int mid = (start-1)+((end-start+1)/2); why this formula is not working for same intution ?

Saturday, November 23, 2024 4:21 PM

int mid = (start-1)+((end-start+1)/2);
why this formula is not working for same intution ?

The formula
int mid = (start - 1) + ((end - start + 1) / 2);
might seem similar to the standard formula but fails because the adjustments it introduces (start - 1 and end - start + 1) **distort the range boundaries**, resulting in incorrect or even **out-of-bounds values** during binary search. Let's break it down step by step to understand why this happens.

How It Differs From the Correct Formula

1. **Correct formula**:
   int mid = start + ((end - start) / 2);
   - Calculates the offset from start using the range size (end - start) and adds it to start to compute the midpoint.
2. **This formula**:
   int mid = (start - 1) + ((end - start + 1) / 2);
   - Subtracts 1 from start, **shifting the starting point**.
   - Adds 1 to (end - start) in the offset calculation, artificially **inflating the range size**.

The subtraction and addition disrupt the alignment of the calculated midpoint with the actual range [start, end].

Key Problems With This Formula

1. Off-By-One Error

By subtracting 1 from start:

- The formula assumes the range starts one position earlier than it actually does.
- This often results in calculating a mid that is **too low** for the intended range.

2. Inflated Range Size

Adding 1 to (end - start) creates an **inflated range size**, especially when the range is small:

- For a range [start, end] of size n, the formula computes the size as n + 1.
- This extra size shifts the calculated midpoint, causing inconsistencies or even out-of-bounds indices.

3. Potential Out-of-Bounds Errors

When start and end are close, such as in edge cases (start = 0, end = 1), the formula can produce negative indices or values beyond the valid range of the array. For example:

- For start = 0 and end = 1,
  (start - 1) = -1, and the midpoint calculation might result in -1 or 2, both invalid.

Why It Doesn't Work With the Same Intuition

The intuition behind the standard formula is to calculate the **offset** from start based on the actual size of the range. However, the faulty formula modifies the range size and starting point, invalidating the offset calculation:

1. In the standard formula:
   mid = start + ((end - start) / 2);
   - The offset (end - start) / 2 correctly divides the range into two halves.
   - Adding this offset to start ensures the midpoint stays within the range [start, end].
2. In the faulty formula:
   mid = (start - 1) + ((end - start + 1) / 2);
   - Subtracting 1 shifts the base point leftward, and inflating the range size (+1) skews the division.
   - This combination leads to an incorrect midpoint, often out of bounds.

Examples

Example 1: Small Range

For arr = {1, 5, 7, 9, 10, 14, 18}, start = 0, end = 1:

- **Correct formula**:
  mid = start + ((end - start) / 2) = 0 + (1 - 0) / 2 = 0.

  Midpoint is index 0, valid within the range.
- **Faulty formula**:
  mid = (start - 1) + ((end - start + 1) / 2) = -1 + ((1 - 0 + 1) / 2) = -1 + (2 / 2) = -1 + 1 = 0.

  Coincidentally correct here, but...

Example 2: Larger Range

For arr = {1, 5, 7, 9, 10, 14, 18}, start = 2, end = 6:

- **Correct formula**:
  mid = start + ((end - start) / 2) = 2 + ((6 - 2) / 2) = 2 + 2 = 4.

  Midpoint is index 4, valid.
- **Faulty formula**:
  mid = (start - 1) + ((end - start + 1) / 2) = (2 - 1) + ((6 - 2 + 1) / 2) = 1 + (5 / 2) = 1 + 2 = 3.

  Midpoint is index 3, skewed toward the left.

Why the Standard Formula Works

The standard formula avoids these problems by:

1. Calculating the offset as **precisely half the range** without inflating the size.
2. Adding the offset directly to start without shifting it unnecessarily.

The faulty formula introduces unnecessary complications, breaking the intuitive division of the range and leading to incorrect results.

From <https://chatgpt.com/c/6749dd0b-6894-8008-ac83-9d8d46fb7b8c>