What is searching?

In computer science searching is the process of finding an item with specified properties from a collection of items. The items may be stored as records in a data base.

Simple data elements in arrays, test in files, nodes in trees, vertices and edges in graphs, or may be elements of other search space.

Why do we need searching?

Searching is one of the core computer science algorithms. We know that today's computer store a lot of information. To retrieve this information proficiently we need very efficient searching algorithms.

There are certain ways of organizing the data which improves the searching process. That means, if we keep the data in proper order. It is easy to search the required element. Sorting is one of the techniques for making the elements ordered.

Types of Searching:

- Unordered Linear Search
- Sorted/Ordered Linear Search
- Binary Search
- Symbol Table and Hashing
- String searching Algorithms: Tries, Ternary Search, and Suffix Trees

Email: aatish.azad7@gmail.com

Unordered Linear Search:

Assume that given array whose element order is not know. That means the elements of the array are not sorted. In this case if we want to search for an element then we have to scan the complete array and see if the element is there in the given list or not.

```
public static int search(int[] arr, int key){
    for(int i=0;i<arr.length;i++){
        if(arr[i] == key){
            return i;
        }
    }
    return -1;
}
```

Sorted/Ordered Linear Search

If the elements of the array are already sorted in the many cases we don't have to scan the complete array to see if the element is there in the given array or not.

In the algorithm below. It can be seen that at any point if the value at arr[i] is greater than data to be searched then we just return -1 without searching the remaining array.

```
public static int search(int[] arr,int data){
    for(int i=0;i<arr.length;i++){
        if(data == arr[i]){
            return i;
        }else if(arr[i]> data){
            return -1;
        }
    }
    return -1;
}
```

Time complexity of this algorithm is O(n). This is because in the worst case we need to scan the complete array.

- But in the average case it reduces the complexity even though the growth rate is same.
- Space Complexity: O(1)

Binary Search

- Binary Search is much more faster that Linear Search
- Rather than eliminating one element at a time, you can eliminate half of the remaining elements at a time.

NOTE:

Binary search works only on sorted arrays!

- The idea behind Binary Search is Divide and Conquer
- Let's see an example:

  [1,3,4,6,8,9,11,12,15,16,17,18,20] ----------> **Sorted Array**

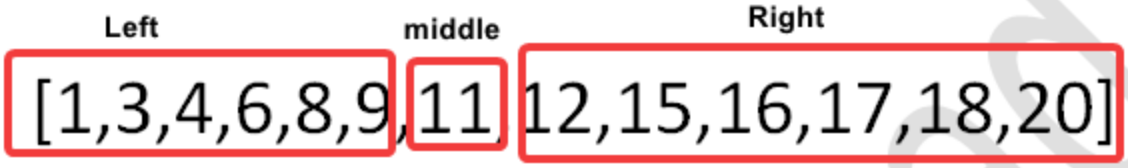
  Let's search of **17.**

Steps:

1. Take roughly the middle element of array

    - In our example: arr.length/2 = 13/2 = 6

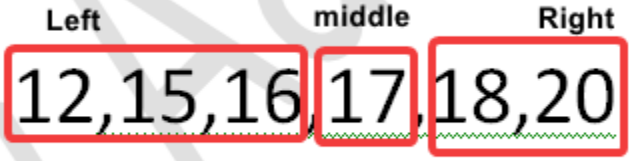
    - arr[6] = 11;



2. Compare the middle element with the search data:

    (11 > data) => 11 > 17 => false

    - If 11 is not less than 17, cancel out the left side



3. Repeat step 1 with the remaining array

arr.length/2 = 3

arr[3] = 17

4. arr[3] is equal to 17, we found our element.

It only takes 2 guess to find the element using Binary Search.

Iterative Binary Search:

```
//Iterative Solutions
public static int binarySearch(int[] inputArr, int key) {

    int start = 0;
    int end = inputArr.length - 1;
    while (start <= end) {
        int mid = start + (end-start)/2;
        if (key == inputArr[mid]) {
            return mid;
        }
        if (key < inputArr[mid]) {
            end = mid - 1;
        } else {
            start = mid + 1;
        }
    }
    return -1;
}
```

Recursive Binary Search:

```
//Recursive solution
public static int binarySearch(int[] arr,int low,int high,int data) {
    int mid = low + (high - low)/2;
    if(low <= high) {
        if(arr[mid] == data) {
            return mid;
        }else if(arr[mid] < data) {
            return binarySearch(arr,mid+1,high,data);
        }else {
            return binarySearch(arr,low,mid-1,data);
        }
    }
    return -1;
}
```

The Big O for binary search is ***O(log n)***

Email: aatish.azad7@gmail.com