**DAY 4 ASSIGNMENT**

**QUE 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?

**ANS:**

There are three reasons:

1. First of all, start + (end - start) / 2 works even if you are using pointers, as long as end - start doesn't overflow1.

int \*start = ..., \*end = ...;

int \*mid = start + (end - start) / 2; // works as expected

int \*mid = (start + end) / 2; // type error, won't compile

1. Second of all, start + (end - start) / 2 won't overflow if start and end are large positive numbers. With signed operands, overflow is undefined:

int start = 0x7ffffffe, end = 0x7fffffff;

int mid = start + (end - start) / 2; // works as expected

int mid = (start + end) / 2; // overflow... undefined

(Note that end - start may overflow, but only if start < 0 or end < 0.)

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

unsigned start = 0xfffffffeu, end = 0xffffffffu;

unsigned mid = start + (end - start) / 2; // works as expected

unsigned mid = (start + end) / 2; // mid = 0x7ffffffe

* Finally, you often want to round towards the start element.

int start = -3, end = 0;

int mid = start + (end - start) / 2; // -2, closer to start

int mid = (start + end) / 2; // -1, surprise!

**QUE 2:**

Write the algorithm/function for Ternary Search.

**ANS:**

**Algorithm to perform Ternary Search:**

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 to perform Ternary Search:**

int ternarySearch(int l, int r, int key, int ar[])

{

    if (r >= l) {

        // Find the mid1 and mid2

        int mid1 = l + (r - l) / 3;

        int mid2 = r - (r - l) / 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

            return ternarySearch(l, mid1 - 1, key, ar);

        }

        else if (key > ar[mid2]) {

            // The key lies in between mid2 and r

            return ternarySearch(mid2 + 1, r, key, ar);

        }

        else {

            // The key lies in between mid1 and mid2

            return ternarySearch(mid1 + 1, mid2 - 1, key, ar);

        }

    }

    // Key not found

    return -1;

}