Searching and Sorting

Searching

Searching means to find out whether a particular element is present in the given array/list. For instance, when you visit a simple google page and type anything that you want to know/ask about, basically you are searching that topic in google's huge database for which google is using some technique in order to provide the desired result to you.

There are basically two types of searching techniques:

- Linear search
- Binary search

Linear Search

It is a simple sequential search over all the elements of the array, and each element is checked for a match, if a match is found return the element otherwise the search continues until we reach the end of the array.

Pseudocode:

```
/*
    array from leftidx(0) to rightidx(arr.length-1) is considered
*/
function linearSearch(arr, leftidx , rightidx , target)

    //  Search for the target from the beginning of arr

    for idx = 0 to arr.length-1
        if arr[idx] == target
            return idx

    //  target is not found
    return -1
```

Time complexity: O(N), as we traverse the array only once to check for a match for the target element.

Space complexity: O(1), as no extra space is required.

Binary Search

Search in a sorted array by repeatedly dividing the array into two halves and searching in one of the halves.

Suppose you want to find a particular element in the sorted array, then following this technique, you have to traverse all the array elements for searching one element but guess if you only have to search at most half of the array indices for performing the same operation. This can be achieved through binary search.

Now, let's look at what binary searching is.

Let us consider the array:

0	1	2	3	4
1	2	3	4	5

Given an array of size 5 with elements inside the boxes and indices above them. Our target element is 2.

Steps:

- Find the middle index of the array.
- Now, we will compare the middle element with the target element. In case they are equal then we will simply return the middle index.
- In case they are not equal, then we will check if the target element is less than or greater than the middle element.

1. In case, the target element is less than the middle element, it means that the target element, if present in the given array, will be on the left side of the middle element as the array is sorted.
2. Otherwise, the target element will be on the right side of the middle element.

- This helps us discard half of the length of the array each time and we reduce our search space to half of the current search space.

In the example above, the middle element is 3 at index 2, and the target element is 2 which is greater than the middle element, so we will move towards the left part. Now marking start = 0, and end = n/2-1 = 1, now middle = (start + end)/2 = 0. Now comparing the 0-th index element with 2, we find that 2 > 1, hence we will be moving towards the right. Updating the start = 1 and end = 1, middle becomes 1, comparing the 1-st index of the array with the target element, we find they are equal, meaning from here we will simply return 1 (the index of the element).

Advantages of Binary search:

- This searching technique is fast and easier to implement.
- Requires no extra space.
- Reduces time complexity of the program to a greater extent i.e O(logN), where N is the number of the elements in the array, provided the given array is already sorted.

Pseudocode:

```
/*
    array of size N from 0 to N-1 is considered
*/
function binarySearch(arr, N, target)

        //   Initializing lo and hi pointers
        lo = 0
        hi = N-1

        //   Searching for the target element until lo<=hi
        while lo <= hi

                //   Finding the mid of search space from lo to hi
                mid = lo + (hi-lo)/2

                //   If the target element is present at mid
                if arr[mid] == target
                        return mid

                /*
                If the target element is less than arr[mid], then if the target is
                present, it must be in the left half.
                */
        if target < arr[mid]
            hi = mid-1

        //   Otherwise if the target is present, it must be in the right half
        else
            lo = mid+1

    //   If the target is not found return -1
    return -1
```

Time complexity: O(logN), where N is the number of elements in the array, given the array is sorted. Since we search for the target element is one of the halves every time, reducing our search space to half of the current search space.

Since we go on searching for the target until our search space reduces to 1, so

Iteration 1- Initial search space: N

Iteration 2 - Search space: N/2

Iteration 3 - Search space: N/4

Let after ‘k’ iterations search space reduces to 1

So, N/(2k) = 1

=> N = 2k

Taking Log2 on both sides:

=> k = log2N

Hence, the maximum number of iterations ‘k’ comes out to be log2N.

Space complexity: O(1), as no extra space is required.

Sorting

Sorting means an arrangement of elements in an ordered sequence either in increasing(ascending) order or decreasing(descending) order. Sorting is very important and many software and programs use this. The major difference is the amount of space and time they consume while being performed in the program.

For a detailed explanation of types of sorting algorithms that are generally used, please refer to the topic Sorting Algorithms.