Odd or Even (Problem ID: 1)

Problem Description

You are given a single integer **N**. Your task is to determine if the number is **odd** or **even**.

- If the number is **odd**, print `"Yes"`.
- If the number is **even**, print `"No"`.

Example Explanation

Input

```
1
```

Explanation

- N = 1
- 1 is an odd number.

Output

```
Yes
```

Input

```
2
```

Explanation

- N = 2
- 2 is an even number.

Output

```
No
```

Constraints & Key Observations

- `1 <= N <= 10^18`
- Time Limit: 1000 ms (1 second)
- The input number can be very large (), which fits within a standard 64-bit integer (`long long` in C++, `int` in Python 3).
- The property of being odd or even depends solely on the **last digit** or the **remainder when divided by 2**.
- We need an time complexity solution.

Intuition

An integer is **even** if it is perfectly divisible by 2 (i.e., `N % 2 == 0`). An integer is **odd** if it leaves a remainder of 1 when divided by 2 (i.e., `N % 2 != 0`).

Alternatively, in binary representation, the **least significant bit (LSB)** determines parity:

- If LSB is `0`, the number is **even**.
- If LSB is `1`, the number is **odd**.

Approaches

Approach 1: Modulo Operator (Standard)

Explanation We compute the remainder of when divided by 2 using the modulo operator `%`.

- If `N % 2 != 0`, print “Yes”.
- Else, print “No”.

Why It Works By definition, even numbers are multiples of 2. The modulo operator directly checks this divisibility.

Why It Fails It does **not fail**. This is the standard, correct approach.

Code

```python
def solve():
    try:
        line = input().strip()
        if not line: return
        n = int(line)

        if n % 2 != 0:
            print("Yes")
        else:
            print("No")

    except ValueError:
        return

if __name__ == "__main__":
    solve()
```

Time Complexity

- **O(1)** — Basic arithmetic operation.

Space Complexity

- **O(1)** — No extra space used.

Approach 2: Bitwise AND (Optimized)

Explanation We check the last bit of the number using the bitwise AND operator `&`. `N & 1` returns the least significant bit.

- If `N & 1 == 1`, the number is **odd** -> Print “Yes”.
- If `N & 1 == 0`, the number is **even** -> Print “No”.

Why It Works In binary, all powers of 2 () are even numbers. The only component that contributes “oddness” is . Therefore, checking the bit (LSB) is sufficient to determine parity.

Code

```python
def solve():
    try:
        line = input().strip()
        if not line: return
        n = int(line)

        if n & 1:
            print("Yes")
        else:
            print("No")

    except ValueError:
        return

if __name__ == "__main__":
    solve()
```

Time Complexity

- **O(1)** — Bitwise operations are extremely fast.

Space Complexity

- **O(1)**

Edge Cases & Common Pitfalls

- **Large Inputs:** The constraint requires using a 64-bit integer type in strictly typed languages like C++ (`long long`) or Java (`long`). Python handles large integers automatically.
- **Input Format:** Sometimes inputs may have leading/trailing whitespace. Always strip inputs.
- **Negative Numbers:** While constraints say , generally `N % 2` works for negative numbers in Python (returns 1) and C++ (returns -1 or 1). The check `N % 2 != 0` correctly identifies odd numbers regardless of sign.

When Not to Use This Approach

- There is practically no scenario where these approaches are inappropriate for this specific problem.

“‘