Question 01.

```
package Q1;
import java.util.LinkedHashSet;
import java.util.Set;

public class Functions {

    public static Queue removeDuplicates(Queue inputQueue, int maxSize) {
        Set<Integer> uniqueElements = new LinkedHashSet<>();
        Queue tempStorage = new Queue(maxSize);

        while (!inputQueue.isQueueEmpty()) {
            int element = inputQueue.serve();
            if (element != -1) {
                 uniqueElements.add(element);
                 tempStorage.append(element);
            }
        }

        Queue resultQueue = new Queue(maxSize);
        for (int uniqueElement : uniqueElements) {
            resultQueue.append(uniqueElement);
        }

        return resultQueue;
    }
}
```

```java
package Q1;

public class Queue {
    private int[] queue;
    private int front;
    private int rear;
    private int maxSize;
    private int count;

    public Queue(int size) {
        maxSize = size;
        queue = new int[maxSize];
        front = 0;
        rear = -1;
        count = 0;
    }

    public boolean isQueueEmpty() {
        return (count == 0);
    }

    public boolean isQueueFull() {
        return (count == maxSize);
    }

    public void append(int item) {
        if (isQueueFull()) {
            System.out.println("Error: Queue is Full. Cannot append " + item);
        } else {
            rear = (rear + 1) % maxSize;
            queue[rear] = item;
```

```
            count++;
        }
    }

    public int serve() {
        if (isQueueEmpty()) {
            System.out.println("Error: Queue is Empty. Cannot serve.");
            return -1;
        } else {
            int item = queue[front];
            front = (front + 1) % maxSize;
            count--;
            return item;
        }
    }

    public int queueSize() {
        return count;
    }

    public void display() {
        if (isQueueEmpty()) {
            System.out.println("Queue is empty.");
            return;
        }
        System.out.print("Queue (front to rear): [");
        int current = front;
        for (int i = 0; i < count; i++) {
            System.out.print(queue[current]);
            if (i < count - 1) {
                System.out.print(", ");
```

```
            }
            current = (current + 1) % maxSize;
        }
        System.out.println("]");
    }
}
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

 /usr/bin/env /Library/Java/JavaVirtualMachines/temurin-24.jdk/Contents/Home/bin/java --ena
kavindus@kavindus-MacBook-Air BECS-21223-Data-Structures-and-Algorithms-LAB-03 %  /usr/bin/
tualMachines/temurin-24.jdk/Contents/Home/bin/java --enable-preview -XX:+ShowCodeDetailsInE
rs/kavindus/Library/Application\ Support/Code/User/workspaceStorage/c9f7464b032ac8edd2b3959
_ws/BECS-21223-Data-Structures-and-Algorithms-LAB-03_431795d5/bin Q1.Test
Appending elements:

Original Queue:
Queue (front to rear): [10, 20, 10, 10, 30, 40, 50, 40, 50, 50, 10, 20, 20]
Original Queue Size: 13

Queue after removing duplicates:
Queue (front to rear): [10, 20, 30, 40, 50]
Unique Queue Size: 5
kavindus@kavindus-MacBook-Air BECS-21223-Data-Structures-and-Algorithms-LAB-03 %
```

```java
package Q1;

public class Test {

    public static void main(String[] args) {
        int[] inputData = {10, 20, 10, 10, 30, 40, 50, 40, 50, 50, 10, 20, 20};
        int maxSize = 20;

        Queue myQueue = new Queue(maxSize);
        System.out.println("Appending elements:");
        for (int item : inputData) {
            myQueue.append(item);
        }

        System.out.println("\nOriginal Queue:");
        myQueue.display();
        System.out.println("Original Queue Size: " + myQueue.queueSize());

        Queue uniqueQueue = Functions.removeDuplicates(myQueue, maxSize);

        System.out.println("\nQueue after removing duplicates:");
        uniqueQueue.display();
        System.out.println("Unique Queue Size: " + uniqueQueue.queueSize());
    }
}
```

Question 02.

```
package Q2;


public class Functions {


    public static Queue interleaveQueue(Queue inputQueue, int maxSize) {
        int size = inputQueue.queueSize();


        if (size % 2 != 0) {
            System.out.println("Error: Input queue size must be even for
interleaving.");
            return null;
        }


        if (size == 0) {
            return new Queue(maxSize);
        }


        int[] tempArray = new int[size];
        int index = 0;
        while (!inputQueue.isQueueEmpty()) {
            int element = inputQueue.serve();
            if (element != -1) {
                tempArray[index++] = element;
            } else {
                System.out.println("Error serving element during interleaving
process.");
                return null;
            }
        }


        Queue resultQueue = new Queue(maxSize);
```

```
        int midPoint = size / 2;

        for (int i = 0; i < midPoint; i++) {
            resultQueue.append(tempArray[i]);
            resultQueue.append(tempArray[i + midPoint]);
        }

        return resultQueue;
    }
}
```

```java
package Q2;

public class Queue {
    private int[] queue;
    private int front;
    private int rear;
    private int maxSize;
    private int count;

    public Queue(int size) {
        maxSize = size;
        queue = new int[maxSize];
        front = 0;
        rear = -1;
        count = 0;
    }

    public boolean isQueueEmpty() {
        return (count == 0);
    }

    public boolean isQueueFull() {
        return (count == maxSize);
    }

    public void append(int item) {
        if (isQueueFull()) {
            System.out.println("Error: Queue is Full. Cannot append " + item);
        } else {
            rear = (rear + 1) % maxSize;
            queue[rear] = item;
```

```
            count++;
        }
    }

    public int serve() {
        if (isQueueEmpty()) {
            System.out.println("Error: Queue is Empty. Cannot serve.");
            return -1;
        } else {
            int item = queue[front];
            front = (front + 1) % maxSize;
            count--;
            return item;
        }
    }

    public int queueSize() {
        return count;
    }

    public void display() {
        if (isQueueEmpty()) {
            System.out.println("Queue is empty.");
            return;
        }
        System.out.print("Queue (front to rear): [");
        int current = front;
        for (int i = 0; i < count; i++) {
            System.out.print(queue[current]);
            if (i < count - 1) {
                System.out.print(", ");
```

```
            }
            current = (current + 1) % maxSize;
        }
        System.out.println("]");
    }
}
```

```
 /usr/bin/env /Library/Java/JavaVirtualMachines/temurin-24.jdk/Contents/Home/bin/java --enab
kavindus@kavindus-MacBook-Air BECS-21223-Data-Structures-and-Algorithms-LAB-03 %  /usr/bin/e
tualMachines/temurin-24.jdk/Contents/Home/bin/java --enable-preview -XX:+ShowCodeDetailsInEx
rs/kavindus/Library/Application\ Support/Code/User/workspaceStorage/c9f7464b032ac8edd2b3959b
_ws/BECS-21223-Data-Structures-and-Algorithms-LAB-03_431795d5/bin Q2.Test
Appending elements:

Original Queue:
Queue (front to rear): [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
Original Queue Size: 10

Queue after interleaving:
Queue (front to rear): [10, 60, 20, 70, 30, 80, 40, 90, 50, 100]
Interleaved Queue Size: 10
kavindus@kavindus-MacBook-Air BECS-21223-Data-Structures-and-Algorithms-LAB-03 %
```

```
package Q2;

public class Test {

    public static void main(String[] args) {
        int[] inputData = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
        int maxSize = 20;

        Queue myQueue = new Queue(maxSize);
        System.out.println("Appending elements:");
        for (int item : inputData) {
            myQueue.append(item);
        }

        System.out.println("\nOriginal Queue:");
        myQueue.display();
        System.out.println("Original Queue Size: " + myQueue.queueSize());

        Queue interleavedQueue = Functions.interleaveQueue(myQueue, maxSize);

        if (interleavedQueue != null) {
            System.out.println("\nQueue after interleaving:");
            interleavedQueue.display();
            System.out.println("Interleaved Queue Size: " +
interleavedQueue.queueSize());
        } else {
            System.out.println("\nInterleaving failed");
        }
    }
}
```

Question 03.

```
package Q3;

public class Functions {

    public static int deleteMiddleDigit(int number) {
        if (number == 0) {
            return 0;
        }
        if (number < 0) {
            System.out.println("Error: Input number cannot be negative.");
            return -1;
        }

        String numStr = Integer.toString(number);
        int len = numStr.length();
        int middleIndex = len / 2;

        Queue digitQueue = new Queue(len + 1);

        for (char c : numStr.toCharArray()) {
            digitQueue.append(c);
        }

        StringBuilder resultStr = new StringBuilder();
        int currentIndex = 0;
        while (!digitQueue.isQueueEmpty()) {
            char digit = digitQueue.serve();
            if (digit == '\0') {
                System.out.println("Error serving digit from queue.");
                return -1;
```

```
            }
            if (currentIndex != middleIndex) {
                resultStr.append(digit);
            }
            currentIndex++;
        }

        try {
            if (resultStr.length() == 0) {
                return 0;
            }
            return Integer.parseInt(resultStr.toString());
        } catch (NumberFormatException e) {
            System.out.println("Error parsing result string back to integer: "
+ e.getMessage());
            return -1;
        }
    }
}
```

```
package Q3;


public class Test {


    public static void main(String[] args) {
        int inputNumber = 12345;


        System.out.println("Input Number: " + inputNumber);


        int resultNumber = Functions.deleteMiddleDigit(inputNumber);


        if (resultNumber != -1) {
            System.out.println("Number after deleting middle digit: " +
resultNumber);
        } else {
            System.out.println("Function execution failed.");
        }
        int evenInput = 1234;
        System.out.println("Input Number: " + evenInput);
        int evenResult = Functions.deleteMiddleDigit(evenInput);
         if (evenResult != -1) {
            System.out.println("Number after deleting middle digit: " +
evenResult);
        } else {
            System.out.println("Function execution failed.");
        }
        int singleInput = 5;
        System.out.println("Input Number: " + singleInput);
        int singleResult = Functions.deleteMiddleDigit(singleInput);
         if (singleResult != -1) {
            System.out.println("Number after deleting middle digit: " +
singleResult);
```

```
        } else {
            System.out.println("Function execution failed.");
        }
        int zeroInput = 0;
        System.out.println("Input Number: " + zeroInput);
        int zeroResult = Functions.deleteMiddleDigit(zeroInput);
         if (zeroResult != -1) {
            System.out.println("Number after deleting middle digit: " +
zeroResult);
        } else {
            System.out.println("Function execution failed.");
        }
    }
}
```

```
23            System.out.println(x:"Function execution failed.");
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

 /usr/bin/env /Library/Java/JavaVirtualMachines/temurin-24.jdk/Contents/Home/bin/java
kavindus@kavindus-MacBook-Air BECS-21223-Data-Structures-and-Algorithms-LAB-03 %  /us
tualMachines/temurin-24.jdk/Contents/Home/bin/java --enable-preview -XX:+ShowCodeDeta
rs/kavindus/Library/Application\ Support/Code/User/workspaceStorage/c9f7464b032ac8edd
_ws/BECS-21223-Data-Structures-and-Algorithms-LAB-03_431795d5/bin Q3.Test
Input Number: 12345
Number after deleting middle digit: 1245
Input Number: 1234
Number after deleting middle digit: 124
Input Number: 5
Number after deleting middle digit: 0
Input Number: 0
Number after deleting middle digit: 0
kavindus@kavindus-MacBook-Air BECS-21223-Data-Structures-and-Algorithms-LAB-03 %
```

```java
package Q3;

public class Queue {
    private char[] queue;
    private int front;
    private int rear;
    private int maxSize;
    private int count;

    public Queue(int size) {
        maxSize = size;
        queue = new char[maxSize];
        front = 0;
        rear = -1;
        count = 0;
    }

    public boolean isQueueEmpty() {
        return (count == 0);
    }

    public boolean isQueueFull() {
        return (count == maxSize);
    }

    public void append(char item) {
        if (isQueueFull()) {
            System.out.println("Error: Queue is Full. Cannot append " + item);
        } else {
            rear = (rear + 1) % maxSize;
            queue[rear] = item;
```

```
            count++;
        }
    }

    public char serve() {
        if (isQueueEmpty()) {
            System.out.println("Error: Queue is Empty. Cannot serve.");
            return '\0';
        } else {
            char item = queue[front];
            front = (front + 1) % maxSize;
            count--;
            return item;
        }
    }

    public int queueSize() {
        return count;
    }

    public void display() {
        if (isQueueEmpty()) {
            System.out.println("Queue is empty.");
            return;
        }
        System.out.print("Queue (front to rear): [");
        int current = front;
        for (int i = 0; i < count; i++) {
            System.out.print(queue[current]);
            if (i < count - 1) {
                System.out.print(", ");
```

```
            }
            current = (current + 1) % maxSize;
        }
        System.out.println("]");
    }
}
```

Question 04.

```
package Q4;

import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

public class Functions {

    public static double calculateMean(List list) {
        if (list.isListEmpty()) {
            System.out.println("Cannot calculate mean of an empty list.");
            return Double.NaN;
        }
        double sum = 0;
        int size = list.listSize();
        for (int i = 0; i < size; i++) {
            sum += list.retrieveList(i);
        }
        return sum / size;
    }

    public static double calculateMedian(List list) {
        if (list.isListEmpty()) {
            System.out.println("Cannot calculate median of an empty list.");
            return Double.NaN;
        }
        int[] sortedArray = list.getInternalArrayCopy();
        Arrays.sort(sortedArray);

        int size = sortedArray.length;
```

```
        if (size % 2 != 0) {
            return sortedArray[size / 2];
        } else {
            int mid1 = sortedArray[size / 2 - 1];
            int mid2 = sortedArray[size / 2];
            return (double) (mid1 + mid2) / 2.0;
        }
    }

    public static int calculateMode(List list) {
        if (list.isListEmpty()) {
            System.out.println("Cannot calculate mode of an empty list.");
            return Integer.MIN_VALUE;
        }

        Map<Integer, Integer> frequencyMap = new HashMap<>();
        int size = list.listSize();
        int maxFrequency = 0;
        int mode = list.retrieveList(0);

        for (int i = 0; i < size; i++) {
            int element = list.retrieveList(i);
            int frequency = frequencyMap.getOrDefault(element, 0) + 1;
            frequencyMap.put(element, frequency);

            if (frequency > maxFrequency) {
                maxFrequency = frequency;
                mode = element;
            }
        }
        return mode;
```

```
    }
    public static int calculateRange(List list) {
        if (list.isListEmpty() || list.listSize() == 1) {
             System.out.println("Cannot calculate range for lists with 0 or 1
element.");
            return -1;
        }
        int min = list.retrieveList(0);
        int max = list.retrieveList(0);
        int size = list.listSize();

        for (int i = 1; i < size; i++) {
            int element = list.retrieveList(i);
            if (element < min) {
                min = element;
            }
            if (element > max) {
                max = element;
            }
        }
        return max - min;
    }
}
```

```
tualMachines/temurin-24.jdk/Contents/Home/bin/java --enable-preview -XX:+ShowCodeDeta
rs/kavindus/Library/Application\ Support/Code/User/workspaceStorage/c9f7464b032ac8edc
_ws/BECS-21223-Data-Structures-and-Algorithms-LAB-03_431795d5/bin Q4.Test
Dataset:
List: [10, 9, 52, 24, 35, 11, 9, 12, 3, 11, 25, 24, 8, 11, 42]
Mean: 19.066667
Median: 11.0
Mode: 11
Range: 49
kavindus@kavindus-MacBook-Air BECS-21223-Data-Structures-and-Algorithms-LAB-03 %
```

```
package Q4;

public class List {
    private int maxSize;
    private int position;
    private int[] listEntry;

    public List(int size) {
        maxSize = size;
        listEntry = new int[maxSize];
        position = -1;
    }

    public boolean isListEmpty() {
        return (position == -1);
    }

    public boolean isListFull() {
        return (position == maxSize - 1);
    }

    public int listSize() {
        return (position + 1);
    }

    public void insertLast(int x) {
        if (isListFull()) {
            System.out.println("Error: Attempt to insert at the end of a full
list");
        } else {
            listEntry[++position] = x;
        }
```

```
    }

    public void insertList(int p, int element) {
        if (isListFull()) {
            System.out.println("Error: Attempt to insert an entry into a full
list");
        } else if (p < 0 || p > listSize()) {
            System.out.println("Error: Attempt to insert at position " + p + "
which is out of bounds [0, " + listSize() + "]");
        } else {
            for (int i = listSize(); i > p; i--) {
                listEntry[i] = listEntry[i - 1];
            }
            listEntry[p] = element;
            position++;
        }
    }

    public int deleteList(int p) {
        int element;
        if (isListEmpty()) {
            System.out.println("Error: Attempt to delete an entry from an empty
list");
            return Integer.MIN_VALUE;
        } else if (p < 0 || p >= listSize()) {
            System.out.println("Error: Attempt to delete position " + p + "
which is not in the list [0, " + (listSize() - 1) + "]");
            return Integer.MIN_VALUE;
        } else {
            element = listEntry[p];
            for (int i = p; i < listSize() - 1; i++) {
                listEntry[i] = listEntry[i + 1];
            }
```

```
            position--;
            return element;
        }
    }

    public int retrieveList(int p) {
        if (isListEmpty()) {
            System.out.println("Error: Attempt to retrieve an entry from an
empty list");
            return Integer.MIN_VALUE;
        } else if (p < 0 || p >= listSize()) {
            System.out.println("Error: Attempt to retrieve entry at position "
+ p + " which is not in the list [0, " + (listSize() - 1) + "]");
            return Integer.MIN_VALUE;
        } else {
            return listEntry[p];
        }
    }

    public void replaceList(int p, int x) {
        if (isListEmpty()) {
            System.out.println("Error: Attempt to replace an entry of an empty
list");
        } else if (p < 0 || p >= listSize()) {
            System.out.println("Error: Attempt to replace entry at position " +
p + " which is not in the list [0, " + (listSize() - 1) + "]");
        } else {
            listEntry[p] = x;
        }
    }

    public void traverseList() {
        if (isListEmpty()) {
```

```
            System.out.println("List is empty.");
            return;
        }
        System.out.print("List: [");
        for (int i = 0; i < listSize(); i++) {
            System.out.print(listEntry[i]);
            if (i < listSize() - 1) {
                System.out.print(", ");
            }
        }
        System.out.println("]");
    }

    public void clearList() {
        position = -1;
    }

    public int[] getInternalArrayCopy() {
        if (isListEmpty()) {
            return new int[0];
        }
        int[] copy = new int[listSize()];
        System.arraycopy(listEntry, 0, copy, 0, listSize());
        return copy;
    }
}
```

```
package Q4;

public class Test {
    public static void main(String[] args) {
        int[] dataset = {10, 9, 52, 24, 35, 11, 9, 12, 3, 11, 25, 24, 8, 11,
42};
        int maxSize = dataset.length;

        List dataList = new List(maxSize);

        for (int item : dataset) {
            dataList.insertLast(item);
        }

        System.out.println("Dataset:");
        dataList.traverseList();

        double mean = Functions.calculateMean(dataList);
        double median = Functions.calculateMedian(dataList);
        int mode = Functions.calculateMode(dataList);
        int range = Functions.calculateRange(dataList);

        System.out.printf("Mean: %.6f\n", mean);
        System.out.printf("Median: %.1f\n", median);
        System.out.println("Mode: " + mode);
        System.out.println("Range: " + range);
    }
}
```

Question 05.

```
package Q5;

public class Functions {


    public static int findSecondLargest(List list) {

        if (list.isListEmpty() || list.listSize() < 2) {

            System.out.println("Error: List must contain at least two
elements");

            return Integer.MIN_VALUE;

        }


        int largest = list.retrieveList(0);

        int secondLargest = Integer.MIN_VALUE;


        for (int i = 1; i < list.listSize(); i++) {

            int current = list.retrieveList(i);


            if (current > largest) {

                secondLargest = largest;

                largest = current;

            } else if (current > secondLargest && current != largest) {

                secondLargest = current;

            }

        }


        if (secondLargest == Integer.MIN_VALUE) {

            System.out.println("Error: All elements in the list are
identical");

            return Integer.MIN_VALUE;

        }


        return secondLargest;
```

```
    }

    public static List sortDescending(List originalList) {
        if (originalList.isListEmpty()) {
            return new List(0);
        }

        List sortedList = new List(originalList.listSize());
        int[] elements = originalList.getInternalArrayCopy();

        bubbleSort(elements);

        for (int element : elements) {
            sortedList.insertLast(element);
        }

        return sortedList;
    }

    private static void bubbleSort(int[] arr) {
        for (int i = 0; i < arr.length - 1; i++) {
            for (int j = 0; j < arr.length - i - 1; j++) {
                if (arr[j] < arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }
}
```

```
package Q5;

public class List {
    private int maxSize;
    private int position;
    private int[] listEntry;

    public List(int size) {
        maxSize = size;
        listEntry = new int[maxSize];
        position = -1;
    }

    public boolean isListEmpty() {
        return (position == -1);
    }

    public boolean isListFull() {
        return (position == maxSize - 1);
    }

    public int listSize() {
        return (position + 1);
    }

    public void insertLast(int x) {
        if (isListFull()) {
            System.out.println("Error: Attempt to insert at the end of a full
list");
        } else {
            listEntry[++position] = x;
        }
```

```
    }

    public void insertList(int p, int element) {
        if (isListFull()) {
            System.out.println("Error: Attempt to insert an entry into a full
list");
        } else if (p < 0 || p > listSize()) {
            System.out.println("Error: Attempt to insert at position " + p + "
which is out of bounds [0, " + listSize() + "]");
        } else {
            for (int i = listSize(); i > p; i--) {
                listEntry[i] = listEntry[i - 1];
            }
            listEntry[p] = element;
            position++;
        }
    }

    public int deleteList(int p) {
        int element;
        if (isListEmpty()) {
            System.out.println("Error: Attempt to delete an entry from an empty
list");
            return Integer.MIN_VALUE;
        } else if (p < 0 || p >= listSize()) {
            System.out.println("Error: Attempt to delete position " + p + "
which is not in the list [0, " + (listSize() - 1) + "]");
            return Integer.MIN_VALUE;
        } else {
            element = listEntry[p];
            for (int i = p; i < listSize() - 1; i++) {
                listEntry[i] = listEntry[i + 1];
            }
```

```
            position--;
            return element;
        }
    }

    public int retrieveList(int p) {
        if (isListEmpty()) {
            System.out.println("Error: Attempt to retrieve an entry from an
empty list");
            return Integer.MIN_VALUE;
        } else if (p < 0 || p >= listSize()) {
            System.out.println("Error: Attempt to retrieve entry at position "
+ p + " which is not in the list [0, " + (listSize() - 1) + "]");
            return Integer.MIN_VALUE;
        } else {
            return listEntry[p];
        }
    }

    public void replaceList(int p, int x) {
        if (isListEmpty()) {
            System.out.println("Error: Attempt to replace an entry of an empty
list");
        } else if (p < 0 || p >= listSize()) {
            System.out.println("Error: Attempt to replace entry at position " +
p + " which is not in the list [0, " + (listSize() - 1) + "]");
        } else {
            listEntry[p] = x;
        }
    }

    public void traverseList() {
        if (isListEmpty()) {
```

```
            System.out.println("List is empty.");
            return;
        }
        System.out.print("List: [");
        for (int i = 0; i < listSize(); i++) {
            System.out.print(listEntry[i]);
            if (i < listSize() - 1) {
                System.out.print(", ");
            }
        }
        System.out.println("]");
    }

    public void clearList() {
        position = -1;
    }

    public int[] getInternalArrayCopy() {
        if (isListEmpty()) {
            return new int[0];
        }
        int[] copy = new int[listSize()];
        System.arraycopy(listEntry, 0, copy, 0, listSize());
        return copy;
    }
}
```

```java
package Q5;


public class Test {
    public static void main(String[] args) {
        int[] dataset = {10, 8, 7, 20, 15, 4};
        int maxSize = dataset.length;

        List myList = new List(maxSize);

        for (int item : dataset) {
            myList.insertLast(item);
        }

        System.out.println("Original List:");
        myList.traverseList();

        int secondLargest = Functions.findSecondLargest(myList);
        if (secondLargest != Integer.MIN_VALUE) {
            System.out.println("Second largest number: " + secondLargest);
        }

        List sortedList = Functions.sortDescending(myList);
        System.out.println("Descending order:");
        sortedList.traverseList();
    }
}
```

```
 /usr/bin/env /Library/Java/JavaVirtualMachines/temurin-24.jdk/Content
kavindus@kavindus-MacBook-Air BECS-21223-Data-Structures-and-Algorithm
tualMachines/temurin-24.jdk/Contents/Home/bin/java --enable-preview -X
rs/kavindus/Library/Application\ Support/Code/User/workspaceStorage/c9
_ws/BECS-21223-Data-Structures-and-Algorithms-LAB-03_431795d5/bin Q5.T
Original List:
List: [10, 8, 7, 20, 15, 4]
Second largest number: 15
Descending order:
List: [20, 15, 10, 8, 7, 4]
kavindus@kavindus-MacBook-Air BECS-21223-Data-Structures-and-Algorithm
```