
# Kunal Kushwaha DSA Bootcamp Sorting

### Bubble Sort -

Bubble sort works on the repeatedly swapping of adjacent elements until they are not in the intended order .

```
import java.util.Arrays;

public class Sorting {
    public static void main(String[] args) {
        int nums[] = { -34, -78, 0, -90, 1, 45, 9, 2 };
        int[] res = Bubblesort(nums);
        System.out.println(Arrays.toString(res));
    }

    public static int[] Bubblesort(int nums[]) {
        int n = nums.length;
        boolean swap;
        for (int i = 0; i < n; i++) {
            swap = false;
            for (int j = 1; j < n - i; j++) {
                if (nums[j] < nums[j - 1]) {
                    int temp = nums[j];
                    nums[j] = nums[j - 1];
                    nums[j - 1] = temp;
                    swap = true;
                }
            }
            if (swap == false)
                break;
        }
        return nums;
    }

}
```

### Selection Sort -

the smallest/largest value among the unsorted elements of the array is selected in every pass and inserted to its appropriate position into the array . Here in the code , I have sorted the largest element to the end of the remaining array .

```
import java.util.Arrays;

public class Sorting {
    public static void main(String[] args) {
        int nums[] = { 7, 3, 5, 1, 9 };
        int[] res = Selectionsort(nums);
        System.out.println(Arrays.toString(res));
    }

    public static int[] Selectionsort(int nums[]) {

        for (int i = 0; i < nums.length; i++) {
            int last = nums.length - i - 1;
            int max = maxIndex(nums, 0, last);
            int temp = nums[last];
            nums[last] = nums[max];
            nums[max] = temp;

        }
        return nums;

    }

    public static int maxIndex(int[] nums, int start, int end) {
        int max = start;
        for (int i = start; i <= end; i++) {
            if (nums[max] < nums[i])
                max = i;
        }
        return max;
    }
}
```

## Insertion sort -

It sorts the elements on the LHS with each iteration .

```
import java.util.Arrays;

public class Sorting {
    public static void main(String[] args) {
        int nums[] = { 7, 2 };
        int[] res = Insertionsort(nums);
        System.out.println(Arrays.toString(res));
    }

    public static int[] Insertionsort(int nums[]) {
        int n = nums.length;
        for (int i = 0; i <= n - 2; i++) { // i<=n-2 because for i = 0 , numbers upto index 1 are sorted ,..... for
                                           // index 3 ,numbers upto index 4 are sorted i.e. i<=(5-2)
            for (int j = i + 1; j > 0; j--) {
                if (nums[j] < nums[j - 1]) {
                    int temp = nums[j];
                    nums[j] = nums[j - 1];
                    nums[j - 1] = temp;
```

```
                } else {
                    break; // if element at j is not smaller than j-1 , it means that the LHS is sorted
                }

            }
        }
        return nums;

    }

}
```

### Cycle Sort -

Works for arrays having first n integers . (1,2,3,4,5,……n)

```
import java.util.Arrays;

public class Sorting {
    public static void main(String[] args) {
        int nums[] = { 4, 1, 5, 3, 2 };
        int[] res = Cyclesort(nums);
        System.out.println(Arrays.toString(res));
    }

    public static int[] Cyclesort(int nums[]) {
        int n = nums.length;
        int i = 0;
        while (i < n) {
            int index = nums[i] - 1; // index = correct index of the number , ex , for 4 index = 3 .
            if (nums[i] != nums[index]) {
                // swap
                int temp = nums[index];
                nums[index] = nums[i];
                nums[i] = temp;

            } else
                i++;
        }
        return nums;
    }

}
```

## 268. Missing Number

```
class Solution {
    public int missingNumber(int[] nums) {
    int n = nums.length;
        int i = 0;
        while (i < n) {
            int index = nums[i];  // index should be the same as the element since arrays tarts from 0
            if (nums[i] < n && nums[i] != nums[index]) { // element = nums.length should be ignored.Hence,the first condition
                int temp = nums[index];
                nums[index] = nums[i];
```

```
                nums[i] = temp;
            } else
                i++;
        }

        for (int j = 0; j < n; j++) {
            if (nums[j] != j) {
                return j;
            }

        }
        return n;

    }
}
```

Note : at the end of swap function , the largest number will  be at the index of the missing number. hence , that index will be returned.

## 448. Find All Numbers Disappeared in an Array

```
class Solution {
    public List<Integer> findDisappearedNumbers(int[] nums) {

        int i =0 ;
        while(i<nums.length){
            int index = nums[i] -1;

            if(nums[i]!=nums[index]){
                int temp = nums[index] ;
                nums[index] = nums[i] ;
                nums[i] = temp;
            }
            else
                i++;
        }

        ArrayList<Integer> list = new ArrayList<>();

        for ( int j =0 ;j< nums.length ; j++){
            if(nums[j]!=j+1)
                list.add(j+1) ;
        }
        return list  ;
    }
}
```

## 287. Find the Duplicate Number

```
class Solution {
    public int findDuplicate(int[] nums) {
  int i = 0;
        while (i < nums.length) {
```

```
            int index = nums[i] - 1;

            if (nums[i] != nums[index]) {
                int temp = nums[index];
                nums[index] = nums[i];
                nums[i] = temp;
            } else
                i++;
        }

        return nums[nums.length - 1];
}
}
```

Note: the repeated number will always be at the last index.

***KK's code -***

Note - here an extra check is made -

if the last element is also present at the index where it is supposed to be found , it is the answer .

```
public class FindDuplicate {
    public int findDuplicate(int[] arr) {
        int i = 0;
        while (i < arr.length) {

            if (arr[i] != i + 1) {
                int correct = arr[i] - 1;
                if (arr[i] != arr[correct]) {
                    swap(arr, i , correct);
                } else {
                    return arr[i];
                }
            } else {
                i++;
            }
        }
        return -1;
    }

    static void swap(int[] arr, int first, int second) {
        int temp = arr[first];
        arr[first] = arr[second];
        arr[second] = temp;
    }
}
```

## 442. Find All Duplicates in an Array

```
class Solution {
    public List<Integer> findDuplicates(int[] nums) {

        int n = nums.length  ;
        int i =0 ;
```

```
        while(i<n){
            int index = nums[i]-1 ;
            if(nums[i]!=nums[index]){
                int temp = nums[index] ;
                nums[index] = nums[i] ;
                nums[i] =temp ;
            }
            else
                i++  ;
        }

        ArrayList<Integer> list = new ArrayList<>() ;
        for( int j =0 ;j<n;j++) {
            if(nums[j]!=j+1){
                list.add(nums[j]) ;
            }
        }
        return list ;
    }
}
```

## 645. Set Mismatch

```
class Solution {
    public int[] findErrorNums(int[] nums) {
        int[] res = new int[2] ;
        int i =0 ;
        while(i<nums.length){
            int index = nums[i] -1 ;

            if(nums[i] != nums[index]){
                int temp  = nums[index] ;
                nums[index] = nums[i] ;
                nums[i] = temp ;
            }
            else
                i++ ;
        }

        for( int j =0; j<nums.length ; j++){
            if(nums[j]!=j+1){
                res[0]  = nums[j];
                res[1] =j+1;
            }
        }

        return res ;
    }
}
```

## 41. First Missing Positive

```
class Solution {
    public int firstMissingPositive(int[] nums) {
   // sorting the whole array in increasing order .
        int i = 0;
        while (i < nums.length) {
            int index = nums[i]-1 ;
```

```
            if (nums[i] > 0 && nums[i] <= nums.length && nums[i] != nums[index]) {
                int temp = nums[index];
                nums[index] = nums[i];
                nums[i] = temp;
            } else
                i++;
        }

//the first element that is not present at its position i the answer .

        for (int j = 0; j < nums.length; j++) {
            if (nums[j] != j+1 )
                return (j+1);

        }
        return nums.length+1;

    }
}
```