Basic Searching Algorithms.

Binary Search

Def:-

"Binary search is more efficient algorithm for finding an element in a sorted array. It works by repeatedly dividing the search intervals in half, making it much faster than linear search for large datasets.

Note:- Array must be sorted. (Ascending order)
for.eg {1, 10, 20 30, 100}

Step-by-step Explanation

0	1	2	3	4
10	20	30	40	50
l				r

1. Divide the array & find the mid, suppose our target is 20.

   mid = (0+4)/2 = 2.

2. Check whether the mid value is equal to target value.

```
 arr[mid]    target
 ( 30    ==   20) {
       False.
 }
```

```
if 20           20
 (mid   ==   target)
 {
   return mid;
          ↑
       index value
 }
```

3. If target value is greater than mid then, ignore ~~right half~~ left half.

i.e mid < target

target = 40

0	1	2	3	4
10	20	30	40	50
l		mid		r

mid = 2

mid < target

30 < 40

0	1	2	3	4
~~10~~	~~20~~	~~30~~	40	50
		mid	l	r

l = mid + 1

4. If target value is smaller than mid, ignore right half.

0	1	2	3	4
10	20	30	40	50
l		mid		r

target = 20;

mid > target

30 > 20

0	1	2	3	4
10	20	30	40	50
l	r	mid		

$r = mid - 1$

5. If the element is not present then return -1.

#Code.

```
public class BinarySearch {

    public static int binarySearch (int[] arr, int target){
        int left = 0;
        int right = arr.length -1;

    while ( left <= right) {

        int mid = (left + right) / 2;

        // check if target is present at mid
        if (arr[mid] == target) {
            return mid;
        }

        // If target greater, ignore left half
        if (arr[ mid] < target) {
            left = mid + 1;
        }
```

```
    }        }

(arr[mid]>tar) // If target is smaller, ignore right half
    else {
        right = mid - 1;
    }
  }

  // Targe was not found
  return -1;
}

public static void main(String[] args){
    int[] array = { 10, 20, 30, 40, 50};
    int target = 20;

    int result = binarySearch(array, target);

    if( result != -1){
       System.out.println("Element " + target +" found at
            index: " + result);
    } else{
       System.out.println("Element" + target + "not found
           in the array.");
    }
  }
}
```

Date
Page

Time Complexity :-

Best case :- $\Omega(1)$
Average case :- $\Theta(\log n)$
Worst case :- $O(\log n)$

* **Best case :-**

target element is in the middle of array. hence $O(1)$.

* **Average case :-**

binary search divides the search space in half with each comparison. This leads to a logarithmic number of comparisons, hence $O(\log n)$.

i.e number of comparisons growing logarithmically with the size of (n) increases.

as array size ↑ increases, the number of comparisons needed to find an element grows very slowly, making binary search very efficient for large datasets.

* **Worst case :-**

$O(\log n)$