Assignment Questions 5 to 7

PAGE NO.
DATE:

Q.1. Topic :- lecture 3 - Array

* The solution is using a Brute force approach.

Ans:-

```
import java.util.*;
public class ThreeSumClosest {
  public static int threeSumClosest
                (int[] nums, int target) {
  Array.sort(nums);
  int closestSum = num[0] + nums[i] + num[2];

  for (int i=0; i<num.length-2; i++) {
    int left = i+1;
    int right = num.length-1;

  while (left< right) {
  int sum = nums[i] + nur[left] + nums[right];

  if (sum == target) {
    return sum;
  }
  if (Math.abr(sum-target) < Math.abr
                (closestSum-target)
  { {   closestSum = sum;

    }
  if (sum < target) {
    left ++;
  { else {
    right--;
  {
 {
}
```

```
    return closestSum;
}
public static void main (String [] args)
{
    int [] num = {-1, 2, 1, -4};
    int target = 1;
    int closestSum = threeSumClosest
                    (nums, target);
    System.out.println
        ("Closest sum: " + closestsum);
}
}
```

Output

Closest sum : 2

Q.2: Brute Approach

Ans:-

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class fourSum {
    public static List<List<Integer>
        fourSum(int[] nums, int target) {
    List<List<Integer>> result = new ArrayList
                                <>();
    Arrays.sort(nums);
    for (int i=0; i< nums.length-3; i++) {
    if(i>0 && nums[i] == nums[i-1])
        continue;
```

```
for (int j = i+1; j < nums.length-2; j++)
    {
    if (j > i+1 && nums[j] == nums[j-1])
        continue;
    int left = j+1;
    int right = num.length-1;
    while (left < right) {
    int sum = nums[i] + nums[j] + num[left]
                + numr[right];
    if (sum == target) {
        result.add (Arrays.aslist (nums[i], nums[j],
                nums[left], numr['right]));
    left++;
    right--;
    while (left < right && num[left] == num
                [left-1])
    left++;
    while (left < right && nums[right] == num
                [right+1])
    right--;
    } else if (sum < target) {
    left++;
    } else {
    right--;
    }
    }
}
```

return result;
}
public static void main (String[] args) {
int [] nums = {1, 0, -1, 0, -2, 2};
int target = 0;
List<List<Integer>> result = fourSum
(nums, target);
System.out.println("Unique quadruplets:");
for (List<Integer> quadruplet : result) {
System.out.println(quadruplet);
}
}
}

Output

unique quadruplets:
[-2, -1, 1, 2]
[-2, 0, 0, 2]
[-1, 0, 0, 1]

Q.3. permutation Arrayr :-

Answer :-

```
import java.util.Arrayr;
public class NextPermutation {
  public static void nextPermutation (int[] num){

int i = num.length - 2;
while (i >= 0 && numr[i] >= numr[i+1]) {
   i--;
}
if (i >=0) {

  int j = num.length - 1;
  while (j >i && numr[j] <= nums[i]) {
     j--;
    {

   swap(nums, i, j);

  }

  reverse(nums, i+1);
  }
private static void swap(int[] nums, int i, int j){
    int temp = nums[i];
 nums[i] = nums[j];
  numr[j] = temp;
{
private static void reverse (int[] numr, int start) {
  int i = start;
 Int j = numr.length - 1;

  while (i<j) {
      swap(numr, i, j);
    i++;
    j--;

  }
}
```

public static void main (String [] args) {
int [] numr = {1, 2, 3};
System.out.println("original array:" + Arrays.
toString(nums));

nextPermutation(numr);

System.out.println
("Next permutation:" + Array
to String(numr));

}

Output :

Original array : [1, 2, 3]

Next permutation : [1, 3, 2]

Q.4) :- O(log n) runtime complexity ~~use~~ I can use Binary search algorithm.

```
public class SearchInsertPosition {
  public static int searchInsert
                (int[] numr, int target) {
int left = 0;
int right = num.length - 1;
while (left <= right) {
    int mid = left + (right - left) / 2;
if (num[mid] == target) {
    return mid;
}
}
```

```
if (nums[mid] < target) {
    left = mid + 1;
} else {
    right = mid - 1;
{
{
return left;
{
public static void main(String[] args) {
    int[] num = {1, 3, 5, 6};
    int target = 5;
    int index = seachInsert(nums, target);
    System.out.println("Index: " + index);
    {
    {
```

Q.5 :- Answer :-

```
import java.util.Arrays;
public class PlusOne {
public static int[] plusOne(int[] digits)
    {
    int n = digits.length;
    for (int i = n-1; i >= 0; i--) {
        if (digits[i] < 9) {
            digits[i]++;
            return digits;
        }
    digits[i] = 0;
{
```

```
[] result = new int[n+1];
result[0] = 1;
return result;
}
public static void main(String[] args) {
    int[] digits = {1, 2, 3};
    System.out.println("Original digits: "
        + Arrays.toString(digits));
    int[] result = plusOne(digits);
    System.out.println("Resulting digits: "
        + Arrays.toString(result));
}
}
```

Output :-

Original digits : [1, 2, 3]

Resulting digits : [1, 2, 4]

Que :- 6 → Answer :-

```
public class SingleNumber {
    public static int singleNumber(int[] nums) {
        int result = 0;
        for (int num : nums) {
            result ^= num;
        }
        return result;
    }
```

```
public static void main (String [] args) {
    int [] nums = {2, 2, 1};
    int single = singleNumber (nums);
    System.out.println ("single number:"
                        + single);
    }
}
```

Output :-
single no :- 1

Que: 7 : Answer :-

```
import java.util.ArrayList;
import java.util.List;

public class MissingRanger {
public static List<String> findMissingRanger
        (int[] nums, int lower, int upper){

{
List<String> missingRanger = new ArrayList<>()

for (int num : nums) {
    if (num > lower) {
        missingRanger.add (formatRange.
                (lower, num - 1));

    }
    lower = num + 1;
}
if (lower <= upper) {
    missingRanger.add (formatRange
                (lower, upper));

{return missingRanger;
}
```

```
private String stringformatRange(int start,
                                 int end) {
    if (start == end) {
        return String.valueOf(start);
    } else {
        return start + "->" + end;
    }
}

public static void main(String[] aegr) {
    int[] nums {0, 1, 3, 50, 75};
    int lower = 0;
    int upper = 99;
    List<String> missingRanger = findMissing
                    Ranger(nums, lower,
                           upper);
    System.out.println("Missing ranger:" +
                       missingRanger);
}
```

output:

Missing ranger: [2,2], [4,49], [51,74]

[76,99]

Que: 8 Answer:- to check the ovelaping interv

```
import java.util.Arrays;
public class MeetingRoom1 {
    public static boolean canAttendMeeting(int
                                [][] intervalr) {
    Arrays.sort(intervalr(a,b) -> a[0] - b[0]);
    for(int i = 1; i < intervalr.length; i++) {
        if(intervals[i][0] < intervalr[i-1][1])
        {
            return false;
        {
    }
    return true;
}
public static void main(String[] args) {
int[][] intervalr = {{0,30},{5,10},{15,20}
                                        };
boolean canAttend = canAttendMeeting
                        (intervalr);
System.out.println("can attend all
                    meetings:" + canAttend
                                        );
{
}
```

Output:

Can attend all meeting : false