**DAY-4 ASSIGNMENT**

**Question 1: In the Binary Search algorithm, it is suggested to calculate the mid as**

**beg + (end - beg) / 2 instead of (beg + end) / 2. Why is it so?**

beg + (end - beg) / 2 won't overflow if beg and end are large positive numbers. With signed operands, overflow is undefined.

 end - beg may overflow, but only if beg < 0 or end < 0.

Or with unsigned arithmetic, overflow is defined but gives you the wrong answer. However, for unsigned operands, beg + (end - beg) / 2 will never overflow as long as end >= beg.

mid = (beg + end)/2, problem with this approach is what if value of beg or end or both is INT\_MAX, it will cause integer overflow.

**beg + (end – beg)/2 also works even if you are using pointers and (beg+end)/2 doesn’t work because** pointer addition is not supported in C while pointer subtraction is supported, the reason being the result of subtraction is the difference (in array elements) between the operands. The subtraction expression yields a signed integral result of type ptrdiff\_t (defined in the standard include file STDDEF.H)**(in short subtraction gives memory distance)**, but addition of two pointers in not meaningful, that’s why not supported.

**Question 2: Write the algorithm/function for Ternary Search.**

**Algorithm:**

1. First, we compare the key with the element at mid1. If found equal, we return mid1.
2. If not, then we compare the key with the element at mid2. If found equal, we return mid2.
3. If not, then we check whether the key is less than the element at mid1. If yes, then recur to the first part.
4. If not, then we check whether the key is greater than the element at mid2. If yes, then recur to the third part.
5. If not, then we recur to the second (middle) part.

Function:

int ternarySearch(int beg, int end, int key, int ar[])

{

    while (end >= beg) {

        // Find the mid1 and mid2

        int mid1 = beg + (end - beg) / 3;

        int mid2 = end - (end - beg) / 3;

        // Check if key is present at any mid

        if (ar[mid1] == key) {

            return mid1;

        }

        if (ar[mid2] == key) {

            return mid2;

        }

        // Since key is not present at mid,

        // check in which region it is present

        // then repeat the Search operation

        // in that region

        if (key < ar[mid1]) {

            // The key lies in between l and mid1

            end = mid1 - 1;

        }

        else if (key > ar[mid2]) {

            // The key lies in between mid2 and r

            beg = mid2 + 1;

        }

        else {

            // The key lies in between mid1 and mid2

            beg = mid1 + 1;

            end = mid2 - 1;

        }

    }

    // Key not found

    return -1;

}