perplexity

how biggest no is 2 53 - 1 and - 2 53 - 1, esplain

Let's break down what **$2^{53} - 1$** and **$-(2^{53} - 1)$** mean, especially in the context of JavaScript and binary numbers.

1. What is $2^{53} - 1$?

- $2^{53} - 1$ means: two raised to the power 53, minus 1.
- This computes to:
  $2^{53} - 1 = 9{,}007{,}199{,}254{,}740{,}991$

In binary, this is **fifty-three 1's in a row**.

Why 53?
JavaScript's Number type (based on IEEE 754 double precision) can accurately represent **all** integer values from
$-(2^{53} - 1)$ to **$2^{53} - 1$**
without losing precision.

2. What about $-(2^{53} - 1)$?

- $-(2^{53} - 1) = -9{,}007{,}199{,}254{,}740{,}991$

This is just the negative of the maximum "safe" integer value.

3. Why is $2^{53} - 1$ important in JavaScript?

- **53 bits** come from:
  - IEEE 754 double-precision floating point can store 53 significant binary digits ("bits") for integers.
  - Therefore, numbers larger than $2^{53} - 1$ **cannot** be accurately represented; after that point, not all integers are uniquely represented—the gaps between representable numbers increase.
- In JavaScript, these boundary values are called "safe" integers:
  - `Number.MAX_SAFE_INTEGER` = $2^{53} - 1$ = **9,007,199,254,740,991**
  - `Number.MIN_SAFE_INTEGER` = $-(2^{53} - 1)$ = **−9,007,199,254,740,991**

4. Summary Table

Expression	Value	Explanation
$2^{53} - 1$	9,007,199,254,740,991	Largest exact integer in JS Number type
$-(2^{53} - 1)$	−9,007,199,254,740,991	Smallest (most negative) safe integer in JS

5. Visual Analogy

- If you count in binary, with 53 digits, the highest unsigned number you can get (all 1s): $2^{53} – 1$.
- In decimal, that's 9,007,199,254,740,991.

6. Why subtract 1?

- Counting starts from 0.
- For n bits, the highest value is always $2^n – 1$ (not 2^n).

In summary:

- **$2^{53} – 1$** is the largest **"safe"** integer in JavaScript's Number type (can be represented exactly),
- **$-(2^{53} – 1)$** is its negative counterpart.
- Any integer outside this range (positive or negative) loses precision.

If you need to represent numbers larger than this, you must use **BigInt** in JavaScript!