Exercise 1: Sliding Differences (5 points)

Part 1: Calculating the list of differences

There are many possible solutions to the problem. Here are some (arguably elegant) of them:

```scala
def differences(ls: List[Int]): List[Int] = {
  ((0 :: ls) zip ls) map { case (l, r) => r - l }
}

def differences2(ls: List[Int]): List[Int] = {
  @tailrec
  def loop(xs: List[Int], acc: List[Int]): List[Int] = xs match {
    case x :: Nil => acc.reverse
    case x :: y :: ys => loop(y :: ys, (y - x) :: acc)
  }
  loop(0 :: ls, Nil)
}
```

Part 2: Rebuilding the original list

There are many possible solutions to the problem. Here are some (arguably elegant) of them:

```scala
def rebuildList(ls: List[Int]): List[Int] = ls match {
  case Nil => Nil
  case x :: Nil => ls
  case x :: y :: ys => x :: rebuildList((x + y) :: ys)
}

def rebuildList2(ls: List[Int]): List[Int] = ls match {
  case Nil => Nil
  case x :: xs =>
    (xs.foldLeft(x :: Nil) {
      case (y :: acc, elem) => (elem + y) :: y :: acc
    }).reverse
}

def rebuildList3(ls: List[Int]): List[Int] =
  xs.scan(x)((l, elem) => l + elem)
```
Exercise 2: Subtyping (5 points)

We first state the following subtyping relation for functions:

\[ A_1 \Rightarrow B_1 \preceq A_2 \Rightarrow B_2 \]

iff

\[ A_1 \succ A_2 \]

and

\[ B_1 \preceq B_2 \]

1. Map\[V, Y\] is not related to Map\[W, X\].
   • Non related as Map is invariant in first type argument and \( V \neq W \).

2. Iterable[Pair[V, Y]] => V <: Map[V, Y] => V.
   • Iterable[Pair[V, Y]] >: Map[V, Y] from definition of Map.
   • V <: V.

3. Map[Map[V, Y], Y] is not related to Map[Iterable[Pair[V, X]], X].
   • Similar to 3.1, Map is invariant in first type argument and Map[V, Y] =!= Iterable[Pair[V, X]].

4. (Y => Y) => V <: (X => Y) => W.
   • As Y <: X and Y >: Y, we have (Y => Y) >: (X => Y).
   • As V <: W and (Y => Y) >: (X => Y), we have (Y => Y) => V <: (X => Y) => W.

5. W => (Y => X) is not related to W => (V => Y).
   • Y => X and V => Y are non related as Y and V are non related.

Exercise 3: Binary Search Tree (5 points)

Question 1:

```scala
def computeMinMax(b: Node): (Int, Int) = b match {
  case Node(Leaf, k, Leaf) => (k, k)
  case Node(left: Node, k, Leaf) => computeMinMax(left) match {
    case (min, max) => (Math.min(min, k), Math.max(k, max))
  }
  case Node(Leaf, k, right: Node) => computeMinMax(right) match {
    case (min, max) => (Math.min(min, k), Math.max(k, max))
  }
  case Node(left: Node, k, right: Node) =>
    (computeMinMax(left), computeMinMax(right)) match {
      case ((min1, max1),(min2, max2)) =>
        (Math.min(Math.min(min1, k), min2), Math.max(Math.max(max1, k), max2))
    }
}
```

Or

```scala
def computeMinMax(b: Node): (Int, Int) = {
  def rec(b: Node, accMin: Int, accMax: Int): (Int, Int) = b match {
    case Leaf = (accMin, accMax)
    case Node(left, k, right) =>

```
val (leftMin, leftMax) = rec(left, accMin, accMax)
val (rightMin, rightMax) = rec(right, leftMin, leftMax)
(Math.min(rightMin, k), Math.min(rightMax, k))
}
rec(b, b.value, b.value)
}

Questions 2 and 3:

There are three main approaches to solving this problem:

1) Using an inner function returning the min/max

```scala
def isBinarySearchTree(b: Tree): Boolean = {
  def rec(b: Node): (Boolean, Int, Int) = b match {
    case Node(Leaf, k, Leaf) =>
      (true, k, k)
    case Node(left: Node, k, Leaf) =>
      rec(left) match { case (b, min, max) => (b && max < k, min, k) }
    case Node(Leaf, k, right: Node) =>
      rec(right) match { case (b, min, max) => (b && min > k, k, max) }
    case Node(left: Node, k, right: Node) =>
      (rec(left), rec(right)) match {
        case ((b1, min1, max1), (b2, min2, max2)) =>
          (b1 && b2 && max1 < k && k < min2, min1, max2)
      }
  }
  b match {
    case Leaf => true
    case b: Node => rec(b)._1
  }
}
```

Or,

```scala
def isBinarySearchTree(tree: Tree): Boolean = {
  def rec(tree: Tree, minAllowed: Int, maxAllowed: Int): Boolean = tree match {
    case Leaf => true
    case Node(left, k, right) =>
      k >= minAllowed && k <= maxAllowed &&
      rec(left, minAllowed, k - 1) && rec(right, k + 1, maxAllowed)
  }
  rec(tree, Int.MinValue, Int.MaxValue)
}
```

2) Using direct functions (can be solved incrementally)

```scala
def isBinarySearchTree(b: Tree): Boolean = {
  def isBoundedStart(t: Tree): Boolean = t match {
    case Leaf => true
    case Node(left, k, right) =>
      k >= min && k <= max &&
      isBoundedStart(left, min, k - 1) && isBoundedStart(right, k + 1, max)
  }
  isBoundedStart(b)
}
```

3) Using an inner function returning the min/max
case Leaf => true
    case Node(left, k, right) =>
        isBoundedRight(left, k) && isBoundedLeft(k, right)
}

def isBoundedLeft(elem: Int, t: Tree): Boolean = t match {
    case Leaf => true
    case Node(left, k, right) =>
        k > elem && isBounded(elem, left, t) && isBoundedLeft(k, right)
}

def isBoundedRight(t: Tree, elem: Int): Boolean = t match {
    case Leaf => true
    case Node(left, k, right) =>
        k < elem && isBounded(k, right, elem) && isBoundedRight(left, k)
}

def isBounded(min: Int, t: Tree, max: Int): Boolean = t match {
    case Leaf => true
    case Node(left, k, right) =>
        k > min && k < max && isBounded(min, left, k) && isBounded(k, right, max)
}

isBoundedStart(b)

3) Using min and max (cumbersome)

def minTree(b: Node): Int = b match {
    case Node(Leaf, elem, _) => elem
    case Node(left, elem, _) => leftMostTree(left)
}

def maxTree(b: Node): Int = b match {
    case Node(_, elem, Leaf) => b
    case Node(_, elem, right) => rightMostTree(left)
}

def isBinarySearchTree(b: Tree): Boolean = b match {
    case Leaf => true
    case b: Node =>
        val min = minTree(b) - 1
        val max = maxTree(b) + 1

        def isWithinBounds(t: Tree, min: Elem, max: Elem): Boolean = t match {
            case Leaf => true
            case Node(left, elem, right) =>
                isWithinBounds(left, min, elem) &&
                min < elem && elem < max &&
                isWithinBounds(right, elem, max)
        }

        isWithinBounds(b, min, max)
Exercise 4: Structural Induction (10 points)

In addition to axioms 1 to 4, we can refer to the following formulas from the definitions of $S$ and $Bst$:

\[
\begin{align*}
(5) & \quad S(Leaf) = \{\} \\
(6) & \quad S(Node(l, e, r)) = S(l) \cup S(r) \cup \{e\} \\
(7) & \quad Bst(Leaf) = \{\} \\
(8) & \quad Bst(Node(l, e, r)) = \\
& \quad \quad Bst(l) \land \\
& \quad \quad Bst(r) \land \\
& \quad \quad (\text{for all } k \text{ in } S(l) : k < e) \land \\
& \quad \quad (\text{for all } k \text{ in } S(r) : k > e)
\end{align*}
\]

Part 1

We have to prove that:

\[
\text{For all } t : \text{Tree such that } Bst(t) \text{ and all } v : \text{Int:} \\
S(add(t, v)) = S(t) \cup \{v\}
\]

We do this by structural induction on the shape of $t$. It turns out that the part of the precondition $Bst(t)$ is irrelevant for this proof.

Base case, i.e., Leaf

We have to prove that:

\[
S(add(Leaf, v)) = S(Leaf) \cup \{v\}
\]

From the left-hand-side:

\[
\begin{align*}
S(add(Leaf, v)) \\
= S(Node(Leaf, v, Leaf)) \\
= S(Leaf) \cup S(Leaf) \cup \{v\} \\
= \{\} \cup S(Leaf) \cup \{v\} \\
= S(Leaf) \cup \{v\}
\end{align*}
\]

which is the right-hand side.
Inductive case, i.e., Node

Induction hypothesis: the property holds for two trees ℓ and r, i.e., \( S(\text{add}(l, v)) = S(l) \cup \{v\} \) and \( S(\text{add}(r, v)) = S(r) \cup \{v\} \) for any \( v: \text{Int} \).

Assuming the IH is true, we have to prove that the property holds for a bigger node \( \text{Node}(l, e, r) \), i.e., that:

\[
S(\text{add}(\text{Node}(l, e, r), v)) = S(\text{Node}(l, e, r)) \cup \{v\}
\]

We decompose this in three cases, \( v = e \), \( v < e \) and \( v > e \).

\[ v = e \] From the left-hand-side:

\[
S(\text{add}(\text{Node}(l, e, r), v))
\]
\[ (2) \]
\[ = S(\text{Node}(l, v, r)) \]  
\[ (6) \]
\[ = S(l) \cup S(r) \cup \{v\} \]

From the right-hand-side:

\[
S(\text{Node}(l, e, r)) \cup \{v\}
\]
\[ (6) \]
\[ = S(l) \cup S(r) \cup \{e\} \cup \{v\} \]
\[ v = e \]
\[ = S(l) \cup S(r) \cup \{v\} \cup \{v\} \]
\[ \text{union of sets, } \{v\} \cup \{v\} \equiv \{v\} \]
\[ = S(l) \cup S(r) \cup \{v\} \]

\[ v < e \] From the left-hand-side:

\[
S(\text{add}(\text{Node}(l, e, r), v))
\]
\[ (3) \]
\[ = S(\text{Node}(\text{add}(l, v), e, r)) \]
\[ (6) \]
\[ = S(\text{add}(l, v)) \cup S(r) \cup \{e\} \]
\[ \text{by induction hypothesis (on l)} \]
\[ = S(l) \cup \{v\} \cup S(r) \cup \{e\} \]

From the right-hand-side:

\[
S(\text{Node}(l, e, r)) \cup \{v\}
\]
\[ (6) \]
\[ = S(l) \cup S(r) \cup \{e\} \cup \{v\} \]
\[ \text{associativity of union} \]
\[ = S(l) \cup \{v\} \cup S(r) \cup \{e\} \]

\[ v > e \] Not asked.
Part 2

We have to prove that:

For all $t$: Tree such that $\text{Bst}(t)$ and all $v$: Int:
$\text{Bst}(\text{add}(t, v))$

We do this by structural induction on the shape of $t$.

Base case, i.e., Leaf

Note that we have to prove this case because $\text{Bst}(\text{Leaf})$ is true (by (7)). If $\text{Bst}(\text{Leaf})$ had been false, we should have skipped this part as trivially satisfied.

We have to prove that:

$\text{Bst}(\text{add}(\text{Leaf}, v)) \implies \text{true}$

From the left-hand-side:

$\text{Bst}(\text{add}(\text{Leaf}, v))$
(1)
$= \text{Bst}(\text{Node}(\text{Leaf}, v, \text{Leaf}))$
(8)
$= \text{Bst}(\text{Leaf}) \land \text{Bst}(\text{Leaf}) \land$
$\quad (\text{for all } k \text{ in } S(l) : k < e) \land$
$\quad (\text{for all } k \text{ in } S(r) : k > e)$
(7) applied twice
$= \text{true} \land \text{true} \land$
$\quad (\text{for all } k \text{ in } S(l) : k < e) \land$
$\quad (\text{for all } k \text{ in } S(r) : k > e)$
(5) applied twice
$= \text{true} \land \text{true} \land$
$\quad (\text{for all } k \text{ in } \{\} : k < e) \land$
$\quad (\text{for all } k \text{ in } \{\} : k > e)$
because (for all $k$ in $\{\}$ : $P$) is always satisfied regardless of $P$
$= \text{true} \land \text{true} \land \text{true} \land \text{true}$
conjunction
$= \text{true}$

Inductive case, i.e., Node

Induction hypothesis: the property holds for two trees $l$ and $r$ such that $\text{Bst}(l)$ and $\text{Bst}(r)$ are true, i.e., $\text{Bst}(\text{add}(l, v))$ and $\text{Bst}(\text{add}(r, v))$ for any $v$: Int.

Assuming the IH is true, we have to prove that the property holds for a bigger node $\text{Node}(l, e, r)$, i.e., that:

$\text{Bst}(\text{add}(\text{Node}(l, e, r), v)) \implies \text{true}$
From the precondition that $Bst(t)$ is true, we know that

$$\text{(9) } Bst(\text{Node}(1, e, r))$$

and we derive the following lemmas from (9) and (8):

$$\text{(10) } Bst(l)$$
$$\text{(11) } Bst(r)$$
$$\text{(12) } \text{(for all } k \text{ in } S(l) : k < e)$$
$$\text{(13) } \text{(for all } k \text{ in } S(r) : k > e)$$

We decompose this in three cases, $v = e$, $v < e$ and $v > e$.

$v = e$

$$Bst(\text{add}(\text{Node}(1, e, r), v))$$
$$= Bst(\text{Node}(1, v, r))$$
$$v = e$$
$$= Bst(\text{Node}(1, e, r))$$
$$= \text{true}$$

$v < e$

$$Bst(\text{add}(\text{Node}(1, e, r), v))$$
$$= Bst(\text{Node}(\text{add}(1, v), e, r))$$
$$= Bst(1, v) \&\& Bst(r) \&\&$$
$$\text{(for all } k \text{ in } S(\text{add}(1, v)) : k < e) \&\&$$
$$\text{(for all } k \text{ in } S(r) : k > e)$$
$$\text{by (10), we know that } Bst(1), \text{ and therefore by induction hypothesis}$$
$$= \text{true } \&\& Bst(r) \&\&$$
$$\text{(for all } k \text{ in } S(\text{add}(1, v)) : k < e) \&\&$$
$$\text{(for all } k \text{ in } S(r) : k > e)$$
$$\text{by (11)}$$
$$= \text{true } \&\& \text{true } \&\&$$
$$\text{(for all } k \text{ in } S(\text{add}(1, v)) : k < e) \&\&$$
$$\text{(for all } k \text{ in } S(r) : k > e)$$
$$\text{by the theorem proved in Part 1, } S(\text{add}(1, v)) === S(1) \cup \{v\}$$
$$= \text{true } \&\& \text{true } \&\&$$
$$\text{(for all } k \text{ in } S(1) \cup \{v\} : k < e) \&\&$$
$$\text{(for all } k \text{ in } S(r) : k > e)$$
$$\text{decomposing 'for all' for } S(1) \text{ and } \{v\}$$
$$= \text{true } \&\& \text{true } \&\&$$
$$\text{(for all } k \text{ in } S(1) : k < e) \&\&$$
$$\text{(for all } k \text{ in } \{v\} : k < e) \&\&$$
$$\text{(for all } k \text{ in } S(r) : k > e)$$
\[(12)\]
\[
= \text{true} \& \& \text{true} \& \& \\
\quad \text{true} \& \& \\
\quad (\text{for all } k \in \{v\} : k < e) \& \& \\
\quad (\text{for all } k \in S(r) : k > e) \\
\]
\[v < e\]
\[
= \text{true} \& \& \text{true} \& \& \\
\quad \text{true} \& \& \\
\quad \text{true} \& \& \\
\quad (\text{for all } k \in S(r) : k > e) \\
\]
\[(13)\]
\[
= \text{true} \& \& \text{true} \& \& \\
\quad \text{true} \& \& \\
\quad \text{true} \& \& \\
\quad \text{true} \\
\quad \text{conjunction} \\
\]
\[= \text{true} \]

\[v > e \quad \text{Not asked.}\]