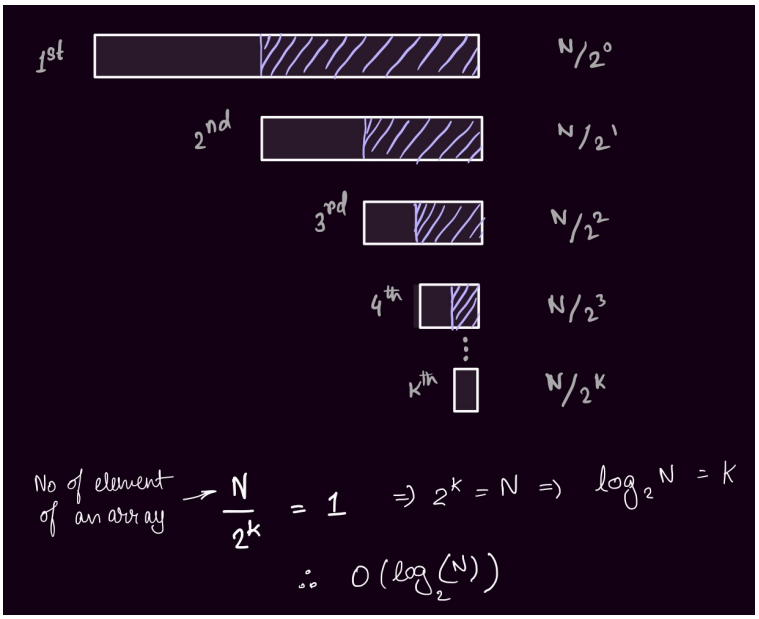
Binary Search

Time Complexity :

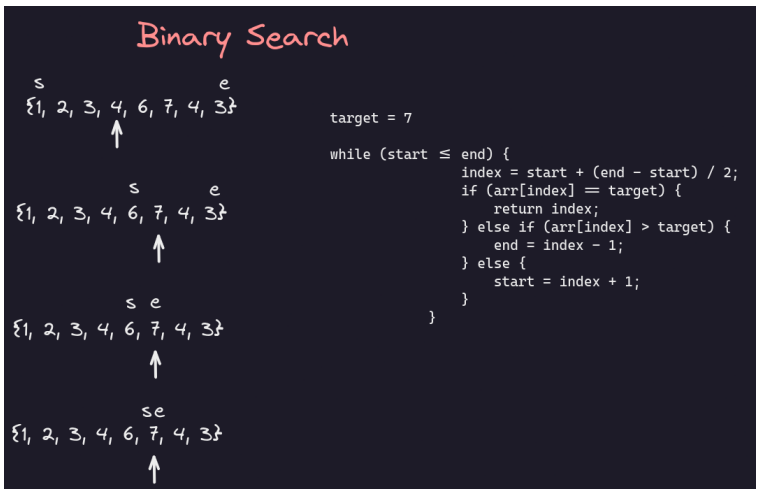
- Best Case : O (1)
- Worst Case : O (log (n)) [where n is number of element in the array]



Steps for Binary Search :

1. Find the middle element.
2. Compare the target element with the middle element :
   a. if target element > middle element :
      1. if Ascending Array : search in left side of the array.
      2. if Descending Array : search in right side of the array.
   b. if target element < middle element :
      1. if Ascending Array : search in right side of the array.
      2. if Descending Array : search in left side of the array.
   c. if target element = middle element :
      1. return element

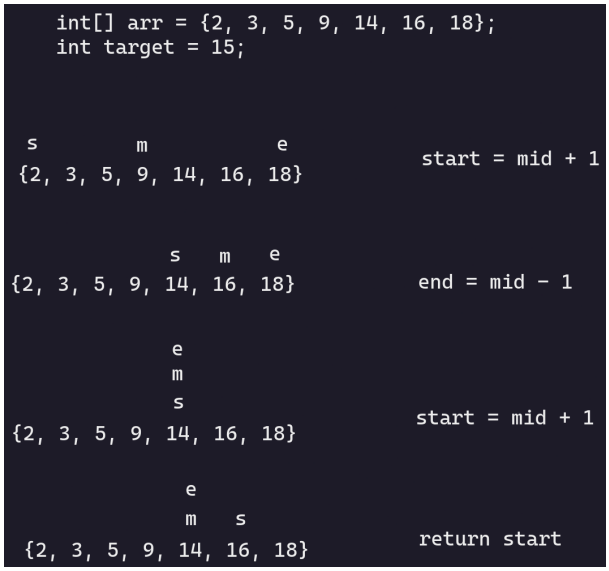
- Q1. Binary Search




```java
package com.inclass;

public class BinarySearch {

    public static void main(String[] args) {
        /*
        Binary Search: (given sorted array (say ascending)
            1. find the middle element
            2. if middle element == target element: ans
            3. if middle element > target: search left
                else middle element < target: search right
         */
        int[] arr = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
        int target = 3;
//        int[] arr = {48, 36, 20, 14, 12, 11, 9, 6, 4, 2};
//        int target = 36;
        if (arr[0] ≥ arr[arr.length - 1]) {
            System.out.println(binarySearchDes(arr, target));
        } else {
            System.out.println(binarySearchAsc(arr, target));
        }
    }

    // Binary Search for Ascending Sorted Array
    static int binarySearchAsc(int[] arr, int target) {
        int start = 0;
        int end = arr.length - 1;
        int index = 0;
        while (start ≤ end) {
            index = start + (end - start) / 2;
            if (arr[index] == target) {
                return index;
            } else if (arr[index] > target) {
                end = index - 1;
            } else {
                start = index + 1;
            }
        }
        return -1;
    }

    // Binary Search for Descending Sorted Array
    static int binarySearchDes(int[] arr, int target) {
        int start = 0;
```

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

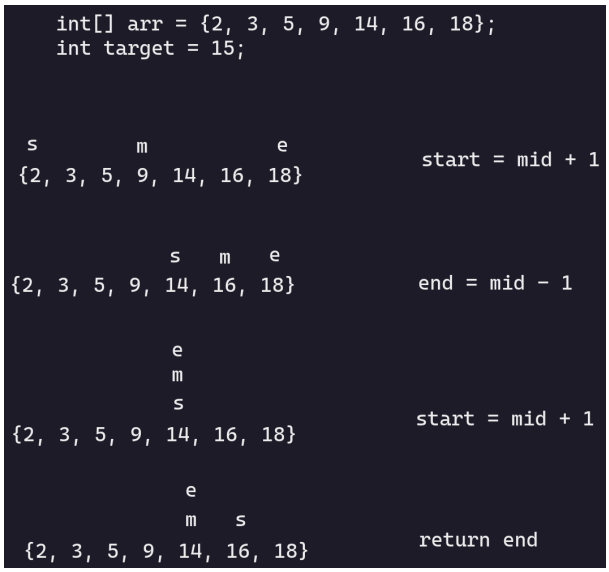
- Q2. Ceiling Element




```
package com.questions;

import java.util.Scanner;

public class Ceiling {
    public static void main(String[] args) {
        int[] arr = {2, 3, 5, 9, 14, 16, 18};
        Scanner in =new Scanner(System.in);
        int target = in.nextInt();
        if (target > arr[arr.length - 1]) {
            System.out.println("No Result Found !");
        }
        else {
            System.out.println(ceiling(arr, target));
        }
    }
    static int ceiling(int[] arr, int target) {
        int start = 0;
        int end = arr.length - 1;
        int index = 0;
        while (start <= end) {
            index = start + (end - start) / 2;
            if (arr[index] == target) {
                return arr[index];
```

```
            }
            else if (arr[index] > target) {
                end = index - 1;
            }
            else {
                start = index + 1;
            }
        }
        return arr[start];
    }
}
```

- Q3. Floor Element




```
package com.questions;

import java.util.Scanner;

public class Floor {

    public static void main(String[] args) {
        int[] arr = {2, 3, 5, 9, 14, 16, 18};
        Scanner in =new Scanner(System.in);
        int target = in.nextInt();
        if (target < arr[0]) {
            System.out.println("No Result Found !");
        }
        else {
            System.out.println(floor(arr, target));
        }
    }
    static int floor(int[] arr, int target) {
        int start = 0;
        int end = arr.length - 1;
        int index;
        while (start ≤ end) {
            index = start + (end - start) / 2;
            if (arr[index] == target) {
                return arr[index];
            }
            else if (arr[index] > target) {
                end = index - 1;
```

```
            }
            else {
                start = index + 1;
            }
        }
        return arr[end];
    }
}
```

- Q4. Find First and Last position of element in Sorted Array

```
package com.questions;

import java.util.Arrays;

// <https://leetcode.com/problems/find-first-and-last-position-of-element-in-sorted-array/>

public class FirstLastIndex {

    public static void main(String[] args) {
        System.out.println(Arrays.toString(selfAttempt()));
        System.out.println(Arrays.toString(solution()));
    }

    /* Self Attempt:
    Runtime: 0 ms, faster than 100.00% of Java online submissions for Find First and Last Position of Element in Sorted Array.
    Memory Usage: 42.4 MB, less than 54·−92% of Java online submissions for Find First and Last Position of Element in Sorted Array.
     */
    static int[] selfAttempt() {
        int[] nums = {5, 7, 7, 8, 8, 8, 10};
        int target = 8;
        int index = firstLastIndex(nums, target);
        int first = firstIndex(nums, target, index);
        int last = lastIndex(nums, target, index);
        return new int[]{first, last};
    }
    static int firstLastIndex(int[] nums, int target) {
        int start = 0;
        int end = nums.length - 1;
        int index = 0;
        while (start ≤ end) {
            index = start + (end - start) / 2;
            if (nums[index] < target) {
                start = index + 1;
            } else if (nums[index] > target) {
                end = index - 1;
            } else {
                return index;
            }
        }
        return -1;
    }

    static int firstIndex(int[] nums, int target, int index) {
        int temp = index;
        for (int i = index; i ≥ 0; i--) {
            if (nums[i] ≠ target) {
```

```
                return i + 1;
            }
            else {
                temp = i;
            }
        }
        return temp;
    }

    static int lastIndex(int[] nums, int target, int index) {
        int temp = index;
        for (int i = index; 0 ≤ i && i < nums.length; i++) {
            if (nums[i] ≠ target) {
                return i - 1;
            }
            else{
                temp = i;
            }
        }
        return temp;
    }

    /* Solution:
    Runtime: 0 ms, faster than 100.00% of Java online submissions for Find First and Last Position of Element in Sorted Array.
    Memory Usage: 44 MB, less than 28.93% of Java online submissions for Find First and Last Position of Element in Sorted Array.
     */
    static int[] solution() {
        int[] nums = {5, 7, 7, 8, 8, 8, 10};
        int target = 8;
        int start = binarySearch(nums, target, true);
        int end = binarySearch(nums, target, false);
        return new int[] {start, end};
    }

    static int binarySearch(int[] nums, int target, boolean firstIndex) {
        int index;
        int start = 0;
        int end = nums.length - 1;
        int ans = -1;
        while (start ≤ end) {
            index = start + (end - start) / 2;
            if (nums[index] < target) {
                start = index + 1;
            } else if (nums[index] > target) {
                end = index - 1;
            } else {
                ans = index;
                if (firstIndex) {
                    end = index - 1;
                } else {
                    start = index + 1;
                }
            }
        }
        return ans;
    }
}
```

– Q5. Find position of an element in a sorted array of infinite array

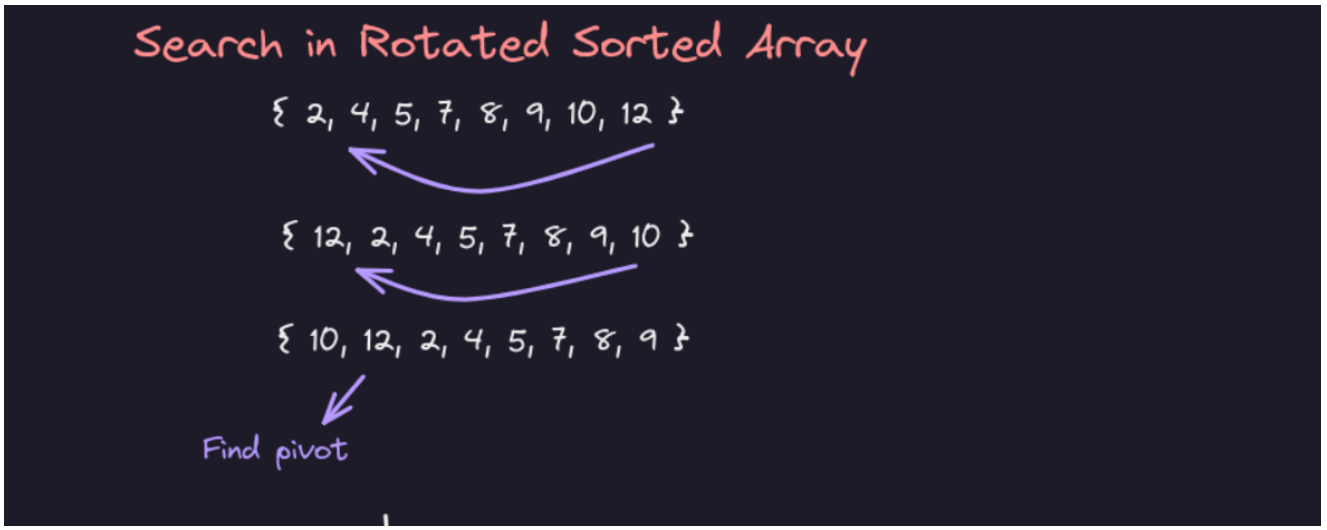
```java
package com.questions;

import java.lang.Math;

public class IndexInfiniteArray {

    public static void main(String[] args) {
        System.out.println(solution());
    }

    static int solution() {
        int[] nums = new int[100];
        for (int i = 0; i < 100; i++) {
            nums[i] = i;
        }
        int target = 13;
        return findRange(nums, target);
    }

    static double log2(int num) {
        return Math.log(num)/Math.log(2);
    }

    static int findRange(int[] nums, int target) {
        int start = 0;
        int end = 0;
        int range = (int) log2(nums.length);
        while(start ≤ end) {
            while (range ≥ 0) {
                if ((nums.length - start + 1) - Math.pow(2, range) ≥ 0) {
                    end = start + (int) Math.pow(2, range) - 1;
                    range--;
                    break;
                }
                range--;
            }
            if (target == nums[end]){
                return end;
            } else if (target < nums[end]){
                return binarySearch(nums, target, start, end - 1);
            } else {
                start = end + 1;
            }
        }
        return -1;
    }

    static int binarySearch(int[] nums, int target, int tempstart, int tempend) {
        while (tempstart ≤ tempend) {
            int index = tempstart + (tempend - tempstart) / 2;
            if (target < nums[index]) {
                tempend = index - 1;
            }
            else if (target > nums[index]) {
                tempstart = index + 1;
            }
            else {
                return index;
```

```
            }
        }
        return -1;
    }
}
```

- Q6. Peak Index in Mountain Array

```
package com.questions;

public class PeakIndexMountainArray {

    // <https://leetcode.com/problems/peak-index-in-a-mountain-array/submissions/>

    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 6, 7, 4, 3, 2, 1};
        System.out.println(solution2(arr));
    }

    /* Solution2 (Error: {1, 2, 2, 3, 1}:
    Runtime: 0 ms, faster than 100.00% of Java online submissions for Peak Index in a Mountain
Array.
    Memory Usage: 39 MB, less than 89.22% of Java online submissions for Peak Index in a
Mountain Array.
     */
    // original solution
    static int solution2 (int[] arr) {
        int start = 0;
        int end = arr.length - 1;

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

- Q7. Search in Rotated Sorted Array



{ 10, 12, | 2, 4, 5, 7, 8, 9 }
asc asc

Finding pivot :

{ 10, 12, 2, 4, 5, 7, 8, 9 }

Case 1 : arr[index] > arr[index + 1]
return index

{ 10, 12, 2, 4, 5, 7, 8, 9 }

Case 2 : arr[index + 1] > arr[index]
return index + 1

{ 10, 12, 2, 4, 5, 7, 8, 9 }

Case 3 : arr[index] <= arr[start]
end = index - 1

{ 10, 12, 2, 4, 5, 7, 8, 9 }

Case 4 : else (arr[index] >= arr[end])
start = index + 1

For duplicate values (without pivot) :

{ 10, 11, 12, 5, 7, 8, 9 }
s e

{ 10, 12, 2, 4, 4, 4, 4 }
s e

Case 1 : arr[index] < arr[end] || arr[index] < arr[start]

Right Side is Sorted

Case 1.1 : arr[index] < target <= arr[end]
start = index + 1

Case 1.2 : else (arr[start] <= target < arr[index])
end = index - 1

{ 10, 12, 2, 4 }
s e

{ 10, 10, 10, 4, 5, 7 }
s e

Case 2 : arr[index] > arr[end] || arr[index] > arr[start]

Left Side is Sorted

Case 2.1 : arr[end] <= target < arr[index]
end = index - 1

Case 2.2 : else (arr[index] < target <= arr[start])
start = index + 1


```java
package com.questions;

public class TargetRotatedSortedArray {

    // <https://leetcode.com/problems/search-in-rotated-sorted-array/>

    public static void main(String[] args) {
        int[] arr = {1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1};
        int target = 2;
        System.out.println(selfAttempt(arr, target));
        System.out.println(solution(arr, target));
        System.out.println(bestSolution(arr, target));
        System.out.println(search(arr, target));
    }

    /* Self Attempt:
    Runtime: 0 ms, faster than 100.00% of Java online submissions for Search in Rotated Sorted Array.
    Memory Usage: 39.7 MB, less than 11.53% of Java online submissions for Search in Rotated Sorted Array.
     */
    static int selfAttempt(int[] arr, int target) {
        int peak = peak(arr);
        if (target ≥ arr[0]) {
            return binarySearch(arr, target, 0, peak);
        } else {
            return binarySearch(arr, target, peak + 1, arr.length - 1);
        }
    }

    static int peak(int[] arr) {
        int start = 0;
        int end = arr.length - 1;
        if (arr[start] < arr[end]) {
            return end;
        } else {
            while (start < end) {
                int index = start + (end - start) / 2;
                if (arr[index] > arr[start]) {
                    start = index;
                } else {
                    end = index;
                }
            }
            return start;
        }
    }

    static int binarySearch(int[] arr, int target, int start, int end){
        while (start ≤ end) {
            int index = start + (end - start) / 2;
            if (target < arr[index]) {
                end = index - 1;
```

```
            } else if (target > arr[index]) {
                start = index + 1;
            } else {
                return index;
            }
        }
        return -1;
    }

    /* Solution:

     */
    static int solution(int[] arr, int target) {
        int pivot = findPivot(arr);
        if (pivot == -1) {
            return binarySearch(arr, target, 0, arr.length - 1);
        } else {
            if (arr[pivot] == target) {
                return pivot;
            } else if (target > arr[0]) {
                return binarySearch(arr, target, 0, pivot);
            } else {
                return binarySearch(arr, target, pivot + 1, arr.length - 1);
            }
        }
    }

    static int findPivot(int[] arr) {
        int start = 0;
        int end = arr.length - 1;
        while(start ≤ end) {
            int index = start + (end - start) / 2;
            if (index < end && arr[index] > arr[index + 1]) {
                return index;
            } else if (index > start && arr[index - 1] > arr[index]) {
                return index - 1;
            } else if (arr[index] == arr[start] && arr[index] == arr[end]) {
                start ++;
                end --;
            } else if (arr[index] ≤ arr[start]) {
                end = index - 1;
            } else if (arr[index] ≥ arr[end]) {
                start = index + 1;
            } else if (arr[start] == arr[end]) {
                start ++;
                end --;
            }
        }
        return -1;
    }

    /* Best Solution (no need to find peak, works with duplicates) :
    Runtime: 0 ms, faster than 100.00% of Java online submissions for Search in Rotated Sorted Array.
    Memory Usage: 39.2 MB, less than 42.77% of Java online submissions for Search in Rotated Sorted Array.
     */
    static int bestSolution (int[] arr, int target) {
        int start = 0;
        int end = arr.length - 1;
```

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

    static int search(int[] nums, int target) {
        int pivot = findPivotKunal(nums);
        // if you did not find a pivot, it means the array is not rotated
        if (pivot == -1) {
            // just do normal binary search
            return binarySearch(nums, target, 0 , nums.length - 1);
        }
        // if pivot is found, you have found 2 asc sorted arrays
        if (nums[pivot] == target) {
            return pivot;
        }
        if (target ≥ nums[0]) {
            return binarySearch(nums, target, 0, pivot - 1);
        }
        return binarySearch(nums, target, pivot + 1, nums.length - 1);
    }

    // this will not work in duplicate values
    static int findPivotKunal(int[] arr) {
        int start = 0;
        int end = arr.length - 1;
        while (start ≤ end) {
            int mid = start + (end - start) / 2;
            // 4 cases over here
            if (mid < end && arr[mid] > arr[mid + 1]) {
                return mid;
            }
            if (mid > start && arr[mid] < arr[mid - 1]) {
                return mid-1;
            }
            if (arr[mid] ≤ arr[start]) {
                end = mid - 1;
            } else {
                start = mid + 1;
            }
        }
        return -1;
    }
```

```
    static int findPivotWithDuplicates(int[] arr) {
        int start = 0;
        int end = arr.length - 1;
        while (start ≤ end) {
            int mid = start + (end - start) / 2;
            // 4 cases over here
            if (mid < end && arr[mid] > arr[mid + 1]) {
                return mid;
            }
            if (mid > start && arr[mid] < arr[mid - 1]) {
                return mid-1;
            }
            // if elements at middle, start, end are equal then just skip the duplicates
            if (arr[mid] == arr[start] && arr[mid] == arr[end]) {
                // skip the duplicates
                // NOTE: what if these elements at start and end were the pivot??
                // check if start is pivot
                if (start < end && arr[start] > arr[start + 1]) {
                    return start;
                }
                start++;
                // check whether end is pivot
                if (end > start && arr[end] < arr[end - 1]) {
                    return end - 1;
                }
                end--;
            }
            // left side is sorted, so pivot should be in right
            else if(arr[start] < arr[mid] || (arr[start] == arr[mid] && arr[mid] > arr[end]))
{
                start = mid + 1;
            } else {
                end = mid - 1;
            }
        }
        return -1;
    }
}
```

- Q8. Split Array Largest Sum

  - Case 1: m = 1; $\Rightarrow sum\ of\ entire\ array$
  - Case 2: m = size of array; $\Rightarrow maximum\ element\ of\ array$