**#Advance DSA** *Assignment\_1.*

1. Find the largest element in a given array.

Ans:- public class Main {

public static int findLargestElement(int[] arr) {

if (arr.length == 0) {

throw new IllegalArgumentException("Array is empty");

}

int largest = arr[0];

for (int num : arr) {

if (num > largest) {

largest = num;

}

}

return largest;

}

public static void main(String[] args) {

int[] array = {3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5};

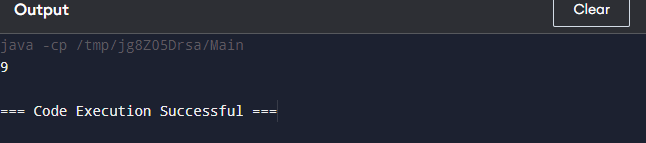
System.out.println(findLargestElement(array)); // Output: 9

}

}

Time Complexity : O(n);

Space Complexity : O(1);



1. Reverse a given array.

Ans :- 2. public class reverseArray {

static void reverse(int a[], int n)

{

int[] b = new int[n];

int j = n;

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

b[j - 1] = a[i];

j = j - 1;

}

System.out.println("Reversed array is: \n");

for (int k = 0; k < n; k++) {

System.out.println(b[k]);

}

}

public static void main(String[] args)

{ int [] arr = {10, 20, 30, 40, 50};

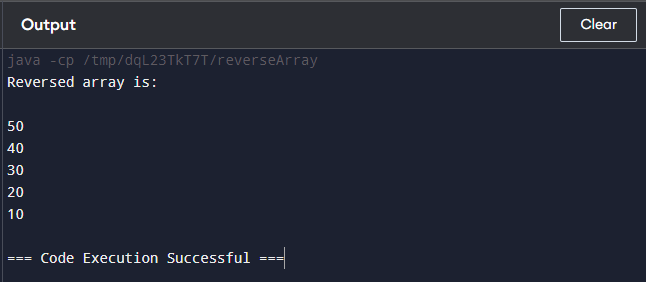
reverse(arr, arr.length);

}

}

Time Complexity : O(n);

Space Complexity : O(1);



1. Find the second largest element in a given array.

Ans :- public class SecondLargest {

public static int findSecondLargest(int[] arr) {

if (arr.length < 2) {

throw new IllegalArgumentException("Array should contain at least two elements.");

}

int largest = Integer.MIN\_VALUE;

int secondLargest = Integer.MIN\_VALUE;

for (int num : arr) {

if (num > largest) {

secondLargest = largest;

largest = num;

} else if (num > secondLargest && num < largest) {

secondLargest = num;

}

}

if (secondLargest == Integer.MIN\_VALUE) {

throw new IllegalArgumentException("There is no second largest element.");

}

return secondLargest;

}

public static void main(String[] args) {

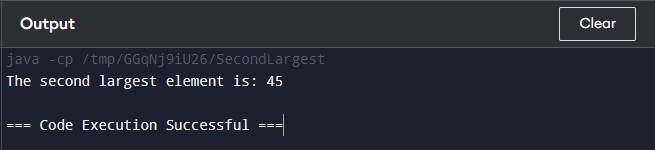
int[] arr = {10, 20, 4, 45, 99};

System.out.println("The second largest element is: " + findSecondLargest(arr));

}

}

Time Complexity : O(n);

Space Complexity : O(1); 

1. Check if a given array is sorted.

Ans :- public class ArraySortCheck {

public static boolean isSorted(int[] arr) {

if (arr.length < 2) {

return true;

}

boolean ascending = true;

boolean descending = true;

for (int i = 1; i < arr.length; i++) {

if (arr[i] < arr[i - 1]) {

ascending = false;

}

if (arr[i] > arr[i - 1]) {

descending = false;

}

}

return ascending || descending;

}

public static void main(String[] args) {

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

int[] arr2 = {5, 4, 3, 2, 1};

int[] arr3 = {1, 3, 2, 4, 5 };

System.out.println("Array 1 is sorted: " + isSorted(arr1));

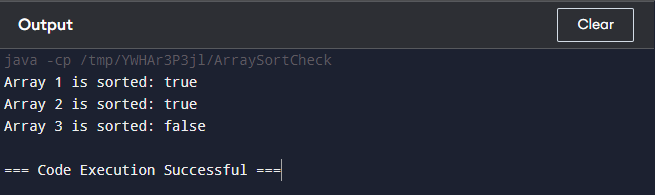
System.out.println("Array 2 is sorted: " + isSorted(arr2));

System.out.println("Array 3 is sorted: " + isSorted(arr3));

}

}

Time Complexity : O(n);

Space Complexity : O(1); 

1. Remove duplicates from a given array.

Ans :- import java.util.HashSet;

import java.util.Set;

public class RemoveDuplicates {

public static int[] removeDuplicates(int[] arr) {

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

for (int num : arr) {

uniqueElements.add(num);

}

int[] result = new int[uniqueElements.size()];

int i = 0;

for (int num : uniqueElements) {

result[i++] = num;

}

return result;

}

public static void main(String[] args) {

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

int[] result = removeDuplicates(arr);

System.out.println("Array without duplicates:");

for (int num : result) {

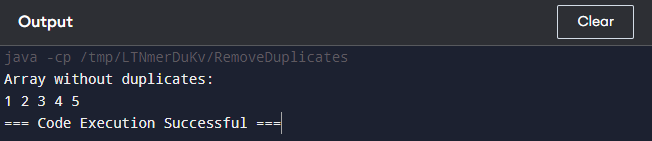
System.out.print(num + " ");

}

}

}

Time Complexity : O(n);

Space Complexity : O(n);

1. Rotate a given array.

Ans :- public class RotateArray {

public static void rotate(int[] arr, int k) {

int n = arr.length;

k = k % n;

reverse(arr, 0, n - 1);

reverse(arr, 0, k - 1);

reverse(arr, k, n - 1);

}

private 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 main(String[] args) {

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

int k = 3; // Rotate right by 3 positions

rotate(arr, k);

System.out.println("Array after rotation:");

for (int num : arr) {

System.out.print(num + " ");

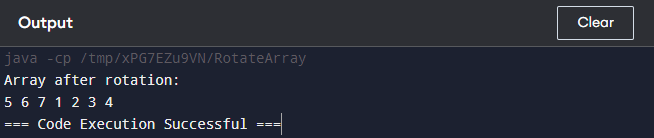
}

}

}

Time Complexity : O(n);

Space Complexity : O(1);



1. Find the frequency of elements in a given array.

Ans :- import java.util.HashMap;

import java.util.Map;

public class ElementFrequency {

public static void findFrequency(int[] arr) {

Map<Integer, Integer> frequencyMap = new HashMap<>();

for (int num : arr) {

frequencyMap.put(num, frequencyMap.getOrDefault(num, 0) + 1);

}

for (Map.Entry<Integer, Integer> entry : frequencyMap.entrySet()) {

System.out.println("Element " + entry.getKey() + " occurs " + entry.getValue() + " times.");

}

}

public static void main(String[] args) {

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

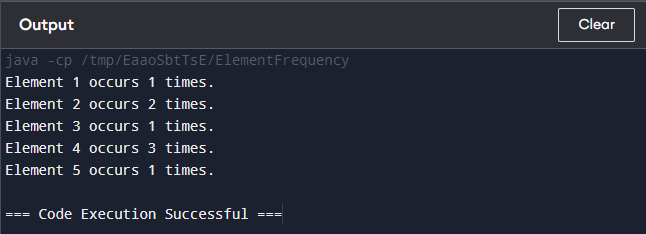
findFrequency(arr);

}

}

Time Complexity : O(n);

Space Complexity : O(m);



1. Merge two sorted arrays.

Ans :- public class MergeSortedArrays {

public static int[] merge(int[] arr1, int[] arr2) {

int n1 = arr1.length;

int n2 = arr2.length;

int[] result = new int[n1 + n2];

int i = 0, j = 0, k = 0;

while (i < n1 && j < n2) {

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

result[k++] = arr1[i++];

} else {

result[k++] = arr2[j++];

}

}

while (i < n1) {

result[k++] = arr1[i++];

}

while (j < n2) {

result[k++] = arr2[j++];

}

return result;

}

public static void main(String[] args) {

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

int[] arr2 = {2, 4, 6, 8};

int[] mergedArray = merge(arr1, arr2);

System.out.println("Merged array:");

for (int num : mergedArray) {

System.out.print(num + " ");

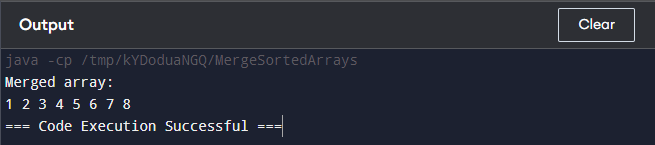
}

}

}

Time Complexity : O(n1 + n2);

Space Complexity : O(n1 + n2);



*By:- Abhinaw singh*

*22scse1011850*