Sum to n Implementations (TypeScript)

Source

```typescript
/**
 * A) Arithmetic-series (Gauss) formula
 * Time: O(1)  |  Space: O(1)
 * Most efficient for normal-sized integers; avoid if result may exceed MAX_SAFE_INTEGER.
 */
export function sum_to_n_a(n: number): number {
    if (!Number.isFinite(n)) throw new Error("n must be finite");
    if (n < 0) throw new Error("n must be a non-negative integer");
    n = Math.trunc(n);
    return (n * (n + 1)) / 2;
}

/**
 * B) Iterative loop
 * Time: O(n)  |  Space: O(1)
 * Simple and safe; slightly slower than (A) for large n, but no recursion depth concerns.
 */
export function sum_to_n_b(n: number): number {
    if (!Number.isFinite(n)) throw new Error("n must be finite");
    if (n < 0) throw new Error("n must be a non-negative integer");
    n = Math.trunc(n);

    let sum = 0;
    for (let i = 1; i <= n; i++) sum += i;
    return sum;
}

/**
 * C) Divide-and-conquer recursion (binary splitting)
 * Time: O(n)  |  Space: O(log n) stack depth
 * Unique vs. plain recursion: pairs ranges to keep recursion depth logarithmic.
 */
export function sum_to_n_c(n: number): number {
    if (!Number.isFinite(n)) throw new Error("n must be finite");
    if (n < 0) throw new Error("n must be a non-negative integer");
    n = Math.trunc(n);

    if (n === 0) return 0;

    const sumRange = (lo: number, hi: number): number => {
```

```
        if (lo === hi) return lo;
        const mid = Math.floor((lo + hi) / 2);
        return sumRange(lo, mid) + sumRange(mid + 1, hi);
    };

    return sumRange(1, n);
}
```