**Assignment-4**

**Qus-1: In the binary search algorithm, it is suggested to calculate mid as beg+(end-beg)/2, instead of (beg+end)/2. Why is it so?**

**Ans: *(beg+beg)/2* 🡨** calculating mid this way is ineffective.

Why? let’s take an example.

Lets us take an integers from an integer beg to an integer end (both included).

|  |  |  |  |
| --- | --- | --- | --- |
| **beg** | **end** | **(beg+end)/2** | **beg +((end-beg)/2)** |
| 3 | 11 | 7 | 7 |
| 3 | 10 | 6 | 6 |
| -11 | -3 | -7 | -7 |
| -10 | -3 | **-6** | **-7** |

🡪 For beg=3 and end=11, the number of elements(#elements)=9.

So there is only 1 mid i.e,7.

***Both formula have computed the mid correctly.***

🡪 for beg=3 and end=10 ,#elements = 8.

So there are two mids, 6(beg mid) and 7(end mid).

***Both formula have computed the end mid correctly.***

🡪 for beg=-11 and end=-3, #elements = 9

So there are only one mid i.e, -7.

***Both formula have computed mid correctly.***

🡪 for beg=-10 and end=-3, #elements = 8

So there are two mids, -7(beg mid) and -8(end mid).

***The formula (beg+end)/2 has failed to compute the beg mid correctly but the***

***other formula has computed it correctly.***

So we should always use the **beg+((end-beg)/2 )** formula to compute beg mid as it is much more reliable;

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

**Ans:** Steps to perform Ternary Search:

1. First, we compute 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: Iterative Approch of Ternary Search**

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

{

While(r<=1)

{

//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(arr[mid1] == key)

{

return mid1;

}

if(arr[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 < arr[mid1])

{

//The key lies in between l and mid1

r = mid1 – 1;

}

elese if(key < arr[mid2])

{

//The key lies in between mid2 and r

l = mid2 + l;

}

else

{

//The key lies in between mid1 and mid2

l = mid1 + l;

r = mid2 – l;

}

}

}