Problem 1

a. $i \leq n \land t = \sum_{j=0}^{i} j$

b. Base case: $i = 0$, $t = 0$. $\sum_{j=0}^{i} j = \sum_{j=0}^{0} j = 0 \rightarrow$ base case holds

c. Let $i \leq n \land t = \sum_{j=0}^{i} j$ hold through the k-th iteration.
$i_{\text{new}} = i + 1$
$t_{\text{new}} = t + i_{\text{new}} = t + i + 1$
$i \leq n$ holds because if $i < n$ was false after the k-th iteration, then the $k+1$-th iteration wouldn't have happened since the loop exit condition would've been met.
$\sum_{j=0}^{i_{\text{new}}} j = i_{\text{new}} + \sum_{j=0}^{i} j = i_{\text{new}} + t = t_{\text{new}} \rightarrow t = \sum_{j=0}^{i} j$ holds.
Thus, the loop invariant holds for all iterations.

d. $\neg(i < n) \land i \leq n \land t = \sum_{j=0}^{i} j$
$i \geq n \land i \leq n \land t = \sum_{j=0}^{i} j = i$
$i == n \land t = \sum_{j=0}^{i} j = i$
$t = \sum_{j=0}^{n} j = n(n+1)/2 \rightarrow$ the loop exit condition and loop invariant implies the postcondition

e. $D = n - i$
Before the loop, $i = 0$, so $D = n - 0 = n \geq 0$.
In each iteration, $i_{\text{new}} = i + 1$, so after each iteration, $D = n - i_{\text{new}} = n - (i+1) = n - i - 1$. So, D decreases after each iteration.
When $D = n - i = 0$, $i == n$, meeting the loop's exit condition ($i \geq n$), terminating the loop.

Problem 2

a. I tested to see if loopysqrt(n) works on $n = 0, 1, 4, 9, 16, 25, 36, 49$ by setting a variable, a, to loopysqrt(n) and then seeing if $a^2 == n$. I also printed their results since assertions don't do anything when a Dafny program is compiled and ran. I got back true for all the tests.

b. I checked to see if loopysqrt(2) * loopsqrt(2) = 2 and found that it doesn't (printed false). This violates the postcondition (root * root != n).

c. I fixed the bug by handling what would happen when root * root != n at the end of the method, making it so that it would return -1 in those cases. This makes it so that the method is expected to return -1 when given an integer n that isn't a perfect square. loopysqrt(2) * loopsqrt(2) = 2 still returns false, but this matches my expectations now.

d. I changed the postcondition to root * root == n || root == -1. This is a necessary change to account for when n isn't a perfect square, showing that -1 will be returned.

e. My code doesn't verify because the postcondition couldn't be verified. This doesn't necessarily mean that there's a bug since I haven't given a loop invariant and decrement function, making it impossible for Dafny to verify the loop does what it should do.

f. I guessed the invariant by looking at the loop's condition, precondition, and postcondition. I knew that if n is a perfect square, root * root == n by the time the loop terminates. I also realized that $\sum_{i=1}^{\text{root}} 2i - 1$ was essentially root * root, so I figured out that $n - a$ had to be root * root while $a \geq 0$. When $a < 0$, root*root is greater than $n - a$, meaning that n isn't a perfect square. So, I handled the two cases by making the loop invariant ($a \geq 0$ && $n - a ==$ root*root) || $a < 0$. Then, I made the decrement function $n-$ root*root since at the end of the loop, root*root must be $\geq n$, making the loop terminate when the decrement function is ≤ 0. My code was failing before adding the invariant and decrement function since Dafny couldn't prove by itself that the loop was returning the expected value and that it was actually terminating.

g. I didn't have to do anything after removing the precondition. This now allows the client to input any integer they want, and now the expectation should be that if loopysqrt wasn't given a perfect square (any non-perfect square integer that can be < 0), -1 will be returned.

h. Prior to the loop, root $= 0 \wedge a = n$.

Loop invariant proof:
<u>Base case</u>: If $n < 0$, then $a < 0$. Else, $a \geq 0 \wedge n - a = 0 \wedge$ root*root $= 0$.

<u>Induction</u>: Assume ($a \geq 0 \wedge n - a =$ root*root) $\vee\ a < 0$ holds through iteration k. a must be > 0 or else iteration $k+1$ wouldn't happen.
$\text{root}_{\text{new}} = \text{root}+1 \rightarrow {\text{root}_{\text{new}}}^2 = \text{root}^2 + 2 * \text{root} + 1$
$a_{\text{new}} = a - (2 * \text{root}_{\text{new}} - 1) = a - (2 * \text{root} + 2 - 1) = a - 2 * \text{root} - 1$
Case 1: $a_{\text{new}} < 0 \rightarrow$ loop invariant holds
Case 2: $a_{\text{new}} \geq 0$
$n - a = \text{root}^2 \rightarrow a = n - \text{root}^2$
$n - a_{\text{new}} = n - (a - 2 * \text{root} - 1) = n - (n - \text{root}^2 - 2 * \text{root} - 1) = \text{root}^2 + 2 * \text{root} + 1 = {\text{root}_{\text{new}}}^2$

<u>Invariant + exit condition implies postcondition</u>: The postcondition of the entire method is root * root $= n \vee$ root $= -1$, but root only gets set to -1 in a check after the loop. So, the postcondition of this loop is ($a = 0 \wedge$ root * root $= n$) $\vee\ a < 0$.
$a \leq 0 \wedge ((a \geq 0 \wedge n - a = \text{root*root}) \vee a < 0) = (a = 0 \wedge n - a = \text{root*root}) \vee a < 0$

<u>Loop termination</u>: $D = n -$ root*root
Prior to the loop, root*root$= 0$ and the loop is only entered if $a = n > 0$, so $D > 0$.
During the loop, root gets incremented, so D decreases in the loop.
When the loop terminates, $a \leq 0$. If $a = 0$, then $n - a = n =$ root*root, so $D = 0$. If $a < 0$, then subtracting $2 *$ root -1 made a go from being positive to negative, meaning that $\sum_{i=0}^{\text{root}} 2i - 1 > n$. $\sum_{i=0}^{\text{root}} 2i - 1 = 2\sum_{i=0}^{\text{root}} i - \sum_{i=0}^{\text{root}} 1 =$ root(root + 1) - root $= \text{root}^2$, and so $\text{root}^2 > n$, making $D < 0$ on termination.

After the loop, we have ($a = 0 \wedge$ root * root $= n$) $\vee\ a < 0$. Then if $a < 0$, root gets set to -1. So, the postcondition becomes ($a = 0 \wedge$ root * root $= n$) $\vee$ ($a < 0 \wedge$ root $= -1$) $\rightarrow$ root * root $= n \vee$ root $= -1$.

Problem 3

a. $\forall n, 0 \leq n < a$: diffs[n] = arr[$n+1$] - arr[n] $\wedge$ $a \leq$ diffs.Length $\wedge$ diffs.Length = arr.Length-1

b. Before the loop, $a = 0 \leq$ diffs.Length, diffs.Length = arr.Length-1, and there are no n in the range $0 \leq n < 0$, satisfying the loop invariant.

c. Assume $\forall n, 0 \leq n < a$: diffs[n] = arr[$n+1$] - arr[n] $\wedge$ $a \leq$ diffs.Length $\wedge$ diffs.Length = arr.Length-1 holds through the k-th iteration.

   After the k-th iteration, $a = k$, diffs.Length = arr.Length-1, and $\forall n, 1 \leq n < k$: diffs[$n-1$] = arr[n] - arr[$n-1$].
   If a = diffs.Length after this iteration, the $(k+1)$-th iteration won't happen, making $a \leq$ diffs.Length and satisfying the loop invariant.

   $(k+1)$-th iteration:
   diffs[a] = arr[$a+1$] - arr[a]
   diffs[k] == arr[$k+1$] - arr[k] $\wedge$ $\forall n, 0 \leq n < k$: diffs[n] = arr[$n+1$] - arr[n]
   $\forall n, 0 \leq n < k+1$: diffs[n] = arr[$n+1$] - arr[n]
   $a_{\text{new}} = a + 1$
   $a_{\text{new}} = k+1 \leq$ diffs.Length $\wedge$ $\forall n, 0 \leq n < k+1$: diffs[n] = arr[$n+1$] - arr[n]
   $a_{\text{new}} = k+1 \leq$ diffs.Length $\wedge$ $\forall n, 0 \leq n < a_{\text{new}}$: diffs[n] = arr[$n+1$] - arr[n]

   diffs.Length never changes, keeping diffs.Length = arr.Length-1 true, and thus, the loop invariant holds through all iterations.

d. $a \geq$ diffs.Length $\wedge$ $\forall n, 0 \leq n < a$: diffs[n] = arr[$n+1$] - arr[n] $\wedge$ $a \leq$ diffs.Length $\wedge$ diffs.Length = arr.Length-1

   a = diffs.Length = arr.Length-1 $\wedge$ $\forall n, 0 \leq n < a$: diffs[n] = arr[$n+1$] - arr[n]

   diffs.Length = arr.Length-1 $\wedge$ $\forall n, 0 \leq n <$ diffs.Length: diffs[n] = arr[$n+1$] - arr[n]

   Thus, the loop invariant and exit condition imply the postcondition.

e. D = diffs.Length$-a$
   Prior to the loop, $a = 0$, so D = diffs.Length-0 = diffs.Length.
   Through each iteration, $a = a+1$, so D decreases by 1.
   When a = diffs.Length, $D = 0$ and the loop's exit condition is met, terminating the loop.

Problem 4

a.
```
method dutch(arr[0..N-1]) {
  i,k = 0
  while i < N {
    if arr[i] == 'r' {
      temp = arr[i]
      arr[i] = arr[k]
      arr[k] = temp
      k++
    }
    i++
  }
  return arr, k
}
```

b. $\forall n, 0 \leq n < k : \text{arr}[n] == \text{'r'} \wedge \forall n, k \leq n < N : \text{arr}[n] == \text{'b'} \wedge 0 \leq k \leq N$

c. $k \leq i \leq N \wedge \forall n, 0 \leq n < k : \text{arr}[n] == \text{'r'} \wedge \forall n, k \leq n < i : \text{arr}[n] == \text{'b'}$