Restricted to integers, what is the largest n-bit binary number we can square:

$$\text{square}(x) \quad = \quad x * x \quad = \quad x^2$$

without worry of arithmetic overflow? This is to say:

$$\max \{x \in \mathbb{Z} \mid x^2 < 2^n\}$$

The obvious solution starts as:

$$x \quad < \quad 2^{\frac{n}{2}}$$

In the case that n is *even*, our solution is immediate as $2^{n/2}$ is a computable integer, and thus:

$$x \quad \leq \quad 2^{\frac{n}{2}} - 1$$

Unfortunately things are not so simple if n is *odd*, in this case we have:

$$x \quad < \quad 2^{\frac{n}{2}} \quad = \quad 2^{\lfloor \frac{n}{2} \rfloor} 2^{\frac{1}{2}} \quad = \quad 2^{\frac{n-1}{2}} \sqrt{2}, \qquad n \geq 1$$

The most direct solution then would be to calculate and round down:

$$x \quad < \quad 2^{\frac{n-1}{2}} \sqrt{2}, \qquad n \geq 1$$

Notice as we're multiplying $\sqrt{2}$ by a power of two, when implementing we only need to achieve binary accuracy of $\frac{n-1}{2}$ bits then we can simply left-shift and coerce our *double floating* point into our max. Unfortunately, the bit precision of *double* is implementation specific with no hard rules in languages like C and C++. In practice it's unlikely to arise in an error, but such a solution still lacks certainly as well as portability.

To this end, it makes sense to take a midpoint guess-and-check approach. Why? Given our bit precision, and the fact that we halve the list of possibilities each time, we would be making at most n guesses and checks. We can improve this though by starting off with a good lower and upper bound:

$$2^{\frac{n-1}{2}} \quad \leq \quad m \quad \leq \quad 2^{\frac{n+1}{2}}$$

Leaving us with only $\frac{n-1}{2}$ possible guesses and checks. This solution scales well against current hardware architecture. If industry gives us 128-bit registers, we then have only 64 possible guesses and checks. We'll see how well this assumption stands the test of time...

We still need an efficient means of testing, where we know we don't have to worry about the exact problem we're trying to solve: arithmetic overflow. We start again with:

$$x^2 \quad < \quad 2^n$$

Dividing by 2^{n-1} on both sides leaves us with:

$$\left(\frac{x}{2^{(n-1)/2}}\right)^2 \quad < \quad 2$$

If we break up our test x into its division and remainder—dividing by $2^{\frac{n-1}{2}}$ we have:

$$x = 2^{\frac{n-1}{2}} q + r$$

applying this we get:

$$\left(q + \tfrac{r}{2^{(n-1)/2}}\right)^2 \quad < \quad 2$$

$$q^2 + \tfrac{2qr}{2^{(n-1)/2}} + \left(\tfrac{r}{2^{(n-1)/2}}\right)^2 \quad < \quad 2$$

Notice if we set $q = 2$ we'd have a contradiction, so $0 \leq q \leq 1$. But given we want x to be as large as possible it makes sense to set $q = 1$:

$$\begin{aligned} \frac{2r}{2^{(n-1)/2}} + \frac{r^2}{2^{n-1}} &< 1 \\ 2 \cdot 2^{\frac{n-1}{2}} r + r^2 &< 2^{n-1} \\ 2r^2 &< 2^{n-1} - 2 \cdot 2^{\frac{n-1}{2}} r + r^2 \\ 2r^2 &< \left(2^{\frac{n-1}{2}} - r\right)^2 \end{aligned}$$

This is ideal as we know from the division remainder theorem that:

$$0 \leq r < 2^{\frac{n-1}{2}}$$

and so

$$0 \leq 2r^2 < 2^n$$

So the left-side of our potential test will not overflow. Additionally:

$$0 < 2^{\frac{n-1}{2}} - r \leq 2^{\frac{n-1}{2}}$$

and so

$$0 < \left(2^{\frac{n-1}{2}} - r\right)^2 \leq 2^{n-1} < 2^n$$

So we need not worry about any overflow on both sides of our test. As such, we only need implement the midpoint method to determine our solution for arbitrary odd n by applying the following test:

$$2r^2 < \left(2^{\frac{n-1}{2}} - r\right)^2, \qquad \text{where} \quad r := x \bmod 2^{\frac{n-1}{2}}$$