Answer 1:

Input: {0,1,2,0,1,2}

Output: {0,0,1,1,2,2}

Sort an array:

**package** multithreading;

**public** **class** Sorting {

**void** arraySort(**int** arr[]) {

**int** n = arr.length;

**for**(**int** i=0; i<n-1; i++) {

**int** max = i;

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

**if**(arr[j]<arr[max]) {

max = j;

}

}

**int** temp = arr[max];

arr[max] = arr[i];

arr[i] = temp;

}

}

**static** **void** print(**int** arr[]) {

**for**(**int** i = 0; i < arr.length; i++) {

System.***out***.print(arr[i] + " ");

}

}

**public** **static** **void** main(String[] args) {

**int**[] arr = {0,1,2,0,1,2};

Sorting s = **new** Sorting();

s.arraySort(arr);

System.***out***.println("Sorted array");

*print*(arr);

}

}

Answer 2: Order of an program

1. Main function will invoke
2. Then we will call arraySort function with the help of Sorting class object s.
3. Then we will print our sorted array.

Answer 3:

**class** Pattern

{

**public** **static** **void** main (String[] args)

{

**int** n = 5;

*pattern*(n, 1);

}

**static** **void** printrow(**int** num)

{

**if** (num == 0)

**return**;

System.***out***.print ("\* ");

*printrow*(num - 1);

}

**static** **void** pattern(**int** n, **int** i)

{

**if** (n == 0)

**return**;

*printrow*(i);

System.***out***.println();

*pattern*(n - 1, i + 1);

}

}

Answer 4: Order of an program:

1. Main function will be called
2. Pattern function will be invoked with parameter n=5 and i=1(starting point)
3. In pattern it will invoke printrow() function

Answer 5:

String s1 and s2 find longest substring present in both

S1 = “abcde” s2 = “bdgek”

Sol: “bde”

**class** Substring {

**static** String *X*,*Y*;

**public** **static** **void** main(String[] args)

{

**int** i, j;

*X* = "abcde";

*Y* = "bdgek";

i = *X*.length();

j = *Y*.length();

System.***out***.println(*recursion*(i, j, 0));

}

**static** **int** recursion(**int** a, **int** b, **int** num)

{

**if** (a == 0 || b == 0)

{

**return** num;

}

**if** (*X*.charAt(a - 1) == *Y*.charAt(b - 1))

{

num = *recursion*(a - 1, b - 1, num + 1);

}

num = Math.*max*(num, Math.*max*(*recursion*(a, b - 1, 0),*recursion*(a - 1, b, 0)));

**return** num;

}

}

Answer 6: Order will be

Main function will invoke

Recursion function will do its task:

Solution -7 :

Height of binary tree

int treeHeight(Node node)

{

if (node == null)

return 0;

Queue<Node> q = new LinkedList();

q.add(node);

int height = 0;

while (1 == 1)

{

int nodeCount = q.size();

if (nodeCount == 0)

return height;

height++;

while (nodeCount > 0)

{

Node newnode = q.peek();

q.remove();

if (newnode.left != null)

q.add(newnode.left);

if (newnode.right != null)

q.add(newnode.right);

nodeCount--;

}

}

}

Answer 8 – Order will be

Answer 9- Algo to reverse deck of cards

Public void reverseDecks()