NAME : WIJAYAWARDHANA W.A.H.A.

REGISTRATION NO. : 2019/E/166

SEMESTER : SEMESTER 04

DATE ASSIGNED : 03 MARCH 2022

DIVIDE AND CONQUER – LAB 02

EC 4070

DATA STRUCTURES AND ALGORITHMS

01.

Code:-

public class RecursiveFunctions {

// Triangular number

public static int triangularNumber(int number) // number variable is to provide the required triangular number.

{

if(number == 1) // Factorial of 1 is 1.

{

return 1;

}

else // If input is higher than one the recursive will call for summation of number.

{

return (number+triangularNumber(number-1)); // Recursively call triangularNumber function.

}

}

// Factorial.

public static int factorial(int number) // Factorial method is here.

{

if(number > 1) // Until 1 the recursive will run.

{

return (number \* factorial(number -1)); // Recursively call the function.

}

else

{

return 1; // When it reaches 1 it will return 1.

}

}

// Anagrams

public static int anagramsMethod(String word , int n) // Insert the word and calling the method for anagrams.

{

if(word.length()==1) // If the word only have 1 letter can not do any change.

{

return 1;

}

for(int i =0; i<word.length(); i++)

{

n++;

return (anagramsMethod(word,n));

}

return 0;

}

// Towers of Hanoi

public static void towersOfHanoi(int top,char from,char inter, char to , int n) // Calling the method.

{

if(top == 1) // Check the changing disc is 1 or other.

{

System.out.println("Disk 1 From : "+from+" to " +to);

n++;

}

else

{

n++;

towersOfHanoi(top-1, from, to, inter,n);

System.out.println("Disk "+top+" from : " + from +" to " + to);

n++;

towersOfHanoi(top-1, inter, from, to,n);

}

}

public static void main(String[] args) {

int n = triangularNumber(5);

System.out.println(n);

int m = factorial(5);

System.out.println(m);

int s = anagramsMethod("hour",0);

System.out.println(s);

towersOfHanoi(3, 'A', 'B', 'C',0);

}

}

Outputs:-

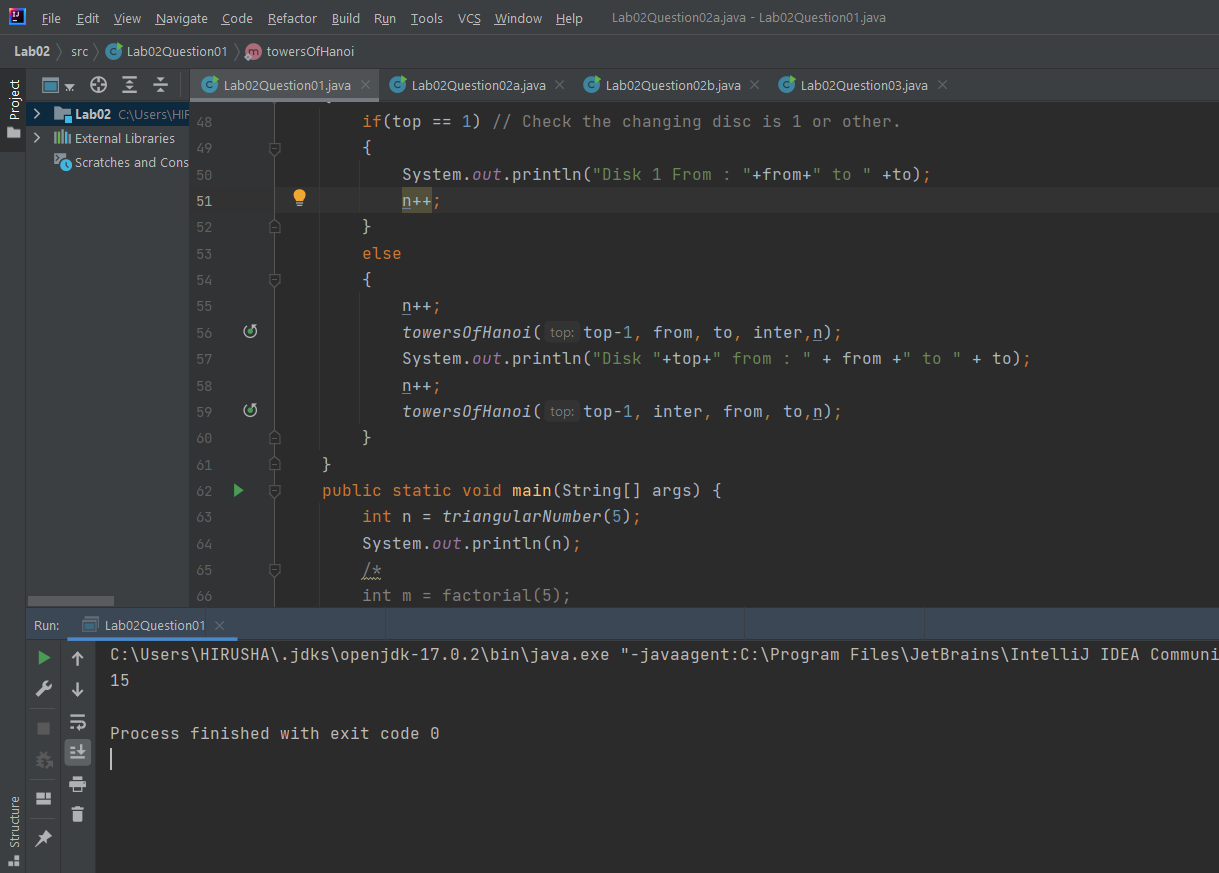


FIGURE 01 – TRIANGULAR NUMBER OUTPUT

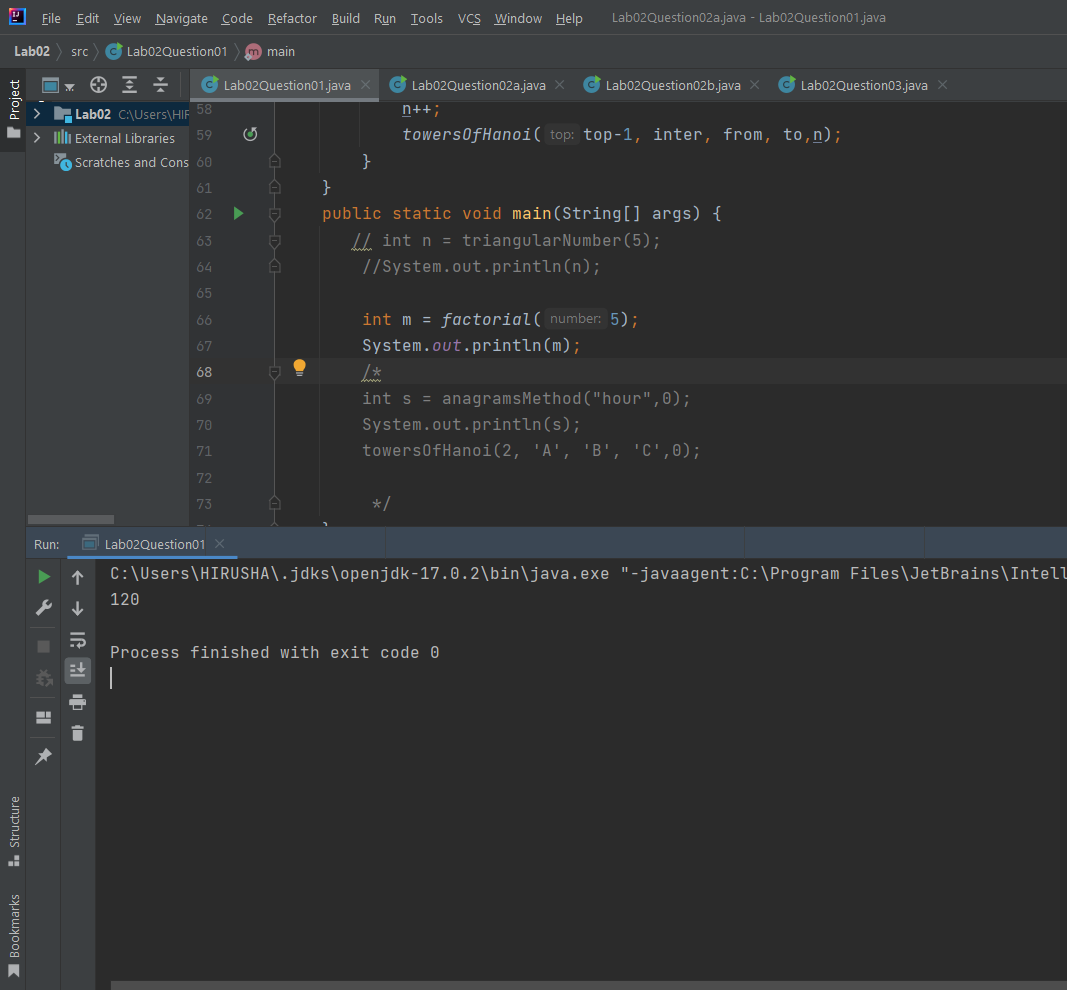


FIGURE 02 – FACTORIAL OUTPUT

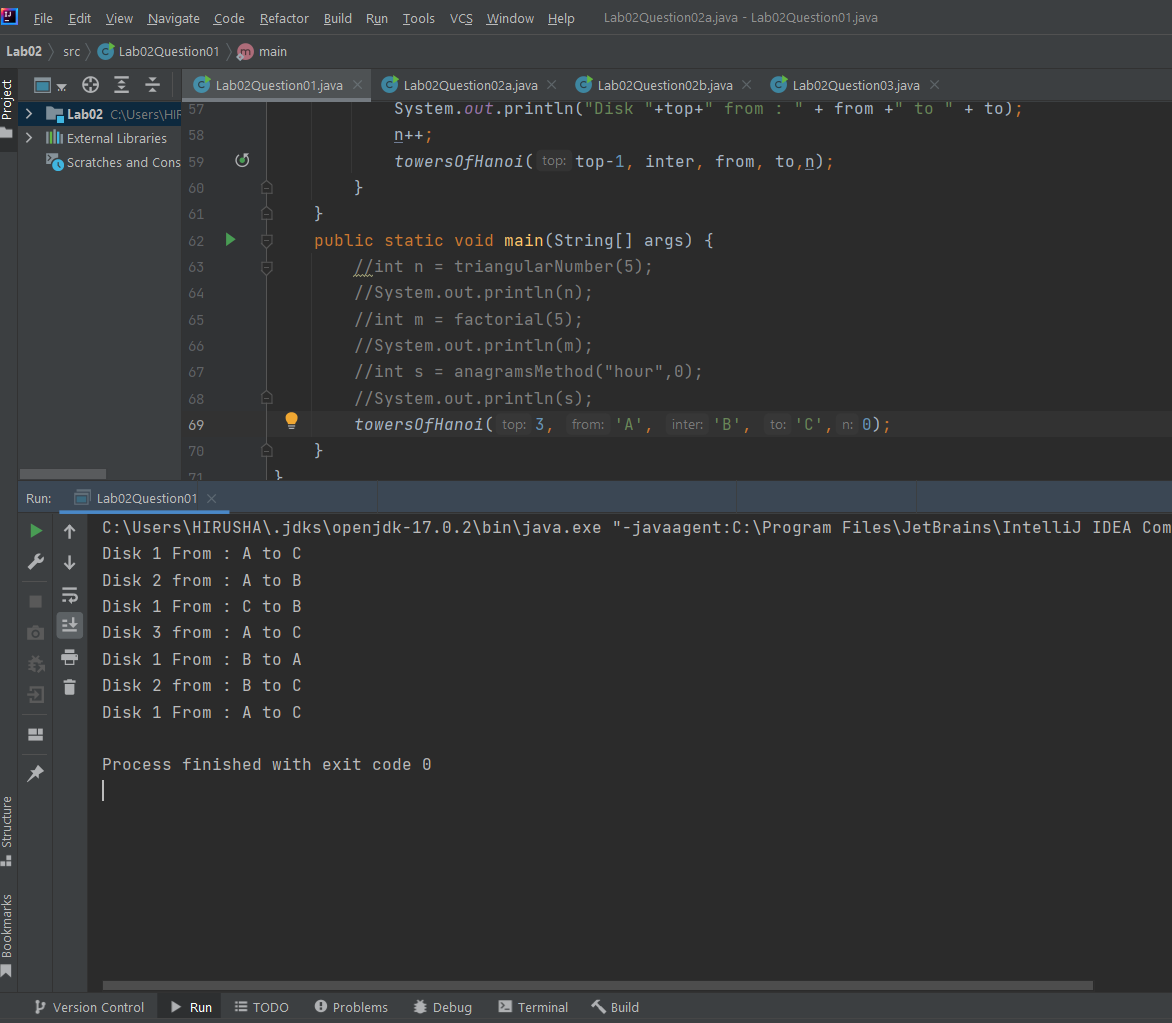


FIGURE 03 – TOWERS OF HANOI OUTPUT

02.

a.

Code:

public class BinarySearchUsingLoop {

public static int binarySearch(long searchKey,int[] numberArray) // Binary search method.

{

int lowerBoundOfSearch = 0; // Lower bound define and assign.

int upperBoundOfSearch = numberArray.length-1; // Upper bound define and assign.

int checkingIndex = 0; // Checking index define and assign as 0.

while(true) // While condition runs true.

{

checkingIndex = (lowerBoundOfSearch+upperBoundOfSearch)/2; // Setting middle index for searching element.

if(numberArray[checkingIndex] == searchKey) // Check the middle element equal to search element.

{

return checkingIndex; // Return the index after equal.

}

else if(lowerBoundOfSearch > upperBoundOfSearch)

{

return numberArray.length-1;

}

else // If upper conditions are not true.

{

if(numberArray[checkingIndex] < searchKey) // Check search element less than the arrays checkingIndex element.

{

lowerBoundOfSearch = checkingIndex + 1; // If it true search element at right side so move to right side.

}

else

{

upperBoundOfSearch = checkingIndex -1; // If not search element is less so element is at left side.

}

}

}

}

public static void main(String[] args) {

int[] numberArray = {3,56,32,33,45,90,190,564,908}; // Define sorted array for searching element.

int n = binarySearch(190, numberArray); // Calling the binarySearch method.

System.out.println("Index is : " + n); // Print the index of searched element.

}

}

Output:-

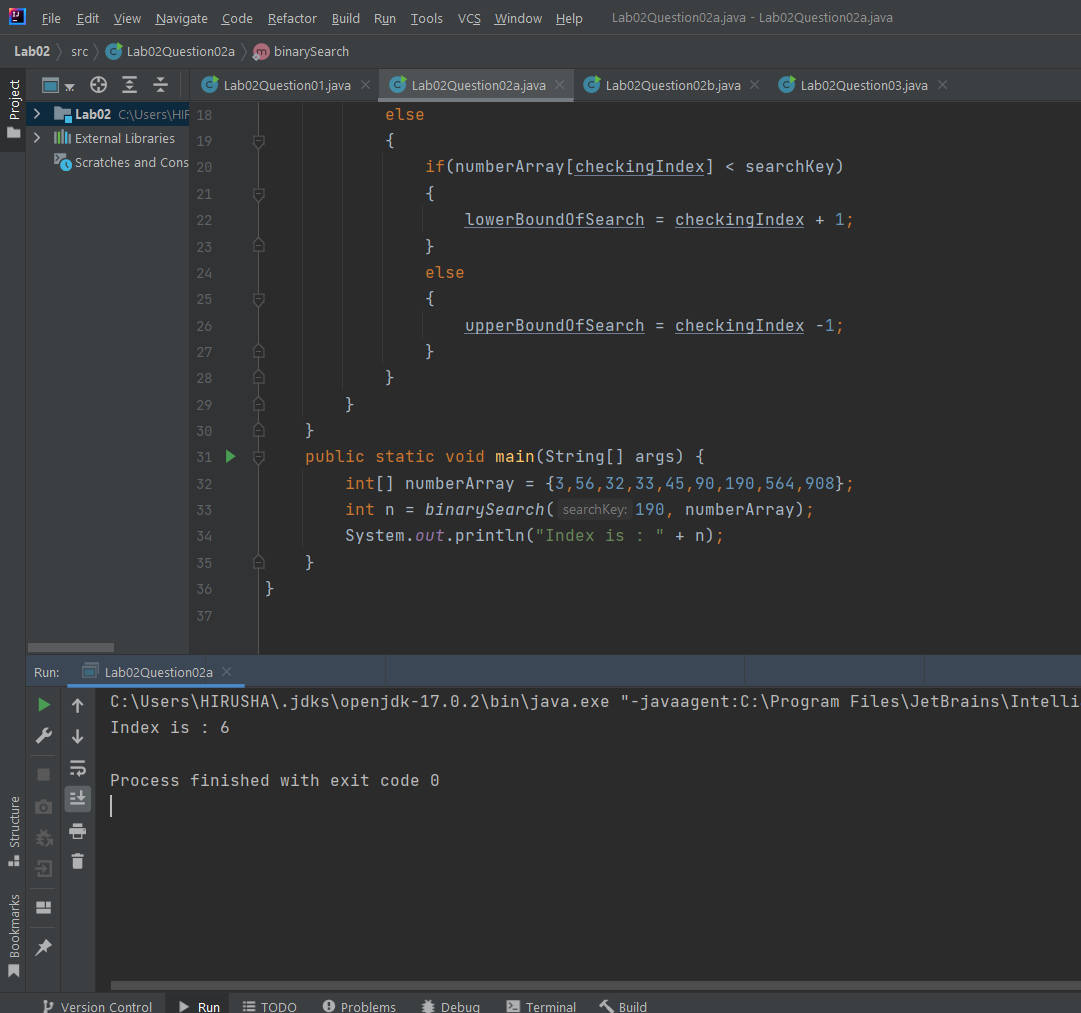


FIGURE 04 – BINARY SEARCH USING LOOP OUTPUT

b.

Code:

public class BinarySearchRecursive {

// Binary searching method.

public static int binarySearch(int searchNumber , int[] numberArray , int upperBoundIndex , int lowerBoundIndex)

{

int middleIndex = (upperBoundIndex + lowerBoundIndex)/2; // Define middleIndex according to passed values.

int upperBoundIndexN; // Define new variable.

int lowerBoundIndexN;

if(searchNumber == numberArray[middleIndex]) // Check the search element equal to middle index's element.

{

System.out.println("Index of " + searchNumber + " : " + middleIndex); // If true that will print and exit the code.

return middleIndex;

}

else if(searchNumber < numberArray[middleIndex]) // If searchNumber less than middleIndex's element that element should at left side of array.

{

upperBoundIndexN = 0; // Define variable according to the identification of element value.

lowerBoundIndexN = middleIndex;

return (binarySearch(searchNumber, numberArray, upperBoundIndexN, lowerBoundIndexN)); // Recall the method.

}

else if(searchNumber == numberArray[numberArray.length-1])

{

System.out.println("Index : " + upperBoundIndex);

}

else if(searchNumber > numberArray[middleIndex]) // If searchNumber higher than middleIndex's element that element should at right side of array.

{

upperBoundIndexN = middleIndex; // Define variable according to the identification of element value.

lowerBoundIndexN = numberArray.length-1;

return (binarySearch(searchNumber, numberArray, upperBoundIndexN, lowerBoundIndexN)); // Recall the method.

}

return -1;

}

public static void main(String[] args) {

int[] numberArray = {10,23,35,45,51,69,78,89,95,100}; // Define the sorted array for searching.

binarySearch(35, numberArray, numberArray.length-1, 0); // Calling binarySearch method.

}

}

Output:

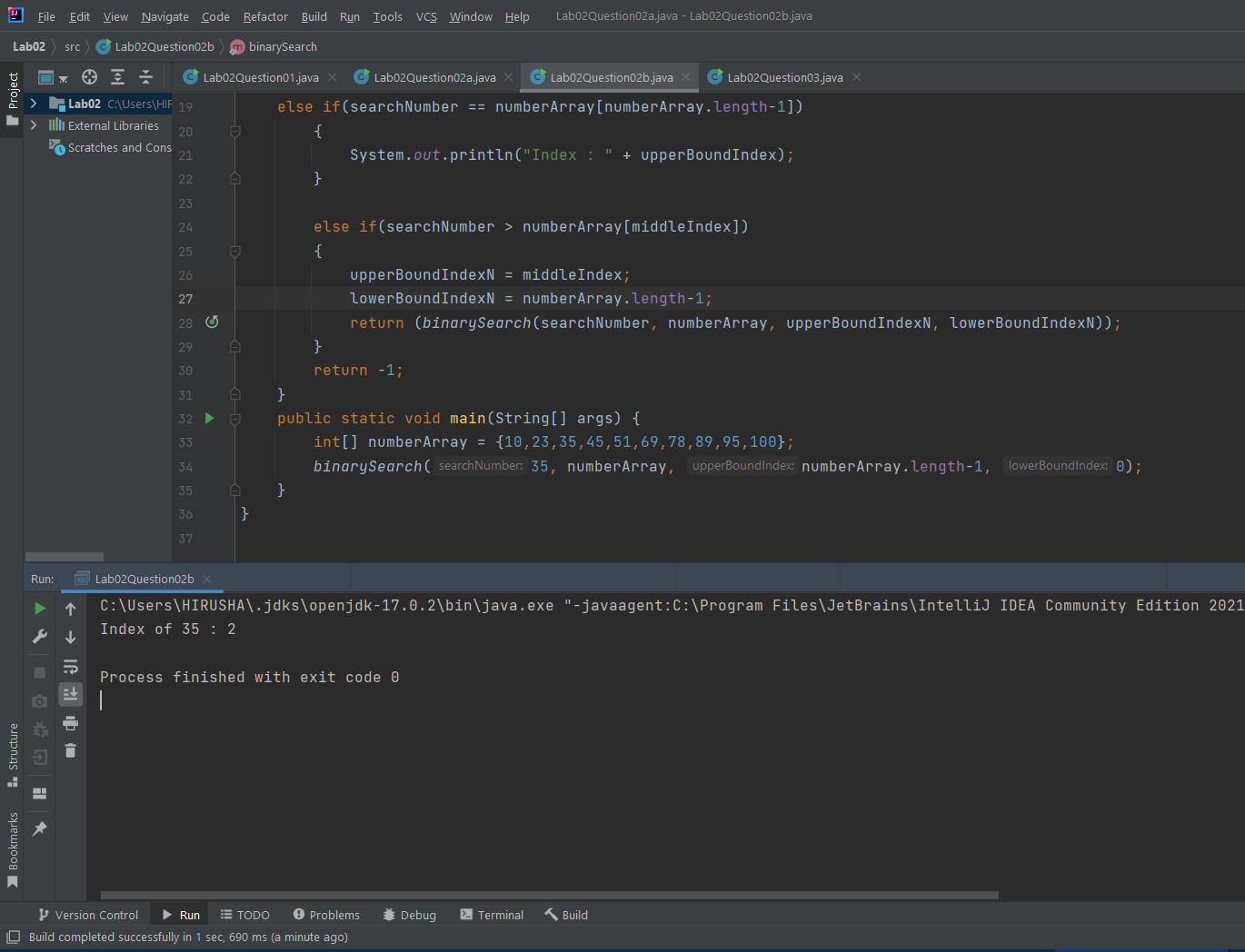


FIGURE 05 – BINARY SEARCH RECURSIVE METHOD OUTPUT

03.

Merge sort

Code:

public class MergeSort {

// Define required variables.

int arraySize;

int[] array = new int[arraySize];

int lowerIndex;

int upperIndex;

int middleIndex;

public void setElement(int[] array,int arraySize) // Setting the array and array size.

{

this.arraySize = arraySize;

this.array = array;

}

// Merge sort method for separating the element.

public void mergeSort(int lowerIndex, int upperIndex)

{

this.lowerIndex = lowerIndex; // Assign values into class variables.

this.upperIndex = upperIndex; // Assign values into class variables.

if(lowerIndex < upperIndex) // Check the condition is true until one element come to array this will true.

{

int middle = (lowerIndex+upperIndex)/2; // Define and assign middle index.

mergeSort(lowerIndex,middle); // Calling mergeSort method for left side of array.

mergeSort(middle+1,upperIndex); // mergeSort will call for right side of array.

merge(lowerIndex,middle,upperIndex); // After finishing dividing array to one element the merge will call for combining the array.

}

}

// Merge method for getting the sorted array.

public void merge(int lowerIndex, int middleIndex, int upperIndex)

{

this.lowerIndex = lowerIndex; // Assign values into class variables.

this.middleIndex = middleIndex; // Assign values into class variables.

this.upperIndex = upperIndex; // Assign values into class variables.

int tempArray[] = new int[upperIndex-lowerIndex+1]; // Define temporary array to store sorted element temporary.

int i = lowerIndex; // Setting values.

int j = middleIndex+1; // Setting values.

for(int k = 0; (i <=middleIndex)||(j<=upperIndex); k++) // Setting values for temporary array from unsorted array.

{

if(i > middleIndex) // According to the value this will put values in sorted array.

{

tempArray[k] = array[j++];

}

else if(j > upperIndex)

{

tempArray[k] = array[i++];

}

else if(array[i] <= array[j])

{

tempArray[k] =array[i++];

}

else

{

tempArray[k] = array[j++];

}

}

for(int index = lowerIndex; index <tempArray.length; index++)

{

array[index] = tempArray[index]; // Put the sorted elements into first array.

}

}

public static void main(String[] args) {

MergeSort newObject = new MergeSort(); // Define an object of the class.

int[] arrayN = new int[]{12,89,65,34,1,66,78,99}; // Define an array for sorting.

newObject.setElement(arrayN,arrayN.length); // Calling setElement method for setting an array for sorting.

newObject.mergeSort(0,arrayN.length-1); // Call mergeSort method for sorting.

for(int i =0; i< arrayN.length; i++) // Print the sorted array.

{

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

}

}

}

Output:-



FIGURE 06 – MERGE SORT OUTPUT

Quick sort

Code:-

public class QuickSort {

int arraySize; // Define arraySize.

int[] arrayElement = new int[arraySize]; // Define array for element.

public int partition(int[] arrayElement, int lowerIndex, int higherIndex)

{

this.arrayElement = arrayElement; // Class object assign for method calling array.

int pivotElement = arrayElement[higherIndex]; // Assign pivot element as last element.

int i = lowerIndex - 1;

for(int j = lowerIndex; j <= higherIndex-1; j++)

{

if(arrayElement[j] < pivotElement) // Check pivot element less than element.

{

i++;

swapElement(i,j); // If condition true swap will do.

}

}

swapElement(i+1,higherIndex); // Otherwise that is the highest value then it will swap.

return(i+1);

}

public void swapElement(int swapElement01 , int swapElement02) // Swap elements method.

{

int temporaryElement = arrayElement[swapElement01]; // Store in temporary value before swap.

arrayElement[swapElement01] = arrayElement[swapElement02]; // Swap elements.

arrayElement[swapElement02] = temporaryElement; // Swap elements.

}

// Quick sort method.

public void quickSort(int[] arrayElement , int lowerElement , int higherElement)

{

this.arrayElement = arrayElement; // Assign the array in class.

if(lowerElement < higherElement) // Condition check high or low.

{

int partitionElement = partition(arrayElement,lowerElement,higherElement); // Call partitionElement method.

quickSort(arrayElement,lowerElement,partitionElement-1); // Call quick sort method for left part.

quickSort(arrayElement,partitionElement+1,higherElement); // Call quick sort method for right part.

}

}

public void printArray(int array[])

{

for(int i =0;i<array.length;i++)

{

System.out.print(array[i]+" "); // Print array element.

}

}

public static void main(String[] args) {

int newArray[]= {23,45,12,78,56,79,90};

QuickSort newObject = new QuickSort(); // Create object of QuickSort.

newObject.quickSort(newArray,0,newArray.length-1); // Calling quickSort method.

newObject.printArray(newArray); // Calling printArray method.

}

}

Output:-

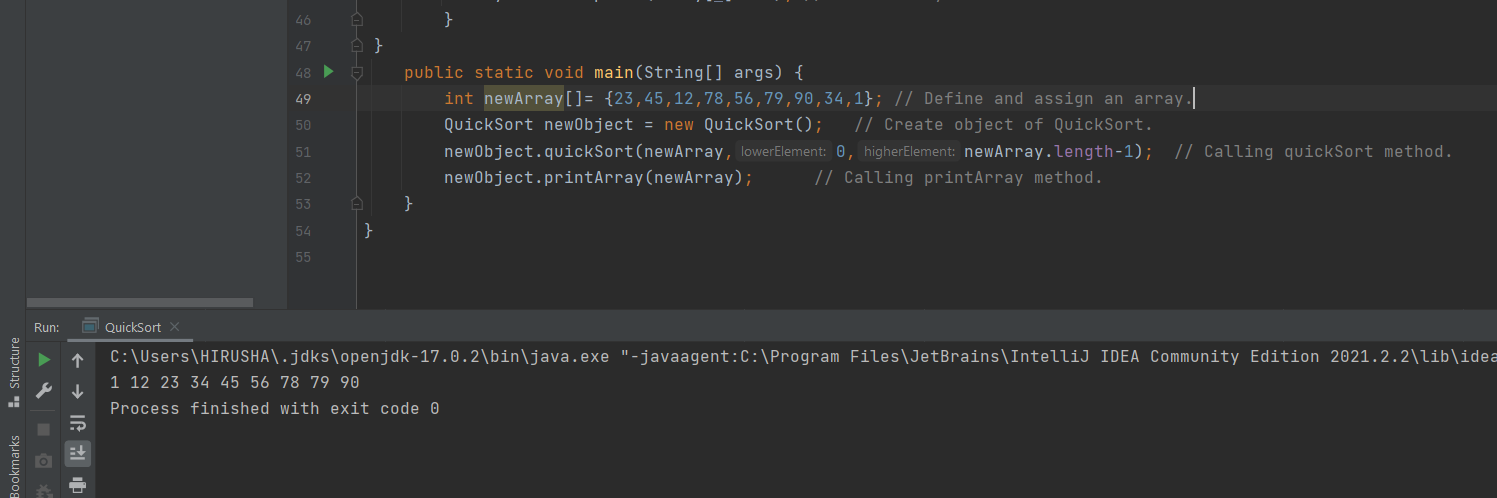


FIGURE 07 – QUICK SORT OUTPUT