Solve the given tasks using any computer programming language or provide pseudocodes as solutions. Each task carries 5 marks [7x5=35 total marks]. Submit your solution to blackboard till 28th May, 2022.

Note: This activity is to evaluate students' participation that comprises 5% of the total score. It also satisfies one of our course learning outcomes that requires students to be able to implement discrete structures in programming.

Task 01:

i. Assume $P(x,y)$ *is* $(x+y>4)$. Write a function/method that can validate the nested quantifier $\forall x \forall y\, P(x,y)$ for elements from two arrays *X* and *Y*.

hint:
The method returns 1 if statement is valid (the sum of all pairs of values from X and Y is greater than 4), and returns 0 otherwise.
Valid case: *For arrays X={2,3,4} and Y={3,4,5}, the method returns 1(since* $\forall x_i \in X$ *and* $\forall y_j \in Y$, $x_i+y_j>4$*)*
Invalid case: *For arrays X={2,3,4} and Y={1,4,5}, the method returns 0 (since* $2 \in X$ *and* $1 \in Y$ *but* $2+1<4$*)*

```
import java.util.*;

public class Task1 {

	public static void main(String[] args) {
		Scanner inp = new Scanner(System.in);

		System.out.println("Input the values for X and Y:");
		int x = inp.nextInt();
		int y = inp.nextInt();

		if (x + y > 4) {
			System.out.println("1");
		} else {
			System.out.println("0");
		}
	}
}

////////////////////////////////////
/*
* Input the values for X and Y:
* 4
* 5
* 1
*/
////////////////////////////////////
/*
* Input the values for X and Y:
* 4
```

```
* 4
* 0
*/
////////////////////////////////////
```

ii. Modify your code to validate nested quantifier $\exists x \exists y P(x,y)$ when $P(x,y)$ *is* $(x+y=9)$

hint:

Valid case: *For X={2,3,4} and Y={1,4,5} method returns 1 (since 4∈X and 5∈Y and 4+5=9)*

Invalid case: *For X={2,3,4} and Y={1,4,4} method returns 0 (since none of the pairs of values from X and Y adds up to 9)*

```
import java.util.*;

public class Task1_ii {

	public static void main(String[] args) {
		Scanner inp = new Scanner(System.in);

		System.out.println("Input the values for X and Y:");
		int x = inp.nextInt();
		int y = inp.nextInt();

		if (x + y >= 9) {
			System.out.println("1");
		} else {
			System.out.println("0");
		}
	}
}

////////////////////////////////////
/*
* Input the values for X and Y:
* 4
* 5
* 1
*/
////////////////////////////////////
/*
* Input the values for X and Y:
* 4
* 4
* 0
*/
////////////////////////////////////
```

Task 02: Review the concept of arithmetic progressions from chapter-2. Write a method that takes an array of *n* integers as input. The method checks whether an arithmetic progression can be formed using all the given elements. If possible print *"Yes"*, else print *"No"*.

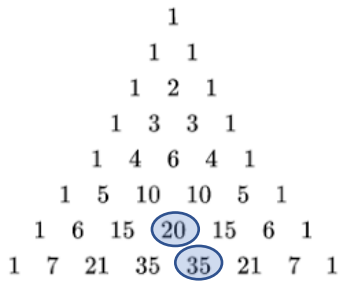
hint: The array of integers {18, 15, 6, 9, 12} can form arithmetic progression if arranged as 6,9,12,15,18 so the method returns "Yes" in this case, but the array {12, 40, 11, 20} cannot form an arithmetic progression. The answer returned in this case will be "No".

```
import java.util.*;

class Task2 {
	public static void main(String[] args) {
		Scanner inp = new Scanner(System.in);
		System.out.print("Please Enter the size of the array: ");
		int size = inp.nextInt();
		System.out.println("Please Enter the numbers: ");
		int[] arr = new int[size];
		for (int i = 0; i < size; i++) {
			arr[i] = inp.nextInt();
		}
		int n = arr.length;
		System.out.println(checkIsAP(arr, n));
	}

	static String checkIsAP(int arr[], int n) {
		HashSet<Integer> set = new HashSet<Integer>();
		int max = Integer.MIN_VALUE;
		int min = Integer.MAX_VALUE;
		for (int i : arr) {
			max = Math.max(i, max);
			min = Math.min(i, min);
			set.add(i);
		}
		int diff = (max - min) / (n - 1);
		int count = 0;

		while (set.contains(max)) {
			count++;
			max = max - diff;
		}
		if (count == arr.length)
			return "YES";

		return "NO";
	}
}
/*
 * Please Enter the size of the array: 5
 * Please Enter the numbers:
 * 18
 * 15
 * 6
 * 9
```

```
 * 12
 * YES
 */
//////////////////////////////////////
```

Task 03: Read the concept of Binomial coefficient given in chapter-6. Write a method that takes two parameters *n* and *k* and returns the value of Binomial Coefficient *C(n, k)*.

For example, your function should return 20 for n = 6 and k = 3, and it should return 35 for n = 7 and k = 4. (confirm it through the Pascal's triangle as given below).




```
import java.util.*;

class Task3 {

	static int binomialCoeff(int n, int k) {

		if (k > n)
			return 0;
		if (k == 0 || k == n)
			return 1;

		return binomialCoeff(n - 1, k - 1) + binomialCoeff(n - 1,
k);
	}

	public static void main(String[] args) {
		Scanner inp = new Scanner(System.in);

		System.out.print("Please choose a number for the n: ");
		int n = inp.nextInt();
		System.out.print("Please choose a number for the k: ");
		int k = inp.nextInt();
		System.out.printf("\nValue of C(" + n + ", " + k + ") is
" + binomialCoeff(n, k));
	}
}

/*
 * Please choose a number for the n: 6
 * Please choose a number for the k: 3
```

```
 *
 * Value of C(6, 3) is 20
 */
////////////////////////////////////
```

Task 04: A *permutation* of a set of distinct objects is an ordered arrangement of these objects. An ordered arrangement of r elements of a set is called an *r-permutation*. Write a method that takes a string and prints all its permutations.

hint: For a string "123" the method returns all its possible permutations as given below.
123
132
213
231
312
321

```
import java.util.*;

public class Task4 {
	public static void main(String[] args) {
		Scanner inp = new Scanner(System.in);

		System.out.print("Please Enter a string: ");
		String str = inp.nextLine();
		int n = str.length();
		Task4 permutation = new Task4();
		permutation.permute(str, 0, n - 1);
	}

	private void permute(String str, int l, int r) {
		if (l == r)
			System.out.println(str);
		else {
			for (int i = l; i <= r; i++) {
				str = swap(str, l, i);
				permute(str, l + 1, r);
				str = swap(str, l, i);
			}
		}
	}

	public String swap(String a, int i, int j) {
		char temp;
		char[] charArray = a.toCharArray();
		temp = charArray[i];
		charArray[i] = charArray[j];
		charArray[j] = temp;
		return String.valueOf(charArray);
	}
}
/*
 * Please Enter a string: 123
 * 123
 * 132
 * 213
```

```
 * 231
 * 321
 * 312
 */
//////////////////////////////
```

Task 05: An *r-combination* of elements of a set is an unordered selection of *r* elements from the set. Thus, an *r*-combination is simply a subset of the set with *r* elements. Given an array of size n, generate and print all possible combinations of r elements in array.

hint, if input array is {1, 2, 3, 4} and r is 3, then output should be:
1 2 3
1 2 4
1 3 4
2 3 4

```
import java.util.*;

class Task5 {

	static void combinationUtil(int arr[], int data[], int start,
int end, int index, int r) {
		if (index == r) {
			for (int j = 0; j < r; j++)
				System.out.print(data[j] + " ");
			System.out.println("");
			return;
		}

		for (int i = start; i <= end && end - i + 1 >= r - index;
i++) {
			data[index] = arr[i];
			combinationUtil(arr, data, i + 1, end, index + 1,
r);
		}
	}

	static void printCombination(int arr[], int n, int r) {
		int data[] = new int[r];

		combinationUtil(arr, data, 0, n - 1, 0, r);
	}

	public static void main(String[] args) {
		Scanner inp = new Scanner(System.in);
		System.out.print("Input the size of the array : ");
		int size = inp.nextInt();
		System.out.print("Input the numbers : \n");
		int arr[] = new int[size];
		for(int i =0;i<size;i++) {
```

```
                arr[i] = inp.nextInt();
            }
            System.out.print("Input the r : ");
            int r = inp.nextInt();
            int n = arr.length;
            printCombination(arr, n, r);
        }
}
/*
 * Input the size of the array : 4
 * Input the numbers :
 * 1
 * 2
 * 3
 * 4
 * Input the r : 3
 * 1 2 3
 * 1 2 4
 * 1 3 4
 * 2 3 4
 */
////////////////////////////////
```

Task 06: Use the recursive definition of Fibonacci series and write a function int fib(int n) that returns F_n (n^{th} element of Fibonacci series). For example, if n = 0, then fib() should return 0. If n = 1, then it should return 1. For n > 1, it should return $F_{n-1} + F_{n-2}$

hint: If n is 8, the function should return 21 (that is 8th element in sequence below).
0, 1, 1, 2, 3, 5, 8, 13, 21, 34

```
import java.util.*;

class Task6 {
      static int fib(int n) {
            if (n <= 1)
                  return n;
            return fib(n - 1) + fib(n - 2);
      }

      public static void main(String args[]) {

            Scanner inp = new Scanner(System.in);
            System.out.print("Input a number for n: ");
            int n = inp.nextInt();
            System.out.println(fib(n));
      }
}
 /*
 * Input a number for n: 8
```

```
 * 21
 */
```

Task 07: The concept of Tower of Hanoi discusses a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3. No disk may be placed on top of a smaller disk.

Write a program that takes number of disks as input and prints a sequence of steps to solve Tower of Hanoi problem.

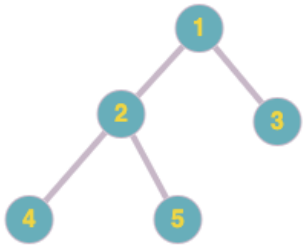
hint: If number of disks are 3, the program prints following sequence:
Move disk 1 from rod A to rod C
Move disk 2 from rod A to rod B
Move disk 1 from rod C to rod B
Move disk 3 from rod A to rod C
Move disk 1 from rod B to rod A
Move disk 2 from rod B to rod C
Move disk 1 from rod A to rod C

```
class Task7 {
	static void towerOfHanoi(int n, char from_rod, char to_rod,
char aux_rod) {
		if (n == 0) {
			return;
		}
		towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);
		System.out.println("Move disk " + n + " from rod " +
from_rod + " to rod " + to_rod);
		towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);
	}


	public static void main(String args[]) {
		int n = 3;
		towerOfHanoi(n, 'A', 'C', 'B');
	}
}
 /*
  * Move disk 1 from rod A to rod C
  * Move disk 2 from rod A to rod B
  * Move disk 1 from rod C to rod B
  * Move disk 3 from rod A to rod C
  * Move disk 1 from rod B to rod A
  * Move disk 2 from rod B to rod C
  * Move disk 1 from rod A to rod C
 */
```

Optional Task:

Write a program to implement post-order traversal of a binary tree.

hint: For the following binary tree, the program prints nodes using post-order traversal i.e., 4, 5, 2, 3, 1.




```
class Node {
	int key;
	Node left, right;

	public Node(int item) {
		key = item;
		left = right = null;
	}
}

class OpTask {
	Node root;

	OpTask() {
		root = null;
	}

	void printPostorder(Node node) {
		if (node == null)
			return;

		printPostorder(node.left);

		printPostorder(node.right);

		System.out.print(node.key + " ");
	}

	void printPostorder() {
		printPostorder(root);
	}

	public static void main(String[] args) {
		OpTask tree = new OpTask();
		tree.root = new Node(1);
		tree.root.left = new Node(2);
		tree.root.right = new Node(3);
```

```
            tree.root.left.left = new Node(4);
            tree.root.left.right = new Node(5);

            System.out.println("Postorder traversal of binary tree is
");
            tree.printPostorder();
        }
}
 /*
 * Postorder traversal of binary tree is
 * 4 5 2 3 1
 */
```