

Q1 (40 points - 10 points per part): Study the following code to answer the question below

```
class BinaryTreeNode[A](var value: A, var left: BinaryTreeNode[A], var right:
BinaryTreeNode[A]) {}

def preOrderTraversal[A](node: BinaryTreeNode[A], f: A => Unit): Unit = {
  if (node != null) {
    f(node.value)
    preOrderTraversal(node.left, f)
    preOrderTraversal(node.right, f)
  }
}

def inOrderTraversal[A](node: BinaryTreeNode[A], f: A => Unit): Unit = {
  if (node != null) {
    inOrderTraversal(node.left, f)
    f(node.value)
    inOrderTraversal(node.right, f)
  }
}

def postOrderTraversal[A](node: BinaryTreeNode[A], f: A => Unit): Unit = {
  if (node != null) {
    postOrderTraversal(node.left, f)
    postOrderTraversal(node.right, f)
    f(node.value)
  }
}

def q1(): Unit = {
  val root = new BinaryTreeNode[String]("Rickard", null, null)
  root.right = new BinaryTreeNode[String]("Lyanna", null, null)
  root.left = new BinaryTreeNode[String]("Eddard", null, null)
  root.right.right = new BinaryTreeNode[String]("Jon", null, null)
  root.left.left = new BinaryTreeNode[String]("Sansa", null, null)
  root.left.right = new BinaryTreeNode[String]("Arya", null, null)
  root.right.right.left = new BinaryTreeNode[String]("Ghost", null, null)
  root.right.right.right = new BinaryTreeNode[String]("Rhaegal", null, null)
  root.left.right.right = new BinaryTreeNode[String]("Nymeria", null, null)
  root.left.left.left = new BinaryTreeNode[String]("Lady", null, null)

  preOrderTraversal(root, println)
  inOrderTraversal(root, println)
  postOrderTraversal(root, println)
}
```

a) Draw the tree created by running `q1()`

(For each of these questions, just list out the Strings in the specified order)

b) Write the pre-order traversal of the tree (`preOrderTraversal(root, println)`)

c) Write the in-order traversal of the tree (`inOrderTraversal(root, println)`)

d) Write the post-order traversal of the tree (`postOrderTraversal(root, println)`)

Q2 (20 points): Study the following code to answer the question below

```
class Avenger(var name: String, var isDust: Int ) { }

class BinarySearchTree[A](comparator: (A, A) => Boolean) {

  var root: BinaryTreeNode[A] = null

  def insert(a: A): Unit = {
    if(this.root == null){
      this.root = new BinaryTreeNode(a, null, null)
    }else{
      insertHelper(a, this.root)
    }
  }

  def insertHelper(a: A, node: BinaryTreeNode[A]): Unit = {
    if(comparator(node.value, a)){
      if(node.right == null){
        node.right = new BinaryTreeNode[A](a, null, null)
      }else{
        insertHelper(a, node.right)
      }
    }else{
      if(node.left == null){
        node.left = new BinaryTreeNode[A](a, null, null)
      }else{
        insertHelper(a, node.left)
      }
    }
  }
}

def q2(): Unit = {

  val comp = (a: Avenger, b: Avenger): Boolean = {
    if (a.isDust == b.isDust)
      a.name < b.name
    else
      a.isDust < b.isDust
  }

  val bst = new BinarySearchTree[Int](comp)

  bst.insert(new Avenger("Ironman", 0))
  bst.insert(new Avenger("Spiderman", 1))
  bst.insert(new Avenger("Doctor Strange", 1))
  bst.insert(new Avenger("Captain America", 0))
  bst.insert(new Avenger("Thor", 0))
}
```

```
bst.insert(new Avenger("Antman",0))
```

```
}
```

Draw the Binary Search Tree created when q2() is called

Q3 (20 points):

Write the following infix expression using postfix notation

$2 - 7 * (4 / 6 * 5) - 10 / 8$

Q4 (20 points): Study the following code to answer the question below

```
class BinaryTreeNode[A](var value: A, var left: BinaryTreeNode[A], var right:
BinaryTreeNode[A]) {
```

```
  def compute(func: (Int, A, Int) => Int): Int = {
    val leftResult = if (this.left != null) this.left.compute(func) else 3
    val rightResult = if (this.right != null) this.right.compute(func) else 7
    func(leftResult, this.value, rightResult)
  }
```

```
}
```

```
def q4(): Unit = {
  val root = new BinaryTreeNode[Int](6, null, null)
  root.left = new BinaryTreeNode[Int](12, null, null)
  root.right = new BinaryTreeNode[Int](17, null, null)
  root.left.right = new BinaryTreeNode[Int](-8, null, null)
  root.right.left = new BinaryTreeNode[Int](-10, null, null)
  root.right.right = new BinaryTreeNode[Int](1, null, null)
```

```
  val funFunction = (a:Int, b:Int, c:Int) => a - 2 * b + c
  println(root.compute(funFunction))
}
```

What is printed by the last line of q4()?