# Classic Riddler

19 June 2020

## Riddle:

King Auric adored his most prized possession: a set of perfect spheres of solid gold. There was one of each size, with diameters of 1 centimeter, 2 centimeters, 3 centimeters, and so on. Their brilliant beauty brought joy to his heart. After many years, he felt the time had finally come to pass the golden spheres down to the next generation—his three children.

He decided it was best to give each child precisely one-third of the total gold by weight, but he had a difficult time determining just how to do that. After some trial and error, he managed to divide his spheres into three groups of equal weight. He was further amused when he realized that his collection contained the *minimum* number of spheres needed for this division. How many golden spheres did King Auric have?

*Extra credit*: How many spheres would the king have needed to be able to divide his collection among other numbers of children: two, four, five, six or even more?

## Solution:

Because the spheres are 3-dimensional, then the weight of each is proportional to the cube of the radius. The proportionality is irrelevant, so for the problem, the weight of the smallest sphere with radius 1 centimeter will simply become $1^3 = 1$, the second-smallest sphere will have weight $2^3 = 8$, etc. The total amount of gold is then the sum of the weights, and there is a well-known formula for the sum of $N$ cubes ($N$ will become the solution):

$$\sum_{i=1}^{N} i^3 = \left(\frac{N(N+1)}{2}\right)^2$$

To solve the riddle, this quantity must be divisible by 3. I will call this value the target sum $S$:

$$S = \frac{1}{3}\left(\frac{N(N+1)}{2}\right)^2$$

The original sum is divisible by 3 (and therefore $S$ is a whole number) when $N$ is of the form $3k$ or $3k-1$. So in a very small sense, this narrows down the possible solutions. Ultimately, the solution requires that there are 3 different ways to obtain the target sum:

$$S = \sum_i n_i^3 = \sum_j n_j^3 = \sum_k n_k^3$$
$$n_i \neq n_j \neq n_k$$

The difficulty in this problem is that there is a huge space to search for solutions. Given some $N$, there are $2^N - 1$ ways to sum up individual cubes less than or equal to $N$. So even for relatively small $N$, say around 10–15, there are several thousand ways to sum up the cubes, and each of these sums must be checked against $S$.

I first tried to look for solutions in an excel spreadsheet, but I couldn't find any solution for $n \leq 15$, at which point the table was getting huge and hard to keep track of. So I decided to try to write a code to perform searches. This is in `sums_of_cubes.C`. The first thing I could do is create an array of $n$ cubes. What I needed to do with that array is access every possible subset of the array. After researching this, I learned this is called a powerset. Specifically, a powerset is the set of all subsets of some set; I didn't need the powerset per se, but I did need a way to generate all such subsets. I learned that this can be done by letting a counter run from 0 (or really 1, since the empty subset is trivially useless) up to $2^n - 1$, and each value will have a binary representation that corresponds to a specific subset. Then I just needed to scan through the bits of the counter and add the corresponding cube if the bit is 1.

Once I implemented this method, I could produce a sum, which then had to be compared to the target sum. If they were the same, I could output the counter value, from which I could separately extract the corresponding cubes being summed. My code does this, and writes the $n$ being tested, the $n$th target sum, and the counter value. I note that the output of this first program does not necessarily produce a solution. The solution requires that there are 3 ways to obtain the target sum, which is not checked in the this program.

I next wrote a script (`sums_of_cubes_final_test.C`) to read in an array of possible solutions (from the output of the first program) and see if these correspond to actual solutions. This is done by comparing the bit values of the counter values two at a time from the first program and checking if there are any 1s in common. If not, then the two values do correspond to independent sums, and because there is only one sum left, it is necessarily an independent sum. Then this becomes the solution. Just for illustration, when testing up to $N = 30$, the first program found possible solutions for $N$=8, 14, 17, 18, 20, and 21, even though these turned out not to be solutions. The first real solution was for $N = 23$. By checking the results of the second program by hand, I could verify that

$$1^3 + 4^3 + 7^3 + 8^3 + 12^3 + 16^3 + 20^3 + 22^3$$
$$=$$
$$2^3 + 5^3 + 9^3 + 11^3 + 14^3 + 15^3 + 17^3 + 23^3$$
$$=$$
$$3^3 + 6^3 + 10^3 + 13^3 + 18^3 + 19^3 + 21^3$$

So the solution is **23 spheres**.

I continued the problem for two, four, and five daughters. While I knew how to continue further, I didn't know how large the solution would be and how long the program would take to run. (I was already around 30 minutes to over-calculate the solution for five daughters). Here are those solutions:

For two daughters, there are **12 spheres**.

$$1^3 + 2^3 + 4^3 + 8^3 + 9^3 + 12^3$$
$$=$$
$$3^3 + 5^3 + 6^3 + 7^3 + 10^3 + 11^3$$

For four daughters, there are **24 spheres**.

$$1^3 + 2^3 + 3^3 + 4^3 + 14^3 + 18^3 + 24^3$$
$$=$$
$$5^3 + 6^3 + 8^3 + 11^3 + 13^3 + 15^3 + 16^3 + 22^3$$
$$=$$
$$7^3 + 9^3 + 21^3 + 23^3$$
$$=$$
$$10^3 + 12^3 + 17^3 + 19^3 + 20^3$$

For five daughters, there are **24 spheres**.

$$1^3 + 18^3 + 23^3$$
$$=$$
$$2^3 + 4^3 + 9^3 + 15^3 + 24^3$$
$$=$$
$$3^3 + 5^3 + 12^3 + 19^3 + 21^3$$
$$=$$
$$6^3 + 7^3 + 10^3 + 11^3 + 13^3 + 17^3 + 20^3$$
$$=$$
$$8^3 + 14^3 + 16^3 + 22^3$$