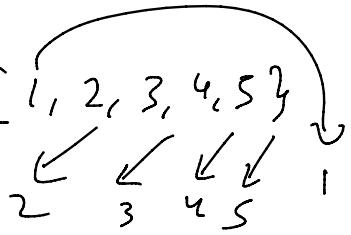
2. Left Rotate the Array by one

arr[] = [1, 2, 3, 4, 5] d = 1

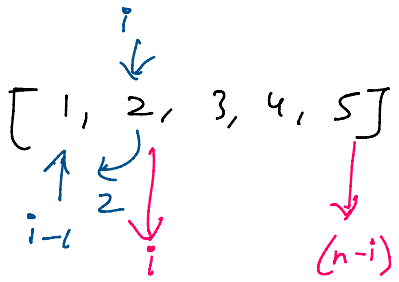
O/P ⇒ [2, 3, 4, 5, 1]

Optimal Approach

arr[] = {1, 2, 3, 4, 5}



→ Store arr[0] = temp



→ Now place ith index at (i-1) index

~~temp~~ = arr[0]
1

```
for(i=1; i<n; i++)
{
    arr[i-1] = arr[i]
}
arr[n-1] = temp //at the last place
```

T.C ⇒ O(N) S.C ⇒ O(1) // in the algorithm

T.C ⇒ O(N) S.C ⇒ O(1) // in the algorithm

Extra S.C ⇒ O(N)

Problem Statement Suggest Edit

Rajesh loves Supriya and wants to propose to Supriya but Supriya is an Array lover and asked Rajesh a problem which is as follows:-
Given an array "ARR' containing 'N' elements, rotate this array Left by once means to shift all elements by one place to the left and move the first element to the last position in the array.
As Rajesh is very bad at array he asks you to help him as he loves her so much.
Example:

```
Input: 'N' = 5, 'ARR' = [1, 2, 3, 4, 5]

Output: [2, 3, 4, 5, 1]

We moved the 2nd element to the 1st position and 3rd element to the 2nd position and 4th element to the 3rd position and 5th element to the 4th position and move 0th element to the 5th position.
```

```cpp
#include <bits/stdc++.h>
vector<int> rotateArray(vector<int>& arr, int n) {
    int temp = arr[0];
    for(int i = 1; i < n; i++){
        arr[i-1] = arr[i];
    }
    arr[n-1] = temp;
    return arr;
}
```

3. Left Rotate the Array by K times

arr[] = {1, 2, 3, 4, 5, 6, 7}

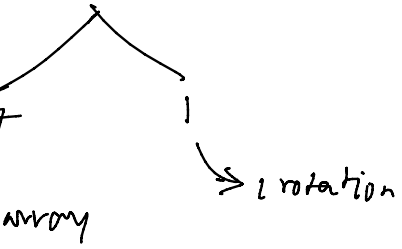
o/p ⇒ [3, 4, 5, 6, 7, 1, 2] d = 2

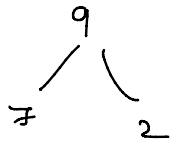
o/p ⇒ d = 3 [4, 5, 6, 7, 1, 2, 3] d = 3

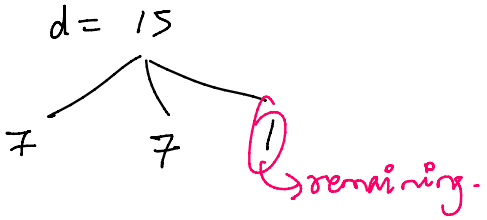
if d = 7 arr[] = {1, 2, 3, 4, 5, 6, 7}

the same original array as array size and k is same.

now for d = 8







So, the no of rotation is always d = d % N

For eg

d = 20 = 20 % 7

= 6 (So you need to '6' times rotation)

Brute force

arr[] = {1, 2, 3, 4, 5, 6, 7} d = 3

temp[] = [1, 2, 3]

Shifting

0 1 2 3 4 5 6

arr[] = {1, 2, 3, 4, 5, 6, 7}

4, 5, 6, 7, 1, 2, 3

d = 3

i → i − d → Shifting

3 → 0

4 → 1

5 → 2

```
for(i=d; i<n; i++)
{
    a[i-d] = a[i]
}
```

temp = [1, 2, 3]

Put Back temp

```
j=0
for(i=n-k; i<n; i++)
{
    a[i] = temp[j]
    j++
}
```

instead of 'j'

OR ⇒

put back temp

n − k

7 − 3 = 4th index

	0	1	2	3	4	5	6
arr	1,	2,	3,	4,	5,	6,	7
temp = [1,	2,	3)					
	0	1	2				

(n − d)

n − d + 1

n + d + 2

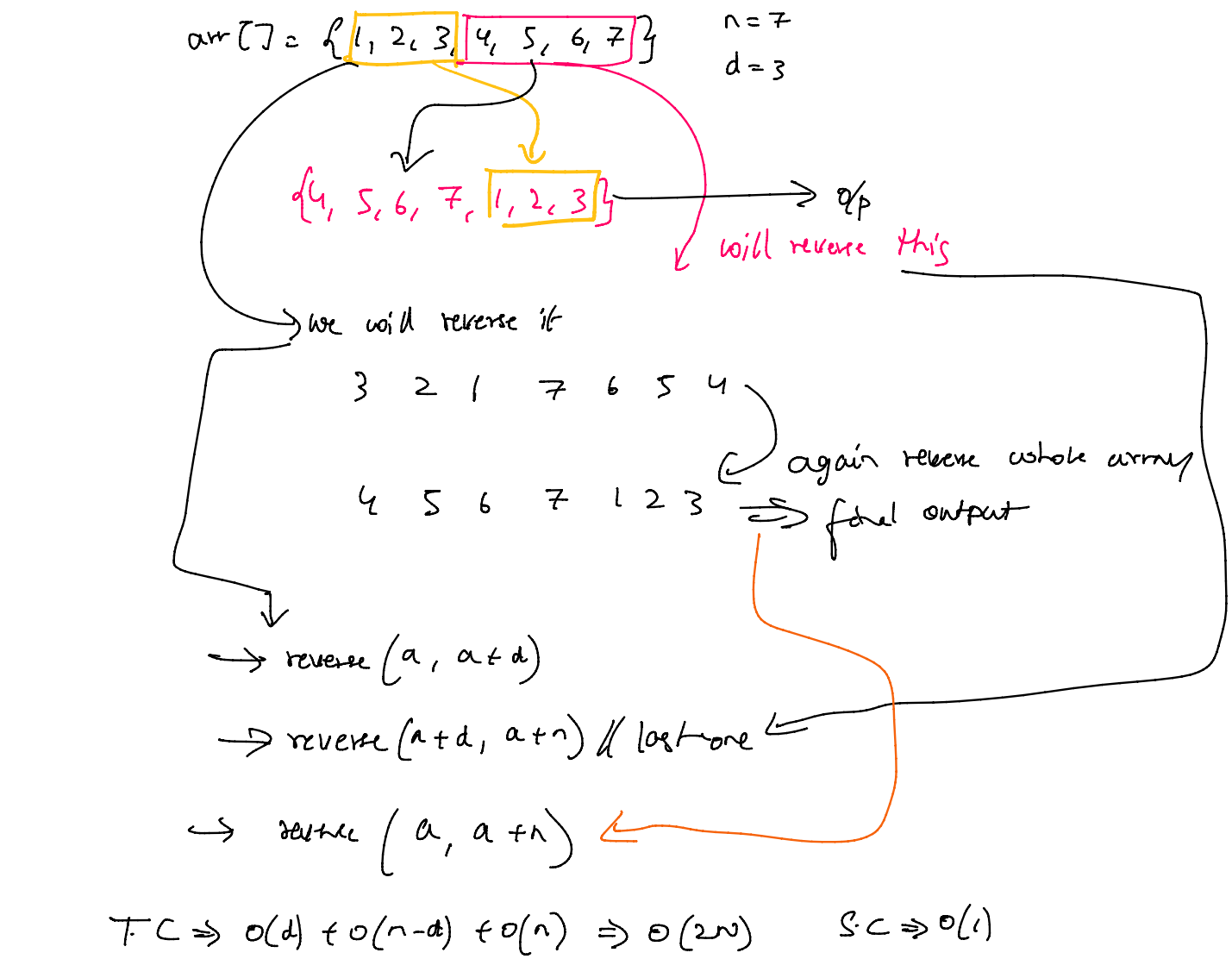
So

```
for(i=n-d; i<n; i++)
{
    a[i] = temp[i-(n-d)];
}
```

T.C ⇒ O(d) + O(n − d) + O(d) ⇒ O(n + d)

Extra Space ⇒ O(d)

Optimial Approach



T.C ⇒ O(d) + O(n-d) + O(n) ⇒ O(2n) S.C ⇒ O(1)

Right Shift code.

4. Linear Search

arr [ ] = [6, 4, 8, 4, 1] nums = 4

0 1 2 3 4

→ return '3'.

```
for (i = 0 ⟶ i++) {
    if (arr[] == num)
        return i
}
```

5. Unios of two sorted arrays.

arr1[] = { 1, 1, 2, 3, 4, 5 } arr2[] = { 2, 3, 4, 4, 5, 6 }

union [] ⇒ { 1, 2, 3, 4, 5, 6 } ⇒ output

unique → map or set

→ iterate over two array.

→ Set | 1 | 2 | 3 | 4 | 5 | 6 |

→ Create a union array of size = = set. and put them into array.

union [] = { 1, 2, 3, 4, 5, 6 }

Don't use unordered set ⇒ The ordering will get disorted.

Step1

```
set<int> st
for (i=0; i<n1; i++)
    st.insert[a1[i]
for (i=0 → n2)
    st.insert(a2[i])
```

Step2

```
union[st.size()]
i=0
for (auto it : st)
union[i++] = it:
union.add(it).
```

T.C ⇒ O(n1 log n) + O(n2 log n) + O(n1+n2)

S.C ⇒ O(n1 + n2) + O(n1 + n2)

↳ to 'return the answer

```cpp
#include <bits/stdc++.h>

vector < int > sortedArray(vector < int > a, vector < int > b) {
    int n1 = a.size();
    int n2 = b.size();

    set<int> st;
    for(int i = 0; i < n1; i++){
        st.insert(a[i]);
    }

    for(int i = 0; i < n2; i++){
        st.insert(b[i]);
    }

    vector<int> temp;
    for(auto it: st){
        temp.push_back(it);
    }
    return temp;
}
```

2nd Approach

two pointer approach

arr1[] = { 1, 1, 2, 3, 4, 5 }

arr2[] = { 2, 3, 4, 4, 5, 6 }

union [] = { 1, 2, 3, 4, 5, 6 }

smallest element

T.C $\Rightarrow O(n_1 + n_2)$ S.C $\Rightarrow O(n_1 + n_2)$

↳ for returning the answer

6. Intersection of two sorted array

→ present in both array

A[] = {1, 2, 2, 3, 3, 4, 5, 6}

B[] = {2, 3, 3, 5, 6, 6, 7}

O/P ⇒ {2, 3, 3, 5, 6}

Brute force → Pick one element

→ Track

→ Created a visited array and mark them '0'

vis[] = {0, 0, 0, 0, 0, 0, 0}

→ take first element from arr[a] and check if the element is there in arr[b]. if not found move to next element.

{2, 3, 3, 5, 6}

Pseudo Code :

```
ans → []        vis[n2] = {0}

for(i=0 ⟶ n1)
  for(j=0 ⟶ n2)
  {
    if(a[i] == b[j] && vis[j] == 0)
    {
      ans.add(a[i])
      vis[j] = 1;
```

break

}

if(b[j] > a[i]) break; // truncate the loop.

T.C ⇒ O(n1 + n2) S.C ⇒ O(n2)

```cpp
#include <bits/stdc++.h>
vector<int> findArrayIntersection(vector<int> &A, int
vector<int> &B, int m)
{
    vector<int> ans;
    int vis[m] = {0};
    for(int i = 0;i<n;i++) {
        for(int j = 0;j<m;j++) {
            if(A[i] == B[j] && vis[j] == 0) {
                ans.push_back(A[i]);
                vis[j] = 1;
                break;
            }

            if(B[j] > A[i]) break;
        }
    }
    return ans;
}
```

2nd Approach (2 pointer)

A[] = { 1, 2, 2, 3, 3, 4, 5, 6 } (i i i i i i i i)

B[] = { 2, 3, 3, ~~5~~, ~~6~~, 6, 7 } (j j j j j)

ans → { 2, 3, 3, 5, 6 } (Output)

→ wherever both pointer push them into ans, and move both.

→ if they don't match the just move (i).

```cpp
#include <bits/stdc++.h>
vector<int> findArrayIntersection(vector<int> &arr1, int n, vector<int> &arr2, int m)
{
    // Write your code here.
    int i = 0;
    int j = 0;
    vector<int> ans;
    while(i<n && j<m){
        if(arr1[i] < arr2[j]){
            i++;    // for arr1 don't have partner so move
        }
        else if(arr2[j] < arr1[i]) {
            j++; // same for arr2 it don't have partner so move
        }
        else {
            // if both match then add.
            ans.push_back(arr1[i]);
            i++;
            j++;
        }
    }
    return ans;
}
```