[1, 3, **5**, **7** , 9 , **11**, **13**, 15, 17, **19** ,21]
Nombres non congru 2n[P] 1 à 21 premiers q de 21 à 42: 5 ; 4 couples (p+q) = 42

*Donnez **N: 22***
[3, 5] P
2
4
crible:
*[**1**, **1**,[0, **1**, 0, 0, **1**, **1**, 0, 0, **1**]*
[1, 3, **5**, **7** , 9 , **11**, 13, 15, **17**, **19** ,21]

Nombres non congru 2n[P] 1 à 22 premiers q de 22 à 44: 6 ;
***3 couples p+q** = 44 sont encore vérifiés , sans compter les deux nouveaux couples avec 5 et 7, ; le but est de montrer la décalage de l'axe des ordonnées sur l'axe des abscisses,à chaque fois que 2n augmente de 2.*

Donnez N: 23
[3, 5]
1
1
crible
*[0, 1, 1, [0, **1**, 0, 0, 1, **1**, 0, 0 , **1**]*
[1, 3, **5**, [**7** , 9 , [11, **13**, 15, **17**, **19** ,21, **23**]

Donnez N: 24
[3, 5]
0
3
crible:
*[1, 0, 1, 1, [0, **1**, 0, 0, **1**, **1**, 0 , 0 , 1]*

[1, 3, **5**, **7** ,[9 , **11**, 13, 15, **17**, **19** , 21 , **23**]

Congruences de Goldbach , entiers A impairs modulo 2:

Nombre premiers **p'** ≤ N où 1 n'est pas un nombre premier, Axe des abscisses d'Ératosthène,
Entiers A impairs:
[1, **3**, **5, 7**, 9, **11, 13**, 15, **17, 19**, 21, **23**, 25, 27, **29, 31**, 35, **37**, 39, **41 43**, 45, **47**, 49, 51**,]** *53, 59, 61, 67, 71, 73, 79, 83, 89, 97**]***

[ce qui se traduit par l'axe d'abscisses des nombres premiers suivant] **: n ≤ 30**
[1, 1, 1, 1, 0**, 1, 1,** 0**, 1, 1,** 0**, 1,** 0**,** 0**, 1] 15** ***p' = 1,*** *ou multiples de P = 0*

N point k, de 15 à X conjecture fausse par supposition,

A : impairs, **congrus** = 0 , **non congrus** =1

[entiers impairs de **1 à n**] et ils sont ***indicé de 0 à n.*** On calcule le reste **R** de 2n = 30 par **P ≤ √2n**
P = 3 , 5 ; **R** = 0 , 0 :
*Si **R**%2= 0 on fait (**R+P**) // 2 et **on part de l'indice(index)** en le marquant 0 , ainsi que tous ses suivants modulo P , si **R**%2 != 0 , **R**//2 on part de l'indice que l'on marque 0 , ainsi que tous ses suivants modulo P.*
*On aura marqué **[,0,]** tous les entiers **congrus à 2n [P]**, à la fin on relève les **[,1,]** qui sont les entiers **non congrus à 2n[P].** « Il est clair que ces indexes de départ sont différents, pour Ératosthène.»*

Illustration :

ordonnées
↓
↓
[1, 3, 5, 7, 9,11,13] → → abscisses
[*secteur des diagonales de congruences qui montre le décalage pour 2n +2:*
*représentés par les A impairs **congru*** = 0 **sinon = 1**] en partant de l'axe d'ordonnées initialisé par la cellule du chiffre **[1,** et ce , pour tout 2n +2 , qui initialise donc une diagonale de congruences , ***par l'algorithme de Goldbach***

Ordonnées, de l'algorithme de Goldbach, chaque diagonale représente les A impairs en progression arithmétique de raison 2.
Crible G
↓
↓
↓
[1], 0, 0, 1, 0, 1, 1,] **15**]n =15 // 2 = 7 +1 entiers impairs de 1 à 15
[1], **1**, **0**, **0**, 1, 0, 1, 1,] 16 … P = 3 et 5 , R = 2 et 2 ; indice (2+3)//2 = **2** et (2+5)//2 = **3 ;** *ensuite par pas de 3 et de 5*
[0], 1, **1**, 0, **0**, 1, 0, 1,] **17**] … P = 3 et 5 , R = 1 et 4 ; indice (1)//2 = **0** et (4+5)//2 = **4**
[0], **0**, 1, **1**, 0, 0, 1, 0, 1,] 18 … P = 3 et 5 , R = 0 et 1 ; indice (0+3)//2 = **1** et (1)//2 = **0**
[1], 0, 0, 1, **1**, 0, 0, 1, 0,] **19**] s,
[0], 1, 0, 0, 1, **1**, 0, 0, 1, 0,] 20
[1], 0, 1, 0, 0, 1, **1**, 0, 0, 1,] **21**]
[1], 1, 0, 1, 0, 0, 1, **1**, 0, 0, 1,] 22
[0], 1, 1, 0, 1, 0, 0, 1, **1**, 0, 0,] **23**]
[1], 0, 1, 1, 0, 1, 0, 0, 1, **1**, 0, 0,] 24
[0], 1, 0, 1, 1, 0, 1, 0, 0, 1, **1**, 0,] **25**]
[0], 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, **1**, 0,] 26
[1], 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, **1**,] **27**]
[0], 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, **1**,] **28 P.**
[0], 0, **1**, 0, 0, **1**, 0, **1**, **1**, 0, 1, 0, 0, 1, **1**] **29**] … 7 nombres premiers ***q*** ***∈ [N,2N].****(« Si la conjecture était fausse il faudrait **supprimer 4 nombres premiers q,** ainsi que pour les limites précédentes N =28 , 27 , 26… etc. **Ce qui est impossible** car déjà vérifiées lors de ces limites précédentes .!*

[1], 1**, 1,** 1, 0, **1,** 1**, 0, 1,** 1**, 0,** 1, 0**, 0, 1,]31] → → → abscisses < N des nombres premiers p' Ératosthène**

crible **É** et **G** : [0, 0, **1**, 0, 0, **1,** 0, 0, **1**, 0, 0, 0, 0, 0]

Nombres **p' non congru 2n[P]** « ou couple **P'+q = 2N** », de (i) à 29 : **3**

4_) Simulation et illustration des diagonales de congruences dans les 8 Familles de nombres premiers > 5, de la forme **30k + i .**

Goldbach modulo → 30, en progression arithmétique de raison 30 de premier terme 7 ou l'égalité récurrente de Goldbach, devient :

(2N – A) ⇔ (2N +30) – (A + 30) par famille i.

À partir d'une limite n = 300 , qui a vérifié tous les entiers pairs < 600 ,

Simulation et vérification en utilisant simplement la famille **30k + 7** et les nombres **P ≤ √2n**
les entiers pairs 2N, seront vérifiés par les diagonales de congruence → 2N = y = 780 ,
on **repousse la limite de 6**

Secteur gradué de : **1** = entier **non congrus** à **2n[P]** ; 0 = entier **congru à 2n[P]**

Donnez **N: 300** , 600 →; [630; 660; 690; 720; 750; 780] y = 780

secteur des congruences représentant les entiers **non congru =1**,ou congrus à 2n[P] = 0

[**1, 1, 1, 1, 1, 1**, 0, 0, 0, **1**] diagonales des congruences initialisées par Goldbach
[**7,37,67,97,127,157**,187, 217, 247, **277]** → *Abscisses* → ∞

on décale d'un rang, les congruences pour tout 2N+2 :

crible G: [**1**, 1, 0, 1, 0, 1, 0, 1, 1, 0] stop *fin du décalage des diagonales vérifiant la conjecture jusqu'à 2n = 780* ,
crible E: [**1, 1, 1, 1, 1, 1**, 0,] 0, 0, **1**] ⇒ entiers d'Ératosthène = [**7,37,67,97127,157**,187, 217, 247,**277]**

En vérifiant pour **N = 390** , les entiers pairs < 2N = 780 , entiers pairs vérifiée → y = 1140 on **repousse la limite de 12**

On réitère à partir de N = 390 dernière limite N vérifiée.

Donnez **N: 390**, **2N = 780** → 810, 840, 870, 900, 930, 960, 990 , 1020 , 1050 ...1080 ,1110 , 1140 , y = 1140
crible G: [[**1**, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0]
crible E: [1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, **1**]

Donnez **N: 1140**, 2N = 1140 → entiers pairs vérifiés jusqu'à y = (1140 +1110) **on repousse la limites de 37** , quasiment le double , C'est à dire : que lorsque l'on vérifie les entiers pairs inférieur à une limite 2n fixée , on vérifie par la même tous les entiers pairs qui seront vérifié **<** à (2n/30)*2 par la propriété récurrente du crible de Goldbach *le décalage d'un rang des diagonales de congruences* .

Crible G, [***vecteur des congruences***] qui se sont décalées pour tout 2n + ...2 : ***ou vecteur B*** des nombres **q**
,,, [**1**, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1]

crible E: [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, 1, 1, 1, 1, 0, 0, 1, **1**].

n=2250 , 2250/30 = 37*2 = 74 et **74*2** → donne la prochaine limite N= 3840 et tous les entiers pairs < 7680 sont vérifiés

G :[[1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1]

É :[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, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0]

Famille 7 modulo 30 :
Diagonales des congruences en progression arithmétique de raison 30

on peut vérifier avec Ératosthène à quel moment la conjecture deviendrait fausse le cas échéant ; c'est à dire lorsque l'axe des ordonnées des congruences [0] ne croise plus un nombre premier d'Ératosthène sur l'axe des abscisses à la ligne Ln+1 = point X

crible: des diagonales, non congru = 1, sinon 0
↓ axe des ordonnées
↓
(7 ,37 ,67 ,97 127 ,157 ,187] →→ *axe des abscisses, avec A modulo 30.*
↓
1[0, 1, 1, 0, 1, 1, 1] → n=210 ; **2n = 420** [« *l'algorithme progresse donc modulo 15(k), diagonale initialisée par 7,*
2[**1**, 0, 1, 1, 0, 1, 1] n=225 [« *pour tout 2n + 2*, on réinitialise une diagonale de congruences
3[0, **1**, 0, 1, 1, 0, 1, 1]
4[**1**, 0, **1**, 0, 1, 1, 0, 1]
[0, **1**, 0, **1**, 0, 1, 1, 0, 1]
: [**1**, 0, **1**, 0, **1**, 0, 1, 1, 0]
: [**1**, **1**, 0, **1**, 0, **1**, 0, 1, 1, 0] n=300
: [0, 1, **1**, 0, **1**, 0, **1**, 0, 1, 1]
: [**1**, 0, 1, **1**, 0, **1**, 0, **1**, 0, 1, 1]
: [**1**, 1, 0, 1, **1**, 0, **1**, 0, **1**, 0, 1]
: [0, 1, 1, 0, 1, **1**, 0, **1**, 0, **1**, 0, 1]
: [**1**, 0, 1, 1, 0, 1, **1**, 0, **1**, 0, **1**, 0] 375
: [**1**, 1, 0, 1, 1, 0, 1, **1**, 0, **1**, 0, **1**, 0] 390
: [0, 1, 1, 0, 1, 1, 0, 1, 1, 0, **1**, 0, **1**]
: [0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, **1**, 0, **1**]420
: [1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, **1**, 0]435
: [0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, **1**, 0] 450
: [0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, **1**] 465
: [1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, **1**]
: [1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0]
: [1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0]510
: [0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1]525
: [**0, 0, 1, 1, 1, 0**, 0, 1, 0, **0, 1, 1, 0, 1**, 1, **0, 1**, 1] **X = 540 conjecture vraie ,** (avec A = 7, il suffit de décaler d'un rang ces congruences sur l'axe d'abscisse d'Ératosthène ci-dessous , pour vérifier jusqu'où la conjecture est vraie ...
[**0, 0, 1, 1, 1,** 0, 0, 1, **0, 0, 1, 1, 0,** 1, **1, 0, 1,** 1] **X+15 = 555** (*avec A = 37 et qui augmente de 30 pour tout X=15(k+1)*
:[1[1, **1, 1, 1, 1**, 0, 0, **0**, 1, 1, **1, 1, 1**, 1, **0, 1**, 1, **0**] **Ératosthène ; p' < 15k = 540 , Fam 30k + 7**

On comprend pourquoi cet effet boule de neige, augmente le nombre de solutions **p'+ q = 2N** qui tend vers l'infini : lorsque **N → ∞ .**
Car on augmente [*le champ des congruences]* qui se décalera d'un rang pour toute limite N+1 successives.
Mais aussi pourquoi si elle était fausse, le nombre de nombre premiers ***q ∈[N,2N] serait faux , alors qu'il a été vérifié lors des limites N précédentes .!***

****

Autre exemple : Crible G, entiers ***A = impairs de 1 à N modulo 2,*** *N = 600 ; 1,3,5,7,...N*

[0, 0, 0, **1**, 0, 0, **1**, 0, 0, **1**, 0, 0, 0, 0, **1**, 0, 0, 0, **1**, 0, 0, 0, 0, **1, 1**, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, **1**, 0, 0, **1**, 0, 0, **1**, 0, 0, 0, 1, 0, 0,
[1, **1, 1, 1**, 0, **1, 1**, 0, **1, 1**, 0, **1**, 0, 0, **1, 1**, 0, 0, **1**, 0, **1, 1**, 0, **1,** 0, 0, **1**, 0, 0, **1, 1**, 0, 0, **1**, 0, **1, 1**, 0, 0, **1**, 0, **1**, 0, 0, **1**, 0, 0, 0,

1, 0, 0, **1**, 0, **1, 1**, 0, **1**, 0, 0, 0, 0, 0, 0, 0, 0, **1**, 0, 0, **1, 1**, 0, 0, 0, 0, **1, 1**, 0, 0, 0, 0, **1**, 0, 0, **1, 1,** 0, 0, 0, 0, **1, 1**, 0, 0, **1**, 0, **1**,
1, 0, **1, 1**, 0, **1, 1**, 0, **1**, 0, 0, 0, 0, 0, 0, 0, **1**, 0, **1,** 0, 0, **1, 1,** 0, 0, 0, 0, **1, 1**, 0, 0, **1,** 0, 0, **1**, 0, **1**, 0, 0, **1**, 0, 0, **1, 1**, 0, 0, 0, 0, 0, **1**,

0, 0, 0, 0, 0, **1**, 0, 0, **1**, 0, 0, 0, **1**, 0, 0, **1**, 0, 0, **1**, 0, **1**, 0, 0, 0, 0, 0, 0, **1**, 0, 0, **1**, 0, 0, **1**, 0, **1**, 0, 0, 0, 0, **1**, 0, 0, 0, 0, 0, **1**, 0, 0, 0,
1, 0, **1, 1,** 0, 0, 0, 0, 0, **1**, 0, 0, 0, 0, 0, 0, **1**, 0, **1, 1**, 0, **1,** 0, 0, **1, 1**, 0, 0, 0, 0, **1,** 0, 0, **1**, 0, 0, **1**, 0, 0, 0, **1**, 0, 0, 0, **1, 1**, 0, 0, **1**, 0, **1, 1**, 0, 0,

1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1,

0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,
0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1,

1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0,

0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1]
1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1]

Crible E entiers A impairs (1,3,5,7,,,,,N

[1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0,
1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1,
1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1,
0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1,
0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0,
1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1]

crible G N = 620
[0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1,[**0, 0, 0**, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1,
1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1,
1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0,
0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1]

Crible E
[1[, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0,[**1, 1, 0**, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0,
1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0,
1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0,
1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, [*1, 0, 0, 1, 0, 0, 1, 0, 1, 1*]

****

Crible E et G fusionné , N = 620 , **2N = 1240**
[0, **1**, 0, 0, 0, **1**, 0, 0, **1**, 0, 0, **1**, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, **1**, 0, 0, **1**, 0, 0, **1**, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, **1**,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Nombres p' non congru 2n[P] ou couples **p'+ q = 2N**, de (i = 1) à 620 pour les entiers A impairs **1 modulo 2** :
33 ----- 0.0

La vérification de la conjecture pour une limite N ⇒ N+1; « plus généralement pour N ≥ 10^9, entraîne la vérification de la conjecture pour des limites N ⇒ N+1+2+3+..x; successives avec un effet boule de neige . »

Le Décalage d'un rang des congruences sur les **,1,** ou les **,0,** non congrus à 2N[P], relatif à la propriété de l'algorithme, vérifiera la conjecture pour les nombres pairs 30(k+1) + 2i , soit la vérification des entiers **A ≢ 2N[P]**, pour la limite
N = 15k + 1 = 3001 , ⇒ le nombre de décompositions de 2N = 6002; Ce qui rend aussi impossible, l'infirmation de cette conjecture , même par famille 30k + i , exemple avec l'algorithme modulo 30 :

Famille 30k + i
Donnez N = 15k = 3000 : on fixe la famille Fam(i) les **1** sont les **p'** non congruent à P
fam1 crible G : et : Fam 1 **É**ratosthène **EG**Crible pour N=3000 : conjecture vérifiée **pour tout n** ≤ 3000.
[0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,0, 0, 0, 0, 1] **19** ∓ 1 couples **P'+ q** = 6000.

On suppose maintenant , que pour **2N + 2** = 6002 il n'y a pas de solution : On peut reprendre cette même Fam i =1 pour la limite =15k+ 1 = 3001 qui a été vérifiée pour la limite précédente N=3000 (ie : *on a criblé les **A** ≢**2N[P]** , ≤ N*) , relative à sa Fam (**i = 1**) pour affirmer que la conjecture serra vérifiée pour la limite suivante 15k + 1 , tel que 30k + 2 = 6002 est somme de deux premiers **p'+ q** ! Ce n'est que le résultat des indexes départ des deux cribles Goledbach et Ératosthène, qui sont incompatibles et en utilisant le même principe.

Le décalage d'un rang des congruences sur leur successeur **A+30**, contredit une supposition fausse !

<u>ECrible</u> les 1 sont premiers P' ≢ 2n[P] les 0 ≢ 2n[P] sont les multiple de P fam 30k+ i = 1;
[1, 1, 1, 0, 0, **1, 1, 1**, 1, 1, 0, 1, 0, 0, **1**, 0, 0, 0, 1, **1**, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, **1**, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0]
<u>Gcrible</u> les 1 sont les entiers A ≢ 2n[P]
[0, 0, 0, 0, 1, **1, 1, 1**, 0, 0, 1, 0, 1, 0, **1**, 0, 1, 0, 0, **1**, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, **1**, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0]

Ce qui donnera au minimum 17 couples **P'+ q** qui vérifieront **30(k+1) + 2i = 6002**.
Vérification avec l'algorithme **EG**Crible fusioné :
[0, 0, 0, 0, 0, **1, 1, 1**, 0, 0, 0, 0, 0, 0, **1**, 0, 0, 0, 0, **1**, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, **1**, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1] **réel** :18 couples p+q = 2N. On peut montrer suivant cette propriété de l'algorithme de Goldbach, qu'il en sera de même sur les 14 limites successives suivantes. Ce ne sont que les antécédents décalés d'un rang, relatif à la famille i concernée !

<u>**Autre EX :**</u> avec 2N+2 = 6004 on vérifie avec la famille i = 11 et la fam complémentaire serra i = 23, pour la limite N = 3001 + 1:

N = 3002; fam i = 11 ; **E**crible P' = 1 ou 0 ⇔ ≢ 2n [P] et en dessous **G**crible Entier A = 1 ⇔ ≢ 2n [P] et P ≤ √2n avec **le décalage des congruences de 1 rangs**, des **A** non congrus **1** ou **0** qui vont se reporter sur les A+30 = **P'= 1 ou :** 1 est congrus ou pas, à 2N modulo P de la limite précédente , qui vérifiera au minimum cette limite N+1 suivante, tel que 2N = 6004 = p'+ q ; relatif à (15(k-1)+ 2) [P].
Eratosthène Fam 11
[1, 1, 1, **1**, 1, 0, **1**, 0, 1, 1, **1**, 0, 0, 1, **1**, 1, 1, **1**, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1] soit 21 p' ≢ 2N [P] .
Goldbach Fam 11
[0, 0, 0, **1**, 0, 1, **1**, 1, 0, 0, **1**, 0, 0, 0, **1**, 0, 0, **1**, 0, 0, **1**, 0, **1, 1, 1**, 0, 0, 0, **1**, 0, 0, 0, 0, **1, 1, 1**, 0, 0, 0, 0, **1**, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1] soit 21 entiers **A** non congrus modulo P, qui ce sont décalés d'un rang sur leur successeur p'
vérification avec **E & G , famille 11 , de i à 3002**
[0, 0, 0, **1**, 0, 0, **1**, 0, 0, 0, **1**, 0, 0, 0, **1**, 0, 0, **1**, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0] réel : **21** couples **p+q = 2N = 6004**

Nous venons de vérifier et de constater le décalage récurrent des congruences des **A = 1** ou **A= 0** où : on ne s'occupe pas de savoir si ce **A** est un nombre premier ou pas, mais qu'il soit non congruent (mod P), d'où le décalage d'un rang des congruences qui se reportent sur leur successeur **A+30 = 1** ou **0** et si **A'+ 30 = p' = 1** il vérifiera la conjecture car ils ont donc pour antécédent, **A ≠ 2N[P] ;** le contraire contredirait le TNP et Le TFA, (cqfd)

La conjecture se montre avec l'une des 8 Fam(i) en fonction de la forme de N et pour toute limite N ≥ 150 :
Les nombres A ≠ 2N[P] ⇔ q ∈ [N;2N] sont donc définis pour 2N + 2 à une exception près !

Car les restes R de 30(k-1)+2i par P, ne peuvent plus être utilisés du simple fait qu'ils ne correspondent pas aux nouveaux R de 30k + 2i par P , lorsque N augmente de 1.
Si on utilisait les R de 30(k -1)+ 2i et les R de 30k + 2i, le cardinal du nombre de nombres premiers **q** serait faux pour la limite 2N+2 qui a déjà été fixé et vérifié avec les nombres P qui ont criblés la limite N – 15 précédente.

Mais qui plus est, l'index de départ des nombres P qui criblent suivant le même principe dans les deux algorithmes est différent entre Ératosthène et Goldbach , ce qui nous assure un minimum de **A=p' ≠ 2N[P] ;**
c'est à dire : un minimum de couples **p' + q = 2N et une infirmation de la conjecture, impossible.**

La répartition des nombres premiers et du TNP, est une conséquences directe d'une conjecture de Goldbach vraie et non l'inverse . !

Densité de couple P+q par limite de 10 000 -10, ci-dessous ; programme C++ utilisé

Rapport nombre du nombre P'≠ 2n(P] ou couples (p+q) par rapport au nombre de P' < limite n criblée par Famille:

N =9991 :
Fam 1, nbr de p' < 9991 = 153 , nbr de couples A ≠ 2N[P] ou p+q = 2n , total 48 ; 153/48 = 3,18...
Fam 13, nbr de p' < 9991 = 154 , nbr de couples A ≠ 2N[P] ou p+q = 2n , total 59 : 154/59 = 2,61.
Fam 19, nbr de p' < 9991 = 150 , nbr de couples A ≠ 2N[P] ou p+q = 2n , total 56 : 150/56 = 2,67.

N =19981 :
Fam 1, nbr de p' < 19981 = 276, nbr de couples A ≠ 2N[P] ou p+q = 2n , total 103 ; 176/103 = 2,67…

Fam 13, nbr de p' < 19981 = 284 , nbr de couples A ≠ 2N[P] ou p+q = 2n , total 106 : 284/106 = 2,67.

Fam 19, nbr de p' < 19981 = 277 , nbr de couples A ≠ 2N[P] ou p+q = 2n , total 112 : 277/112 = 2,47.

N = 29971 :
Fam 1, nbr de p' < 29971 = 403, nbr de couples A ≠ 2N[P] ou p+q = 2n , total 103 ; 403/150 = **2,68**…

Fam 13, nbr de p' < 29971 = 405, nbr de couples A ≠ 2N[P] ou p+q = 2n , total 106 : 405/151 = **2,68.**..

Fam 19, nbr de p' < 29971 = 394, nbr de couples A ≠ 2N[P] ou p+q = 2n , total 112 : 394/151 = **2,60.**..

N = 39961 :
Fam 1, nbr de p' < 39961 = 514, nbr de couples A ≠ 2N[P] ou p+q = 2n , total 103 ; 514/159 = **3,32**…

Fam 13, nbr de p' < 39961 = 531, nbr de couples A ≠ 2N[P] ou p+q = 2n , total 106 : 531/180 = **2,95.**..

Fam 19, nbr de p' < 39961 = 513, nbr de couples A ≠ 2N[P] ou p+q = 2n , total 112 : 513/161 = **3,18**…

N = 49951 :
Fam 1, nbr de p' < 49951 = 632, nbr de couples A ≠ 2N[P] ou p+q = 2n , total 103 ; 632/233= **2,71**…

Fam 13, nbr de p' < 49951 = 648, nbr de couples A ≠ 2N[P] ou p+q = 2n , total 106 : 648/249 = **2,60**...

Fam 19, nbr de p' < 49951 = 628, nbr de couples A ≠ 2N[P] ou p+q = 2n , total 112 : 628/213 = **2,94**…

en poussant la limite jusqu'à la limite N = 6 600 000 000 010 , soit 660660660 * 9990 on obtient une rapport de 8,49…
exemple :
Fam 1, nbr de p' < 6 600 000 000 010 = 28967518720, nbr de couples A ≠2N[P] ou p+q = 2n , total 3411256593; 28967518720 ÷ 3411256593 = 8,49…

D'où on peut conjecturer : que pour tout limite n >3000 000 criblée, il exsite un entier premier **p'≠ 2n[P]** entre 1 et racine carré de 3000 000 , et au minimum jusqu à **9* 10^{18} pour toute famille 30k+i. Ie :** qu'il existe toujours une solution telle que **2n = p'+ q** , « *voir progamme en C++ , page 19 ci après.* »,

Annexe 1

En fonction de la limite **N = 15k** fixée, on fixe lune des **8 Fam (i)** correspondante à la forme de **N** qui implique le complémentaire par rapport à **2N .**

Pour **N** de la forme **15k**, on peut choisir n'importe la quelle des 8 Fam.

Le programme est fixé avec une limite **N** début = 3000000000 et Fin = 3000000330 il progressera de raison 15,
Il n'y a que la fam(i) à rentrer à la demande:

Pour toute Limite **N = 15k + n'**, avec **n' ∈ [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14]** on rentre la Fam(i) correspondante

N = 15k+1; Fam =(1,13,19); ⇒ **2n = 30k + 2** ⇒ **32 – 19 = 13** , ou **32 – 1 = 31** ⇒ la Fam complémentaire
N = 15k+2; Fam =(11,17,23); ⇒ **2n = 30k + 4** ⇒ **34 – 11 = 23** , ou **34 – 17 = 17**
N = 15k+3; Fam =(7,29,13,23,17,19); **2n = 30k + 6**
N = 15k+4; Fam =(1,7,19); 2n = **30k + 8**
N = 15k+5; Fam =(1,7,13,19); **2n = 30k + 10**
N = 15k+6; Fam =(1,11,13,19,23,29); **2n = 30k + 12**
N = 15k+7; Fam =(1,7,13); 2n = **30k + 14**
N = 15k+8; Fam =(17,23,29); 2n = **30k + 16**
N = 15k+ 9; Fam =(1,7,11,17,19,29); 2n = **30k + 18**
N = 15k+10; Fam =(11,29,17,23); 2n = **30k + 20**
N = 15k+11; Fam =(11,23,29); **2n = 30k + 22**
N = 15k+ 12; Fam =(1,7,11,13,17,23); 2n = **30k + 24**
N = 15k+ 13; Fam =(7,13,19); 2n = **30k + 26**
N = 15k+ 14; Fam =(11,17,29); 2n = **30k + 28**

Pour les multiples de 30, avec N = 15k; l'une des 8 Fam , 2n = 30k

---

Programme en C++ *paramétré pour cribler* ***jusqu'à la racine carrée de n > 90 0000 000 000 000*** *par famille* ***i*** *tel que définie. Attention la limite* ***n*** *minimum doit être* ***≥ 3 000 000 , sinon on remet la valeur n normale :***
size_t lencrible = sqrt(limite)/30 ou **size_t lencrible = limite/30** pour cribler des valeurs n entières

Attention, *on choisie la Fam ou les Fam 30k+i en fonction de la forme de 2n montrée ci dessus .*

****

```
// -*- 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 ECrible(const vector<ulonglong> &premiers, ulonglong n, int fam, vector<bool> &crible, size_t lencrible)
{ //on va contruire un tableau de 1 modulo 30 et rappeler les premiers p
  int cl = clock();
  // size_t lencrible = n / 30,
  size_t nbpremiers = premiers.size(); //on va contruire un tableau de 1 modulo 30 en divisant N par 30
```

```
  //vector<bool> crible(lencrible, true);                  // on rappelle les nombres premiers p d'Eratotene
  // 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}; // on va calculer le produit de p par un element du groupe GM
    for (size_t j = 0; j < sizeof(GM) / sizeof(int); j++)
    {
      produit = p * GM[j]; // calcul du produit, jusqu'a ce que le produit soit égale à fam modulo 30
      if (produit % 30 == fam)
      {
        produit /= 30; // puis on va va calculer l'indice, afin de commencer à cribler de l'indice à n/30 et on réitère
        break;
      }
    }
    indices[i] = produit;
  }
  ulonglong nslices = lencrible / 45000000, currentslice = 0;   //on peut modifier la valeur des slices
  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 << " Nbr p' criblés Fam " << fam << " < a " << n << ": " << total; // << " time " << (clock() - cl) * 1e-6 <<
endl;
  return total; // à la fin du crible on return le résultat est le temps mis
}

size_t GCrible(const vector<ulonglong> &premiers, ulonglong n, int fam, vector<bool> &crible, size_t lencrible)
{
  int cl = clock();
  //size_t lencrible = n / 30,
  size_t nbpremiers = premiers.size(); //on rappelle le tableau d'ératosthène criblé
  //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; // on calcule le reste de 2n par p
    if (reste % 2 == 0)
      reste += p;
    ulonglong pi2 = 2 * p;
    while (reste % 30 != fam) // tant que le reste += p n'est pas = à Fam % 30 on rajoute 2*p
      reste += pi2;
```

```
    reste /= 30; // ensuite on va calculer l'indice de départ de p pour commencer à cribler les 1.1.1...on remplace 1 par 0 si congru mod p, de l'indice à n/30
    indices[i] = reste;
  }
  ulonglong nslices = lencrible / 45000000, currentslice = 0;   //on peut modifier la valeur des slices
  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]); // le criblage du tableau de 1 modulo 30 jusqu'a n/30 (1.1.1.1...etc) est fini on va retourner le résultat
  cout << " Nbr couple p+q criblés Fam " << fam <<  " : " << total;// " time " << (clock() - cl) * 1e-6 << endl;
  return total;
}

int main(int argc, char **argv)
{
  vector<unsigned> temp;  // on entre la valeur de la limite n à cribler et la fin augmenté de 15 , pour cribler jusqu'à n ou la sqrt de n, voir ligne 186
  ulonglong debut = 900000000000000;
  ulonglong fin = 900000000000015;

  vector<int> familles;  // On choisit la ou les familles en fonction de la forme de 2n, pour avoir le bon complémentaire q premier
  //familles.push_back(1);
  familles.push_back(7);
  familles.push_back(11);
  //familles.push_back(13);
  //familles.push_back(17);
  //familles.push_back(19);
  //familles.push_back(23);
  //familles.push_back(29);

  for (int i = 0; i < familles.size(); i++)
  {
    int fam = familles[i];

    for (ulonglong limite = debut; limite < fin; limite += 15){
      cout << "--> limite : " << limite << endl;
      double sqrt2N = unsigned(std::sqrt(2 * double(limite)));
      fill_crible(temp, sqrt2N);
      vector<ulonglong> premiers;
      for (ulonglong p = 7; p <= sqrt2N;)
      {
        premiers.push_back(p);
        p = nextprime(temp, p);
        if (p == unsigned(-1))
```

```
        break;
      }

      size_t lencrible = sqrt(limite)/30;   // attention on crible les p' non congru 2N[P] ≤ √ de la limite n fixée
      vector<bool> crible(lencrible, true);
      ECrible(premiers, limite, fam, crible, lencrible);
      GCrible(premiers, limite, fam, crible, lencrible);
    }
   }
}
```

****

*programme **crible de Goldbach** ci-joins, python modulo 2 utilisé :*

Programme Python (fam 1 **modulo 2):**

```
from time import time
from os import system
import math

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 = [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)
    return prime_list[0:]


def demander_N():
    n = input("Donnez N: ")
    n = int(n.strip().replace(" ", ""))

    return n


def G_crible(premiers, n, fam):
    start_crible = time()
    crible = (n//2)*[1]     ## Ou: on rappel le tableau Ératosthène criblé de N/2 cases
    lencrible = len(crible)

    ## On calcule les restes: R = 2*n modulo P
    nbpremiers = len(premiers)
    n2 = 2*n
```

```
    for i, premier in enumerate(premiers):
        reste = n2 % premier
        #print(reste)
        if reste % 2 == 0:
            reste += premier
        pi2 = 2*premier
        while reste % 2 != fam:   ## tant que R % 2 != fam on fait R+= 2P
            reste += pi2
                        ## Ensuite on divise R par 2 pour obtenir l'indexe , indice.
        reste //= 2
                        ## On crible directement à partir du n° indice l'index que l'on marque 0
        for index in range(reste, lencrible, premier):
            crible[index] = 0

    total = sum(crible)
    print("crible:", crible)
    print(f"Nombres non congru 2n[P] de {1} à {n} famille {fam} nbr premiers q 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 de 3 à √2N
    premiers = eratostene(n)
    #lprint("premiers:", premiers)
    #print(f"nombres premiers de3 à {int((2*n)**0.5)}: {len(premiers)}")

    start_time = time()
    ## On crible
    G_crible(premiers, n, 1)
    temps = time()-start_time
    print(f"--- Temps total: {int(temps*100)/100} sec ---")


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

Programme Python (fam 1 **modulo 2) [Ératosthène et Goldbach unifié]:**
Attention j'ai repris: l'algorithme par famille 30k + i ; modulo 30 pour le transformer en crible modulo 2 , afin de l'utiliser dans les entiers A impairs de premier terme 1 en progression arithmétique de raison 2 , uniquement pour les petite valeurs afin d'illustrer les commentaires ci-dessus.
On peut réduire la taille du tableau à la racine carrée de n > 90 000

```
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)  ##(si on fusionne les deux cribles il faudra rentrer, int((2n)**0.5) pour Goldbach.
    prime_E =[3]
    sieve_list = [True for _ in range(n+1)] ## c'est plus propre comme ça
    for each_number in candidate_range(n):
        if sieve_list[each_number]:
            prime_E.append(each_number)
            for multiple in range(each_number*each_number, n+1, each_number):
                sieve_list[multiple] = False
    print(prime_E[0:])
    return prime_E[0:]

def E_Crible(premiers, n, fam):   ## on peut réduire la taille du taleau à cribler avec n1 = int((n)**0,5)
    start_crible = time()

    ## On génère un tableau de N/2 cases rempli de 1
    lencrible = ((n//2)*1)  ## ou ((n1//2)*1)
    crible=[1 for _ in range(lencrible)] ## c'est plus propre comme ça
    GM = [3,5,7,11,13,17,19,23,29,31]  ## attention GM > 5 est utilisé pour l'algorithme modulo 30, fam 30k + i
    # On calcule les produits :
    for a in premiers:
        for b in GM:
            j = a * b
            if j%2 == fam:
                index = j // 2  ## Je calcule l'index n° indice et On crible directement à partir du n° indice
                for idx in range(index, lencrible, a):  ## index qui est réutilisé ici...
                    crible[idx] = 0

    total = sum(crible)-1
    print("crible Ératosthène :", crible)  ## pour éditer le tableau Ératosthène criblé
    print(f"Nombre premiers criblés >2, famille {fam} : {total} ----- {int((time()-start_crible)*100)/100}")
    return crible,lencrible

def GCrible_2N(premiers, crible, lencrible, n, fam):
    start_crible = time()
    ## On calcule les restes: R = 2*n modulo P
    nbpremiers = len(premiers)
    n2 = 2*n
    for premier in premiers:
        reste = n2 % premier
        print(reste)
        if reste % 2 == 0:
            reste += premier
        pi2 = 2*premier
       ## tant que reste % 2 != fam on fait reste += pi2 , pi = P(int((2*n)**0.5))
        while reste % 2 != fam:
            reste += pi2
        ## Ensuite on divise reste par 2 pour obtenir l'index
        reste //= 2
        ## On crible directement à partir de l'index
        for index in range(reste, lencrible, premier):
            crible[index] = 0

    total = sum(crible)-1  ## car 1 n'est pas premier donc il n'est pas un couple p'+ q
    print("crible É ET G:", crible)  ## éditer le tableau criblé É et G
    print(f"Nombres p' non congru 2n[P] ou couple P'+q = 2N, de (i) à {n} famille {fam} : {total} ----- {int((time()-
start_crible)*100)/100}")
```

```
def demander_N():
    n = input("Donnez N: ")
    n = int(n.strip().replace(" ", ""))
    #n = int(30 * round(float(n)/30))
    return n

def main():
    ## On demande N a l'utilisateur
    n = demander_N()
    ## On récupère les premiers de 7 à √2N
    premiers = eratostene(n)
    start_time = time()
    ## On crible
    fam=1 ## ou (1, 7, 11, 13, 17, 19, 23, 29, utilisé par famille 30k + i au choix en fonction de n)
    crible,lencrible=E_Crible(premiers, n, fam)
    GCrible_2N(premiers, crible, lencrible, n, fam)

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

****

Programme Python Goldbach : Par Famille 30k + i , déjà réglé sur la famille 30k +7 **,** avec i ∈ [1,7,11,13,17,19,23,29]

```
from time import time
from os import system
import math

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))
    return n


def GCrible(premiers, n, fam):
    start_crible = time()
    crible = (n//30)*[1]     # Ou: on rapelle le tableau Ératosthène criblé de N/30 cases
    lencrible = len(crible)

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

    for i, premier in enumerate(premiers):
        reste = n2 % premier
        #print(reste)
            # tant que ri % 30 != fam on fait R+= 2P
        if reste % 2 == 0:
            reste += premier
        pi2 = 2*premier
        while reste % 30 != fam:
            reste += pi2
        # Ensuite on divise R par 30 pour obtenir l'indexe, indice
        reste //= 30
        # On crible directement à partir du n°d'indice avec P où on remplace le 1 par 0
        for index in range(reste, lencrible, premier):
            crible[index] = 0

    total = sum(crible)
    print("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)
    temps = time()-start_time
    print(f"--- Temps total: {int(temps*100)/100} sec ---")
```

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

Programme Python Ératosthène : D*éjà réglé sur la famille 30k +7 , les deux familles dans les deux algorithmes doivent toujours être les mêmes pour la même limite* ***n*** *fixée ; car on crible Goldbach en utilisant les congruences de* ***1 à n.***
Ce qui évite de s'occuper des nombres premiers ***q*** *complémentaires de la famille des nombres premiers* ***p'*** *d'Ératosthène* ***≤ n*** *; il devient donc sans intérêt et inutile de savoir quel nombres premier* ***q*** *appartenant à* ***[n;2n]*** *est le complémentaire de* ***p'****.*

lorsque 7 est congru à 2n modulo 30 , on sait très bien que le complémentaire q = 23 [30], autrement dit ,
60 – 7= 53 =23[30] ,
attention pour la famille 1 [30] , 1 n'est pas premier donc enlever 1 au résultat final du nombres de nombres premiers.

Ératosthène :

```
from itertools import product
from time import time
from os import system
import math

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)  ##(si on fusionne les deux cribles il faudra rentrer, int((2n)**0.5) pour Goldbach.
    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))
    return n


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 et On crible directement à partir de l'index, du n° d'indice
        for idx in range(index, lencrible, a):  # index qui est réutilisé ici...
            crible[idx] = 0
        #print(index)

    total = sum(crible)  ## à la place, pour utiliser le tableau d'Ératosthène criblé dans le crible de Goldbach, on re-
turn "crible:", crible
    print("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)
    #print(f"nombres premiers entre 7 et n: {len(premiers)}")

    start_time = time()
    ## On crible
    E_Crible(premiers, n, 7)  ## au choix (1,7,11,13,17,19,23,29)
    temps = time()-start_time
    print(f"--- Temps total: {int(temps*100)/100} sec ---")


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

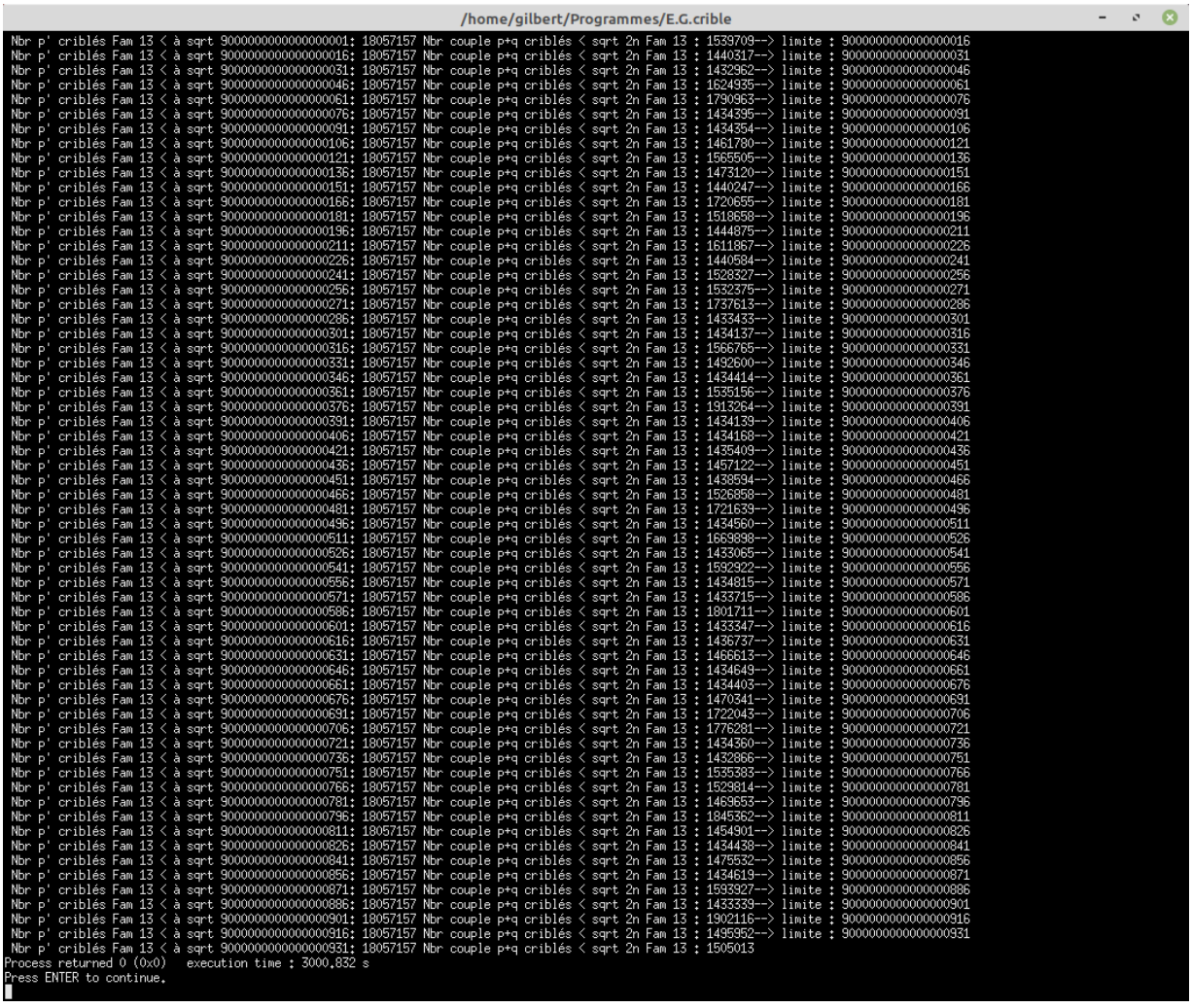
Programme unifié Goldbach et Ératosthène , qui donne le résultat du nombre de décompositions pour tout entier pair **2n = 30k + 2i** , **pour toute limite : n = 15k + i ≤ (√n / 30)** et au minimum, pour **n >3000 000** fixée .
(« Fait par Y sur le forum Bibm@th.net ») et Modifié le 04/11/2024

```
from time import time
from os import system

def candidats(n):
    start_crible = time()

    n = int((2*n)**0.5)
    m = (n-1) // 2
    b = [True]*m
    i = 0
    p = 3
    premiers = [2]
    while p*p < n:
```

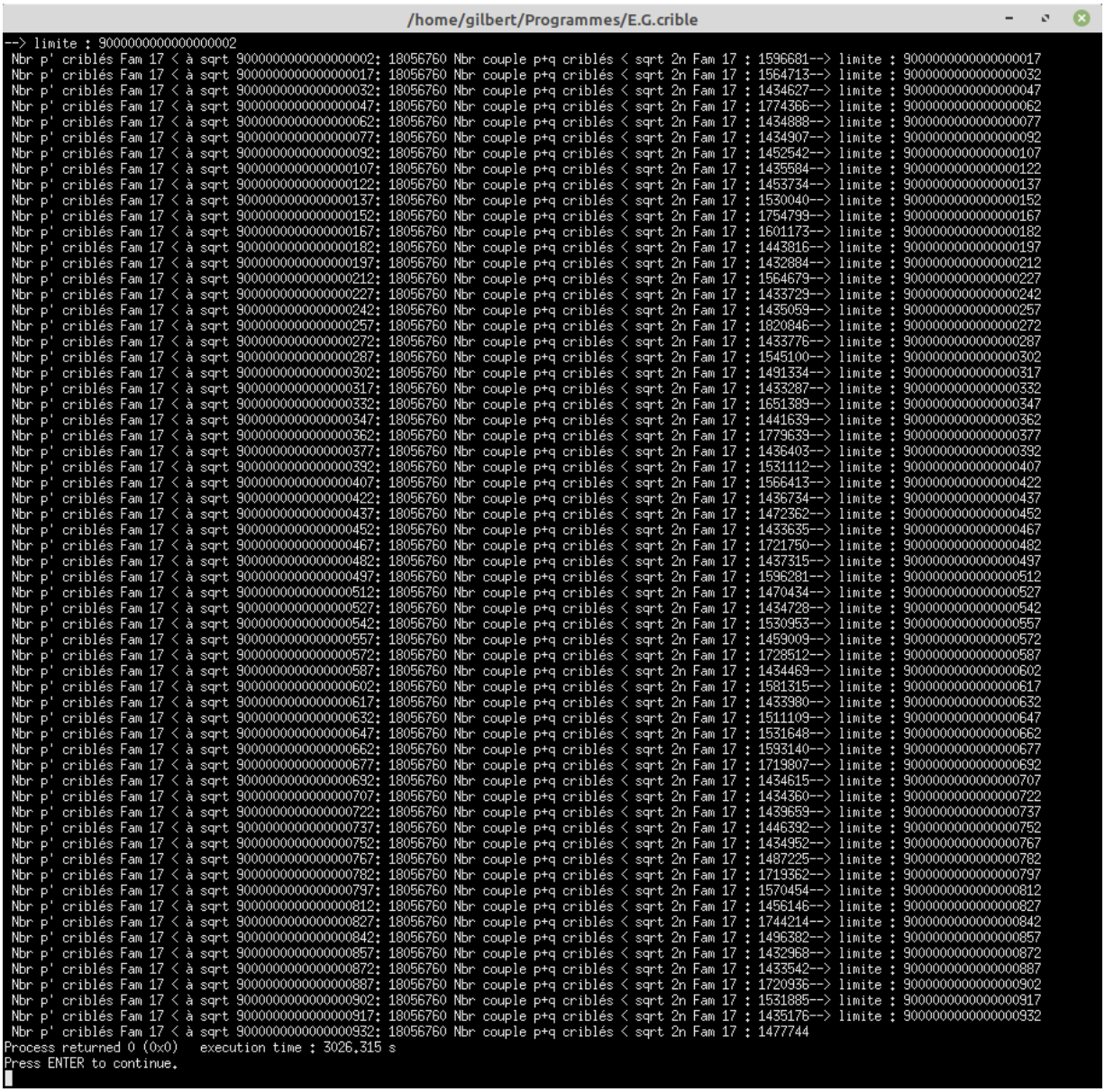
```
        if b[i]:
            premiers.append(p)
            j = 2*i*i + 6*i + 3
            while j < m:
                b[j] = False
                j = j + 2*i + 3
        i += 1
        p += 2
    while i < m:
        if b[i]:
            premiers.append(p)
        i += 1
        p += 2
    print(f"Nombre premiers[3:] : {int((time()-start_crible)*100)/100}")
    return premiers[3:]

def E_Crible(premiers, n, fam):
    start_crible = time()
    n1 = int(n**0.5)

    # On génère un tableau de N/30 cases rempli de 1 < racine carrée de n
    lencrible = n1//30
    crible=[1 for i in range(lencrible)] ## c'est plus propre comme ça
    GM = [7,11,13,17,19,23,29,31]
    # On calcule les produits :
    for a in premiers:
        for b in GM:
            j = a * b
            if j%30 == fam:
                index = j // 30  ## Je calcule l'index et On crible directement à partir de l'index
                for idx in range(index, lencrible, a):  ## index qui est réutilisé ici...
                    crible[idx] = 0

    total = sum(crible)
    #print("crible Ératosthène :", crible)  ## pour éditer le tableau Ératosthène criblé
    print(f"Nombre premiers criblés famille {fam} : {total} ----- {int((time()-start_crible)*100)/100}")
    return crible,lencrible

def GCrible_2n(premiers, crible, lencrible, n, fam):
    start_crible = time()
    # On calcule les restes: r = 2*n/P
    nbpremiers = len(premiers)
    n2 = 2*n
    for premier in premiers:
        reste = n2 % premier
        #print(reste)
        if reste % 2 == 0:
            reste += premier
        p2 = 2*premier
       ## tant que reste % 30 != fam on fait reste += p2
        while reste % 30 != fam:
            reste += p2
        ## Ensuite on divise reste par 30 pour obtenir l'index
        reste //= 30
        ## On crible directement à partir de l'index le tableau d'Ératosthène
        for index in range(reste, lencrible, premier):
            crible[index] = 0

    total = sum(crible)
    #print("crible É ET G:", crible) ## éditer le tableau criblé É et G
```

```
    print(f"Nombres p' non congru 2n[P] < sqrt n , ou couple p'+q = 2n, de (1) à sqrt de {n} famille {fam} : {total}
----- {int((time()-start_crible)*100)/100}")

def demander_n():
    n = input("Donnez n: ")
    n = int(n.strip().replace(" ", ""))
    #n = int(30 * round(float(n)/30))
    return n

def main():
    ## On demande n a l'utilisateur
    n = demander_n()
    ## On récupère les premiers de 7 à √2n
    premiers = candidats(n)
    start_time = time()
    ## On crible
    fam=7   ## ou 1, 7, 11, 13, 17, 19, 23, 29, au choix en fonction de n
    crible,lencrible=E_Crible(premiers, n, fam)
    GCrible_2n(premiers, crible, lencrible, n, fam)

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

estimation asymptotique Nombre **N** ((≤√**n**)/3) de p' à cribler =***18057157*** , soit 2N: 18057157 *2 = **36 114 314**
18 057 157 **/ Ln 36114314 = 1 037 636 p' ≠ 2N[P] ou** couple p'+ q , (≤ √ **à 2*(**9*10^18)) environ



,


/home/gilbert/Programmes/E.G.crible
--> limite : 900000000000000002
Nbr p' criblés Fam 17 < à sqrt 900000000000000002: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1596681--> limite : 900000000000000017
Nbr p' criblés Fam 17 < à sqrt 900000000000000017: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1564713--> limite : 900000000000000032
Nbr p' criblés Fam 17 < à sqrt 900000000000000032: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1434627--> limite : 900000000000000047
Nbr p' criblés Fam 17 < à sqrt 900000000000000047: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1774366--> limite : 900000000000000062
Nbr p' criblés Fam 17 < à sqrt 900000000000000062: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1434888--> limite : 900000000000000077
Nbr p' criblés Fam 17 < à sqrt 900000000000000077: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1434907--> limite : 900000000000000092
Nbr p' criblés Fam 17 < à sqrt 900000000000000092: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1452542--> limite : 900000000000000107
Nbr p' criblés Fam 17 < à sqrt 900000000000000107: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1435584--> limite : 900000000000000122
Nbr p' criblés Fam 17 < à sqrt 900000000000000122: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1453734--> limite : 900000000000000137
Nbr p' criblés Fam 17 < à sqrt 900000000000000137: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1530040--> limite : 900000000000000152
Nbr p' criblés Fam 17 < à sqrt 900000000000000152: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1754799--> limite : 900000000000000167
Nbr p' criblés Fam 17 < à sqrt 900000000000000167: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1601173--> limite : 900000000000000182
Nbr p' criblés Fam 17 < à sqrt 900000000000000182: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1443816--> limite : 900000000000000197
Nbr p' criblés Fam 17 < à sqrt 900000000000000197: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1432884--> limite : 900000000000000212
Nbr p' criblés Fam 17 < à sqrt 900000000000000212: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1564679--> limite : 900000000000000227
Nbr p' criblés Fam 17 < à sqrt 900000000000000227: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1433729--> limite : 900000000000000242
Nbr p' criblés Fam 17 < à sqrt 900000000000000242: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1435059--> limite : 900000000000000257
Nbr p' criblés Fam 17 < à sqrt 900000000000000257: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1820846--> limite : 900000000000000272
Nbr p' criblés Fam 17 < à sqrt 900000000000000272: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1433776--> limite : 900000000000000287
Nbr p' criblés Fam 17 < à sqrt 900000000000000287: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1545100--> limite : 900000000000000302
Nbr p' criblés Fam 17 < à sqrt 900000000000000302: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1491334--> limite : 900000000000000317
Nbr p' criblés Fam 17 < à sqrt 900000000000000317: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1433287--> limite : 900000000000000332
Nbr p' criblés Fam 17 < à sqrt 900000000000000332: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1651389--> limite : 900000000000000347
Nbr p' criblés Fam 17 < à sqrt 900000000000000347: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1441639--> limite : 900000000000000362
Nbr p' criblés Fam 17 < à sqrt 900000000000000362: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1779639--> limite : 900000000000000377
Nbr p' criblés Fam 17 < à sqrt 900000000000000377: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1436403--> limite : 900000000000000392
Nbr p' criblés Fam 17 < à sqrt 900000000000000392: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1531112--> limite : 900000000000000407
Nbr p' criblés Fam 17 < à sqrt 900000000000000407: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1566413--> limite : 900000000000000422
Nbr p' criblés Fam 17 < à sqrt 900000000000000422: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1436734--> limite : 900000000000000437
Nbr p' criblés Fam 17 < à sqrt 900000000000000437: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1472362--> limite : 900000000000000452
Nbr p' criblés Fam 17 < à sqrt 900000000000000452: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1433635--> limite : 900000000000000467
Nbr p' criblés Fam 17 < à sqrt 900000000000000467: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1721750--> limite : 900000000000000482
Nbr p' criblés Fam 17 < à sqrt 900000000000000482: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1437315--> limite : 900000000000000497
Nbr p' criblés Fam 17 < à sqrt 900000000000000497: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1596281--> limite : 900000000000000512
Nbr p' criblés Fam 17 < à sqrt 900000000000000512: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1470434--> limite : 900000000000000527
Nbr p' criblés Fam 17 < à sqrt 900000000000000527: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1434728--> limite : 900000000000000542
Nbr p' criblés Fam 17 < à sqrt 900000000000000542: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1530953--> limite : 900000000000000557
Nbr p' criblés Fam 17 < à sqrt 900000000000000557: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1459009--> limite : 900000000000000572
Nbr p' criblés Fam 17 < à sqrt 900000000000000572: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1728512--> limite : 900000000000000587
Nbr p' criblés Fam 17 < à sqrt 900000000000000587: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1434469--> limite : 900000000000000602
Nbr p' criblés Fam 17 < à sqrt 900000000000000602: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1581315--> limite : 900000000000000617
Nbr p' criblés Fam 17 < à sqrt 900000000000000617: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1433980--> limite : 900000000000000632
Nbr p' criblés Fam 17 < à sqrt 900000000000000632: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1511109--> limite : 900000000000000647
Nbr p' criblés Fam 17 < à sqrt 900000000000000647: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1531648--> limite : 900000000000000662
Nbr p' criblés Fam 17 < à sqrt 900000000000000662: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1593140--> limite : 900000000000000677
Nbr p' criblés Fam 17 < à sqrt 900000000000000677: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1719807--> limite : 900000000000000692
Nbr p' criblés Fam 17 < à sqrt 900000000000000692: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1434615--> limite : 900000000000000707
Nbr p' criblés Fam 17 < à sqrt 900000000000000707: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1434360--> limite : 900000000000000722
Nbr p' criblés Fam 17 < à sqrt 900000000000000722: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1439659--> limite : 900000000000000737
Nbr p' criblés Fam 17 < à sqrt 900000000000000737: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1446392--> limite : 900000000000000752
Nbr p' criblés Fam 17 < à sqrt 900000000000000752: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1434952--> limite : 900000000000000767
Nbr p' criblés Fam 17 < à sqrt 900000000000000767: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1487225--> limite : 900000000000000782
Nbr p' criblés Fam 17 < à sqrt 900000000000000782: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1719362--> limite : 900000000000000797
Nbr p' criblés Fam 17 < à sqrt 900000000000000797: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1570454--> limite : 900000000000000812
Nbr p' criblés Fam 17 < à sqrt 900000000000000812: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1456146--> limite : 900000000000000827
Nbr p' criblés Fam 17 < à sqrt 900000000000000827: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1744214--> limite : 900000000000000842
Nbr p' criblés Fam 17 < à sqrt 900000000000000842: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1496382--> limite : 900000000000000857
Nbr p' criblés Fam 17 < à sqrt 900000000000000857: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1432968--> limite : 900000000000000872
Nbr p' criblés Fam 17 < à sqrt 900000000000000872: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1433542--> limite : 900000000000000887
Nbr p' criblés Fam 17 < à sqrt 900000000000000887: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1720936--> limite : 900000000000000902
Nbr p' criblés Fam 17 < à sqrt 900000000000000902: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1531885--> limite : 900000000000000917
Nbr p' criblés Fam 17 < à sqrt 900000000000000917: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1435176--> limite : 900000000000000932
Nbr p' criblés Fam 17 < à sqrt 900000000000000932: 18056760 Nbr couple p+q criblés < sqrt 2n Fam 17 : 1477744
Process returned 0 (0x0) execution time : 3026,315 s
Press ENTER to continue.

,