Question-1
https://leetcode.com/problems/binary-search/ (https://leetcode.com/problems/binary-search/)

Circular Queue
https://leetcode.com/problems/design-circular-queue/ (https://leetcode.com/problems/design-circular-queue/)

```
In [ ]:
```

DIY:
Question-1
https://leetcode.com/problems/find-peak-element/ (https://leetcode.com/problems/find-peak-element/)

Question-2
Given sorted rotated array, find point of rotation. Return the index.
4 5 1 2 3 Brute Force: iterate from start till end and find where nums[i] < nums[i+1] breaks O(N)

```
In [ ]:
```

Question-2
https://leetcode.com/problems/search-in-rotated-sorted-array/ (https://leetcode.com/problems/search-in-rotated-sorted-array/)

```
class Solution {
    public int search(int[] nums, int target) {
        int left = 0;
        int right = nums.length-1;

        while(left <= right){
            int mid = left + (right-left) /2;

            if(nums[mid] == target) return mid;

            if(nums[left] <= nums[mid]){ //left to right is sorted

                // identify if target is present in sorted part or not
                if(nums[left] <= target && target < nums[mid])
                    right = mid -1;
                else
                    left = mid +1;

            }else{ // mid to right is sorted
                if(nums[mid] < target && target <= nums[right])
                    left = mid +1;
                else
                    right = mid -1;
            }
        }
        return -1;
    }
}

arr 1 3 4 6 9
x

t=0
  4,5,6,7,0,1,2
  0 1 2 3 4 5 6
l 0 4 4
r 6 6 4
m 3 5 4
```

```
In [ ]:
```

Insertion sort

Insert an element from unsorted section of array at correct position in sorted section.

```
[] empty array
[1] array with 1 element

5 | 6 1 3 4 7 2
5 6 | 1 3 4 7 2
1 5 6 | 3 4 7 2  temp=1
1 3 5 6 | 4 7 2

Worst Case
5 4 3 2 1
1 + 2 + ... n-1 => O(N^2)

Best Case
1 2 3 4 5
1 + 1 + .... (n times) => O(N)
```

In [4]:

```
def i_sort(data): # data is array
    for i in range(1, len(data)):
        temp = data[i]
        j = i-1
        while (j >= 0 and data[j] > temp):
            data[j+1] = data[j]
            j -= 1

        data[j+1] = temp

data = [5,6,1,3,4,7,2]
i_sort(data)
print(data)

data = [4,3,2,1]
i_sort(data)
print(data)

data = [1,2,3,4]
i_sort(data)
print(data)
```

```
[1, 2, 3, 4, 5, 6, 7]
[1, 2, 3, 4]
[1, 2, 3, 4]
```

In [ ]:

In [ ]:

Selection sort

```
| 5 6 1 3 4 7 2 # find pos of min element, and swap with the first element in unsorted array
1 | 6 5 3 4 7 2
1 2 | 5 3 4 7 6
1 2 3 | 5 4 7 6
1 2 3 4 | 5 7 6
1 2 3 4 5 | 7 6
1 2 3 4 5 6 | 7

TC: O(N^2)

1 2 3 4 5 6
TC: O(N^2)

6 5 4 3 2 1
TC: O(N^2)
```

In [7]:
```
def s_sort(data):
    for i in range(len(data)): # (0.. n-1)
        min_pos = i

        # find minimum pos
        for j in range(i, len(data)):
            if data[j] < data[min_pos]:
                min_pos = j

        # swap
        t = data[min_pos]
        data[min_pos] = data[i]
        data[i] = t

data = [5,6,1,3,4,7,2]
s_sort(data)
print(data)

data = [4,3,2,1]
s_sort(data)
print(data)

data = [1,2,3,4]
s_sort(data)
print(data)
```

```
[1, 2, 3, 4, 5, 6, 7]
[1, 2, 3, 4]
[1, 2, 3, 4]
```

In [ ]:

Question-3
https://leetcode.com/problems/max-consecutive-ones-ii/ (https://leetcode.com/problems/max-consecutive-ones-ii/) : https://www.lintcode.com/problem/883/ (https://www.lintcode.com/problem/883/)

In [ ]:
~~~
```Python

def find_max_consecutive_ones(self, nums: List[int]) -> int:
    # write your code here
    left = -1
    ones = 0
    longest = 0
    for num in nums:
        if num == 1:
            ones = ones + 1
        else:
            longest = max(longest,left+ones + 1)
            left,ones = ones,0
    return longest
```
1 1 0 1 1 1 0 0 1
0 1 2 3 4 5 6 7 8

l       -1     2       3 0
o        0 1 2 0 1 2 3 0 0 1
longest  0   2 2       6 6
~~~

```
**Question-4**
https://leetcode.com/problems/max-consecutive-ones-iii/
```

~~~
```Java
k = 2
[1,1,1,0,0,0,1,1,1,1,0]
 0 1 2 3 4 5 6 7 8 9 10
s=0          4 4 4 4 4
e=0 1 2 3 4 5 5 6 7 8 9
z=0     1 2 3 2 2 2 2 2
m=1 2 3 4 5   5       6

 class Solution {
    public int longestOnes(int[] nums, int k) {
        int start=0;

    int zeroCount=0;
    int maxConsecutiveOne=0;
    for(int end =0;end<nums.length;end++) {
        if(nums[end]==0){
~~~

```
            zeroCount++;
        }

        // Fix the start pointer once we have more than k zeros in the current sliding window.
        while(zeroCount>k){
            if(nums[start]==0){
                zeroCount--;
            }
            start++;
        }

        maxConsecutiveOne=Integer.max(maxConsecutiveOne,end-start+1);
    }

    return maxConsecutiveOne;
    }
}
TC: O(N)
SC: O(1)
```

In [ ]:

```
public int longestOnes(int[] A, int K) {
       int i = 0, j;
       for (j = 0; j < A.length; ++j) {
            if (A[j] == 0) K--;
            if (K < 0 && A[i++] == 0) K++;
       }
      return j - i;
}

[1,1,1,0,0,0,1,1,1,1,0]
       - - -         -

j 0
i 0
k 2
```

In [ ]:

```
        i = 0
        for j in range(len(nums)):
            if nums[j]==0:
                k -= 1
            if k<0:
                if  nums[i]==0:
                    k += 1
                i += 1
        return j-i+1
```