```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""Reproduction des calculs du document Produit local et décompositions de
Goldbach.

Definitions:
 G(n) = nombre de decompositions non ordonnees n=p+q, p<=q, p,q premiers.
 B(n) = produit_{q<=sqrt(n), q>2 premier} (q-2)/q
 C(n) = produit_{q|n, q>2 premier} (q-1)/(q-2)
 R(n) = G(n)/(n*B(n)*C(n))

Dependance: numpy
Exemple: python produit_local_goldbach.py --max-n 300000
"""
from __future__ import annotations
import argparse, csv, math, statistics
import numpy as np

def sieve(n):
    p=np.ones(n+1,dtype=bool); p[:2]=False
    for q in range(2, math.isqrt(n)+1):
        if p[q]: p[q*q:n+1:q]=False
    return p

def goldbach_counts(max_n,is_prime):
    a=is_prime.astype(np.float64)
    size=1
    while size <= 2*max_n: size <<= 1
    conv=np.fft.irfft(np.fft.rfft(a,size)**2,size)
    ordered=np.rint(conv[:max_n+1]).astype(np.int64)
    G=np.zeros(max_n+1,dtype=np.int64)
    ev=np.arange(4,max_n+1,2)
    diagonal=is_prime[ev//2].astype(np.int64)
    G[ev]=(ordered[ev]+diagonal)//2
    return G

def B_values(max_n,is_prime):
    B=np.ones(max_n+1,dtype=np.float64); cur=1.0
    for q in np.flatnonzero(is_prime):
        if q<=2: continue
        q=int(q); q2=q*q
        if q2>max_n: break
        cur *= (q-2.0)/q
        B[q2:]=cur
    return B

def C_values(max_n,is_prime):
    C=np.ones(max_n+1,dtype=np.float64)
    for q in np.flatnonzero(is_prime):
        if q<=2: continue
        q=int(q)
        if q>max_n: break
        C[q::q] *= (q-1.0)/(q-2.0)
    return C

def compute(max_n):
    isp=sieve(max_n)
    G=goldbach_counts(max_n,isp)
    B=B_values(max_n,isp); C=C_values(max_n,isp)
    R=np.full(max_n+1,np.nan,dtype=np.float64)
    ev=np.arange(4,max_n+1,2)
    R[ev]=G[ev]/(ev*B[ev]*C[ev])
    return isp,G,B,C,R
```

```
def fam_2p(primes,m): return [2*int(p) for p in primes if p>2 and 2*p<=m]
def fam_2p2(primes,m): return [2*int(p)*int(p) for p in primes if p>2 and
2*p*p<=m]
def fam_4p(primes,m): return [4*int(p) for p in primes if p>2 and 4*p<=m]
def vals(R,fam,lo=100): return [(n,float(R[n])) for n in fam if n>=lo and
np.isfinite(R[n])]
def med(v,lo,hi):
    x=[r for n,r in v if lo<=n<hi]
    return statistics.median(x) if x else None

def report(max_n):
    isp,G,B,C,R=compute(max_n); primes=np.flatnonzero(isp)
    print('============================================================')
    print('Produit local et decompositions de Goldbach')
    print('============================================================')
    print(f'Calcul jusqu a n = {max_n}\n')
    for lo,hi in [(100,1000),(1000,5000),(5000,10000),(10000,50000),
(50000,100000)]:
        x=R[lo:min(hi,max_n+1)]; x=x[np.isfinite(x)]
        if len(x): print(f'{lo:6d}-{min(hi,max_n+1)-1:6d}:
med={np.median(x):.5f} mean={np.mean(x):.5f}')
    for n in (1000,10000,100000):
        if n<=max_n: print(f'R({n}) = {R[n]:.7f}')

families={'2p':vals(R,fam_2p(primes,max_n)),'2p^2':vals(R,fam_2p2(primes,max_n))
,'4p':vals(R,fam_4p(primes,max_n))}
    print('\nFamilles n=2p, 2p^2, 4p (n>=100)')
    for name,v in families.items():
        x=[r for _,r in v]; n,r=min(v,key=lambda z:z[1])
        p=n//2 if name=='2p' else n//4 if name=='4p' else math.isqrt(n//2)
        print(f'{name:4s}: median={statistics.median(x):.5f}
mean={statistics.mean(x):.5f} min n={n} p={p} R={r:.5f}')
    print('\nTranches: interval       2p       2p^2      4p')
    for lo,hi in [(100,1000),(1000,5000),(5000,10000),(10000,50000),
(50000,100000),(100000,300000)]:
        if lo>max_n: continue
        hi2=min(hi,max_n+1); a,b,c=[med(families[k],lo,hi2) for k in
('2p','2p^2','4p')]
        if a is not None and b is not None and c is not None: print(f'{lo:6d}-
{hi2-1:<6d}   {a:.5f}   {b:.5f}   {c:.5f}')
    print('\nMinima des trois familles')
    for name,v in families.items():
        n,r=min(v,key=lambda z:z[1]); p=n//2 if name=='2p' else n//4 if
name=='4p' else math.isqrt(n//2)
        print(f'{name:4s}: n={n}, p={p}, R={r:.5f}')
    if max_n>=68:
        print('\nCas n=68:')
        print(f'G(68)={G[68]} ; 68=7+61=31+37 ; R(68)={R[68]:.7f}')
    print('\nVrais minima parmi tous les pairs')
    for lo,hi in [(100,1000),(1000,10000),(10000,100000),(100000,300000)]:
        if lo>max_n: continue
        hi2=min(hi,max_n+1); cand=[(n,float(R[n])) for n in range(lo,hi2,2) if
np.isfinite(R[n])]
        if cand:
            n,r=min(cand,key=lambda z:z[1]); print(f'{lo:6d}-{hi2-1:<6d}: n={n},
R={r:.5f}')
    print('\nReference heuristique: e^(2 gamma)/16 ~= 0.1982637')

def export_csv(filename,max_n):
    isp,G,B,C,R=compute(max_n)
    with open(filename,'w',newline='',encoding='utf-8') as f:
        w=csv.writer(f); w.writerow(['n','G(n)','B(n)','C(n)','R(n)'])
        for n in range(4,max_n+1,2):
w.writerow([n,int(G[n]),f'{B[n]:.15g}',f'{C[n]:.15g}',f'{R[n]:.15g}'])
```

```
    print('CSV ecrit:',filename)

def main():
    ap=argparse.ArgumentParser(); ap.add_argument('--max-
n',type=int,default=300000); ap.add_argument('--csv'); a=ap.parse_args()
    report(a.max_n)
    if a.csv: export_csv(a.csv,a.max_n)
if __name__=='__main__': main()
```