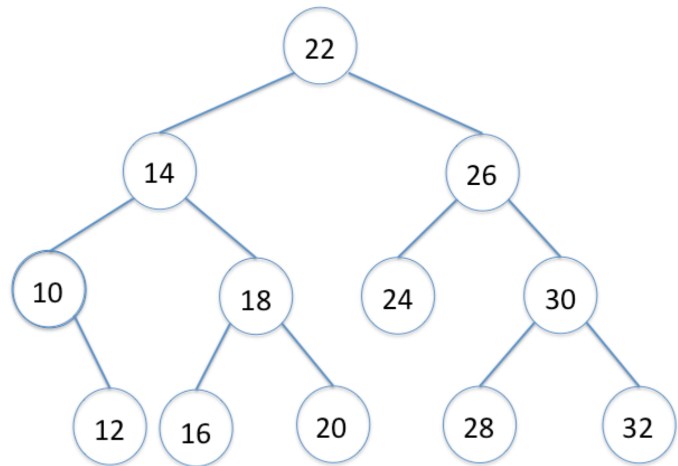
1. (20 Points) Given the following binary tree containing **int** s:



   a. (10 Points) What is the output of the following recursive method if it is initially called with the root node as a parameter:

```
public void preOrder(TreeNode n) {
   if (n != null) {
      System.out.print(n.getInt() + "  ");
      preOrder(n.getLeftChild());
      preOrder(n.getRightChild());
   } // end if
} // end preorder
```

22 14 10 12 18 16 20 26 24 30 28 32

   b. (10 Points) What is the output of the following recursive method if it is initially called with the root node as a parameter:

```
public void postOrder(TreeNode n) {
   if (n != null) {
      postOrder(n.getLeftChild());
      postOrder(n.getRightChild());
      System.out.print(n.getInt() + "  ");
   } // end if
} // end postOrder
```

12 10 16 20 18 14 24 28 32 30 26 22

2. (20 Points) What is the output of the following code:

```
String[] strings = new String[10];

for ( int i = 0 ; i < strings.length ; i++ ) {
   strings[i] = null;
} // end for

strings[2] = "Serigne"; strings[5] = "Raymond"; strings[7] = "Esmeralda";
strings[9] = "Brian";   strings[4] = "Amon";    strings[6] = "Alix";
strings[0] = "Sameh";   strings[1] = "Nicholas";

for ( int i = 0 ; i < strings.length ; i++ ) {
   try {
      switch (i) {
         case 0: System.out.println(strings[i+1].toUpperCase());
         case 1: System.out.println(strings[i+3].toUpperCase());
         case 2: System.out.println(strings[i+4].toUpperCase());
         case 3: System.out.println(strings[i-1].toUpperCase());
         case 4: System.out.println(strings[i-4].toUpperCase());
         case 5: System.out.println(strings[i+4].toUpperCase());
         case 6: System.out.println(strings[i-1].toUpperCase());
         case 7: System.out.println(strings[i+1].toUpperCase());
         case 8: System.out.println(strings[i-5].toUpperCase());
         default:
            System.out.println(strings[i-7].toUpperCase());
      } // end switch
   } catch (Exception e) {
      System.out.println("Something Wrong!!!");
   } // end try
} // end for
```

```
NICHOLAS
Something Wrong!!!
AMON
RAYMOND
SAMEH
Something Wrong!!!
ALIX
NICHOLAS
Something Wrong!!!
SERIGNE
Something Wrong!!!
SAMEH
Something Wrong!!!
BRIAN
AMON
ALIX
SAMEH
Something Wrong!!!
RAYMOND
ESMERALDA
NICHOLAS
Something Wrong!!!
Something Wrong!!!
Something Wrong!!!
SERIGNE
```

3. (10 Points) Given that the following numbers are inserted, in the given order, into a binary tree. Draw the resulting tree.
(15, 19, 13, 14, 16, 25, 18, 23, 27, 9, 11, 6, 7, 12, 26)

```
15 is the root
19 is the right child of 15
13 is the left child of 15
14 is the right child of 13
16 is the left child of 19
25 is the right child of 19
18 is the right child of 16
23 is the left child of 25
27 is the right child of 25
9 is the left child of 13
11 is the right child of 9
6 is the left child of 9
7 is the right child of 6
12 is the right child of 11
26 is the left child of 27
28 is the right child of 27
17 is the left child of 18
```

4. (10 Points) Write a recursive method to compute the nth Fibonacci number. Your method should have the following signature:

```
public static double fibonacci(double n)
```

x

```
public static double fibonacci(double n) {
      if (n == 0) {
            return 0;
      } else if (n == 1) {
            return 1;
      } else {
            return (fibonacci(n-1) + fibonacci(n-2));
      } // end if
}
```

5. (20 Points) Assuming you have a class that implements a variable length array of `int`'s, and assume that this class already has the following `private` attributes:

   a. **`ARRAY_SIZE`**, an `int` constant that defines the initial size of **`myInts`**.
   b. **`myInts`**, an array of `int`'s that is initialized to have **`ARRAY_SIZE`** capacity.
   c. **`numInts`**, an `int` variable that keeps track of the number of elements in **`myInts`**.

   Write a method that has the following signature:

   ```
   public int removeInts(int startIndex, int numToRemove)
   ```

   Your method will remove **`numToRemove`** integers from **`myInts`** starting at **`StartIndex`**, and return a **`0`** if they all were removed. In case the array does not contain enough integers to satisfy the request, do not remove any integers and return a **`-1`**.

   Note: when integers are removed from **`myInts`** you need to pack the array. Finally, if there are more that **`ARRAY_SIZE`** unused elements in **`myInts`**, you must shrink **`myInts`** so that there is never more that **`ARRAY_SIZE`** unused elements left in **`myInts`**.

```
public int removeInts(int startIndex, int numToRemove) {
        int returnCode = -1;

        // check for valid index and enough elements
        if (startIndex >= numInts) {
                return returnCode;
        } else {
                if ((startIndex + numToRemove - 1) >= numInts) {
                        return returnCode;
                } // end if
        } // end if

        // We can remove the elements
        // pack the array
        for ( int i = startIndex ; (i + numToRemove) < numInts ; i++ ) {
                int j = i + numToRemove;
                myInts[i] = myInts[j];
        } // end for

        // update numInts
        numInts -= numToRemove;

        int newMyIntsSize = myInts.length;
        while ((newMyIntsSize - numInts) > ARRAY_SIZE) {
                newMyIntsSize -= ARRAY_SIZE;
        } // end while

        // do we need to shrink myInts?
        if (myInts.length != newMyIntsSize) {
                // yes we must shrink myInts
                int[] newMyInts = new int[newMyIntsSize];
                // copy myInts into the new array
                for (int i = 0 ; i < numInts ; i++ ) {
                        newMyInts[i] = myInts[i];
                } // end for
                // switch myInts to be the new one
                myInts = newMyInts;
        } // end if

        return numToRemove;
}
```

6. (40 Points) Programming using **eclipse**:

   a. (20 Points) Create a class `SchoolKid` that is the base class for children at a school. The class should Implement the `Comparable` interface using the `name` for the comparison, The class contains the following `private` attributes:

      i. `name`, a `String` variable representing the name of the child.
      ii. `age`, an `int` variable representing the age of the child.
      iii. `teacherName`, a `String` variable representing the name of the teacher.
      iv. `greeting`, a `String` variable representing the child's favorite way to greet people.

   You should also define the following methods:

      i. `SchoolKid(String name, int age, String teacherName, String greeting)` – The only constructor for the class.
      ii. `equals(Object otherObject)` – Two SchoolKid objects are equal if, and only if, all their attributes are equal.
      iii. Getter methods for all attributes.
      iv. Setter methods for all attributes.

```
public class SchoolKid implements Comparable<SchoolKid> {
      private String name;
      private int age;
      private String teacherName;
      private String greeting;

      public SchoolKid (String name, int age, String teacherName, String
greeting) {
            this.name = name;
            this.age = age;
            this.teacherName = teacherName;
            this.greeting = greeting;
      }

      public String getName() {
            return name;
      }

      public void setName(String name) {
            this.name = name;
      }

      public int getAge() {
            return age;
      }

      public void setAge(int age) {
            this.age = age;
      }
```

```
        public String getTeacherName() {
              return teacherName;
        }

        public void setTeacherName(String teacherName) {
              this.teacherName = teacherName;
        }

        public String getGreeting() {
              return greeting;
        }

        public void setGreeting(String greeting) {
              this.greeting = greeting;
        }

        @Override
        public boolean equals (Object o) {
              if (o == null) {
                    return false;
              } else if (o instanceof SchoolKid) {
                    SchoolKid sk = (SchoolKid) o;
                    return this.name.equals(sk.getName()) &&
                           this.age == sk.getAge() &&
                           this.teacherName.equals(sk.getTeacherName()) &&
                           this.greeting.equals(sk.getGreeting());
              } // end if
              return false;
        }

        @Override
        public int compareTo(SchoolKid sk) {
              int returnCode;
              if (equals((Object) sk)) {
                    returnCode = 0;
              } else {
                    returnCode = name.compareTo(sk.getName());
              } // end if
              return returnCode;
        }
  }
```

b. (20 Points) Create a class **ExaggeratingSchoolKid** that inherits from **SchoolKid**. This class should contain the following `private` attributes:

   i. ageDiff, an `int` variable representing the number of years the exaggerating child will augment his/her age.

   Your class will override the following methods:

   i. The constructor to accept **ageDiff** as a parameter.
   ii. The accessor method for the age, reporting the age to be the actual age + **ageDiff.**
   iii. The accessor method for the greeting, returning the child's greeting concatenated with the words "I am the best".
   iv. The **equals** method to include **ageDiff** in the equality test.

```
public class ExaggeratingSchoolKid extends SchoolKid {
      private int ageDiff;

      public ExaggeratingSchoolKid(String name,
                                    int age,
                                    String teacherName,
                                    String greeting,
                                    int ageDiff) {
            super(name, age, teacherName, greeting);
            this.ageDiff = ageDiff;
      }

      public int getAgeDiff() {
            return ageDiff;
      }

      @Override
      public int getAge() {
            return super.getAge() + ageDiff;
      }

      @Override
      public String getGreeting() {
            return super.getGreeting().concat("I am the best");
      }
```

```
    @Override
    public boolean equals(Object o) {
        boolean isEqual = false;
        if (super.equals(o)) {
            if ((o != null) && (o instanceof ExaggeratingSchoolKid)) {
                ExaggeratingSchoolKid esk = (ExaggeratingSchoolKid) o;
                if (ageDiff == esk.getAgeDiff()) {
                    isEqual = true;
                } // end if
            } // end if
        } // end if
        return isEqual;
    }
}
```