Three Sum

Problem Description

You are given an array of integers `arr` and a target value `target`. Your task is to determine if there exists a triplet (three distinct elements) in the array such that their sum equals the `target`.

The output should be "Yes" if such a triplet exists, otherwise "No".

Example Explanation

Input

```
1 2 3
6
```

Explanation

- Array: `[1, 2, 3]`
- Target: `6`
- Triplet: `1 + 2 + 3 = 6`

Output

```
Yes
```

Input

```
4 5 6 1
10
```

Explanation

- Array: `[4, 5, 6, 1]`
- Target: `10`
- Triplet: `4 + 5 + 1 = 10`

Output

```
Yes
```

Constraints & Key Observations

- `3 <= n <= 10^5`
- `1 <= arr[i] <= 10^9`
- Time Limit: 1000 ms (1 second)
- Finding three numbers suggests we need to check combinations.
- A naive approach will be too slow for (total operations).

- We need an approach closer to or .

Intuition

The problem asks for . This can be rewritten as finding a pair such that .

If we fix one number (`a`), the problem reduces to the classic **Two Sum** problem for the remaining array with a new target (`target - a`).

Sorting the array first allows us to use the **Two Pointer** technique efficiently.

Approaches

Approach 1: Brute Force (Naive)

Explanation Use three nested loops to check every possible triplet .

Why It Works It checks every single combination, so it is guaranteed to find the answer if it exists.

Why It Fails

- **Time Complexity:** .
- For , this performs operations, which might pass.
- For (as per constraints), this is impossibly slow and will get **Time Limit Exceeded (TLE)**.

Code

```python
# Pseudo-code for logic
for i in range(n):
    for j in range(i+1, n):
        for k in range(j+1, n):
            if arr[i] + arr[j] + arr[k] == target:
                return "Yes"
return "No"
```

Approach 2: Sorting + Two Pointers (Optimal)

Explanation

1. **Sort** the array first. ()
2. Iterate through the array with a fixed pointer `i` from `0` to `n-3`.

3. For each `i`, use two pointers (`left` and `right`) on the remaining part of the array:

- `left = i + 1`
- `right = n - 1`

4. Calculate `current_sum = arr[i] + arr[left] + arr[right]`.

- If `current_sum == target`: Return "Yes".
- If `current_sum < target`: We need a larger sum, so move `left` forward (`left++`).
- If `current_sum > target`: We need a smaller sum, so move `right` backward (`right--`).

Why It Works Sorting gives order to the numbers. This allows us to make intelligent decisions (moving left or right) instead of blindly checking every pair, reducing the inner complexity from to .

Code

```
def solve():
    # Read inputs
    try:
        line1 = input().split()
        if not line1: return # Handle empty input
        arr = list(map(int, line1))

        line2 = input().strip()
        if not line2: return
        target = int(line2)
    except EOFError:
        return

    n = len(arr)

    # 1. Sort the array
    arr.sort()

    # 2. Fix one element and use two pointers
    for i in range(n - 2):
        left = i + 1
        right = n - 1

        while left < right:
            current_sum = arr[i] + arr[left] + arr[right]

            if current_sum == target:
                print("Yes")
                return
            elif current_sum < target:
                left += 1
            else:
                right -= 1

    print("No")

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

Time Complexity

- Sorting:
- Two Pointers Loop: (Outer loop runs times, inner `while` runs times).
- Total: ****. This fits well within the 1-second time limit for typical constraints.

Space Complexity

- **O(1)** (ignoring space for sorting), as we only use pointers.

Edge Cases & Common Pitfalls

- **Duplicates:** The problem asks *if* a triplet exists. We don't need to count unique triplets, so duplicates generally don't break the boolean logic, but handling them correctly is important if we needed to print the numbers.
- **Array Size < 3:** If , a triplet cannot exist. The loops naturally handle this, but an explicit check is good practice.
- **Integer Overflow:** Constraints say up to . The sum of three such numbers can reach , which exceeds a standard 32-bit integer (limit). In Python, this is handled automatically. In C++/Java, use `long long`.

When Not to Use This Approach

- If the array is **immutable** (cannot be sorted). In that case, use a Hash Map approach (time, space).

“‘