Data Structures and Algo in Java - Day 11

1. Left Rotate by 1;
2. Left Rotate by D places;
3. Right Rotate by D places;
4. Move Zeros to end;
5. Linear Search;
6. Find Union;
7. Find Intersection;

Worked on these problems and understood both Brute Force Approach and Optimal Approach.

import java.util.\*;

public class arrayss

{

public static void main(String[] args)

{

int arr1 [] = {1,3,5,7,9,10,2,1,3,5,7};

int arr2 [] = {1,2,3,4,5,6,11,12,0,1,2};

int d = 3;

intersectionOptimal(arr1,arr2);

}

public static void left(int arr [])

{

int n = arr.length;

int temp = arr[0];

for(int i=1;i<n;i++)

{

arr[i-1] = arr[i];

}

arr[n-1]= temp;

System.out.println(Arrays.toString(arr));

}

public static void leftd(int arr[],int d)

{

int n=arr.length;

for(int i=0;i<d;i++)

{

int temp = arr[0];

for(int j=1;j<n;j++)

{

arr[j-1]=arr[j];

}

arr[n-1]=temp;

}

System.out.println(Arrays.toString(arr));

}

public static void rightd(int arr[],int d)

{

int n=arr.length;

for(int i=0;i<d;i++)

{

int temp = arr[n-1];

for(int j=n-1;j>0;j--)

{

arr[j]=arr[j-1];

}

arr[0]=temp;

}

System.out.println(Arrays.toString(arr));

}

public static void leftDTimeOptimized(int arr[],int d)

{

int n=arr.length;

reverse(arr,0,d-1);

reverse(arr,d,n-1);

reverse(arr,0,n-1);

System.out.println(Arrays.toString(arr));

}

public static void reverse(int arr[],int start, int end)

{

while(start<end)

{

int temp = arr[start];

arr[start]=arr[end];

arr[end]=temp;

start++;

end--;

}

}

public static void rightDTimeOptimized(int arr[],int d)

{

int n=arr.length;

reverse(arr,0,n-1);

reverse(arr,0,d-1);

reverse(arr,d,n-1);

System.out.println(Arrays.toString(arr));

}

public static void moveNonZeros(int arr[])

{

int n=arr.length;

for(int i=0;i<n;i++)

{

for(int j=0;j<n;j++)

{

if(arr[j]==0 && arr[i]!=0)

{

int temp = arr[j];

arr[j]=arr[i];

arr[i]=temp;

}

}

}

System.out.println(Arrays.toString(arr));

}

public static void moveNonZerosOptimal(int arr[])

{

int n = arr.length;

int j=-1;

for(int i=0;i<n;i++)

{

if(arr[i]==0)

{

j=i;

break;

}

}

for(int i=j+1;i<n;i++)

{

if(arr[i]!=0)

{

int temp = arr[i];

arr[i]=arr[j];

arr[j]=temp;

j++;

}

}

System.out.println(Arrays.toString(arr));

}

public static void union(int arr1[],int arr2[])

{

int n = arr1.length;

int m = arr2.length;

Set<Integer> set = new HashSet<>();

for(int i=0;i<n;i++)

{

set.add(arr1[i]);

}

for(int j=0;j<m;j++)

{

set.add(arr2[j]);

}

System.out.println(set);

}

public static void unionOptimal(int arr1[],int arr2[])

{

Set<Integer> set = new HashSet<>();

for(int num:arr1)

{

set.add(num);

}

for(int num:arr2)

{

set.add(num);

}

System.out.println(set);

}

public static void intersection(int arr1[],int arr2[])

{

int n = arr1.length;

int m = arr2.length;

Set<Integer> set = new HashSet<>();

for(int i=0;i<n;i++)

{

for(int j=0;j<m;j++)

{

if(arr1[i]==arr2[j])

{

set.add(arr1[i]);

}

}

}

System.out.println(set);

}

public static void intersectionOptimal(int arr1[],int arr2[])

{

Set<Integer> set = new HashSet<>();

Set<Integer> intersection = new HashSet<>();

for(int num:arr1){

set.add(num);

}

for(int num:arr2){

if(set.contains(num)){

intersection.add(num);

}

}

System.out.println(intersection);

}

}