Understanding the problem statement

Given an array, we have to rotate the array of size n by d elements. We need to perform left rotation. For example given an arr=[1,2,3,4,5,6] and d=2, rotated array should look like output : [3,4,5,6,1,2]

Approach 1

1. Create a temporary array of size d.
2. Copy first d elements from given array to temporary array.
3. Shift remaining (n-d) elements to beginning of array arr[0] = arr[d+1] arr[1] = arr[d+2] ...
4. Then copy elements from temporary array to given array at end. arr[d+1] = temp_arr[0] arr[d+2] = temp_arr[1] ..
5. We are iterating the entire array once and temporary array of size d once. Time Complexity : O(n) Space Complexity : O(d) for temporary array

Time Complexity : O(n)

Space Complexity : O(d)

Implementation

In [3]:
```
def rotateArray(arr,size,d):
    temp_arr = []
    for i in range(d):
        temp_arr.append(arr[i])
    pos = 0
    for i in range(d,size):
        arr[pos] = arr[i]
        pos+=1
    pos = 0
    for i in range(size-d,size):
        arr[i] = temp_arr[pos]
        pos+=1
    return arr
```

In [13]:
```
arr = [1,2,3,4,5,6]
size = len(arr)
d = 1
print(rotateArray(arr,size,d))
# ans
# [2, 3, 4, 5, 6, 1]
```

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

Approach 2

1. Copy first element to temporary variable
2. Shift all the elements of given array by one position left.
3. Copy element from temporary variable to the end of given array.
4. Repeat the process for d times to rotate the array by d elements.

5. We rotate the n elements, O(n) and repeat this d times. So total time complexity is O(nd). We are using only one variable irrespective of the size of array. So Space Complexity is O(1)

Time Complexity : O(n * d)

Space Complexity : O(1)

Implementation

In [10]:
```
def rotateArrOnce(arr,size):
    temp = arr[0]
    for i in range(1,size):
        arr[i-1] = arr[i]
    arr[size-1] = temp
    return arr

def rotateArray2(arr,size,d):
    for i in range(d):
        arr = rotateArrOnce(arr,size)
    return arr
```

In [14]:
```
arr = [1,2,3,4,5,6]
size = len(arr)
d = 2
print(rotateArray2(arr,size,d))
# ans
# [3, 4, 5, 6, 1, 2]
```

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

Approach 3

```
This is also called juggling algorithm for array rotation
```

1. Find the gcd of size and d i.e g size = 6, d = 2, gcd(6,2) = 2
2. Divide the given array into multiple blocks of size g. blocks [1,2] [3,4] [5,6]
3. From first block copy first element to temporary variable. temp = 1
4. From second block copy first element to first element of first block. Copy first element from third block to first element of second block repeat the process for all blocks. Copy temp to first element of last block [3,2] [5,4] [1,6]
5. Similarly repeat the process for other elements of blocks i.e second element temp = 2 [3,4] [5,6] [1,2]
6. We are scanning each element once, so time complexity becomes O(n). We are using only one variable irrespective of size of array, so space complexity is O(1)

Time Complexity : O(n)

Space Complexity : O(1)

Implementation

In [20]:
```
def gcd(size,d):
    if d == 0:
        return size
    else:
        return gcd(d,size%d)

def rotateArray3(arr,size,d):
    block_size = gcd(size,d)
    for i in range(block_size):
        temp = arr[i]
        j = i
        while(1):
            k = (j+d)%size # for cyclic increment
            if k == i:
                break # break when d reaches i. cycle done
            arr[j] = arr[k]
            j = k
        arr[j] = temp
    return arr
```

In [21]:
```
arr = [1,2,3,4,5,6]
size = len(arr)
d = 2
print(rotateArray3(arr,size,d))
# ans
# [3, 4, 5, 6, 1, 2]
```

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

In [22]:
```
arr = [1,2,3,4,5,6]
size = len(arr)
d = 3
print(rotateArray3(arr,size,d))
# ans
# [4, 5, 6, 1, 2, 3]
```

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

In [24]:
```
arr = [1,2,3,4,5,6,7]
size = len(arr)
d = 2
print(rotateArray3(arr,size,d))
# ans
# [3, 4, 5, 6, 7, 1, 2]
```

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

Approach 4

1. First reverse the first d elements [1,2,3,4,5,6] --> [2,1,3,4,5,6]
2. Next reverse the remaining (n-d) elements [2,1,3,4,5,6] --> [2,1,6,5,4,3]
3. Reverse the complete resulting array [3,4,5,6,1,2]
4. We are scanning the entire array twice. O(2n) and we are not using any extra space O(1)

Time Complexity : O(n)

Space Complexity : O(1)

Implementation

In [34]:
```
def revr_list(arr,index1,index2):
    while(index1<=index2):
        temp = arr[index1]
        arr[index1] = arr[index2]
        arr[index2] = temp
        index1+=1
        index2-=1
    return arr

def rotateArray4(arr,size,d):
    arr = revr_list(arr,0,d-1)
    arr = revr_list(arr,d,size-1)
    arr = revr_list(arr,0,size-1)
    return arr
```

In [35]:
```
arr = [1,2,3,4,5,6,7]
size = len(arr)
d = 2
print(rotateArray4(arr,size,d))
# ans
# [3, 4, 5, 6, 7, 1, 2]
```

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

In [36]:
```
arr = [1,2,3,4,5,6]
size = len(arr)
d = 3
print(rotateArray4(arr,size,d))
# ans
# [4, 5, 6, 1, 2, 3]
```

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

In [38]:
```
arr = [1,2,3,4,5,6]
size = len(arr)
d = 1
print(rotateArray4(arr,size,d))
# ans
# [2, 3, 4, 5, 6, 1]
```

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

In [ ]: