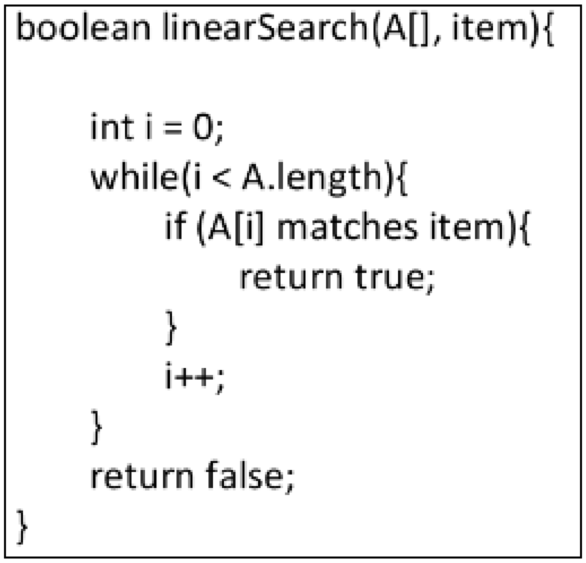
Searching Algorithms

Linear Search:

This is the most basic form of searching. Given an item to search from a bucket of items, we search one by one. If we find it, then bingo! Otherwise, we look until the end of the bucket.

Eg. Given A = [1,2,3,4,5] and the item we are looking for is 5.



```
boolean linearSearch(A[], item){

    int i = 0;
    while(i < A.length){
        if (A[i] matches item){
            return true;
        }
        i++;
    }
    return false;
}
```

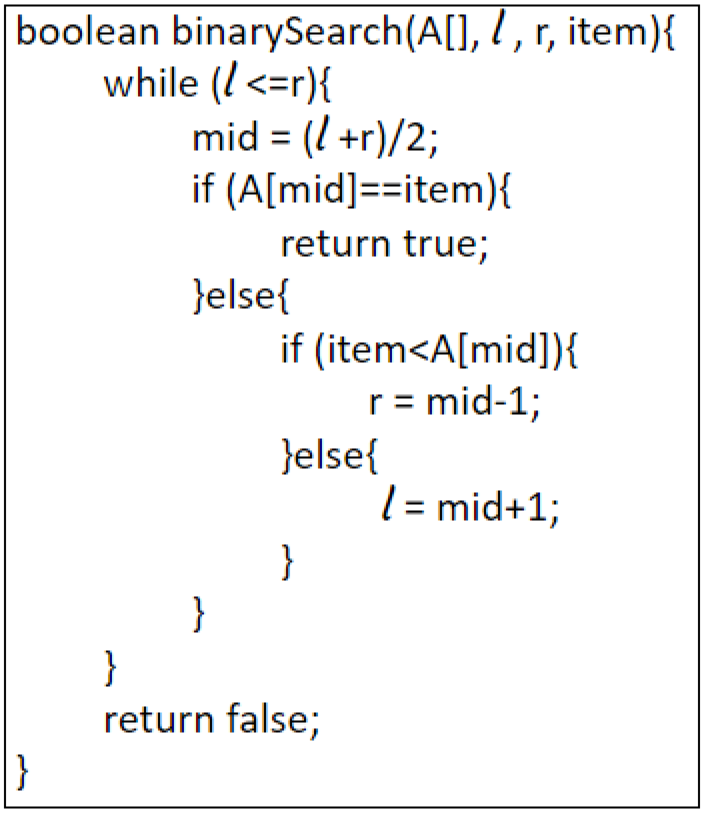
Follow-up Question:

Q) What is the Time Complexity of Linear Search?

Ans: O(n)

Binary Search:

This search follows a divide and conquer algorithm, It does not search the entire array like linear search. Given an array of sorted elements and the item, k, to search for, we first check if the middle element matches k. If we find then bingo! Otherwise, we check if k > middle element or k < middle element. If the first condition is true we look into the right half of the array. If the second condition is true we look into the left half of the array.



```
boolean binarySearch(A[], ℓ, r, item){
    while (ℓ <=r){
        mid = (ℓ +r)/2;
        if (A[mid]==item){
            return true;
        }else{
            if (item<A[mid]){
                r = mid-1;
            }else{
                ℓ = mid+1;
            }
        }
    }
    return false;
}
```

Simulation:

A = [1, 2, 3, 4, 5]	item = 5

A = [1, 2, 3, 4, 5]
l m r
	l = 0, r = 4, m = (l + r) // 2 = 2
	A[2] = 3, 3 < 5
	l = m + 1 = 3

A = [1, 2, 3, 4, 5]
l r
m
	l = 3, r = 4, m = (l + r) // 2 = 3
	A[3] = 4, 4 < 5
	l = m + 1 = 4

A = [1, 2, 3, 4, 5]
l
m
r
	l = 4, r = 4, m = (l + r) // 2 = 4
	A[4] = 5, 5 == 5
	TRUE

Follow-up Question:

Q) What is the Recurrence Equation?

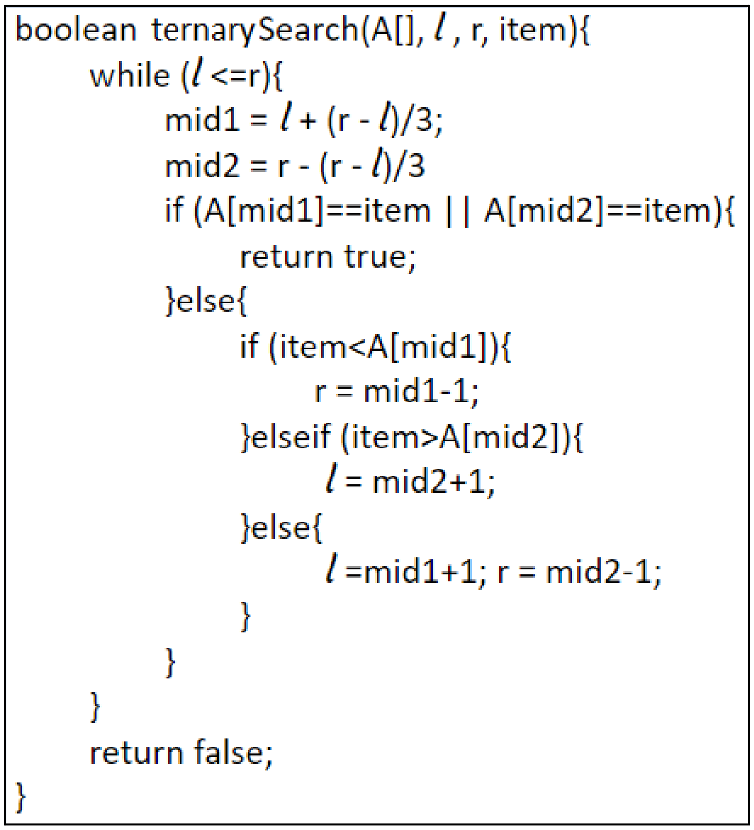
Ans: It depends on the number of subproblems, each subproblem size, and the work done at each step. In each step, there is 1 subproblem where size gets divided by 2 (n/2) and work done is 1 at each step. $T(n) = T(n/2) + 1$

Q) What is the Time Complexity of Binary Search?

Ans: $O(\log_2 n)$

Ternary Search:

This search also follows a divide-and-conquer algorithm. It is similar to binary search except that the array is divided into 3 portions. There are 2 “mids” and we check for the item, k inside both of them. If we find then bingo! Else we check if k < first mid or k > second mid or k > first mid and k < second mid? If the first condition is true we look into the rightmost portion of the array. If the second condition is true we look into the leftmost portion of the array else we look into the center portion.



```
boolean ternarySearch(A[], l , r, item){
        while (l <=r){
                mid1 = l + (r - l)/3;
                mid2 = r - (r - l)/3
                if (A[mid1]==item || A[mid2]==item){
                        return true;
                }else{
                        if (item<A[mid1]){
                                r = mid1-1;
                        }elseif (item>A[mid2]){
                                l = mid2+1;
                        }else{
                                l =mid1+1; r = mid2-1;
                        }
                }
        }
        return false;
}
```

Simulation:

A = [1, 2, 3, 4, 5, 6, 7]	item = 7

A = [1, 2, 3, 4, 5, 6, 7]
l m_1 m_2 r
	l = 0, r = 6, m_1 = l + (r- l) // 3 = 2, m_2 = r - (r- l) // 3 = 4
	A[4] = 5, 5 < 7
	l = m_2 + 1 = 5

A = [1, 2, 3, 4, 5, 6, 7]
l r
m_1 m_2
	l = 5, r = 6, m_1 = l + (r- l) // 3 = 5, m_2 = r - (r- l) // 3 = 6
	A[6] = 7, 7 == 7
	TRUE

Follow-up Question:

Q) What is the Recurrence Equation?

Ans: It depends on the number of subproblems, each subproblem size, and the work done at each step. In each step, there is 1 subproblem where size gets divided by 3 (n/3) and work done is 1 at each step. $T(n) = T(n/3) + 1$

Q) What is the Time Complexity of the Ternary Search?

Ans: $O(\log_3 n)$

Q) Which is faster, binary search or ternary search?

Ans: Binary Search is faster even though it is $O(\log_2 n)$ which is bigger than $O(\log_3 n)$. The reason behind this is the number of comparisons required for each search. There is 1 comparison per iteration in binary search (left part or right part) whereas there are 2 comparisons per iteration in ternary search (left part, mid part, right part). This makes the running time of binary search $1*(\log_2 n)$ and ternary search $2*(\log_3 n)$. Hence, binary search is faster.