Boldly Going – Abundant but Rare

Welcome to a new article in my *"Boldly Going"* series, this time dealing with a most fascinating topic in *Number Theory*, namely finding a particularly rare kind of so-called *abundant* numbers. The qualifier *"abundant"* refers to a classification of the positive integers which considers them as being either *deficient*, *perfect* or *abundant* depending on the sum of their proper divisors, and we're boldly going to find a most rare variety of the latter, so rare in fact that up to 10^{210} there are only *seven* of them, A_1 to A_7, the last being a 205-digit number no less. Nevertheless we'll use our trusty vintage *HP-71B* handheld to quickly find and display all seven, *exactly*. Oh, and the abundant numbers may hold the key to settle the *Riemann Hypothesis*.

Introduction

About two millennia ago, a respected mathematician called Nicomachus wrote a treatise titled *"Introduction to Arithmetic"*, in which he describes mathematical properties of numbers, to which he also adscribed mystical (and even *moral* !) properties. In particular he writes about the significance of *prime* numbers and *perfect* numbers, where the qualifier *"perfect"* refers to his classification of the integers, which he considers as being either *deficient*, *perfect* or *abundant* depending on the sum of their proper[1] divisors (*SPD* henceforth). For instance:

12's proper divisors are 1, 2, 3, 4 and 6, and its *SPD* is 1+2+3+4+6 = **16** > 12, so **12** is *abundant*.

15's proper divisors are 1, 3 and 5, and its *SPD* is 1+3+5 = **9** < 15, so **15** is *deficient*.

28's proper divisors are 1, 2, 4, 7 and 14, and its *SPD* is 1+2+4+7+14 = **28** = 28, so **28** is *perfect*.

The *perfect* numbers, the ones whose *SPD* exactly equals the number itself, have a long, interesting story going back to Euclid, who gave a formula which could be used to find *even* perfect numbers. Some 2,000 years later, Euler proved that *all even* perfect numbers are given by Euclid's formula. This solved half of the problem of finding all perfect numbers but there was still the question of whether there existed any *odd perfect* numbers. Euler gave some necessary requirements for an odd perfect number to comply with, but ultimately could neither find any nor prove that none existed.

Many first-line mathematicians gave serious attention to the problem of finding an odd perfect number (or proving that none existed) but to no avail, they only succeeded in finding many additional necessary requirements which, regrettably, don't exclude the possibility that some numbers might *"miraculously"* comply with them all. One of the most striking requirements is that any odd perfect number must be greater than 10^{1500}, i.e.: the hypothetical *smallest* odd perfect number must have > 1,500 digits[2]. No wonder Euler couldn't find one.

Things would have been much easier if we could prove that there are no *odd* abundant numbers, i.e.: that the *SPD* of an odd number could never be greater than the number itself, which at first sight seems plausible. Consider for example the numbers **20** *(even)* and **21** *(odd)*. Their respective *SPD* and classification are:

20's proper divisors: 1, 2, 4, 5 and 10, and its *SPD* = 1+2+4+5+10 = **22** > 20, so **22** is *even* and *abundant*.
21's proper divisors: 1, 3 and 7, and its *SPD* is 1+3+7 = **11** < 21, so **21** is *odd* and *deficient*.

All in all, about *24.76%* of all numbers are abundant and a few tries will convince you that it's relatively easy to find *even* abundant numbers and matter of fact we find many of them when checking up to *100, 200, 300, ..., 900*, but we find *no* odd ones, all odd numbers in that range are *deficient*. Intuitively, this seems quite logical: *even* numbers have a large divisor which by itself contributes about 50% of the sum, while *odd* numbers don't.

---

[1] The *proper* divisors of a positive number include all its positive divisors *except* the number itself. For instance, the divisors of *6* are *1, 2, 3* and *6*, but *6* is not a proper divisor as it equals the number itself. Thus, the *proper* divisors are *1, 2* and *3*.

[2] Although this seems at first to be seriously hinting at no such numbers existing, actually integers of 1,500 digits and more tend to raise their heads in *Number Theory* quite frequently. Consider, for instance, that the largest known primes, the *Mersenne* primes (primes of the form $2^p - 1$ with *p* prime, related to *even* perfect numbers: each *Mersenne* prime originates one) have *millions* of digits (24,862,048 digits for M_{51}), thus a mere 1,500 digits is next to nothing in comparison.

For instance, as seen above, ***20*** has the divisor ***10***, which is already one half of *20*, so the other divisors need only contribute the other half (the number would be *perfect*) or more (the number would be *abundant*), which actually is not that difficult and they succeed: *SPD(**20**) = **10** + (1+2+4+5) = **10** + 12 = **22** > 20, abundant*.

But *odd* numbers lack that advantage: their largest divisor is only *1/3* (at best) of the original number, and the other, smaller divisors need to contribute more than the remaining *2/3*, and they usually fail to, as seen above for ***21***, where the largest divisor is just ***7*** and *SPD(**21**) = **7** + (1+3) = **7** + 4 = **11** < 21, deficient*. Worse, if the odd number is *not* divisible by *3*, then the largest divisor's contribution to the sum will be even smaller, say *1/5* of the number, and the other smaller divisors must then add up to the remaining *4/5* or more. *Very* difficult indeed.

This kind of reasoning makes it seem plausible that perhaps for odd numbers their proper divisors never manage to add up to a sum greater than the number itself and thus there would be no *abundant* odd numbers. Even adding up to *exactly* the number itself might be impossible too if we could prove that the proper divisors always add up to some limit *less* than the number, in which case *odd perfect* numbers wouldn't exist either. Case closed.

But lo and behold, if we don't get bored and stop in haste at *900* but continue checking up to *1,000* then we suddenly stumble upon ***945*** and we have:

945's proper divisors are 1, 3, 5, 7, 9, 15, 21, 27, 35, 45, 63, 105, 135, 189 and 315, which add up to **975** > 945

so ***945*** is indeed *odd* and *abundant* (the first of its kind). Now this is bad news as far as proving the inexistence of odd perfect numbers is concerned because if an odd number can be *deficient* (most are) and thus its *SPD* is <u>*smaller*</u> than the number, and we now see that an odd number can also be *abundant*, so its *SPD* is <u>*greater*</u> than the number, then there's no obvious reason why the *SPD* can't be exactly <u>*equal*</u> to the number, which thus would be *perfect*, an *odd perfect number.* Their existence doesn't seem that utterly impossible now.

Now that we've seen the rarity of *odd* abundant numbers as compared with *even* ones, this article will deal with a particularly rare and interesting variety of abundant numbers, so rare in fact that their rarity might be comparable or superior to that of *even perfect numbers*[1]. Abundant but rare !

Boldly going ...

We've seen in the ***Introduction*** above that *odd* abundant numbers are much scarcer than *even* ones because the largest proper divisor *(LPD)* of the latter is already about *1/2* of the sum and the remaining divisors must account for just the other *1/2*, while the *LPD* of the former can't be more than *1/3* of the sum and the other divisors must add up to the remaining *2/3*, which is much more difficult, about *120 times* more difficult as we'll see below.

But what happens if the odd number isn't divisible by *3* either ? Then the *LPD* will contribute at most *1/5* of the sum and the rest must complete the remaining *4/5*, which proves tremendously difficult. And if the number isn't divisible by any of *2, 3* and *5* ? Then the *LPD* contributes only *1/7* or less and the rest will have to complete the remaining *6/7*. *Excruciatingly* difficult. And so on, raising the bar all the time by orders of magnitude ...

Now the question is: is it even *possible* ?, i..e.: can an *abundant* number not divisible by any of the first primes *(2, 3, 5, 7, 11, 13, ...)* even *exist* ? The answer is ***yes***, it's possible (because there's an infinity of *prime* numbers and the sum of their reciprocals *diverges*) and such abundant numbers A_n *do* actually exist but get increasingly larger at a super-exponential rate and by the time we get to A_7, which is the *smallest* abundant number not divisible by any of the first 7 primes (*2, 3, 5, 7, 11, 13, 17*), we're talking about a *205-digit* number !

How to proceed to try and find them ? Can we do it using a small, vintage *HP* model, namely the 1984 ***HP-71B***[2] handheld computer/calculator ? **You bet !**. And we'll proceed in two stages in order to get additional data and to better highlight the difficulties. First of all, for the very smallest one, A_1, which is not divisible by the first prime $p_1 = 2$ (so it's *odd*), we can use a simple *brute-force* search, for which we'll need a suitably *fast* implementation of the *SPD* function, and this one I've concocted in just 5 lines will do nicely:

---

[1] As of 2020, only 51 *even perfect* numbers are known, the largest is $2^{82589932}\times(2^{82589933}-1)$ with *49,724,095* digits.

[2] Unless otherwise specified, times given are for a *virtual* ***HP-71B***, the ***go71b*** emulator using images of the original *HP-71B*'s *System ROMs* and running on a *Samsung* tablet under *Android*, which is about *128x* faster than a physical *HP-71B.*

Program listing for the *SPD* *(Sum of Proper Divisors)* Function

Code:

```
100  DEF FNS(M) @ D=PRIM(M) @ IF NOT D THEN FNS=1 @ END
110  S=1 @ N=M/D @ E=D @ C=1
120  REPEAT @ D=PRIM(N) @ D=D+N*NOT D @ N=N/D @ C=C+1
130  IF D#E THEN S=S*(E^C-1)/(E-1) @ E=D @ C=1
140  UNTIL N=1 @ S=S*(E^(C+1)-1)/(E-1) @ FNS=S-M @ END DEF
```

Specs:

- *182 bytes*
- *uses parameter M.*
- *uses variables C, D, E, N and S.*
- *requires the JPC ROM (*PRIM, REPEAT*)*
- *line numbers are arbitrary, no branching*

The *SPD* function is implemented as a multi-line user-defined function **FNS** by the code above[1], so it's callable either from the command line or from a user's program. We'll use it both ways to check the results stated above, to gather additional information and results on abundant numbers, and to try and find A_1, as described above.

First of all, let's check all results already given and more. From the command line prompt (>), execute this:

```
>FNS(12);FNS(15);FNS(28);FNS(20);FNS(21);FNS(945);FNS(1234567891)  [END LINE]

     16  9   28  22  11  975   1
```

which confirms all the *SPD*s previously presented, plus the fact that the *SPD* of a *prime* number is always ***1***, thus the *"**1**"* result at the end corresponding to *1234567891*, which is prime.

Now let's use **FNS** to find an approximation to the *density* of abundant numbers in general (what percentage of numbers are abundant, which we said was ~ *24.76%*, i.e.: a density of *0.2476*), as well as the particular densities for *even* and *odd* abundant numbers separately. For the general case, add this code before the function definition, which when run asks for *N*, the upper limit to try, and outputs both the *tally* and the resulting *density*:

```
 1  DESTROY ALL @ STD @ INPUT K @ T=0
10  FOR M=2 TO K @ IF FNS(M)>M THEN T=T+1
20  NEXT M @ DISP T;T/K @ END

[RUN]  →       ?
 5000   [END LINE]  →   1239    0.2478
```

so among the first 5,000 numbers exactly *1,239* are abundant, a density of *0.2478*. For the tally and density of just the *even* abundant numbers, change **FOR M=2 TO K** in line *10* above to **FOR M=2 TO K STEP 2** , and for *odd* abundant numbers change line *10* again, this time to **FOR M=3 TO K STEP 2**.

Trying all the above changes and various values of *N* from *5,000* to *100,000*, we obtain the following table:

N	#Abundant	Density	#Even ab.	Density	#Odd ab.	Density
5,000	1,239	0.2478	1,232	0.2464	7	0.0014
10,000	2,488	0.2488	2,465	0.2465	23	0.0023
20,000	4,953	0.24765	4,910	0.2455	43	0.00215
50,000	12,394	0.24788	12,280	0.2456	114	0.00228
100,000	24,795	0.24795	24,585	0.24585	210	0.0021

which shows that indeed the density of abundant numbers in general is about *0.24795* but *even* abundant numbers have a density of *0.24585* while *odd* abundant numbers have a much lower density, *0.0021*, some *120* times lower, which accounts for why the first *even* abundant number is ***12*** while the first *odd* one is ***945***.

[1] **FNS** works by factorizing its argument and applying the formula $SPD(M) = \left\{\prod_{i=1}^{n} (p_i^{(e_i+1)} - 1)/(p_i - 1)\right\} - M$, where n is the number of distinct prime factors of M, p_i is the i-th prime factor and e_i is its maximum exponent in the factorization.

We can have a lot of fun with **`FNS`**, which proves invaluable to explore all kinds of questions having to do with *deficient*, *perfect* and *abundant* numbers, for instance:

1) To find all *even perfect numbers* up to *2,000*, add this code before the function definition:

```
 1  DESTROY ALL @ STD
10  FOR M=2 TO 2000 STEP 2 @ IF FNS(M)=M THEN DISP M;
20  NEXT M @ DISP @ END
```

RUN → *6 28 496* { *SPD(496)* **=** *496* }

2) To find all *even deficient numbers* up to *2,000* whose *deficience* is ***1***, change **`FNS(M)=M`** in line *10* above to **`FNS(M)=M-1`**

RUN → *2 4 8 16 32 64 128 256 512 1024* { *SPD(64) = 63 = 64***-1** }

3) To find all *even abundant numbers* up to *2,000* whose *abundance* is ***2***, change **`FNS(M)=M`** in line *10* above to **`FNS(M)=M+2`**

RUN → *20 104 464 650 1952* { *SPD(1952) = 1954 = 1952***+2** }

4) To find all *even abundant numbers* up to *2,000* whose *SPD* is twice the number itself, change **`FNS(M)=M`** in line *10* above to **`FNS(M)=M*2`**

RUN → *120 672* { *SPD(672) = 1344 =* **2*** *672* }

And at long last, to find A_1, the very first abundant number which is not divisible by the very first prime $p_1 = 2$ (so it's *odd*), now that we have **`FNS`** we can use the simple *brute-force* search by just changing line *10* above to:

```
10  FOR M=3 TO INF STEP 2 @ IF FNS(M)>M THEN DISP M @ END
```

RUN → ***945*** { *found in just 1.1"* }

so we have our first result, $A_1 = 945$

Now that we've got A_1, can we use this approach to find at least a few of the rest: A_2, A_3, etc.? Well, *no*, regrettably we can't. ***945*** is a *3-digit* number that we found in ~*1* second[1], and if A_2 were a *6-,7-digit* number, say, we'd be able to find it in anything form *20'* to ~*4* hours. But as it happens, A_2 is already a *10-digit* number, and finding it this way would take ~2.10^7 seconds = ~*5,600* hours = ~*231 days* (you get the drift), which is unfeasible, never mind A_3 *(26 digits)*, A_4 *(53 digits)*, A_5 *(88 digits)*, A_6 *(140 digits)* and last but not least, A_7 *(205 digits)*.

It's plainly clear that such *brute-force search won't do* and a radically different approach is needed. So far we've attempted to simply check each suitable candidate (i.e.: odd for A_1) number in turn using **`FNS`** to find its *SPD*. But **`FNS`** works by *factorizing* the number, which might take a while for numbers up to 10 digits (and 15 digits would be the limit for the present implementation), thus A_3 *(26 digits)* and the rest are simply out of the question.

A much better approach is to *avoid* factorization altogether and instead go the *reverse* way: for each ***n*** from ***2*** to ***7***, iteratively select a certain, carefully chosen set of prime factors and their multiplicities so that the resulting number A_n having those factors is indeed *abundant* <u>and</u> *not divisible* by the first ***n*** primes <u>and</u> it's the *smallest* possible such. That's what my program below achieves:

[1] As stated in the footnote at page *2*, this is the timing for a *virtual* ***HP-71B***, a physical one would take *2-3 min*. Finding A_2 this way using the *emulated 71B* would take ~*231 days*, while the *physical 71B* would need some *80 years* (!).

Program Listing for the HP-71B

Code:

```
10  DESTROY ALL @ STD @ OPTION BASE 0 @ INTEGER P(100),M(10)
20  P(1)=2 @ FOR I=2 TO 100 @ P(I)=FPRIM(P(I-1)+1) @ NEXT I
30  FOR K=2 TO 7 @ T=K+1 @ E=K DIV 3 @ H=E+1
40  U=K @ W=1 @ REPEAT @ U=U+1 @ W=W*(1+1/(P(U)-1)) @ UNTIL W>2
50  V=K @ W=1 @ REPEAT @ V=V+1 @ W=W*(1+1/P(V)) @ UNTIL W>2
60  FOR I=0 TO E @ M(I+1)=P(T+I) @ NEXT I @ S=1
70  FOR I=T TO T+E @ J=P(I) @ S=S*(1+1/J+1/J^2) @ NEXT I
80  FOR I=I TO U @ S=S*(1+1/P(I)) @ NEXT I
90  FOR I=U+1 TO V @ J=P(I) @ H=H+1 @ M(H)=J @ S=S*(1+1/J)
100 IF S>2 THEN DISP "A";STR$(K);": Abundancy:";S;","; @ CALL DMUL(M,H,P,T,U) @ I=V
110 NEXT I @ PAUSE @ NEXT K @ END

120 SUB DMUL(M(),H,P(),T,U) @ DIM A(1),S$[256] @ D=9 @ K=10^D @ R=0 @ A(0)=1
130 FOR I=1 TO H @ MAT A=(M(I))*A @ GOSUB 180 @ NEXT I @ W=50
140 FOR I=T TO U @ MAT A=(P(I))*A @ GOSUB 180 @ NEXT I @ S$=STR$(A(R))
150 FOR R=R-1 TO 0 STEP -1 @ A$=STR$(A(R)) @ S$=S$&SPACE$("0",D-LEN(A$))&A$
160 NEXT R @ D=LEN(S$) @ DISP D;"digits"
170 FOR I=0 TO D DIV W @ DISP S$[1+W*I,W+W*I] @ NEXT I @ END
180 FOR J=0 TO R @ A(J+1)=A(J+1)+A(J) DIV K @ A(J)=MOD(A(J),K) @ NEXT J
190 R=R+SGN(A(R+1)) @ DIM A(R+1) @ RETURN
```

Specs:

- *759 bytes*
- *requires the JPC ROM.*
- *requires the Math ROM.*

Notes:

The **FPRIM** *(Find next prime) keyword of the JPC ROM is used for simplicity and speed, to generate an array of prime numbers. It can be replaced by simple BASIC code if no JPC ROM is available.*

Likewise, **SPACE$** *can be replaced by* **RPT$** *and* **REPEAT..UNTIL** *can be replaced by* **IF..GOTO***.*

The **MAT=()*** *keyword is used to speed multi-digit multiplications. It can be replaced by a loop if no Math ROM is available.*

Program details

For this new approach, I won't be using *SPD*, the sum of the *proper* divisors, but the sum of *all* the divisors (i.e.: including the number itself), which goes in the literature by the name ***σ(n)***[1]. Matter of fact, actually I'll be using the so-called *index*[2] *of n*, which is $\sigma_{-1}(n) = \sigma(n)/n$. This means that if this *index* is **< 2** then the number is *deficient*, if exactly equal to **2** then the number is *perfect* and if **> 2** then the number is *abundant*, in which case I'll call it the *abundancy* of the number. That said, let's succinctly comment the source code:

Line 10: main entry point: initialization and dimensioning of the arrays for primes and certain prime factors.

Line 20: fills up the array with the first 100 primes *(2, 3, 5, 7, ..., 523, 541)*, enough to search for up to A_7.

Line 30: start of the loop to search for A_2 to A_7.

Line 40: computes an array index used to define an upper bound for A_n and loops until the *abundancy* is ***>2***.

Line 50: computes an array index used to define a lower bound for A_n and loops until the *abundancy* is ***>2***.

Lines 60-90: iteratively computes the *abundancy* of a candidate set of prime factors.

Line 100: if the *abundancy* is ***>2*** then we've found A_n , so display its *abundancy* and call the separate **DMUL** subprogram to assemble from its factors and display A_n itself in full.

Line 110: if the abundancy isn't ***>2*** then iterate for the next candidate or, if A_n was output, **PAUSE** for the user to take note and upon executing **CONT** loop for the next A_n . Once A_7 is output, the program ends.

Lines 120-190:

The subprogram **DMUL** accepts as parameters the arrays holding certain prime factors of the A_n just found and a list of primes, as well as the indexes to the relevant factors, and assembles the multiprecision number A_n in another array by simply multiplying them one at a time and releasing the carries after each multiplication *(*subroutine at line *180)*.

Once finished, an auxiliar string is formed by concatenating the array elements (taking due care of intermediate 0's), which is finally displayed in lines *W* characters long (*50* in the code but you can previously specify the **WIDTH/PWIDTH** you want and/or change **W=50** at line *130* to some other value of your choice.

---

[1] You can turn *SPD(n)* into *σ(n)* by changing **FNS=1** in line *100* to **FNS=1+M** and **FNS=S-M** in line *140* to **FNS=S**.

[2] You can turn *SPD(n)* into $\sigma_{-1}(n)$ by changing **FNS=1** in line *100* to **FNS=1+1/M** and **FNS=S-M** in line *140* to **FNS=S/M**.

Delivering the goods

If you need to, first specify the **`WIDTH/PWIDTH`** you want and/or change **`W=50`** at line *130* to some other value of your choice, perhaps **`W=20`** if you're using the real/emulated *LCD* display of a physical/virtual ***HP-71B***, in which case you'll probably want to change the **`DELAY`** setting as well, to output lines at a rate comfortable for either seeing them or writing them down; consult the *HP-71 Owner's Manual* and/or *Reference Manual* for details.

That addressed, now run the program by pressing **[RUN]** *{timings below for both virtual and physical 71B }*

→

A2: *Abundancy: 2.00305796572* , ***10 digits*** *{virtual: 0.03"; physical: 3.58" }*

5391411025 *{abundant and not divisible by 2 or 3}*

[CONT] →

A3: *Abundancy: 2.00961403492* , ***26 digits*** *{virtual: 0.05"; physical: 7.11" }*

20169691981106018776756331 *{abundant and not divisible by 2, 3 or 5}*

[CONT] →

A4: *Abundancy: 2.00420624527* , ***53 digits*** *{virtual: 0.11"; physical: 14.57" }*

49061132957714428902152118459264865645885092682687
973 *{abundant and not divisible by 2, 3, 5 or 7}*

[CONT] →

A5: *Abundancy: 2.00121428261* , ***88 digits*** *{virtual: 0.21"; physical: 26.81" }*

79704663275245715382257095454345062559700269697100
12787303278390616918473506860039424701 *{abundant and not divisible by 2, 3, 5, 7 or 11}*

[CONT] →

A6: *Abundancy: 2.00014597776* , ***140 digits*** *{virtual: 0.39"; physical: 49.94" }*

64702979560966356488598858364510932416917957393726
35681325649411274876738261599028442624375057781562
9801062581738756314644292387919578086291 *{abundant and not divisible by 2, 3, 5, 7, 11 or 13}*

[CONT] →

A7: *Abundancy: 2.00005198195* , ***205 digits*** *{virtual: 0.68"; physical: 87.78" }*

30992145361707757163162116848915935971910152112597
26905017058067280971249287656549276771862918446292
41948770133600229536130441901801403770323543210611
52694925865576180906246071055135468567378389159244
25547 *{abundant and not divisible by 2, 3, 5, 7, 11, 13 or 17}*

[CONT]

And the program ends. As you can see, the execution times are *really fast*: the *205-digit* A_7 is found, assembled and displayed in *0.68"* on the *emulated 71B* at *128x*, and in just ***87.78"*** on a *physical*, *real* vintage ***HP-71B***. How's that for speed ?

Notes

1. As far as I know, as of *August 2020* the *explicit* values of $\boldsymbol{A_6}$ and $\boldsymbol{A_7}$ have never been published anywhere and they don't appear in the Internet either. *OEIS* only lists explicit values up to $\boldsymbol{A_5}$ but $\boldsymbol{A_6}$ and $\boldsymbol{A_7}$ are nowhere to be found in explicit form, not even under the *"list"* option, so this might be considered *Original Research* on my part.

2. Although my program can be trivially modified to also produce $\boldsymbol{A_8}$, $\boldsymbol{A_9}$ and $\boldsymbol{A_{10}}$ and the values produced are indeed guaranteed to be *abundant* <u>and</u> *not divisible* by the first eight *(2, 3, ..., 19)*, nine *(2, 3, ..., 23)* and ten *(2, 3, ..., 29)* primes, respectively, regrettably I can't guarantee that they're the *smallest* ones. They might be but for now I can't prove that they are, so my program is only *guaranteed* to produce correct results for $\boldsymbol{A_2}$ to $\boldsymbol{A_7}$ but no farther.

3. Last but not least, in case you're thinking something about the lines of *"Abundant numbers ? Who cares ? What are they good for ? ..."*, let me tell you that they might actually *hold the key* to settle for good the ***Riemann Hypothesis***, the most important unsolved problem, the *Holy Grail* in mathematics. See for instance the fourth reference I give below, if in doubt.

References

Dominic Klyve *et al.* (2019)	*Estimating the Density of the Abundant Numbers*
William Dunham (1999)	*Euler: The Master of Us All*
G. Hardy & E. Wright (1979)	*An Introduction to the Theory of Numbers*
S. Nazardonyavi *et al.* (2014)	*Extremely Abundant Numbers and the Riemann Hypothesis*

Copyrights