**Search:**

Given an array, searching for an element can be one of the most vital operation that a programmer may need to handle or design. The searching techniques can vary depending on the nature of the array.

If the array is **unsorted**, then mostly linear searching makes sense. (although divide and conquer can be applied. Thats for another day!)

If the array is sorted, then different methods can be used. Such as binary search and ternary search.

**Linear Search:**

As the name suggests, its a pretty straightforward approach to linearly search each element and search for the item. This slide will be enough to understand the solution and it’s time complexity (worst case O(N)). [**Linear Search Slide**](https://drive.google.com/file/d/1vAJ2EbjdNmv6_ZgbEcDwDJj3RnwPs7qp/view)

**Binary Search:**

Binary search can be both done iteratively and recursively. The primary concept for this algorithm is to divide the array into two parts each time. **KEEP IN MIND, THIS TECHNIQUE CAN ONLY BE APPLIED CONSIDERING THAT THE ARRAY IS SORTED.** To put it simply, we fix a mid point, if the number is less than the mid then we search on the left half and if the number is greater than the mid then we search on the right half. Refer to this slide for better understanding [**Divide & Conquer Slide**](https://drive.google.com/file/d/1JUGTeiVhRnwaMKVcvwruklKGymlq-U25/view) **[**read the part regarding binary search only for now**]**

**Optional:** You can also watch this playlist that addresses different combinations of binary search: [Binary Search related playlist [OPTIONAL]](https://youtube.com/playlist?list=PL2_aWCzGMAwL3ldWlrii6YeLszojgH77j&si=XtOfwdO8rukMjgIH)

**Ternary Search:**

Its similar to the binary search and the condition that array must be sorted is also applied here. The difference from binary search is that instead of having 1 mid point at the n/2; it divides the array into 3 parts; meaning : [ start—->mid1—->mid2—->end ] The rules are the same, in whichever portion the number is found we check them. So in this case the array is divided by 3rd of its size. [**Ternary Search Slide**](https://drive.google.com/file/d/1tP_ebyE3uHpGto6NIcDf5-VIzUsgoyOe/view)

**Binary vs Ternary:**

Although at a glance its easily assumed that ternary should be a better solution than binary search since one is cutting the search area by half and other is cutting the area by one-third; however, if you really compare these two algorithms:

**Binary search**

**Procedure binary\_search**

**A ← sorted array**

**n ← size of array**

**x ← value to be searched**

**Set lowerBound = 1**

**Set upperBound = n**

**while x not found**

**if upperBound < lowerBound**

**EXIT: x does not exists.**

**set midPoint = lowerBound + ( upperBound - lowerBound ) / 2**

**if A[midPoint] < x**

**set lowerBound = midPoint + 1**

**if A[midPoint] > x**

**set upperBound = midPoint - 1**

**if A[midPoint] = x**

**EXIT: x found at location midPoint**

**end while**

**Ternary Search**

**Begin**

**if start <= end then**

**midFirst := start + (end - start) /3**

**midSecond := midFirst + (end - start) / 3**

**if array[midFirst] = key then**

**return midFirst**

**if array[midSecond] = key then**

**return midSecond**

**if key < array[midFirst] then**

**call ternarySearch(array, start, midFirst-1, key)**

**if key > array[midSecond] then**

**call ternarySearch(array, midFirst+1, end, key)**

**else**

**call ternarySearch(array, midFirst+1, midSecond-1, key)**

**else**

**return invalid location**

**End**

The recurrance relation equation of these two:

T1(n) = T(n/2) + 2c [only 2 comparison per instance in binary search]

T2(n) = T(n/3) + 4c [4 comparisons per instance for ternary]