LAB01: ARRAYS

i. Write a function to convert array to number. (10pts)

Suppose we have loaded an array with the digits of an integer, where the highest power is kept in position 0, next highest in position 1, and so on.

The ones position is always at position array.Length - 1:

int[] digits = { 2, 0, 1, 8 };

```java
import java.util.Scanner;

public class convertArray {
    public static int convertArrayToNumber(int[] digits) {
        int number = 0;
        int power = digits.length - 1;

        for (int digit : digits) {
            number += digit * Math.pow(10, power);
            power--;
        }

        return number;
    }

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

        System.out.print("Enter the number of digits: ");
        int numDigits = scanner.nextInt();

        int[] digits = new int[numDigits];

        System.out.println("Enter the digits:");
        for (int i = 0; i < numDigits; i++) {
            digits[i] = scanner.nextInt();
        }

        int result = convertArrayToNumber(digits);
        System.out.println("Converted number: " + result);

        scanner.close();
    }
}
```

ii. Write a function to input a list of integer numbers and return the median of that list (10pts).

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

public class medianCal {
    public static double calculateMedian(List<Integer> numbers) {
        // Sort the list in ascending order
        Collections.sort(numbers);

        int size = numbers.size();
        if (size % 2 == 0) {
            // If the list size is even, return the average of the middle two numbers
            int middleIndex = size / 2;
            int num1 = numbers.get(middleIndex - 1);
            int num2 = numbers.get(middleIndex);
            return (num1 + num2) / 2.0;
        } else {
            // If the list size is odd, return the middle number
            int middleIndex = size / 2;
            return numbers.get(middleIndex);
        }
    }

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

        System.out.println("Enter the numbers (separated by spaces):");
        String input = scanner.nextLine();
        String[] numberStrings = input.split(" ");

        List<Integer> numbers = new ArrayList<>();
        for (String numberString : numberStrings) {
            int number = Integer.parseInt(numberString);
            numbers.add(number);
        }

        double median = calculateMedian(numbers);
        System.out.println("Median: " + median);

        scanner.close();
    }
}
```

iii. Find the min-gap (10pts)

Write a method named minGap that accepts an integer array and a number of elements as parameters and returns the minimum 'gap' between adjacent values in the array. The gap between two adjacent values in an array is defined as the second value minus the first value.

For example, suppose a variable called array is an array of integers that stores the following sequence of values:

int[] array = {1, 3, 6, 7, 12};

The first gap is 2 (3 - 1), the second gap is 3 (6 - 3), the third gap is 1 (7 - 6) and the fourth gap is 5 (12 - 7).

Thus, the call of minGap(array, n) should return 1 because that is the smallest gap in the array. If you are passed an array with fewer than 2 elements, you should return 0.

```
import java.util.Scanner;

public class minGap1 {
   public static int minGap(int[] array, int numElements) {
       if (numElements < 2) {
           return 0; // Return 0 if array has fewer than 2 elements
       }

       int minGap = Integer.MAX_VALUE;

       for (int i = 0; i < numElements - 1; i++) {
           int gap = array[i + 1] - array[i];
           if (gap < minGap) {
               minGap = gap;
           }
       }

       return minGap;
   }

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

       System.out.print("Enter the number of elements in the array: ");
       int numElements = scanner.nextInt();

       int[] array = new int[numElements];

       System.out.println("Enter the elements of the array:");
       for (int i = 0; i < numElements; i++) {
           array[i] = scanner.nextInt();
       }

       int result = minGap(array, numElements);
```

```
        System.out.println("Minimum gap: " + result);

        scanner.close();
    }
}
```

iv. GasMileage.java (input, the use of Scanner class to read numeric data)

What happens if you enter 20.5 for miles? Fix the problem in two ways.

What happens if enter miles and gallons on the same line?

Modify the program so that it reads two lines of inputs with car, miles, and gallons (separated by blanks) and prints mpg for each car.

```
import java.util.Scanner;

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

        System.out.print("Enter the number of cars: ");
        int numCars = scan.nextInt();

        scan.nextLine(); // Consume the newline character

        for (int i = 1; i <= numCars; i++) {
            System.out.println("Car " + i);

            System.out.print("Enter the miles and gallons (separated by a space): ");
            double miles = scan.nextDouble();
            double gallons = scan.nextDouble();

            double mpg = miles / gallons;

            System.out.println("Miles Per Gallon: " + mpg);
            System.out.println();
            scan.nextLine(); // Consume the newline character
        }

        scan.close();
    }
}
```

v. Student.java, Students.java, students.txt (text files, loops, decision making, access modifiers)

Add public/private to the declaration of student names and print students with System.out.println(st.lname), explain.

Change the while-loop with a for-loop

Use grade to determine the type of the student: excellent (> 89), ok [60,89], and failure (< 60)

Do counting and averaging within each student type (excellent, ok, and failure)

Students.java

```
import java.util.Scanner;
import java.io.*;

public class Students {
   public static void main(String[] args) throws IOException {
       int total = 0;
       int count = 0;
       int excellentCount = 0;
       int okCount = 0;
       int failureCount = 0;

       Scanner fileInput = new Scanner(new File("students.txt"));
       while (fileInput.hasNext()) {
           String first_name = fileInput.next();
           String last_name = fileInput.next();
           int grade = fileInput.nextInt();

           Student st = new Student(first_name, last_name, grade);

           System.out.println(st);
           total += grade;
           count++;

           String studentType = st.getStudentType();
           if (studentType.equals("Excellent")) {
               excellentCount++;
           } else if (studentType.equals("OK")) {
               okCount++;
           } else {
               failureCount++;
           }
       }
       fileInput.close();

       double average = (double) total / count;
       System.out.println("There are " + count + " students with an average grade of " + average);
```

```
        System.out.println("Excellent students: " + excellentCount);
        System.out.println("OK students: " + okCount);
        System.out.println("Failure students: " + failureCount);
    }
}
```

Student.java

```
public class Student {
    private String fname;
    private String lname;
    private int grade;

    public Student(String fname, String lname, int grade) {
        this.fname = fname;
        this.lname = lname;
        this.grade = grade;
    }

    public String getFullName() {
        return fname + " " + lname;
    }

    public int getGrade() {
        return grade;
    }

    public String getStudentType() {
        if (grade > 89) {
            return "Excellent";
        } else if (grade >= 60 && grade <= 89) {
            return "OK";
        } else {
            return "Failure";
        }
    }

    public String toString() {
        return getFullName() + "\t" + grade;
    }
}
```

Students.txt

John Smith 90
Barack Obama 95
Al Clark 80
Sue Taylor 55
Ann Miller 75
George Bush 58

1.3. Problem 2

i. Compile and Run the Example Programs

- Array, LowArray, HighArray
- OrderedArray
- ClassDataArray

ii. Programming Projects 2.2 in Text-Book (lowArray.java)

```
// lowArray.java
// demonstrates array class with low-level interface
// to run this program: C>java LowArrayApp
////////////////////////////////////////////////////////////////
class LowArray {
   private long[] a; // ref to array a
   // --------------------------------------------------------------

   public LowArray(int size) // constructor
   {
       a = new long[size];
   } // create array
       // --------------------------------------------------------------

   public void setElem(int index, long value) // set value
   {
       a[index] = value;
   }

   // --------------------------------------------------------------
   public long getElem(int index) // get value
   {
       return a[index];
   }
   // --------------------------------------------------------------
} // end class LowArray
   ////////////////////////////////////////////////////////////////

class LowArrayApp {
   public static void main(String[] args) {
```

```
        LowArray arr; // reference
        arr = new LowArray(100); // create LowArray object
        int nElems = 0; // number of items in array
        int j; // loop variable

        arr.setElem(0, 77); // insert 10 items
        arr.setElem(1, 99);
        arr.setElem(2, 44);
        arr.setElem(3, 55);
        arr.setElem(4, 22);
        arr.setElem(5, 88);
        arr.setElem(6, 11);
        arr.setElem(7, 00);
        arr.setElem(8, 66);
        arr.setElem(9, 33);
        nElems = 10; // now 10 items in array

        for (j = 0; j < nElems; j++) // display items
            System.out.print(arr.getElem(j) + " ");
        System.out.println("");

        int searchKey = 26; // search for data item
        for (j = 0; j < nElems; j++) // for each element,
            if (arr.getElem(j) == searchKey) // found item?
                break;
        if (j == nElems) // no
            System.out.println("Can't find " + searchKey);
        else // yes
            System.out.println("Found " + searchKey);

        // delete value 55
        for (j = 0; j < nElems; j++) // look for it
            if (arr.getElem(j) == 55)
                break;
        for (int k = j; k < nElems; k++) // higher ones down
            arr.setElem(k, arr.getElem(k + 1));
        nElems--; // decrement size

        for (j = 0; j < nElems; j++) // display items
            System.out.print(arr.getElem(j) + " ");
        System.out.println("");
    } // end main()
} // end class LowArrayApp
    ////////////////////////////////////////////////////////////////
```

iii. HighArray.java

Add a method called getMax() that returns the value of the highest key in the array, or –1 if the array is empty. Add some code in main() to exercise this method. You can assume all the keys are positive numbers.

Write a noDups() method for the HighArray class. This method should remove all duplicates from the array. That is, if three items with the key 17 appear in the array, noDups() should remove two of them. Don't worry about maintaining the order of the items. One approach is to first compare every item with all the other items and overwrite any duplicates with a null (or a distinctive value that isn't used for real keys). Then remove all the nulls. Of course, the array size will be reduced.

Insert 100 random items (generated with nextInt()) and find one of them chosen at random. Print the number of comparisons (add a counter of comparisons in class HighArray and set it in the find method).

Compute and print the average number of comparisons to find a random item over 100 trials.

Print the average number of comparisons to find a random item in arrays with 100, 200, 300,...,1000 items. Analyze the trend.

```
import java.util.Random;

class HighArray {
   private long[] a; // ref to array a
   private int nElems; // number of data items
   private int comparisons; // counter for comparisons

   public HighArray(int max) // constructor
   {
       a = new long[max]; // create the array
       nElems = 0; // no items yet
       comparisons = 0; // initialize comparisons counter
   }

   public void insert(long value) { // insert value into array
       a[nElems] = value; // insert it
       nElems++; // increment size
   }

   public boolean find(long searchKey) { // find specified value
       int j;
       comparisons = 0; // reset comparisons counter
       for (j = 0; j < nElems; j++) { // for each element,
           comparisons++; // increment comparisons counter
           if (a[j] == searchKey) // found item?
               break; // exit loop before end
       }
       if (j == nElems) // gone to end?
           return false; // yes, can't find it
       else
           return true; // no, found it
```

```
    }

    public int getComparisons() {
        return comparisons;
    }

    public int getNElems() {
        return nElems;
    }

    public long[] getArray() {
        return a;
    }
}

class HighArrayApp {
    public static void main(String[] args) {
        int maxSize = 100; // array size
        HighArray arr; // reference to array
        arr = new HighArray(maxSize); // create the array

        // Insert 100 random items
        Random random = new Random();
        for (int i = 0; i < 100; i++) {
            long value = random.nextInt(1000); // generate a random number
            arr.insert(value);
        }

        // Find a random item chosen at random
        int randomIndex = random.nextInt(arr.getNElems());
        long randomItem = arr.getArray()[randomIndex];
        System.out.println("Random Item: " + randomItem);

        // Print the number of comparisons to find the random item
        boolean found = arr.find(randomItem);
        int comparisons = arr.getComparisons();
        if (found) {
            System.out.println("Number of comparisons: " + comparisons);
        } else {
            System.out.println("Item not found.");
        }

        // Compute and print the average number of comparisons to find a random item over 100 trials
        int totalComparisons = 0;
        int numTrials = 100;
        for (int i = 0; i < numTrials; i++) {
            randomItem = random.nextInt(1000); // generate a random item
            found = arr.find(randomItem);
```

```
            comparisons = arr.getComparisons();
            totalComparisons += comparisons;
        }
        double averageComparisons = (double) totalComparisons / numTrials;
        System.out.println("Average number of comparisons over 100 trials: " + averageComparisons);

        // Print the average number of comparisons to find a random item in arrays with 100, 200,
300, ..., 1000 items
        for (int size = 100; size <= 1000; size += 100) {
            arr = new HighArray(size); // create a new array with the given size
            totalComparisons = 0;
            numTrials = 100;
            for (int i = 0; i < numTrials; i++) {
                randomItem = random.nextInt(1000); // generate a random item
                found = arr.find(randomItem);
                comparisons = arr.getComparisons();
                totalComparisons += comparisons;
            }
            averageComparisons = (double) totalComparisons / numTrials;
            System.out.println("Average number of comparisons for array size " + size + ": " +
averageComparisons);
        }
    }
}
```

iv. OrderedApp.java

- Insert 100 random items (generated with nextInt()) and find one of them chosen at random. Print the number of comparisons (add a counter of comparisons in class HighArray and set it in the find method).
- Compute and print the average number of comparisons to find a random item over 100 trials.
- Print the average number of comparisons to find a random item in arrays with 100, 200, 300,...,1000 items. Analyze the trend.
- Compare the complexity of linear (HighArrayApp.java) and binary (OrderedApp.java) search.

```
import java.util.Random;

class OrdArray {
   private long[] a;
   private int nElems;

   public OrdArray(int max) {
       a = new long[max];
       nElems = 0;
   }
```

```
    public int size() {
        return nElems;
    }

    public int find(long searchKey) {
        int lowerBound = 0;
        int upperBound = nElems - 1;
        int curIn;
        int comparisons = 0;

        while (true) {
            curIn = (lowerBound + upperBound) / 2;
            comparisons++;
            if (a[curIn] == searchKey)
                return comparisons;
            else if (lowerBound > upperBound)
                return comparisons;
            else {
                if (a[curIn] < searchKey)
                    lowerBound = curIn + 1;
                else
                    upperBound = curIn - 1;
            }
        }
    }

    public void insert(long value) {
        int j;
        for (j = 0; j < nElems; j++)
            if (a[j] > value)
                break;
        for (int k = nElems; k > j; k--)
            a[k] = a[k - 1];
        a[j] = value;
        nElems++;
    }

    public long[] getArray() {
        return a;
    }

    public void display() {
        for (int j = 0; j < nElems; j++)
            System.out.print(a[j] + " ");
        System.out.println("");
    }
}
```

```
class OrderedApp {
   public static void main(String[] args) {
       int maxSize = 100;
       OrdArray arr = new OrdArray(maxSize);
       Random rand = new Random();
       int randomIndex = rand.nextInt(maxSize);

       for (int i = 0; i < maxSize; i++) {
           long value = rand.nextInt(1000);
           arr.insert(value);
       }

       long searchKey = arr.getArray()[randomIndex];
       int comparisons = arr.find(searchKey);
       System.out.println("Number of comparisons to find the random item: " + comparisons);

       int totalComparisons = 0;
       int numTrials = 100;
       for (int i = 0; i < numTrials; i++) {
           randomIndex = rand.nextInt(maxSize);
           searchKey = arr.getArray()[randomIndex];
           comparisons = arr.find(searchKey);
           totalComparisons += comparisons;
       }
       double averageComparisons = (double) totalComparisons / numTrials;
       System.out.println("Average number of comparisons over 100 trials: " + averageComparisons);

       int[] arraySizes = { 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000 };
       for (int size : arraySizes) {
           OrdArray newArray = new OrdArray(size);
           for (int i = 0; i < size; i++) {
               long value = rand.nextInt(1000);
               newArray.insert(value);
           }
           totalComparisons = 0;
           numTrials = 100;
           for (int i = 0; i < numTrials; i++) {
               randomIndex = rand.nextInt(size);
               searchKey = newArray.getArray()[randomIndex];
               comparisons = newArray.find(searchKey);
               totalComparisons += comparisons;
           }
           averageComparisons = (double) totalComparisons / numTrials;
           System.out.println("Average number of comparisons for array size " + size + ": " +
averageComparisons);
       }
   }
}
```

Class DataArrayApp.java

```
// classDataArray.java
// data items as class objects
// to run this program: C>java ClassDataApp
////////////////////////////////////////////////////////////////
class Person
  {
  private String lastName;
  private String firstName;
  private int age;
//-------------------------------------------------------------
  public Person(String last, String first, int a)
     {                              // constructor
     lastName = last;
     firstName = first;
     age = a;
     }
//-------------------------------------------------------------
  public void displayPerson()
     {
     System.out.print("   Last name: " + lastName);
     System.out.print(", First name: " + firstName);
     System.out.println(", Age: " + age);
     }
//-------------------------------------------------------------
  public String getLast()           // get last name
     { return lastName; }
  }  // end class Person
////////////////////////////////////////////////////////////////
class ClassDataArray
  {
  private Person[] a;               // reference to array
  private int nElems;               // number of data items

  public ClassDataArray(int max)    // constructor
     {
     a = new Person[max];               // create the array
     nElems = 0;                        // no items yet
     }
//-------------------------------------------------------------
  public Person find(String searchName)
     {                              // find specified value
     int j;
     for(j=0; j<nElems; j++)            // for each element,
        if( a[j].getLast().equals(searchName) )  // found item?
           break;                       // exit loop before end
     if(j == nElems)                    // gone to end?
        return null;                    // yes, can't find it
```

```
    else
       return a[j];                    // no, found it
    }  // end find()
//--------------------------------------------------------------
// put person into array
  public void insert(String last, String first, int age)
    {
    a[nElems] = new Person(last, first, age);
    nElems++;                          // increment size
    }
//--------------------------------------------------------------
  public boolean delete(String searchName)
    {                                  // delete person from array
    int j;
    for(j=0; j<nElems; j++)            // look for it
       if( a[j].getLast().equals(searchName) )
          break;
    if(j==nElems)                      // can't find it
       return false;
    else                               // found it
       {
       for(int k=j; k<nElems; k++)     // shift down
          a[k] = a[k+1];
       nElems--;                       // decrement size
       return true;
       }
    }  // end delete()
//--------------------------------------------------------------
  public void displayA()               // displays array contents
    {
    for(int j=0; j<nElems; j++)        // for each element,
       a[j].displayPerson();           // display it
    }
//--------------------------------------------------------------
  }  // end class ClassDataArray
////////////////////////////////////////////////////////////////
class ClassDataArrayApp
  {
  public static void main(String[] args)
    {
    int maxSize = 100;                 // array size
    ClassDataArray arr;                // reference to array
    arr = new ClassDataArray(maxSize);  // create the array
                                       // insert 10 items
    arr.insert("Evans", "Patty", 24);
    arr.insert("Smith", "Lorraine", 37);
    arr.insert("Yee", "Tom", 43);
    arr.insert("Adams", "Henry", 63);
```

```
    arr.insert("Hashimoto", "Sato", 21);
    arr.insert("Stimson", "Henry", 29);
    arr.insert("Velasquez", "Jose", 72);
    arr.insert("Lamarque", "Henry", 54);
    arr.insert("Vang", "Minh", 22);
    arr.insert("Creswell", "Lucinda", 18);

    arr.displayA();                 // display items

    String searchKey = "Stimson";  // search for item
    Person found;
    found=arr.find(searchKey);
    if(found != null)
       {
       System.out.print("Found ");
       found.displayPerson();
       }
    else
       System.out.println("Can't find " + searchKey);

    System.out.println("Deleting Smith, Yee, and Creswell");
    arr.delete("Smith");            // delete 3 items
    arr.delete("Yee");
    arr.delete("Creswell");

    arr.displayA();                 // display items again
    }  // end main()
  }  // end class ClassDataApp
```

Array.java

```
// array.java
// demonstrates Java arrays
// to run this program: C>java arrayApp
////////////////////////////////////////////////////////////////
class ArrayApp
  {
  public static void main(String[] args)
    {
    long[] arr;                   // reference to array
    arr = new long[100];          // make array
    int nElems = 0;               // number of items
    int j;                        // loop counter
    long searchKey;               // key of item to search for
//-----------------------------------------------------------
    arr[0] = 77;                  // insert 10 items
    arr[1] = 99;
    arr[2] = 44;
```

```
     arr[3] = 55;
     arr[4] = 22;
     arr[5] = 88;
     arr[6] = 11;
     arr[7] = 00;
     arr[8] = 66;
     arr[9] = 33;
     nElems = 10;                  // now 10 items in array
//--------------------------------------------------------------
     for(j=0; j<nElems; j++)       // display items
        System.out.print(arr[j] + " ");
     System.out.println("");
//--------------------------------------------------------------
     searchKey = 66;               // find item with key 66
     for(j=0; j<nElems; j++)           // for each element,
        if(arr[j] == searchKey)        // found item?
           break;                      // yes, exit before end
     if(j == nElems)                   // at the end?
        System.out.println("Can't find " + searchKey); // yes
     else
        System.out.println("Found " + searchKey);      // no
//--------------------------------------------------------------
     searchKey = 55;               // delete item with key 55
     for(j=0; j<nElems; j++)           // look for it
     if(arr[j] == searchKey)
        break;
     for(int k=j; k<nElems; k++)       // move higher ones down
        arr[k] = arr[k+1];
     nElems--;                         // decrement size
//--------------------------------------------------------------
     for(j=0; j<nElems; j++)       // display items
        System.out.print( arr[j] + " ");
     System.out.println("");
     }  // end main()
  }  // end class ArrayApp
```

2.2. Problem 1: BubbleSortApp.java

- Trace the algorithm (display the array inside after inner or outer loop)
- Display the number of swaps after the inner loop
- Display the number of comparisons after the inner loop and the total number of comparisons, and estimate the algorithms' complexity (n*(n-1)/2, O(n^2))

```
class ArrayBub {
  private long[] a; // ref to array a
  private int nElems; // number of data items
  private int nSwaps; // number of swaps
  private int nComparisons; // number of comparisons

  public ArrayBub(int max) // constructor
  {
     a = new long[max]; // create the array
     nElems = 0; // no items yet
     nSwaps = 0; // no swaps yet
     nComparisons = 0; // no comparisons yet
  }

  public void insert(long value) // put element into array
  {
     a[nElems] = value; // insert it
     nElems++; // increment size
  }

  public void display() // displays array contents
  {
     for (int j = 0; j < nElems; j++) // for each element,
        System.out.print(a[j] + " "); // display it
     System.out.println("");
  }

  public void bubbleSort() {
     int out, in;

     for (out = nElems - 1; out > 1; out--) {
        System.out.print("Array: ");
        display(); // display the array
        nComparisons += out; // increment number of comparisons

        for (in = 0; in < out; in++) {
           if (a[in] > a[in + 1]) {
              swap(in, in + 1); // swap them
           }
           nComparisons++; // increment number of comparisons
```

```
        }

        System.out.println("Number of swaps after inner loop: " + nSwaps);
        System.out.println("Number of comparisons after inner loop: " + nComparisons);
      }
    }

    private void swap(int one, int two) {
      long temp = a[one];
      a[one] = a[two];
      a[two] = temp;

      nSwaps++; // increase number of swaps by 1
    }

    public int getSwapNumber() {
      return nSwaps;
    }

    public int getComparisonNumber() {
      return nComparisons;
    }
}

class BubbleSortApp {
    public static void main(String[] args) {
      int maxSize = 100; // array size
      ArrayBub arr; // reference to array
      arr = new ArrayBub(maxSize); // create the array

      arr.insert(77); // insert 10 items
      arr.insert(99);
      arr.insert(44);
      arr.insert(55);
      arr.insert(22);
      arr.insert(88);
      arr.insert(11);
      arr.insert(00);
      arr.insert(66);
      arr.insert(33);

      System.out.print("Original array: ");
      arr.display(); // display items

      arr.bubbleSort(); // bubble sort them

      System.out.print("Sorted array: ");
      arr.display(); // display them again
```

```
    // display the number of swaps and comparisons
    System.out.println("Number of swaps: " + arr.getSwapNumber());
    System.out.println("Number of comparisons: " + arr.getComparisonNumber());
    System.out.println("Total number of comparisons: " + (arr.getComparisonNumber() +
arr.getSwapNumber()));
    System.out.println("Algorithm complexity: O(n^2)");
  }
}
```

2.3. Problem 2: SelectSortApp.java

- Trace the algorithm (display the array after the inner loop)
- Print the items that are swapped. Are swaps always needed?
- Display the number of comparisons after the inner loop and the total number of comparisons, and estimate the algorithms' complexity (n*(n-1)/2, O(n^2))

```
class ArraySel {
  private long[] a;                 // ref to array a
  private int nElems;               // number of data items

  public ArraySel(int max) {        // constructor
     a = new long[max];                 // create the array
     nElems = 0;                        // no items yet
  }

  public void insert(long value) {    // put element into array
     a[nElems] = value;             // insert it
     nElems++;                      // increment size
  }

  public void display() {             // displays array contents
     for (int j = 0; j < nElems; j++) {       // for each element,
        System.out.print(a[j] + " ");  // display it
     }
     System.out.println("");
  }

  public void selectionSort() {
     int out, in, min;
     int comparisons = 0;

     for (out = 0; out < nElems - 1; out++) {   // outer loop
        min = out;                      // minimum
        for (in = out + 1; in < nElems; in++) { // inner loop
           comparisons++;  // Increment the comparison count
           if (a[in] < a[min]) {         // if min greater,
              min = in;                  // we have a new min
```

```
            }
        }
        swap(out, min);                 // swap them
        System.out.print("Swapped: " + a[out] + " and " + a[min] + " | ");
        display();
      }

      System.out.println("Number of comparisons: " + comparisons);
      System.out.println("Total comparisons: " + (comparisons + (nElems - 1)));
   }

   private void swap(int one, int two) {
      long temp = a[one];
      a[one] = a[two];
      a[two] = temp;
   }
}

class SelectSortApp {
   public static void main(String[] args) {
      int maxSize = 100;            // array size
      ArraySel arr;                 // reference to array
      arr = new ArraySel(maxSize);  // create the array

      arr.insert(77);               // insert 10 items
      arr.insert(99);
      arr.insert(44);
      arr.insert(55);
      arr.insert(22);
      arr.insert(88);
      arr.insert(11);
      arr.insert(00);
      arr.insert(66);
      arr.insert(33);

      arr.display();                // display items

      arr.selectionSort();          // selection-sort them

      arr.display();                // display them again
   }
}
```

2.4. Problem 3: InsertSortApp.java

- Trace the algorithm (display the array after each pass of the outer loop)
- Display the number of passes of the inner loop and total number of passes, and estimate the algorithms' complexity (n*(n-1)/4, O(n^2))

```
class ArrayIns {
  private long[] a;                 // ref to array a
  private int nElems;               // number of data items

  public ArrayIns(int max) {        // constructor
     a = new long[max];                 // create the array
     nElems = 0;                        // no items yet
  }

  public void insert(long value) {    // put element into array
     a[nElems] = value;             // insert it
     nElems++;                      // increment size
  }

  public void display() {             // displays array contents
     for (int j = 0; j < nElems; j++) {       // for each element,
        System.out.print(a[j] + " ");  // display it
     }
     System.out.println("");
  }

  public void insertionSort() {
     int in, out;
     int innerLoopCount = 0;
     int totalPasses = 0;

     for (out = 1; out < nElems; out++) {   // out is dividing line
        long temp = a[out];             // remove marked item
        in = out;                       // start shifts at out
        while (in > 0 && a[in - 1] >= temp) { // until one is smaller
           a[in] = a[in - 1];             // shift item to right
           --in;                        // go left one position
           innerLoopCount++;            // increment inner loop count
        }
        a[in] = temp;                   // insert marked item
        totalPasses++;                  // increment total passes
        System.out.print("Pass " + totalPasses + ": ");
        display();
     }

     System.out.println("Number of passes of the inner loop: " + totalPasses);
```

```
    System.out.println("Total number of passes: " + totalPasses);
    System.out.println("Estimated algorithm complexity: O(n^2)");
  }
}

class InsertSortApp {
  public static void main(String[] args) {
    int maxSize = 100;            // array size
    ArrayIns arr;                 // reference to array
    arr = new ArrayIns(maxSize);  // create the array

    arr.insert(77);               // insert 10 items
    arr.insert(99);
    arr.insert(44);
    arr.insert(55);
    arr.insert(22);
    arr.insert(88);
    arr.insert(11);
    arr.insert(00);
    arr.insert(66);
    arr.insert(33);

    arr.display();                // display items

    arr.insertionSort();          // insertion-sort them

    arr.display();                // display them again
  }
}
```

2.5. Problem 4

Create an array of integer numbers, fill the array with random data and print the number of **comparisons, copies, and swaps** made for sorting 10000, 15000, 20000, 25000, 30000, 35000, 40000, 45000 and 50000 items and fill in the table below. Analyze the trend for the three different algorithms.

COPIES/ COMPARISONS/ SWAPS			
	Bubble Sort	**Selection Sort**	**Insertion Sort**
10000			
15000			
20000			
25000			
30000			
35000			
40000			
45000			
50000			

```java
import java.util.Arrays;
import java.util.Random;

class SortingAnalysis {
   private static int comparisons;
   private static int copies;
   private static int swaps;

   public static void resetCounters() {
       comparisons = 0;
       copies = 0;
       swaps = 0;
   }

   public static void insertionSort(int[] arr) {
       resetCounters();

       int n = arr.length;
       for (int i = 1; i < n; ++i) {
           int key = arr[i];
           int j = i - 1;

           while (j >= 0 && arr[j] > key) {
               comparisons++;
               arr[j + 1] = arr[j];
               copies++;
               swaps++;
               j--;
           }
```

```
        if (j >= 0) {
            comparisons++;
        }

        arr[j + 1] = key;
        copies++;
    }
}

public static void selectionSort(int[] arr) {
    resetCounters();

    int n = arr.length;
    for (int i = 0; i < n - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < n; j++) {
            comparisons++;
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }

        if (minIndex != i) {
            swaps++;
            int temp = arr[i];
            arr[i] = arr[minIndex];
            arr[minIndex] = temp;
            copies += 3;
        }
    }
}

public static void bubbleSort(int[] arr) {
    resetCounters();

    int n = arr.length;
    boolean swapped;
    for (int i = 0; i < n - 1; i++) {
        swapped = false;
        for (int j = 0; j < n - i - 1; j++) {
            comparisons++;
            if (arr[j] > arr[j + 1]) {
                swaps++;
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                copies += 3;
```

```
                    swapped = true;
                }
            }

            if (!swapped) {
                break;
            }
        }
    }

    public static void main(String[] args) {
        int[] arraySizes = {10000, 15000, 20000, 25000, 30000, 35000, 40000, 45000, 50000};

        System.out.println("Array Size\tInsertion Sort\tSelection Sort\tBubble Sort");
        System.out.println("-------------------------------------------------------------");

        for (int size : arraySizes) {
            int[] arr = generateRandomArray(size);

            int[] arrCopy = Arrays.copyOf(arr, arr.length);
            insertionSort(arrCopy);
            int insertionSortComparisons = comparisons;
            int insertionSortCopies = copies;
            int insertionSortSwaps = swaps;

            arrCopy = Arrays.copyOf(arr, arr.length);
            selectionSort(arrCopy);
            int selectionSortComparisons = comparisons;
            int selectionSortCopies = copies;
            int selectionSortSwaps = swaps;

            arrCopy = Arrays.copyOf(arr, arr.length);
            bubbleSort(arrCopy);
            int bubbleSortComparisons = comparisons;
            int bubbleSortCopies = copies;
            int bubbleSortSwaps = swaps;

            System.out.printf("%d\t\t%d\t\t%d\t\t%d\n", size, insertionSortComparisons,
selectionSortComparisons, bubbleSortComparisons);
            System.out.printf(" \t\t%d\t\t%d\t\t%d\n", insertionSortCopies, selectionSortCopies,
bubbleSortCopies);
            System.out.printf(" \t\t%d\t\t%d\t\t%d\n", insertionSortSwaps, selectionSortSwaps,
bubbleSortSwaps);
            System.out.println("-------------------------------------------------------------");
        }
    }

    private static int[] generateRandomArray(int size) {
```

```
        int[] arr = new int[size];
        Random random = new Random();
        for (int i = 0; i < size; i++) {
            arr[i] = random.nextInt();
        }
        return arr;
    }
}
```

2.6. Problem 5: ObjectSortApp.java (sort the array by first name or by age)

(Option 2) Given the class Person.java that has variables of first name, last name, grade

- Add a main() method and add create an array of 10 people

- Add methods to sort the array by first name, last name, and by age.

```
import java.util.Arrays;

class Person {
    private String lastName;
    private String firstName;
    private int age;

    public Person(String last, String first, int a) {
        lastName = last;
        firstName = first;
        age = a;
    }

    public void displayPerson() {
        System.out.print("   Last name: " + lastName);
        System.out.print(", First name: " + firstName);
        System.out.println(", Age: " + age);
    }

    public String getLastName() {
        return lastName;
    }

    public String getFirstName() {
        return firstName;
    }
```

```
    public int getAge() {
        return age;
    }
}

class ArrayInOb {
    private Person[] a;
    private int nElems;

    public ArrayInOb(int max) {
        a = new Person[max];
        nElems = 0;
    }

    public void insert(String last, String first, int age) {
        a[nElems] = new Person(last, first, age);
        nElems++;
    }

    public void display() {
        for (int j = 0; j < nElems; j++) {
            a[j].displayPerson();
        }
    }

    public void insertionSortByLastName() {
        int in, out;

        for (out = 1; out < nElems; out++) {
            Person temp = a[out];
            in = out;

            while (in > 0 && a[in - 1].getLastName().compareTo(temp.getLastName()) > 0) {
                a[in] = a[in - 1];
                in--;
            }

            a[in] = temp;
        }
    }

    public void insertionSortByFirstName() {
        int in, out;

        for (out = 1; out < nElems; out++) {
            Person temp = a[out];
            in = out;
```

```
            while (in > 0 && a[in - 1].getFirstName().compareTo(temp.getFirstName()) > 0) {
                a[in] = a[in - 1];
                in--;
            }

            a[in] = temp;
        }
    }

    public void insertionSortByAge() {
        int in, out;

        for (out = 1; out < nElems; out++) {
            Person temp = a[out];
            in = out;

            while (in > 0 && a[in - 1].getAge() > temp.getAge()) {
                a[in] = a[in - 1];
                in--;
            }

            a[in] = temp;
        }
    }
}

public class ObjectSortApp {
    public static void main(String[] args) {
        int maxSize = 10;
        ArrayInOb arr = new ArrayInOb(maxSize);

        arr.insert("Evans", "Patty", 24);
        arr.insert("Smith", "Doc", 59);
        arr.insert("Smith", "Lorraine", 37);
        arr.insert("Smith", "Paul", 37);
        arr.insert("Yee", "Tom", 43);
        arr.insert("Hashimoto", "Sato", 21);
        arr.insert("Stimson", "Henry", 29);
        arr.insert("Velasquez", "Jose", 72);
        arr.insert("Vang", "Minh", 22);
        arr.insert("Creswell", "Lucinda", 18);

        System.out.println("Before sorting:");
        arr.display();

        arr.insertionSortByLastName();
        System.out.println("\nAfter sorting by last name:");
        arr.display();
```

```
        arr.insertionSortByFirstName();
        System.out.println("\nAfter sorting by first name:");
        arr.display();

        arr.insertionSortByAge();
        System.out.println("\nAfter sorting by age:");
        arr.display();
    }
}
```

LAB 03: STACK AND QUEUE

3.2. Problem 1: Simple stack application

Write a program to

- Convert a decimal number and convert it to octal form.
- Concatenate two stacks.
- Determine if the contents of one stack are identical to that of another.

```
import java.util.Stack;

public class StackOperations {
    public static void main(String[] args) {
        int decimalNumber = 123; // Change this to the decimal number you want to convert
        int originalNumber = decimalNumber;
        Stack<Integer> octalStack = new Stack<>();

        // Convert decimal to octal
        while (decimalNumber > 0) {
            int remainder = decimalNumber % 8;
            octalStack.push(remainder);
            decimalNumber /= 8;
        }

        System.out.print("Decimal " + originalNumber + " in octal: ");
        while (!octalStack.isEmpty()) {
            System.out.print(octalStack.pop());
        }
        System.out.println();

        // Concatenate two stacks
        Stack<Integer> stack1 = new Stack<>();
        stack1.push(1);
```

```
        stack1.push(2);
        stack1.push(3);
        Stack<Integer> stack2 = new Stack<>();
        stack2.push(4);
        stack2.push(5);
        stack2.push(6);

        Stack<Integer> concatenatedStack = new Stack<>();
        concatenatedStack.addAll(stack1);
        concatenatedStack.addAll(stack2);

        System.out.println("Concatenated Stack: " + concatenatedStack);

        // Determine if the contents of two stacks are identical
        boolean areIdentical = areStacksIdentical(stack1, stack2);
        System.out.println("Stack 1 and Stack 2 are identical: " + areIdentical);
    }

    // Function to determine if two stacks are identical
    public static boolean areStacksIdentical(Stack<Integer> stack1, Stack<Integer> stack2) {
        if (stack1.size() != stack2.size()) {
            return false;
        }

        for (int i = 0; i < stack1.size(); i++) {
            if (!stack1.get(i).equals(stack2.get(i)) ) {
                return false;
            }
        }

        return true;
    }
}
```

3.3. Problem 3: Undo and redo

Write a class *SpecialArray* that has:

- (8 points) an array of 20 random values
- (8 points) a function to update the value at a position in the array.
- (8 points) a function to undo the updating.
- (8 points) a function to redo the updating.
- (8 points) a function to display content of the array.

Hint: use two stacks to store the array after each operation

```
import java.util.Random;
import java.util.Stack;

public class SpecialArray {
    private int[] array = new int[20];
    private Stack<int[]> undoStack = new Stack<>();
    private Stack<int[]> redoStack = new Stack<>();

    public SpecialArray() {
        // Initialize the array with random values
        Random random = new Random();
        for (int i = 0; i < 20; i++) {
            array[i] = random.nextInt(100); // You can adjust the range of random values as needed
        }
    }

    // Function to update the value at a specific position
    public void updateValue(int position, int newValue) {
        if (position >= 0 && position < array.length) {
            undoStack.push(array.clone()); // Save the current state to the undo stack
            array[position] = newValue; // Update the value
            redoStack.clear(); // Clear the redo stack since a new change has been made
        } else {
            System.out.println("Invalid position.");
        }
    }

    // Function to undo the last update
    public void undo() {
        if (!undoStack.isEmpty()) {
            redoStack.push(array.clone()); // Save the current state to the redo stack
            array = undoStack.pop(); // Restore the previous state
        } else {
```

```
            System.out.println("Nothing to undo.");
        }
    }

    // Function to redo the last undo operation
    public void redo() {
        if (!redoStack.isEmpty()) {
            undoStack.push(array.clone()); // Save the current state to the undo stack
            array = redoStack.pop(); // Restore the previously undone state
        } else {
            System.out.println("Nothing to redo.");
        }
    }

    // Function to display the content of the array
    public void displayArray() {
        for (int value : array) {
            System.out.print(value + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        SpecialArray specialArray = new SpecialArray();
        specialArray.displayArray();

        // Example usage:
        specialArray.updateValue(5, 42);
        specialArray.displayArray();

        specialArray.undo();
        specialArray.displayArray();

        specialArray.redo();
        specialArray.displayArray();
    }
}
```

3.4. QueueApp.java

- Write a method to display the queue array and the front and rear indices. Explain how wraparound works.
- Write a method to display the queue (loop from 1 to nItems and use a temporary front for wraparound).
- Display the aray, the queue, and the front and rear indices.
- Insert fewer items or remove fewer items and investigate what happens when the queue is empty or full.
- Extend the insert and remove methods to deal with a full and empty queue.
- Add processing time to the queue. Create a new remove method that removes item N after N calls to the method.
- Simulate a queue of customers each one served for a random amount of time. Investigate how simulation is affected by:
    - the size of the queue
    - the range of time for wich each customer is served
    - the rate at which customers arrive at the queue

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Random;

class Customer {
   private int id;
   private int serviceTime;

   public Customer(int id, int serviceTime) {
       this.id = id;
       this.serviceTime = serviceTime;
   }

   public int getId() {
       return id;
   }

   public int getServiceTime() {
       return serviceTime;
   }
}

class CustomerQueue {
   private Queue<Customer> queue = new LinkedList<>();

   public void addCustomer(Customer customer) {
       queue.add(customer);
   }

   public Customer serveCustomer() {
```

```
        return queue.poll();
    }

    public boolean isEmpty() {
        return queue.isEmpty();
    }
}

public class CustomerQueueSimulation {
    public static void main(String[] args) {
        int maxQueueSize = 10; // Maximum size of the customer queue
        int maxServiceTime = 10; // Maximum service time for customers

        Random rand = new Random();
        CustomerQueue customerQueue = new CustomerQueue();

        // Simulate customer arrivals and add them to the queue
        for (int customerId = 1; customerId <= maxQueueSize; customerId++) {
            int serviceTime = rand.nextInt(maxServiceTime) + 1;
            Customer customer = new Customer(customerId, serviceTime);
            customerQueue.addCustomer(customer);
            System.out.println("Customer " + customer.getId() + " entered the queue with a service
time of " + customer.getServiceTime() + " units.");
        }

        // Serve customers in the order they arrived
        while (!customerQueue.isEmpty()) {
            Customer customer = customerQueue.serveCustomer();
            System.out.println("Serving Customer " + customer.getId() + " with a service time of " +
customer.getServiceTime() + " units.");
            try {
                Thread.sleep(customer.getServiceTime() * 100); // Simulate service time
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

3.5. ReverseApp.java

- Create a stack of objects of class Person and use to reverse a list of persons.

```java
import java.util.*;

class Person {
  private String name;
  private int age;

  public Person(String name, int age) {
     this.name = name;
     this.age = age;
  }

  public String getName() {
     return name;
  }

  public int getAge() {
     return age;
  }
}

public class ReversePersonList {
  public static void main(String[] args) {
     Stack<Person> stack = new Stack<>();
     List<Person> persons = new ArrayList<>();
     persons.add(new Person("Alice", 25));
     persons.add(new Person("Bob", 30));
     persons.add(new Person("Charlie", 35));

     for (Person person : persons) {
        stack.push(person);
     }

     List<Person> reversedPersons = new ArrayList<>();
     while (!stack.isEmpty()) {
        reversedPersons.add(stack.pop());
     }

     for (Person person : reversedPersons) {
        System.out.println(person.getName() + " " + person.getAge());
     }
  }
}
```

3.6. PriorityQApp.java

- Write a method to display the queue and use to trace the queue operation.
- Modify the insert method to insert the new item at the rear. Compare this queue with QueueApp.java. Which one is more efficient?
- Use a priority queue instead of an ordinary one in the simulation experiments described above.

```
// priorityQ.java
// demonstrates priority queue
// to run this program: C>java PriorityQApp
////////////////////////////////////////////////////////////////
class PriorityQ
  {
  // array in sorted order, from max at 0 to min at size-1
  private int maxSize;
  private long[] queArray;
  private int nItems;
//--------------------------------------------------------------
  public PriorityQ(int s)          // constructor
     {
     maxSize = s;
     queArray = new long[maxSize];
     nItems = 0;
     }
//--------------------------------------------------------------
  public void insert(long item)    // insert item
     {
     int j;

     if(nItems==0)                          // if no items,
        queArray[nItems++] = item;          // insert at 0
     else                                   // if items,
        {
        for(j=nItems-1; j>=0; j--)          // start at end,
           {
           if( item > queArray[j] )         // if new item larger,
              queArray[j+1] = queArray[j]; // shift upward
           else                             // if smaller,
              break;                        // done shifting
           }  // end for
        queArray[j+1] = item;               // insert it
        nItems++;
        }  // end else (nItems > 0)
     }  // end insert()
//--------------------------------------------------------------
```

```
  public long remove()              // remove minimum item
     { return queArray[--nItems]; }
//-------------------------------------------------------------
  public long peekMin()             // peek at minimum item
     { return queArray[nItems-1]; }
//-------------------------------------------------------------
  public boolean isEmpty()          // true if queue is empty
     { return (nItems==0); }
//-------------------------------------------------------------
  public boolean isFull()           // true if queue is full
     { return (nItems == maxSize); }
//-------------------------------------------------------------
  public void displayQueue() {      // display the queue
     for (int i = 0; i < nItems; i++) {
        System.out.print(queArray[i] + " ");
     }
     System.out.println("");
  }
//-------------------------------------------------------------
  }  // end class PriorityQ
////////////////////////////////////////////////////////////////
class PriorityQApp
  {
  public static void main(String[] args)
     {
     PriorityQ thePQ = new PriorityQ(5);
     thePQ.insert(30);
     thePQ.insert(50);
     thePQ.insert(10);
     thePQ.insert(40);
     thePQ.insert(20);

     thePQ.displayQueue(); // display the queue

     while( !thePQ.isEmpty() )
        {
        long item = thePQ.remove();
        System.out.print(item + " ");  // 10, 20, 30, 40, 50
        }  // end while
     System.out.println("");
     }  // end main()
//-------------------------------------------------------------
  }  // end class PriorityQApp
////////////////////////////////////////////////////////////////
```

LAB 04: LINK LIST

4.2. LinkList2App.java

- add a method insertAfter to insert after a particular item in the list

```
class Link {
   public int iData;
   public double dData;
   public Link next;
   public Link(int id, double dd) {
       iData = id;
       dData = dd;
   }
   public void displayLink() {
       System.out.print("{" + iData + ", " + dData + "} ");
   }
}
class LinkList {
   private Link first;
   public LinkList() {
       first = null;
   }
   public void insertFirst(int id, double dd) {
       Link newLink = new Link(id, dd);
       newLink.next = first;
       first = newLink;
   }
   public Link find(int key) {
       Link current = first;
       while (current != null) {
           if (current.iData == key) {
               return current;
           }
           current = current.next;
       }
       return null;
   }
   public Link delete(int key) {
       Link current = first;
       Link previous = first;
       while (current != null) {
           if (current.iData == key) {
               if (current == first) {
                   first = first.next;
               } else {
```

```
                previous.next = current.next;
            }
            return current;
        }
        previous = current;
        current = current.next;
    }
    return null;
}
public void displayList() {
    System.out.print("List (first-->last): ");
    Link current = first;
    while (current != null) {
        current.displayLink();
        current = current.next;
    }
    System.out.println("");
}
public void insertAfter(int key, int id, double dd) {
    Link current = first;
    while (current != null) {
        if (current.iData == key) {
            Link newLink = new Link(id, dd);
            newLink.next = current.next;
            current.next = newLink;
            return;
        }
        current = current.next;
    }
    System.out.println("Item with key " + key + " not found in the list.");
}
}
class LinkList2App {
public static void main(String[] args) {
    LinkList theList = new LinkList();

    theList.insertFirst(22, 2.99);
    theList.insertFirst(44, 4.99);
    theList.insertFirst(66, 6.99);
    theList.insertFirst(88, 8.99);
    theList.displayList();

    Link f = theList.find(44);
    if (f != null)
        System.out.println("Found link with key " + f.iData);
    else
        System.out.println("Can't find link");
```

```
        Link d = theList.delete(66);
        if (d != null)
            System.out.println("Deleted link with key " + d.iData);
        else
            System.out.println("Can't delete link");

        theList.displayList();

        theList.insertAfter(44, 55, 5.55); // Insert a new link with key 55 and data 5.55 after the
link with key 44
        theList.displayList();
    }
}
```

4.3. LinkStackApp.java

- write an application to reverse a list using a stack

```
import java.util.Stack;

class Link {
    public long dData;              // data item
    public Link next;               // next link in list

    public Link(long dd)            // constructor
    {
        dData = dd;
    }

    public void displayLink()       // display ourself
    {
        System.out.print(dData + " ");
    }
}

class LinkList {
    private Link first;             // ref to the first item on the list

    public LinkList()               // constructor
    {
        first = null;               // no items on the list yet
    }

    public Link getFirst() {
```

```
        return first;
    }

    public boolean isEmpty()        // true if the list is empty
    {
        return (first == null);
    }

    public void insertFirst(long dd) // insert at the start of the list
    {
        Link newLink = new Link(dd);
        newLink.next = first;        // newLink --> old first
        first = newLink;             // first --> newLink
    }

    public long deleteFirst()       // delete the first item
    {
        Link temp = first;           // save a reference to the link
        first = first.next;          // delete it: first --> old next
        return temp.dData;           // return the deleted link
    }

    public void displayList() {
        Link current = first;        // start at the beginning of the list
        while (current != null)     // until the end of the list
        {
            current.displayLink();   // print data
            current = current.next;  // move to the next link
        }
        System.out.println("");
    }
}

class LinkStack {
    private LinkList theList;

    public LinkStack()              // constructor
    {
        theList = new LinkList();
    }

    public void push(long j)      // put an item on top of the stack
    {
        theList.insertFirst(j);
    }

    public long pop()             // take an item from the top of the stack
    {
```

```
        return theList.deleteFirst();
    }

    public boolean isEmpty()        // true if the stack is empty
    {
        return (theList.isEmpty());
    }

    public void displayStack() {
        System.out.print("Stack (top-->bottom): ");
        theList.displayList();
    }
}

public class LinkStackApp {
    public static void reverseListUsingStack(LinkList originalList) {
        // Create a stack to store the elements from the original list.
        Stack<Long> stack = new Stack<>();

        // Traverse the original list and push each element onto the stack.
        Link current = originalList.getFirst();
        while (current != null) {
            stack.push(current.dData);
            current = current.next;
        }

        // Create a new list to store the reversed elements.
        LinkList reversedList = new LinkList();

        // Pop elements from the stack and insert them into the new list.
        while (!stack.isEmpty()) {
            long data = stack.pop();
            reversedList.insertFirst(data);
        }

        // Display the reversed list.
        System.out.print("Reversed List: ");
        reversedList.displayList();
    }

    public static void main(String[] args) {
        LinkList originalList = new LinkList();
        originalList.insertFirst(10);
        originalList.insertFirst(20);
        originalList.insertFirst(30);
        originalList.insertFirst(40);
        originalList.insertFirst(50);
```

```
        System.out.print("Original List: ");
        originalList.displayList();

        reverseListUsingStack(originalList);
    }
}
```

4.4. LinkQueueApp.java

- Put different classes in separate files.
- Create a new remove() method that removes item N after N calls to the method.
- Simulate a queue of customers each one served for a random amount of time.
- Add a size() method and investigate how simulation is affected by the time needed to serve a customer and the rate at which customers join the queue.

LinkQueue.java

```
public class LinkQueue {
    private FirstLastList theList;
    private int size;

    public LinkQueue() // constructor
    {
        theList = new FirstLastList(); // make a 2-ended list
        size = 0;
    }

    public boolean isEmpty() // true if the queue is empty
    {
        return theList.isEmpty();
    }

    public void insert(long j) // insert, rear of the queue
    {
        theList.insertLast(j);
        size++;
    }

    public long remove() // remove, front of the queue
    {
        if (isEmpty()) {
            System.out.println("Queue is empty.");
            return -1; // Replace with an appropriate value or exception handling.
        }
        size--;
        return theList.deleteFirst();
    }
```

```
    public long removeN(int n) // remove item N after N calls
    {
        if (isEmpty() || n <= 0 || n > size) {
            System.out.println("Invalid input for N.");
            return -1; // Replace with appropriate value or exception handling.
        }
        size--;
        long item = -1; // Default value if item N is not found.
        for (int i = 1; i < n; i++) {
            item = theList.deleteFirst();
            theList.insertLast(item);
        }
        return theList.deleteFirst();
    }

    public int size() {
        return size;
    }

    public void displayQueue() {
        System.out.print("Queue (front-->rear): ");
        theList.displayList();
    }
}
```

QueueSimulator.java

```
public class QueueSimulator {
    public static void main(String[] args) {
        LinkQueue customerQueue = new LinkQueue();
        int simulationTime = 100; // Total simulation time
        int currentTime = 0;

        while (currentTime < simulationTime) {
            Customer newCustomer = new Customer();
            customerQueue.insert(newCustomer.getServiceTime());

            if (customerQueue.isEmpty()) {
                System.out.println("No customers in the queue.");
            } else {
                long serviceTime = customerQueue.remove();
                System.out.println("Customer served for " + serviceTime + " units of time.");
            }

            currentTime++;
        }
    }
```

```
}
```

Link.java

```
public class Link {
   public long dData; // data item
   public Link next; // next link in list

   public Link(long d) // constructor
   {
       dData = d;
   }

   public void displayLink() // display this link
   {
       System.out.print(dData + " ");
   }
}
```

FirstLastList.java

```
public class FirstLastList {
   private Link first; // ref to first item
   private Link last; // ref to last item

   public FirstLastList() // constructor
   {
       first = null; // no items on the list yet
       last = null;
   }

   public boolean isEmpty() // true if no links
   {
       return first == null;
   }

   public void insertLast(long dd) // insert at the end of the list
   {
       Link newLink = new Link(dd); // make a new link
       if (isEmpty()) // if empty list,
           first = newLink; // first --> newLink
       else
           last.next = newLink; // old last --> newLink
       last = newLink; // newLink <-- last
   }

   public long deleteFirst() // delete the first link
   { // (assumes non-empty list)
       long temp = first.dData;
```

```
        if (first.next == null) // if only one item
            last = null; // null <-- last
        first = first.next; // first --> old next
        return temp;
    }

    public void displayList() {
        Link current = first; // start at the beginning
        while (current != null) // until the end of the list
        {
            current.displayLink(); // print data
            current = current.next; // move to the next link
        }
        System.out.println("");
    }
}
```

Customer.java

```
import java.util.Random;

public class Customer {
    private int serviceTime;

    public Customer() {
        Random rand = new Random();
        serviceTime = rand.nextInt(10) + 1; // Random service time between 1 and 10 units.
    }

    public int getServiceTime() {
        return serviceTime;
    }
}
```

4.5. Problem II: Josephus Problem

The Josephus Problem (https://en.wikipedia.org/wiki/Josephus_problem) is a famous mathematical puzzle that goes back to ancient times. There are many stories to go with the puzzle. One is that Josephus was one of a group of Jews who were about to be captured by the Romans. Rather than be enslaved, they chose to commit suicide. They arranged themselves in a circle and, starting at a certain person, started counting off around the circle. Every nth person had to leave the circle and commit suicide. Josephus decided he didn't want to die, so he arranged the rules so he would be the last person left. If there were (say) 41 people, and he was the 16^{th} person from the start of the circle, what number should he tell them to use for counting off? The problem is made much more complicated because the circle shrinks as the counting continues.

The problem—given the number of people, starting point, direction, and number to be skipped—is to choose the position in the initial circle to avoid execution.

Your task:

- Create an application that uses a circular linked list (like that in Programming Project 5.3) to model this problem.
- The application must ask for user inputs:
    - the number of people in the circle
    - the number used for counting off
    - the number of the person where counting starts (usually 1).

- The application must print out the output: the list of people being eliminated in order (assuming you go around clockwise).

See Figure 1 below for a visualization of an example. There are 41 soldiers numbered 1 through 41. They start counting at 1 and with a step size of three. The last two soldiers remaining are #16 and #31. ***Note:*** *if the counting continues with those two soldiers, then the last one remaining is #31 only.* Figure 2 shows the example input and output that your application should have.

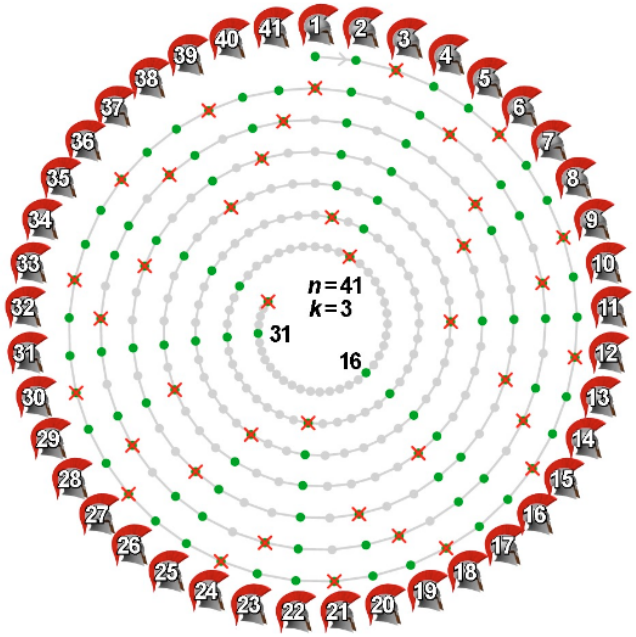


Figure 1: Claude Gaspar Bachet de Méziriac's interpretation of the Josephus problem with 41 soldiers and a step size of 3, showing that places 16 and 31 are last to be killed – time progresses inwards along the spiral, green dots denoting live soldiers, grey dead soldiers, and crosses killings.

```
============================================================

Enter the number of people in the circle: 41
Enter the number used for counting off: 3
Enter the number of the person where counting starts: 1
Elimination order:
3 6 9 12 15 18 21 24 27 30 33 36 39 1 5 10 14 19 23 28 32 37 41 7 13 20
26 34 40 8 17 29 38 11 25 2 22 4 35 16
Last person standing: 31
```

Figure 2: Example input and output

```java
import java.util.LinkedList;
import java.util.List;
import java.util.Scanner;

public class JosephusProblem {
    public static int findSurvivor(int n, int k) {
        List<Integer> people = new LinkedList<>();
        for (int i = 1; i <= n; i++) {
```

```
            people.add(i);
        }

        int index = 0;
        while (people.size() > 1) {
            index = (index + k - 1) % people.size();
            people.remove(index);
        }

        return people.get(0);
    }

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

        System.out.print("Enter the number of people in the circle: ");
        int n = scanner.nextInt();

        System.out.print("Enter the number used for counting off: ");
        int k = scanner.nextInt();

        System.out.print("Enter the number of the person where counting starts: ");
        int start = scanner.nextInt();

        int survivor = findSurvivor(n, k);
        System.out.println("Last person standing: " + survivor);
    }
}
```

LAB 05: Recursion

5.3. Problem 1: Use the following function puzzle(..) to answer problems 1 - 3.

```
int puzzle(int base, int limit)
{       //base and limit are nonnegative numbers
        if ( base > limit )
                return -1;
        else if ( base == limit )
                return 1;
        else
                return base * puzzle(base + 1, limit);
}
```

1. (10 points) Identify the base case(s) of function puzzle(..)
2. (10 points) Identify the recursive case(s) of function puzzle(..)
3. (10 points) Show what would be displayed by the following calls.
    a. System.out.print(puzzle(14,10));
    b. System.out.print(puzzle(4,7));
    c. System.out.print(puzzle(0,0));

Problem 1

1. The base case of this recursive function is when base is equal to limit. In this case, it returns 1.

2. The recursive case is when base is less than limit. In this case, it calls itself with an incremented base and multiplies the current base with the result of the recursive call.

3.

a. System.out.print(puzzle(14,10));
This call has base (14) greater than limit (10), so it will return -1.
Output: -1

b. System.out.print(puzzle(4,7));
This call has base (4) less than limit (7). It will enter the recursive case and call itself with puzzle(5, 7), then puzzle(6, 7), and finally puzzle(7, 7). When base becomes equal to limit, it returns 1. The multiplication will then unwind as follows: 1 * 1 * 1 * 1 = 1.
Output: 1

c. System.out.print(puzzle(0,0));
This call has base (0) equal to limit (0), which matches the base case. It returns 1 directly.
Output: 1

5.4. Problem 3: Write a recursive function that computes the sum of all numbers from 1 to n, where n is given as a parameter.

```
//return the sum 1+ 2+ 3+ ...+ n
int sum(int n)
```

```java
public class Problem03 {
    public static void main(String[] args) {
        int n = 5;
        int result = sum(n);
        System.out.println("The sum of numbers from 1 to " + n + " is: " + result);
    }

    public static int sum(int n) {
        if (n == 1) {
            return 1; // Base case: The sum of 1 is 1.
        } else {
            return n + sum(n - 1); // Recursive case: Add n to the sum of the first (n-1) numbers.
        }
    }
}
```

5.5. Problem 5: Write a recursive function that computes and returns the sum of all elements in an array, where the array and its size are given as parameters.

```
//return the sum of all elements in a[]
int findsum(int a[], int n)
```

```java
public class Problem05 {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        int sum = findSum(arr, arr.length);
        System.out.println("The sum of the elements in the array is: " + sum);
    }

    public static int findSum(int[] a, int n) {
        // Base case: If the array has only one element, return that element.
        if (n == 1) {
            return a[0];
        } else {
            // Recursive case: Sum the last element with the sum of the rest of the array.
            return a[n - 1] + findSum(a, n - 1);
        }
    }
}
```

5.6. Problem 7: Write a recursive function that takes a string as input and reverses it using recursion.

```java
public class Problem07 {
    public static void main(String[] args) {
        String input = "Hello, World!";
        String reversed = reverseString(input);
        System.out.println("Original String: " + input);
        System.out.println("Reversed String: " + reversed);
    }

    public static String reverseString(String str) {
        // Base case: If the string is empty or has only one character, return itself.
        if (str.isEmpty() || str.length() == 1) {
            return str;
        } else {
            // Recursive case: Reverse the substring excluding the first character,
            // and concatenate it with the first character.
            return reverseString(str.substring(1)) + str.charAt(0);
        }
    }
}
```

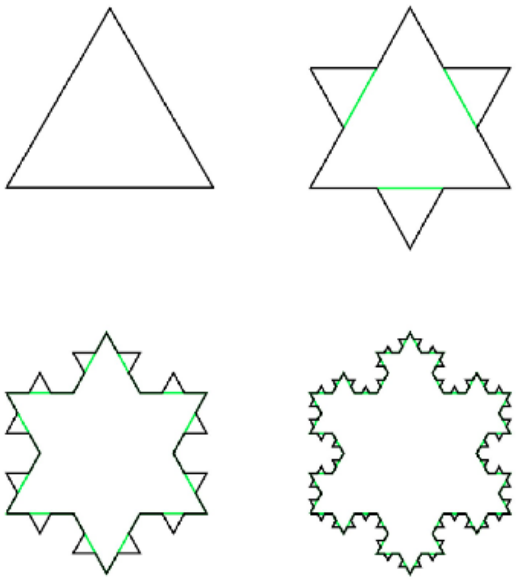
5.7. Problem 8: Write a recursive function to generate all subsets of a given set.

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

public class Problem08 {
    public static void main(String[] args) {
        int[] nums = {1, 2, 3};
        List<List<Integer>> subsets = generateSubsets(nums);
        System.out.println("All subsets of the given set:");
        for (List<Integer> subset : subsets) {
            System.out.println(subset);
        }
    }

    public static List<List<Integer>> generateSubsets(int[] nums) {
        List<List<Integer>> result = new ArrayList<>();
        generateSubsetsHelper(nums, 0, new ArrayList<>(), result);
        return result;
    }
```

```
    private static void generateSubsetsHelper(int[] nums, int index, List<Integer> currentSubset,
List<List<Integer>> result) {

        result.add(new ArrayList<>(currentSubset));


        for (int i = index; i < nums.length; i++) {

            currentSubset.add(nums[i]);

            generateSubsetsHelper(nums, i + 1, currentSubset, result);

            currentSubset.remove(currentSubset.size() - 1);
        }
    }
}
```

5.8. Problem 11: Use recursion to generate a Von Koch snowflake

https://en.wikipedia.org/wiki/Koch_snowflake



```
import javax.swing.*;
import java.awt.*;
import java.awt.geom.Line2D;
import java.util.ArrayList;
```

```
import java.util.List;

public class KochSnowflake extends JPanel {
    private List<Line2D> lines = new ArrayList<>();

    public KochSnowflake(int order) {
        setBackground(Color.WHITE);
        setPreferredSize(new Dimension(800, 650));
        // Start with an equilateral triangle
        Point p1 = new Point(400, 100);
        Point p2 = new Point(100, 650);
        Point p3 = new Point(700, 650);
        // Recursively create the snowflake
        generateSnowflake(p1, p2, order);
        generateSnowflake(p2, p3, order);
        generateSnowflake(p3, p1, order);
    }

    private void generateSnowflake(Point p1, Point p2, int order) {
        if (order == 0) {
            // Draw the line segment
            lines.add(new Line2D.Double(p1.x, p1.y, p2.x, p2.y));
        } else {
            int dx = p2.x - p1.x;
            int dy = p2.y - p1.y;
            Point p3 = new Point(p1.x + dx / 3, p1.y + dy / 3);
            Point p4 = new Point(p1.x + dx * 2 / 3, p1.y + dy * 2 / 3);
            Point p5 = new Point((int) (p3.x + Math.cos(Math.toRadians(60)) * (p4.x - p3.x) -
Math.sin(Math.toRadians(60)) * (p4.y - p3.y)),
                                 (int) (p3.y + Math.sin(Math.toRadians(60)) * (p4.x - p3.x) +
Math.cos(Math.toRadians(60)) * (p4.y - p3.y)));
            // Recurse on the four new lines
            generateSnowflake(p1, p3, order - 1);
            generateSnowflake(p3, p5, order - 1);
            generateSnowflake(p5, p4, order - 1);
            generateSnowflake(p4, p2, order - 1);
        }
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;
        for (Line2D line : lines) {
            g2d.draw(line);
        }
    }
```

```
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("Koch Snowflake");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.getContentPane().add(new KochSnowflake(4));
            frame.pack();
            frame.setLocationRelativeTo(null);
            frame.setVisible(true);
        });
    }

    static class Point {
        int x, y;

        public Point(int x, int y) {
            this.x = x;
            this.y = y;
        }
    }
}
```

LAB 06: TREES

7.2. Problems 1

(10 points) Add a method that counts the elements in a binary tree into the *Tree Class*. Specifically, the method takes no parameters and returns an integer equal to the number of elements in the tree.

7.3. Problems 2

(10 points) Add a method that computes the height of a binary tree into the *Tree Class*. Specifically, this method has no parameters and returns an integer equal to the height of the tree.

7.4. Problems 2

(10 points) Add a method that counts a binary tree's leaves tree into the *Tree Class*. Specifically, this method has no parameters and returns an integer equal to the number of leaves in the tree.

7.5. Problems 3

(10 points) Add a method that determines whether a binary tree is fully balanced. This method takes no parameters and returns a Boolean value: true if the tree is fully balanced and false if not.

7.6. Problems 4

(10 points) Define two binary trees to be identical if both are empty or their roots are equal, their left subtrees are identical, and their right subtrees are identical. Design a method that determines whether two binary trees are identical *(this method takes a second binary tree as its only parameter and returns a Boolean value: true if the tree receiving the message is identical to the parameter, and false otherwise)*.

Tree.java

```
// -------------------------------------------------------------
// Representing arithmetic expressions by binary tree
// CS 501
// Zdravko Markov
// -------------------------------------------------------------
class Tree
{
 public static void main(String[] args)
 {
     Node a = node(2);
     Node b = node(3);
     Node c = node('+',a,b);
     Node d = node(5);
     Node e = node(1);
     Node f = node('-',d,e);
     Node g = node('*',c,f);
     Node h = node(8);
     Node i = node('/',g,h);
```

```
//    Node i = node('/',node('*',node('+',node(2),node(3)),node('-',node(5),node(1))),node(8));

    System.out.println("Tree:");
    showTree(0,i);
    System.out.print("Prefix: ");
    prefix(i);
    System.out.print("\nPostfix: ");
    postfix(i);
    System.out.print("\nInfix: ");
    infix(i);
    System.out.println("\nValue: "+eval(i));
 }
// ---------------------------------------------------------------
  public static Node node (char op, Node l, Node r)
  {
     Node a = new Node();
     a.operation=op;
     a.leftChild=l;
     a.rightChild=r;
     return a;
  }
// ---------------------------------------------------------------
  public static Node node (int val)
  {
     Node a = new Node();
     a.value=val;
     return a;
  }

// ---------------------------------------------------------------
  public static void prefix (Node t)
  {
     if (t.leftChild==null && t.rightChild==null)
           System.out.print(t.value+" ");
     else
     {
         System.out.print(t.operation+" ");
         prefix(t.leftChild);
         prefix(t.rightChild);
     }
  }
// ---------------------------------------------------------------
  public static void postfix (Node t)
  {
     if (t.leftChild==null && t.rightChild==null)
           System.out.print(t.value+" ");
     else
     {
```

```
        postfix(t.leftChild);
        postfix(t.rightChild);
        System.out.print(t.operation+" ");
    }
  }
// -----------------------------------------------------------
  public static void infix (Node t)
  {
     if (t.leftChild==null && t.rightChild==null)
           System.out.print(t.value);
     else
     {
         System.out.print("(");
         infix(t.leftChild);
         System.out.print(t.operation);
         infix(t.rightChild);
         System.out.print(")");
     }
  }
// -----------------------------------------------------------
  public static double eval (Node t)
  {
     double val=0;
     if (t.leftChild==null && t.rightChild==null)
         val = t.value;
     else
         switch(t.operation)
         {
           case '+':
               val = eval(t.leftChild) + eval(t.rightChild);
               break;
           case '-':
               val = eval(t.leftChild) - eval(t.rightChild);
               break;
           case '*':
               val = eval(t.leftChild) * eval(t.rightChild);
               break;
           case '/':
               val = eval(t.leftChild) / eval(t.rightChild);
         }
         return val;
  }
// -----------------------------------------------------------
  public static void showTree (int n, Node t)
  {
      tab(n);
      if (t.leftChild==null && t.rightChild==null)
           System.out.println(t.value);
```

```
      else
      {
          System.out.println(t.operation);
          showTree(n+2,t.leftChild);
          showTree(n+2,t.rightChild);
      }
  }
// --------------------------------------------------------------
  public static void tab(int n)
  {
      for (int i=0;i<n;i++) System.out.print(" ");
  }
}
// --------------------------------------------------------------

class Node
{
  char operation;
  int value;
  Node leftChild;
  Node rightChild;
}
```

TreeApp.java

```
// TreeApp.java
// demonstrates binary tree
import java.io.*;
import java.util.*;               // for Stack class
////////////////////////////////////////////////////////////////

class TreeApp
  {
  public static void main(String[] args) throws IOException
     {
     int value;
     Tree theTree = new Tree();

     theTree.insert(50, 1.5);
     theTree.insert(25, 1.2);
     theTree.insert(75, 1.7);
     theTree.insert(12, 1.5);
     theTree.insert(37, 1.2);
     theTree.insert(43, 1.7);
     theTree.insert(30, 1.5);
     theTree.insert(33, 1.2);
     theTree.insert(87, 1.7);
     theTree.insert(93, 1.5);
```

```
    theTree.insert(97, 1.5);

    while(true)
       {
       System.out.print("\nEnter first letter of show, ");
       System.out.print("insert, find, delete, traverse, or quit: ");
       int choice = getChar();
       switch(choice)
          {
          case 's':
             System.out.print("horizontal or vertical (1 or 2)? ");
             value = getInt();
             if (value==1)
             {
                 System.out.println();
                 showTree(0,theTree.root);
             }
             else
                theTree.displayTree();
             break;
          case 'i':
             System.out.print("Enter value to insert: ");
             value = getInt();
             theTree.insert(value, value + 0.9);
             System.out.println("Comparisons = "+theTree.comps);
             break;
          case 'f':
             System.out.print("Enter value to find: ");
             value = getInt();
             Node found = theTree.find(value);
             if(found != null)
                {
                System.out.print("Found: ");
                found.displayNode();
                System.out.print("\n");
                }
             else
                {
                System.out.print("Could not find ");
                System.out.println(value);
                }
             System.out.println("Comparisons = "+theTree.comps);
             break;
          case 'd':
             System.out.print("Enter value to delete: ");
             value = getInt();
             boolean didDelete = theTree.delete(value);
             if(didDelete)
```

```
                System.out.print("Deleted " + value + '\n');
             else
                {
                System.out.print("Could not delete ");
                System.out.println(value);
                }
             System.out.println("Comparisons = "+theTree.comps);
             break;
          case 't':
             System.out.print("Enter type 1, 2 or 3: ");
             value = getInt();
             theTree.traverse(value);
             break;
          case 'q':
             return;
          default:
             System.out.print("Invalid entry\n");
          }  // end switch
       }  // end while
    }  // end main()
// --------------------------------------------------------------
 public static String getString() throws IOException
    {
    InputStreamReader isr = new InputStreamReader(System.in);
    BufferedReader br = new BufferedReader(isr);
    String s = br.readLine();
    return s;
    }
// --------------------------------------------------------------
 public static char getChar() throws IOException
    {
    String s = getString();
    return s.charAt(0);
    }
//--------------------------------------------------------------
 public static int getInt() throws IOException
    {
    String s = getString();
    return Integer.parseInt(s);
    }
// --------------------------------------------------------------
public static Node node (int data,Node l, Node r)
 {
     Node a = new Node();
     a.iData = data;
     a.leftChild=l;
     a.rightChild=r;
     return a;
```

```
  }
// --------------------------------------------------------------
  public static void showTree (int n, Node t)
  {
      tab(n);
      if (t==null)
          System.out.println("*");
      else
      {
          n=n+3;
          System.out.println(t.iData);
          if (t.leftChild==null && t.rightChild==null) return;
          showTree(n,t.leftChild);
          showTree(n,t.rightChild);
      }
  }
// --------------------------------------------------------------
  public static void tab(int n)
  {
      for (int i=0;i<n;i++) System.out.print(" ");
  }

// --------------------------------------------------------------
}  // end class TreeApp
////////////////////////////////////////////////////////////////

class Node
  {
  public int iData;              // data item (key)
  public double dData;           // data item
  public Node leftChild;         // this node's left child
  public Node rightChild;        // this node's right child

  public void displayNode()      // display ourself
     {
     System.out.print('{');
     System.out.print(iData);
     System.out.print(", ");
     System.out.print(dData);
     System.out.print("} ");
     }
  }  // end class Node
////////////////////////////////////////////////////////////////
class Tree
  {
  int comps=0;
  Node root;             // first node of tree
```

```
// --------------------------------------------------------------
  public Tree()                   // constructor
     { root = null; }             // no nodes in tree yet
// --------------------------------------------------------------
  public Node find(int key)       // find node with given key
     { comps=0;                             // (assumes non-empty tree)
     Node current = root;                   // start at root
     while(current.iData != key)            // while no match,
        { comps++;
        if(key < current.iData)             // go left?
           current = current.leftChild;
        else                                // or go right?
           current = current.rightChild;
        if(current == null)                 // if no child,
           return null;                     // didn't find it
        }
     return current;                        // found it
     }  // end find()
// --------------------------------------------------------------
  public void insert(int id, double dd)
     {comps=0;
     Node newNode = new Node();     // make new node
     newNode.iData = id;            // insert data
     newNode.dData = dd;
     if(root==null)                 // no node in root
        root = newNode;
     else                           // root occupied
        {
        Node current = root;        // start at root
        Node parent;
        while(true)                 // (exits internally)
           {
           parent = current;
           comps++;
           if(id < current.iData)   // go left?
              {
              current = current.leftChild;
              if(current == null)   // if end of the line,
                 {                  // insert on left
                 parent.leftChild = newNode;
                 return;
                 }
              }  // end if go left
           else                     // or go right?
              {
              current = current.rightChild;
              if(current == null)   // if end of the line
                 {                  // insert on right
```

```
                parent.rightChild = newNode;
                return;
                }
             }  // end else go right
          }  // end while
       }  // end else not root
    }  // end insert()
// --------------------------------------------------------------
  public boolean delete(int key) // delete node with given key
    {comps=0;                              // (assumes non-empty list)
    Node current = root;
    Node parent = root;
    boolean isLeftChild = true;

    while(current.iData != key)        // search for node
       { comps++;
       parent = current;
       if(key < current.iData)         // go left?
          {
          isLeftChild = true;
          current = current.leftChild;
          }
       else                            // or go right?
          {
          isLeftChild = false;
          current = current.rightChild;
          }
       if(current == null)             // end of the line,
          return false;                // didn't find it
       }  // end while
    // found node to delete

    // if no children, simply delete it
    if(current.leftChild==null &&
                                current.rightChild==null)
       {
       if(current == root)             // if root,
          root = null;                 // tree is empty
       else if(isLeftChild)
          parent.leftChild = null;     // disconnect
       else                            // from parent
          parent.rightChild = null;
       }

    // if no right child, replace with left subtree
    else if(current.rightChild==null)
       if(current == root)
          root = current.leftChild;
```

```
        else if(isLeftChild)
           parent.leftChild = current.leftChild;
        else
           parent.rightChild = current.leftChild;

     // if no left child, replace with right subtree
     else if(current.leftChild==null)
        if(current == root)
           root = current.rightChild;
        else if(isLeftChild)
           parent.leftChild = current.rightChild;
        else
           parent.rightChild = current.rightChild;

     else  // two children, so replace with inorder successor
        {
        // get successor of node to delete (current)
        Node successor = getSuccessor(current);

        // connect parent of current to successor instead
        if(current == root)
           root = successor;
        else if(isLeftChild)
           parent.leftChild = successor;
        else
           parent.rightChild = successor;

        // connect successor to current's left child
        successor.leftChild = current.leftChild;
        }  // end else two children
     // (successor cannot have a left child)
     return true;                                 // success
     }  // end delete()
// -------------------------------------------------------------
  // returns node with next-highest value after delNode
  // goes to right child, then right child's left descendents
  private Node getSuccessor(Node delNode)
     {
     Node successorParent = delNode;
     Node successor = delNode;
     Node current = delNode.rightChild;   // go to right child
     while(current != null)               // until no more
        {                                 // left children,
        successorParent = successor;
        successor = current;
        current = current.leftChild;      // go to left child
        }
                                          // if successor not
```

```
    if(successor != delNode.rightChild)  // right child,
       {                                 // make connections
       successorParent.leftChild = successor.rightChild;
       successor.rightChild = delNode.rightChild;
       }
    return successor;
    }
// -----------------------------------------------------------
  public void traverse(int traverseType)
    {
    switch(traverseType)
       {
       case 1: System.out.print("\nPreorder traversal: ");
               preOrder(root);
               break;
       case 2: System.out.print("\nInorder traversal:  ");
               inOrder(root);
               break;
       case 3: System.out.print("\nPostorder traversal: ");
               postOrder(root);
               break;
       }
    System.out.println();
    }
// -----------------------------------------------------------
  private void preOrder(Node localRoot)
    {
    if(localRoot != null)
       {
       System.out.print(localRoot.iData + " ");
       preOrder(localRoot.leftChild);
       preOrder(localRoot.rightChild);
       }
    }
// -----------------------------------------------------------
  private void inOrder(Node localRoot)
    {
    if(localRoot != null)
       {
       inOrder(localRoot.leftChild);
       System.out.print(localRoot.iData + " ");
       inOrder(localRoot.rightChild);
       }
    }
// -----------------------------------------------------------
  private void postOrder(Node localRoot)
    {
    if(localRoot != null)
```

```
        {
        postOrder(localRoot.leftChild);
        postOrder(localRoot.rightChild);
        System.out.print(localRoot.iData + " ");
        }
    }
// --------------------------------------------------------------
  public void displayTree()
    {
    Stack globalStack = new Stack();
    globalStack.push(root);
    int nBlanks = 32;
    boolean isRowEmpty = false;
    System.out.println(
    "......................................................");
    while(isRowEmpty==false)
       {
       Stack localStack = new Stack();
       isRowEmpty = true;

       for(int j=0; j<nBlanks; j++)
          System.out.print(' ');

       while(globalStack.isEmpty()==false)
          {
          Node temp = (Node)globalStack.pop();
          if(temp != null)
             {
             System.out.print(temp.iData);
             localStack.push(temp.leftChild);
             localStack.push(temp.rightChild);

             if(temp.leftChild != null ||
                                  temp.rightChild != null)
                isRowEmpty = false;
             }
          else
             {
             System.out.print("--");
             localStack.push(null);
             localStack.push(null);
             }
          for(int j=0; j<nBlanks*2-2; j++)
             System.out.print(' ');
          }  // end while globalStack not empty
       System.out.println();
       nBlanks /= 2;
       while(localStack.isEmpty()==false)
```

```
            globalStack.push( localStack.pop() );
         }  // end while isRowEmpty is false
      System.out.println(
      ".....................................................");
      }  // end displayTree()
// --------------------------------------------------------------
   }  // end class Tree
////////////////////////////////////////////////////////////////
```

TestIdentical.java

```
public class TestIdentical {

   public static void main(String[] args) {
       int value;
       Tree theTree = new Tree();

       theTree.insert(50, 1.5);
       theTree.insert(25, 1.25);
       theTree.insert(75, 1.75);
       theTree.insert(12, 1.12);
       theTree.insert(36, 1.36);
       theTree.insert(60, 1.60);
       theTree.insert(85, 1.85);

       Tree theTree2 = new Tree();
       theTree2.insert(50, 1.5);
       theTree2.insert(25, 1.25);
       theTree2.insert(75, 1.75);
       theTree2.insert(12, 1.12);
       theTree2.insert(36, 1.36);
       theTree2.insert(60, 1.60);
       theTree2.insert(85, 1.85);

       if (theTree.checkIdentical(theTree2))
           System.out.println("Identical");
       else
           System.out.println("Not identical");
   }

}
```

7.7. Huffman coding

(15 points) Draw a Huffman coding tree for the following text with **YOUR_NAME** below is your full name.

"I am a student at International University. My name is YOUR_NAME. I am working on a DSA lab"

```java
import java.util.HashMap;
import java.util.Map;
import java.util.PriorityQueue;

class Node implements Comparable<Node> {
    char character;
    int frequency;
    Node left, right;

    public Node(char character, int frequency) {
        this.character = character;
        this.frequency = frequency;
    }

    @Override
    public int compareTo(Node other) {
        return this.frequency - other.frequency;
    }
}

public class HuffmanTree {
    public static void main(String[] args) {
        String text = "I am a student at International University. My name is Pham Duc Dat. I am working on a DSA lab";
        Map<Character, Integer> frequencyMap = buildFrequencyMap(text);
        Node root = buildHuffmanTree(frequencyMap);

        // Visualization or further processing can be done here
        System.out.println("Huffman coding tree built successfully!");
    }

    private static Map<Character, Integer> buildFrequencyMap(String text) {
        Map<Character, Integer> frequencyMap = new HashMap<>();
        for (char c : text.toCharArray()) {
            frequencyMap.put(c, frequencyMap.getOrDefault(c, 0) + 1);
        }
        return frequencyMap;
    }

    private static Node buildHuffmanTree(Map<Character, Integer> frequencyMap) {
```

```
        PriorityQueue<Node> priorityQueue = new PriorityQueue<>();

        for (Map.Entry<Character, Integer> entry : frequencyMap.entrySet()) {
            priorityQueue.offer(new Node(entry.getKey(), entry.getValue()));
        }

        while (priorityQueue.size() > 1) {
            Node left = priorityQueue.poll();
            Node right = priorityQueue.poll();

            Node parent = new Node('\0', left.frequency + right.frequency);
            parent.left = left;
            parent.right = right;

            priorityQueue.offer(parent);
        }

        return priorityQueue.poll(); // The root of the Huffman tree
    }
}
```

7.8. Tree.java

Create different arithmetic expressions, write a method to read prefix notation and create a tree.

Hint: https://www.cs.colostate.edu/~cs165/.Fall19/recitations/L15/doc/traversal-order.html

```
import java.util.Stack;

class Tree {
    public static void main(String[] args) {
        Node a = node(2);
        Node b = node(3);
        Node c = node('+', a, b);
        Node d = node(5);
        Node e = node(1);
        Node f = node('-', d, e);
        Node g = node('*', c, f);
        Node h = node(8);
        Node i = node('/', g, h);

        System.out.println("Tree:");
        showTree(0, i);
```

```
        System.out.print("Prefix: ");
        prefix(i);
        System.out.print("\nPostfix: ");
        postfix(i);
        System.out.print("\nInfix: ");
        infix(i);
        System.out.println("\nValue: " + eval(i));

        // Example of a new prefix expression: "* + 4 5 2"
        String prefixExpression = "* + 4 5 2";
        Node treeFromPrefix = buildTreeFromPrefix(prefixExpression);

        System.out.println("\nTree from Prefix Expression:");
        showTree(0, treeFromPrefix);
        System.out.print("Prefix: ");
        prefix(treeFromPrefix);
        System.out.print("\nPostfix: ");
        postfix(treeFromPrefix);
        System.out.print("\nInfix: ");
        infix(treeFromPrefix);
        System.out.println("\nValue: " + eval(treeFromPrefix));
    }

    public static Node buildTreeFromPrefix(String prefix) {
        Stack<Node> stack = new Stack<>();

        String[] tokens = prefix.split("\\s");

        for (int i = tokens.length - 1; i >= 0; i--) {
            String token = tokens[i];

            if (isOperator(token)) {
                char operator = token.charAt(0);
                Node right = stack.pop();
                Node left = stack.pop();
                Node newNode = node(operator, left, right);
                stack.push(newNode);
            } else {
                int value = Integer.parseInt(token);
                Node newNode = node(value);
                stack.push(newNode);
            }
        }

        return stack.pop();
    }

    private static boolean isOperator(String token) {
```

```
    return token.length() == 1 && "+-*/".contains(token);
}

public static Node node(char op, Node l, Node r) {
    Node a = new Node();
    a.operation = op;
    a.leftChild = l;
    a.rightChild = r;
    return a;
}

public static Node node(int val) {
    Node a = new Node();
    a.value = val;
    return a;
}

public static void prefix(Node t) {
    if (t.leftChild == null && t.rightChild == null)
        System.out.print(t.value + " ");
    else {
        System.out.print(t.operation + " ");
        prefix(t.leftChild);
        prefix(t.rightChild);
    }
}

public static void postfix(Node t) {
    if (t.leftChild == null && t.rightChild == null)
        System.out.print(t.value + " ");
    else {
        postfix(t.leftChild);
        postfix(t.rightChild);
        System.out.print(t.operation + " ");
    }
}

public static void infix(Node t) {
    if (t.leftChild == null && t.rightChild == null)
        System.out.print(t.value);
    else {
        System.out.print("(");
        infix(t.leftChild);
        System.out.print(t.operation);
        infix(t.rightChild);
        System.out.print(")");
    }
}
```

```
    public static double eval(Node t) {
        double val = 0;
        if (t.leftChild == null && t.rightChild == null)
            val = t.value;
        else
            switch (t.operation) {
                case '+':
                    val = eval(t.leftChild) + eval(t.rightChild);
                    break;
                case '-':
                    val = eval(t.leftChild) - eval(t.rightChild);
                    break;
                case '*':
                    val = eval(t.leftChild) * eval(t.rightChild);
                    break;
                case '/':
                    val = eval(t.leftChild) / eval(t.rightChild);
            }
        return val;
    }

    public static void showTree(int n, Node t) {
        tab(n);
        if (t.leftChild == null && t.rightChild == null)
            System.out.println(t.value);
        else {
            System.out.println(t.operation);
            showTree(n + 2, t.leftChild);
            showTree(n + 2, t.rightChild);
        }
    }

    public static void tab(int n) {
        for (int i = 0; i < n; i++)
            System.out.print(" ");
    }
}

class Node {
    char operation;
    int value;
    Node leftChild;
    Node rightChild;
}
```

LAB 07: HASH TABLES

8. Lab 7: Hash Tables

8.1. Objectives

- Understand and implement Hash table

8.2. Problem

HashDoubleApp

- Display the key sequence for the initial filling of the table
- Display the hash value, the step, and the probe sequence for insert and find.
- Display the probe length for each find and insert
- Display the average probe length for the initial filling of the table
- Investigate how the load factor affects the average probe length
- Demonstrate the importance of using a prime number for the table size

HashChainApp

- Display the key sequence for the initial filling of the table
- Display the probe length for each find and insert
- Display the average probe length for the initial filling of the table
- Investigate how the load factor affects the average probe length

HashDoubleApp

```
public class HashDoubleApp {
 private int size;
 private int[] table;
 private int probeCount;

 public HashDoubleApp(int size) {
   this.size = size;
   this.table = new int[size];
 }

 public void insert(int key) {
   int hash = key % size;
   int probes = 0;
```

```
    while (table[hash] != 0) {
      hash = (hash + 1) % size;
      probes++;
    }
    table[hash] = key;
    probeCount += probes;
    System.out.println("Inserted key " + key + " at position " + hash + " with " + probes + "
probes.");
  }

  public void find(int key) {
    int hash = key % size;
    int probes = 0;
    while (table[hash] != key) {
      hash = (hash + 1) % size;
      probes++;
    }
    System.out.println("Found key " + key + " at position " + hash + " with " + probes + "
probes.");
  }

  public void printAverageProbeLength() {
    System.out.println("Average probe length: " + (double) probeCount / size);
  }

  public static void main(String[] args) {
    HashDoubleApp hashTable = new HashDoubleApp(10); // Create a hash table of size 10

    // Insert keys into the hash table
    for (int i = 1; i <= 5; i++) {
      hashTable.insert(i);
    }

    // Find keys in the hash table
    for (int i = 1; i <= 5; i++) {
      hashTable.find(i);
    }

    // Print the average probe length
    hashTable.printAverageProbeLength();
  }

}
```

HashChainApp

```
class HashChainApp {
  public HashChainApp() {
    hashTable = new HashTable();
```

```
}

// Assuming you have a HashTable class
private HashTable hashTable;
private int totalProbeLengthForInserts = 0;
private int totalInserts = 0;

// Method to insert a key into the hash table
public void insert(int key) {
  int probeLength = hashTable.insert(key);
  totalProbeLengthForInserts += probeLength;
  totalInserts++;
  System.out.println("Inserted key " + key + " with probe length " + probeLength);
}

// Method to find a key in the hash table
public void find(int key) {
  int probeLength = hashTable.find(key);
  System.out.println("Found key " + key + " with probe length " + probeLength);
}

// Method to get the average probe length for inserts
public double getAverageProbeLengthForInserts() {
  return (double) totalProbeLengthForInserts / totalInserts;
}

// Method to get the load factor of the hash table
public double getLoadFactor() {
  int capacity = (int) hashTable.getCapacity();
  if (capacity == 0) {
    return 0.0; // or some other value that makes sense in your context
  }
  return (double) hashTable.getSize() / capacity;
}

public static void main(String[] args) {
  HashChainApp app = new HashChainApp();
  app.hashTable = new HashTable();
  // Insert keys
  app.insert(1);
  app.insert(2);
  // Find keys
  app.find(1);
  // Print average probe length for inserts
  System.out.println("Average probe length for inserts: " +
app.getAverageProbeLengthForInserts());
  // Print load factor
  System.out.println("Load factor: " + app.getLoadFactor());
```

```
 }
}
```

HashTable

```
public class HashTable {

 public int insert(int key) {
   return 0;
 }

 public int find(int key) {
   return 0;
 }

 public double getSize() {
   return 0;
 }

 public double getCapacity() {
   return 0;
 }

}
```