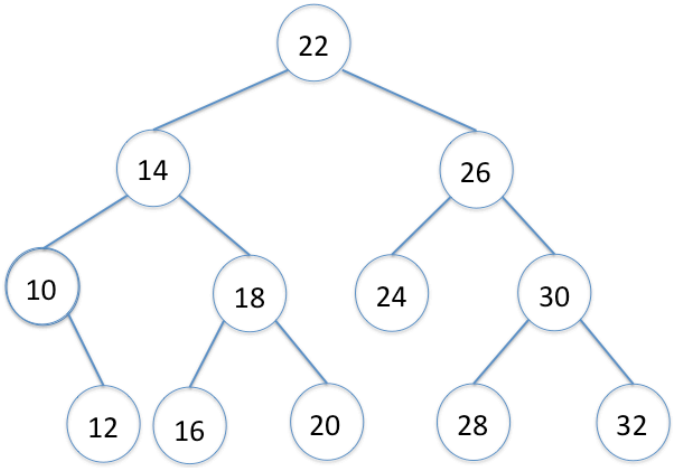
1. (20 Points) Given the following binary tree containing **int** s:

   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
```



   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
```

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
```

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)

4. (10 Points) Write a recursive method to compute the n^{th} Fibonacci number. Your method should have the following signature:

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

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`**.

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.

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.