# Practice Exam 1 Solutions

You will have 1 hour for this exam, although you should not need that long. This exam is closed-book and closed-note. Please take some time to check your work. If you need extra space, write on the back. You must show your work to receive any partial credit. There are a total of 25 points on this exam.

1. (8 points) Consider the following Scala function:

```
def m(a: Int, b: Int): (Int, Int) = {
  var x = a
  var y = 0
  while (x >= b) {
    x = x - b
    y = y + 1
  }
  (y, x)  // Return this pair
}
```

| x | y |
|---|---|
| 10 | 0 |
| 7 | 1 |
| 4 | 2 |
| 1 | 3 |

(a) What is the result of `m(10, 3)`?

(3, 1)

(b) Give an invariant relating the values of `x` and `y` each time the `while` test is evaluated:

$x = a - y \cdot b$

(c) What function is computed by `m(a, b)`? Support your claim using your invariant. You should assume that $a \geq 0$ and $b > 0$.

at end, $a = y \cdot b + x$, (invariant)

and $x < b$, (loop exit)

So $m(a, b) = (a/b, a\% b)$

quotient remainder

2. (5 points) Suppose the running time $T(N)$ of some algorithm is given by the following recurrence:

$$\begin{cases} T(1) = 1 \\ T(N) = T(N-1) + 2N - 1, & (N > 1) \end{cases}$$

(a) Fill in the following table of values. For the last entry, give a closed-form expression for $T(N)$, either by solving the recurrence or by guessing:

| $T(1)$ | $T(2)$ | $T(3)$ | $T(4)$ | $T(N)$ |
|---|---|---|---|---|
| 1 | 4 | 9 | 16 | $N^2$ |

$$\begin{aligned} T(N) &= T(N-1) + 2N - 1 \\ &= T(N-2) + 2(N-1) - 1 + 2N - 1 \\ &= T(N-3) + 2((N-2) + (N-1) + N) - 3 \\ &= \ldots = T(1) + 2(2 + \ldots + (N-1) + N) - (N-1) \\ &= 1 + 2(2 + \ldots + N) - N + 1 \\ &= 2 + 2 \cdot \frac{(N-1)(N+2)}{2} - N \\ &= N^2 \end{aligned}$$

(b) Prove by induction that your closed-form expression for $T(N)$ is correct.

Base case: $T(1) = 1^2 = 1$ ✓

Ind. step: suppose $T(N) = N^2$, for some $N \geq 1$,

then $T(N+1) = T(N) + 2(N+1) - 1$

$= N^2 + 2N + 1$, by I.H.

$= (N+1)^2$ ✓

So true for all $N \geq 1$.

3. (12 points) Here is our Scala code for inserting a value in a binary search tree:

```
trait Tree
case object Empty extends Tree
case class Node(left: Tree, value: Int, right: Tree) extends Tree

def insert(t: Tree, n: Int): Tree = t match {
  case Empty => Node(Empty, n, Empty)
  case Node(l, v, r) =>
    if (n == v)  // No change -- already in tree
      t
    else if (n < v)
      Node(insert(l, n), v, r)
    else  // n > v
      Node(l, v, insert(r, n))
}
```

(a) Complete the following skeleton to define a function `insertAll` which takes a tree and a list of numbers and returns a new tree with all of the numbers inserted into the original tree:

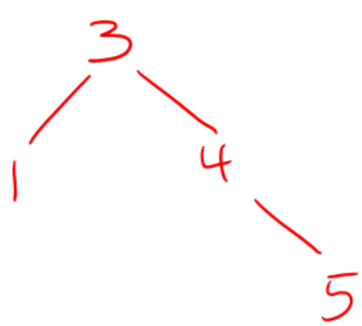
```
def insertAll(t: Tree, nums: List[Int]): Tree = nums match {
  case Nil =>  t

  case head :: tail =>
        insert(insertAll(t, tail), head)

}   (or,  insertAll(insert(t, head), tail)   also works)
```

(b) Show the tree which results from evaluating `insertAll(Empty, List(3, 1, 4, 1, 5))`:

or



(continued)

(c) Give a tight big-oh upper bound on the average running time of `insertAll` in terms of the size of the list, $N$ (assume that the initial tree is empty, and that the resulting tree is "balanced"):

insert is $O(\log N)$, so

insertAll is $O(\log 1 + \log 2 + \ldots + \log(N-1) + \log N)$

$= O(N \log N)$

(d) Here is a version of inorder traversal which returns the visited items in a list (the `:::` operator concatenates two lists; assume for this problem that this can be done in constant time):

```
def inorder(t: Tree): List[Int] = t match {
  case Empty => Nil
  case Node(l, v, r) => inorder(l) ::: List(v) ::: inorder(r)
}
```

Now we may define the following function:

```
def doSomething(nums: List[Int]): List[Int] = inorder(insertAll(Empty, nums))
```

What is the result of `doSomething(List(3, 1, 4, 1, 5))`?

= inorder( insertAll (Empty, List(3,1,4,1,5)))

= inorder (


) = List(1,3,4,5)

(e) Describe the effect of `doSomething(nums)` on an arbitrary list `nums` of type `List[Int]`:

it sorts nums and removes duplicates

(f) Give a tight big-oh upper bound on the average running time of `doSomething` in terms of the size of its argument, $N$:

insertAll (nums) is $O(N \log N)$, giving a tree w/ $\leq N$ nodes;

for inorder, $T(N) = T(N/2) + O(1) + T(N/2)$, assuming balance

$= 2T(N/2) + O(1)$

so $T(N) = O(N)$

hence doSomething is $O(N \log N)$