Binary Search:
- Operates on a contiguous sequence with a specified left and right index (Search Space).
- Maintains the left, right, and middle indices of the search space.
- Compares the search target or applies the search condition to the middle value of the collection:
    - if the condition is unsatisfied or values unequal, keep searching within satisfied range.
    - If the search ends with an empty half, the condition cannot be fulfilled and target is not found.

Procedure:
1.Pre-processing - Sort if collection is unsorted.
2.Binary Search - Using a loop or recursion to divide search space in half after each comparison.
3.Post-processing - Determine viable candidates in the remaining space.

Notice:
- Instead of searching a specific value, Binary Search can take many alternate forms.
- Sometimes you will have to apply a specific condition or rule to **determine which side** (left or right) to search next.

When to Use?
- Should be considered every time you need to search for an index or element in a collection.

Q1:

Given a **sorted** (in ascending order) integer array nums of n elements and a target value, write a function to search target in nums. If target exists, then return its index, otherwise return -1

**Example 1:**
**Input:** nums = [-1,0,3,5,9,12], target = 9 **Output:** 4
**Explanation:** 9 exists in nums and its index is 4

**Example 2:**
**Input:** nums = [-1,0,3,5,9,12], target = 2 **Output:** -1
**Explanation:** 2 does not exist in nums so return -1

# Solution1:

```python
def binarySearch(nums, target):
    """
    :type nums: List[int]
    :type target: int
    :rtype: int
    """
    if len(nums) == 0:
        return -1

    left, right = 0, len(nums) - 1
    while left <= right:
        mid = (left + right) // 2
        if nums[mid] == target:
            return mid
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    # End Condition: left > right
    return -1
```

Key Syntax:

- Initial Condition: left = 0, right = length-1
- Termination: left > right
- Searching Left: right = mid-1
- Searching Right: left = mid+1

**Most basic and elementary form of Binary Search !!!**

# Solution2:

```python
class Solution(object):
    def search(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """
        if len(nums) == 0:
            return -1

        left, right = 0, len(nums)
        while left < right:
            mid = (left + right) // 2
            if nums[mid] == target:
                return mid
            elif nums[mid] < target:
                left = mid + 1
            else:
                right = mid

        # Post-processing:
        # End Condition: left == right
        if left != len(nums) and nums[left] == target:
            return left
        return -1
```

Key Syntax:
- Initial Condition: left = 0, right = length
- Termination: left == right
- Searching Left: right = mid
- Searching Right: left = mid+1

Attributes:
- An advanced way to implement Binary Search.
- Search Condition needs to access element's immediate right neighbor
- Use element's right neighbor to determine if condition is met and decide whether to go left or right
- Gurantee Search Space is at least 2 in size at each step
- Post-processing required. Loop/Recursion ends when you have 1 element left. Need to assess if the remaining element meets the condition.

# Solution3:

```python
class Solution(object):
    def search(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """
        if len(nums) == 0:
            return -1

        left, right = 0, len(nums) - 1
        while left + 1 < right:
            mid = (left + right) // 2
            if nums[mid] == target:
                return mid
            elif nums[mid] < target:
                left = mid
            else:
                right = mid

        # Post-processing:
        # End Condition: left + 1 == right
        if nums[left] == target: return left
        if nums[right] == target: return right
        return -1
```

Key Syntax:
- Initial Condition: left = 0, right = length - 1
- Termination: left + 1== right
- Searching Left: right = mid
- Searching Right: left = mid

Attributes:
- An alternative way to implement Binary Search
- Search Condition needs to access element's immediate left and right neighbors
- Use element's neighbors to determine if condition is met and decide whether to go left or right
- Gurantee Search Space is at least 3 in size at each step
- Post-processing required. Loop/Recursion ends when you have 2 elements left. Need to assess if the remaining elements meet the condition.

# Summary:

Template #1:

```
// Pre-processing
...
left = 0; right = length - 1;
while (left <= right) {
 mid = left + (right - left) / 2;
 if (nums[mid] == target) {
   return mid;
 } else if(nums[mid] < target) {
   left = mid + 1;
 } else
   right = mid - 1;
 }
...
// right + 1 == left
// No more candidate
```

Template #2:

```
// Pre-processing
...
left = 0; right = length;
while (left < right) {
 mid = left + (right - left) / 2;
 if(nums[mid] < target) {
   left = mid  + 1;
 } else {
   right = mid;
 }

...
// left == right
// 1 more candidate
// Post-Processing
```

Template #3:

```
// Pre-processing
...
left = 0; right = length - 1;
while (left + 1 < right) {
 mid = left + (right - left) / 2;
 if (num[mid] < target) {
   left = mid;
 } else {
   right = mid;
 }

...
// left + 1 == right
// 2 more candidates
// Post-Processing
```

- 99% of binary search problems that you see online will fall into 1 of these 3 templates.
- Template 1 and 3 are the most commonly used and almost all binary search problems can be easily implemented in one of them.

Q2:
Implement `int sqrt(int x)`.
Compute and return the square root of *x*, where *x* is guaranteed to be a non-negative integer.
Since the return type is an integer, the decimal digits are truncated and only the integer part of the result is returned.

**Example 1:**
**Input:** 4
**Output:** 2

**Example 2:**
**Input:** 8
**Output:** 2
**Explanation:** The square root of 8 is 2.82842..., and since the decimal part is truncated, 2 is returned.

Solution:

```python
class Solution(object):
    def mySqrt(self, x):
        """
        :type x: int
        :rtype: int
        """
        l, r = 1, x
        while l <= r:
            mid = (l + r)/2
            if mid*mid == x:
                return mid
            elif mid*mid < x:
                l = mid + 1
            else:
                r = mid - 1
        return r
```

Q3:
First Bad Version:
You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.
Suppose you have `n` versions `[1, 2, ..., n]` and you want to find out the first bad one, which causes all the following ones to be bad.
You are given an API `bool isBadVersion(version)` which will return whether `version` is bad

**Example 1:**
**Input:** 8

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| good | good | good | good | bad | bad | bad | bad |

**Output:** 5

# Solution:

```python
class Solution(object):
    def firstBadVersion(self, n):
        """
        :type n: int
        :rtype: int
        """
        l, r = 1, n
        while l < r:
            mid = (l + r)/2
            if not isBadVersion(mid): #Good Version
                l = mid + 1
            else:
                r = mid
        return l
```

## Q4: Find K Closest Elements

Given a sorted array, two integers k and x, find the k closest elements to x in the array. The result should also be sorted in ascending order. If there is a tie, the smaller elements are always preferred.

**Example 1:**
**Input:** [1,2,3,4,5], k=4, x=3
**Output:** [1,2,3,4]

**Example 2:**
**Input:** [1,2,3,4,5], k=4, x=-1
**Output:** [1,2,3,4]

# Solution:

**Analysis:**
The final result should be a list of *k* consecutive elements. i.e. *arr[start: start + k]*
How to find this *start* point?
Using binary search by comparing a and b:

- a = x - arr[start]
- b = arr[start + k] – x
- If a > b: start point is too left, left->mid + 1
- If a < b: start point is too right, righr ->mid

```python
class Solution(object):
    def findClosestElements(self, arr, k, x):
        """
        :type arr: List[int]
        :type k: int
        :type x: int
        :rtype: List[int]
        """
        left, right = 0, len(arr) - k
        while left < right:
            mid = (left + right) / 2
            if x - arr[mid] > arr[mid + k] - x:
                left = mid + 1
            else:
                right = mid
        return arr[left:left + k]
```

## Q5: Search in Rotated Sorted Array

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand. (i.e., [0,1,2,4,5,6,7] might become [4,5,6,7,0,1,2]).

You are given a target value to search. If found in the array return its index, otherwise return -1.

You may assume no duplicate exists in the array.

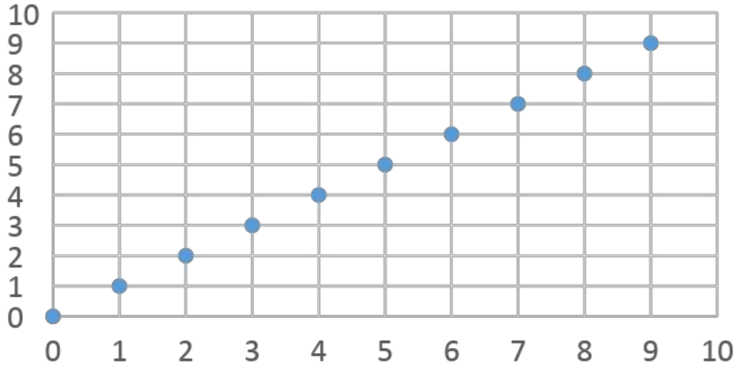
**Example 1:**
**Input:** nums = [4,5,6,7,0,1,2], target = 0
**Output:** 4

**Example 2:**
**Input:** nums = [4,5,6,7,0,1,2], target = 3
**Output:** -1

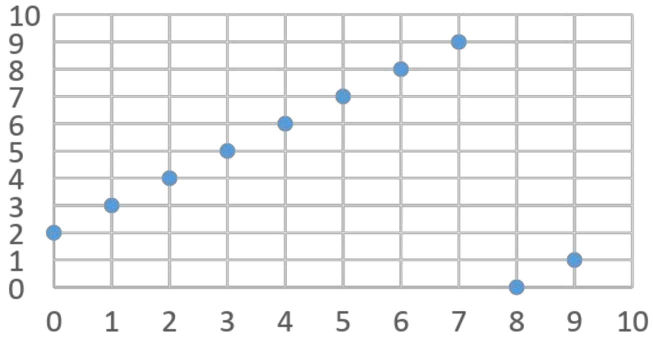
# Solution:

**Analysis:**

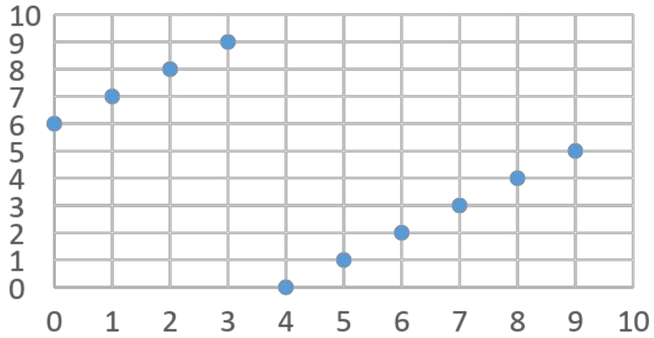
Not Rotated



Rotated 1



Rotated 2



```
class Solution(object):
    def search(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """
        low = 0
        high = len(nums) - 1
        while low <= high:
            mid = (low + high)/2
            if nums[mid] == target:
                return mid
            if nums[low] <= nums[mid]:
                if nums[low] <= target <= nums[mid]:
                    high = mid - 1
                else:
                    low = mid + 1
            else:
                if nums[mid] <= target <= nums[high]:
                    low = mid + 1
                else:
                    high = mid - 1
        return -1
```

In rotated cases: find **monotone increasing range** and **JUDGE**!!!