## Liner Search:

Start searching from the first element till you find the element that you are looking for.

arr = [18, 12, 9, 14, 77, 56, 50] search for 14 in arr.
(0) (1) (2) (3) (4) (5) (6)
✗ ✗ ✗ i

14 exists at index 3

If no value found return -1

**Time complexity:** time grows as input grows

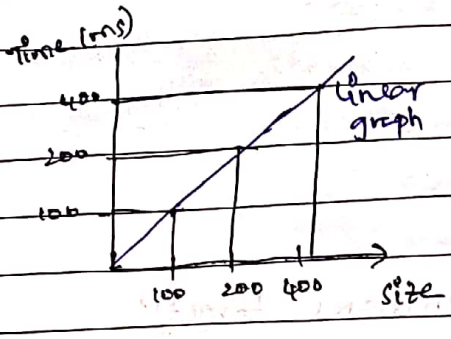
Best case: O(1) → constant
Worst case: O(N) → size of the array

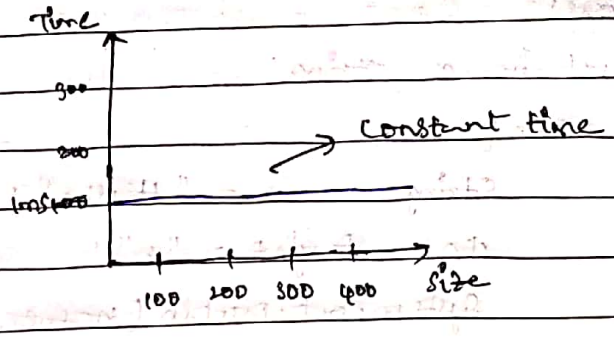
How many checks will the loop make in best case
i.e element found at 0th index
arr = [8, 9, 10 . . . . 200 items]
target = 8
only 1 comparison in best case

arr = [18, 10, 12, . . . . 1 lakh items]
target = 18
here also only 1 comparison made
since element is found at 0th index, there is nothing to do with size of the array.
Hence time complexity is one in best case.

**Worst case:**
we do not find the target item after iterating/going through every item.

Size of the array = 100 → 100 comparisons → 100 ms
1 lakh → 1 lakh comparisons → 100000 ms
200 → 200 comparisons → 200 ms




worst case

Ex:

```
psvm {
    int[] num = {23, 40, 1, 3, 19, 24, 39, 56, -11, 28};
    int target = 19;
    int ans = linearSearch(num, target);
    System.out.println(ans);
}
static int linearSearch(int[] arr, int target){
    if (arr.length == 0){
        return -1;
    }
    for(int index = 0; index < arr.length; index++){
        int element = arr[index];
        if (element == target){
            return index;
        }
    }
    return -1; → This will execute if none of the above return statements executed
}
```

Page No.:
Date: YOUVA

Questions:

1. Search in a string:

```
psvm{
    String name = "Harshe";
    char target = "r";
    System.out.println( name, target) search(name, target);
    }
static boolean search(String str, char target){
    if (str.length() == 0){
        return false;
    }
    for (int i = 0 ; i< str.length(); i++){
        if (target == str.charAt(i)){
            return true;
        }
    }
    return false;
}
```

2. Search in Range:

```
        0   1   2   3   4   5
arr = [18, 12, -7, 3, 14, 28]
```

Search for 3 in range of index [1,4]

psvm{

```
    int[] num = { 18, 12, -7, 3, 14, 28};
    int target = 3;
    System.out.println(search(num, target, 1, 4));
}
static int search(int[] arr, int target, int start, int end){
    if (arr.length == 0){
        return -1;
    }
    for (int i = start; i <= end; i++){
        // int element = arr[i];
        if (element == target){
```

```
            return true;
        }
    }
    return false;
}
```

Q3. Minimum Number:

```
psvm{
    int[] arr = {5,8,24,3,7,9,39};
    System.out.println(min(arr));
}
static int min(int[] arr){
    int ans = arr[0];
    for(i=1; i<arr.length; i++){
        if(arr[i] < ans){
            ans = arr[i];
        }
    }
    return ans;
}
```

Q4. Search in 2D Arrays: Here we are searching and returning row, col as array

```
psvm{
    int[][] arr = {
        {23,4,1},
        {18,12,3,9},
        {78,99,34,56},
        {18,12}
    };
    int target = 3;
    int[] ans = search(arr, target); // format of return value {row, col}
    System.out.println(Arrays.toString(ans));
}
```

```
static int[] search(int[][] arr, int target){
    for(int row=0; row<arr.length; row++){
        for(int col=0; col<arr[row].length; col++){
            if arr[row][col] == target;{
                return new int[]{row, col};
            }
        }
    }
    return new int[]{-1, -1};
}
```

Max in 2D Array:

```
static int max(int[][] arr){
    int max = arr[0][0] (or) Integer.MIN_VALUE;   // Min value Integer can hold
    for(int row=0; row<arr.length; row++){
        for(int col=0; col<arr[row].length; col++){
            if(max > arr[row][col]){
            if(arr[row][col] > max){
                max = arr[row][col];
            }
        }
    }
    return max;
}
```

Q5. Even Digits:

```
public class EvenDigits{
    psvm{
        int[] nums = {12, 345, 2, 6, 7896};
        System.out.println(findNumbers(nums));
    }
    static int findNumbers(int[] nums){
        int count=0;
        for(int num: nums){
            if(even(nums)){
```

```
            count++;
        }
    }
    return count;
}
//function to check whether a number contains even
digits or not
static boolean even(int num){
    int noofdigits = digits(num)
    return noofdigits % 2 == 0;
}
static int digits(int num){ ⟶ count no. of digits
    if(num < 0){                in a number
        num = num * -1;
    }
    if(num == 0){
        return 1;
    }
    int count = 0;
    while(num > 0){
        count++;
        num = num / 10;
    }
    return count;
}
}
shortcut to find no. of digits:
    static int digits2(int num){
        return (int)(Math.log10(num)) + 1;
    }
```

M T W T F S S

Page No.: YOUVA

Date:

Q5. Max Wealth:

```
Psvm {

}
```

person → rows
account → col

```
public int maxWealth(int[][] accounts){
    int ans = Integer.MIN_VALUE;
    for(int person=0; person < accounts.length; person++)
    {
        int sum = 0;
        for(int account=0; account < accounts[person].length;
                                                account++){
            sum = sum + accounts[person][account];
        }
        if(sum > ans){
            ans = sum;
        }
    }
    return ans;
}
}
```

Binary Search:

Assume sorted array

1. Find the middle element
2. target > mid → search in the right side
   else search in left
3. if target element == middle element

```
0 1 2 3 4 5 6 7 8 9
2, 4, 6, 9, 11, 12, 14, 20, 36, 48
S        M          e
```

target = 36

$m = \frac{0+9}{2}$

$= 4$

```
            S          M     e
           12, 14, 20, 36, 48
```

$a = \frac{5+9}{2}$

$= 7$

36, 48
M

$m = \frac{8+9}{2}$

$= 8$

target == Middle

S M e

0 1 2 3 4 5 6 7 8
2, 4, 6, 9, (11) | 12, 14, 20, 36, 48

search 12

target 12 > Middle 11

$\frac{5+9}{2} = 7$

S M e

12, 14, (20), 36, 48

target 12 < Middle 20

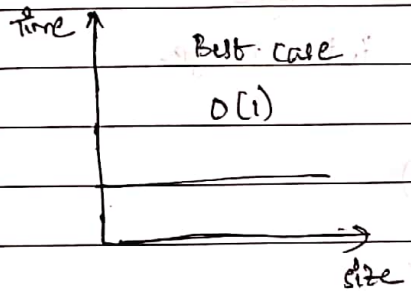
s e

(12) 14

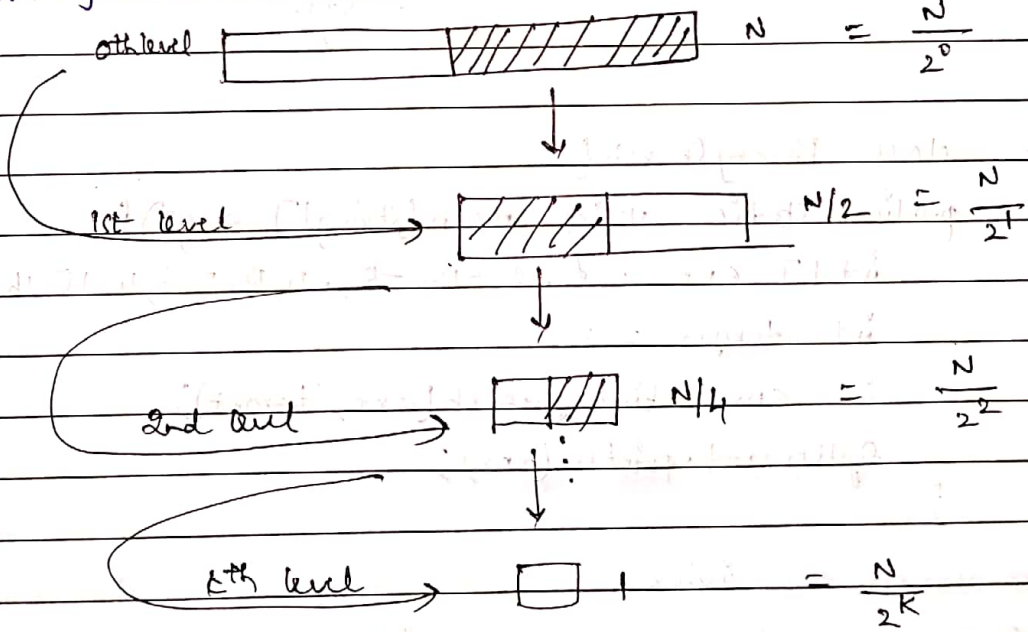
$\frac{5+6}{2} = 5$

Middle == target

if s > e, element not found




Why Binary Search?




$\frac{N}{2^K} = 1$

$N = 2^K$

$\log N = \log 2^K \Rightarrow \log N = K \log 2$

$K = \frac{\log N}{\log 2} = \boxed{K = \log_2 N}$

Total comparisons in worst case = logN

For searching in 100000

Linear
1 mn comparison

Binary search
$\log_2(100000)$
20 comparisons
O(logN)

Better way to find mid value

$m = \frac{s+e}{2}$ → if s & e values are large s+e they may exceed int range

$m = s + \frac{(e-s)}{2}$

$s + \frac{(e-s)}{2}$

$\frac{2s + e - s}{2}$

$= \frac{s+e}{2}$

Ex:

```
Public class binarySearch{
    public static void main(String[] args){
        int[] arr = {-18, -12, -5, -4, 0, 2, 3, 4, 15, 16, 22, 45, 89};
        int target = 22
        int ans = binarySearch(arr, target);
        System.out.println(ans);
    }

// return the index
// return -1 if doll not exist
    static int binarySearch(int[] arr, int target){
        int start = 0;
        int end = arr.length - 1;
```

```
while (start <= end){
    int mid = start + (end - start)/2;
    if (target < arr[mid]){
        end = mid - 1;
    } else if (target > arr[mid]){
        start = mid + 1;
    } else{
        return mid;
    }
}
return -1;
}
}
```

Order-Agnostic Binary Search:

In Binary search, we assumed array is sorted in an ascending order. But what if we do not know the order, in that case we use order Agnostic Binary Search.

To know the order

arr = [3, 4, 7, 9, 15, 35, 49, 84, 99]

s ↓ e

if s > e → decreasing order

else

increasing order

```
public class OrderAgnosticBS{
    public static void main(String[] args){
        int[] arr = {-18, -12, -4, 0, 2, 3, 4, 15, 16, 18, 22, 30, 80};
        int target = 22;
        int ans = orderAgnosticBS(arr, target);
        System.out.println(ans);
    }
```

```
static int orderAgnosticBS(int[] arr, int target){
    int start = 0;
    int end = arr.length-1;
// find arr in Asc or Desc
    boolean isAsc = arr[start] < arr[end];
    while (start <= end){
        int mid = start + (end-start)/2;
        if (arr[mid] == target){
            return mid;
        }
        if (isAsc){
            if (target < arr[mid]){
                end = mid-1;
            } else {
                start = mid+1;
            } else{
                if (target < arr[mid]){
                    start = mid+1;
                } else{
                    end = mid-1;
                }
            }
        }
        return -1;
    }
}
```