ASSIGNMENT DAY – 4

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.

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

```
int *start = ..., *end = ...;
int *mid = start + (end - start) / 2; // works as expected
int *mid = (start + end) / 2;         // type error, won't compile
```

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
```

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

Or with unsigned arithmetic, overflow is defined but gives us 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, we 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,
```

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

Ans.

```
#include <stdio.h>


// 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;
}
```

```
// Driver code
int main()
{
    int l, r, p, key;

    // Get the array
    // Sort the array if not sorted
    int ar[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

    // Starting index
    l = 0;

    // length of array
    r = 9;

    // Checking for 5

    // Key to be searched in the array
    key = 5;

    // Search the key using ternarySearch
    p = ternarySearch(l, r, key, ar);

    // Print the result
    printf("Index of %d is %d\n", key, p);

    // Checking for 50

    // Key to be searched in the array
    key = 50;

    // Search the key using ternarySearch
    p = ternarySearch(l, r, key, ar);

    // Print the result
    printf("Index of %d is %d", key, p);
}
```