```
1. 14 points
One possible solution (possibly not the best one):

Loop Invariant 3 points:
i <= N &&
    total = a[0]+ … a[i-1] &&
    L = [a[j] :: (forall j :: 0 <= j < i ➔ a[j] > a[0] + … a[j-1])]
The last condition can be expressed in words something like L contains all
elements such that a[i] is greater than the sum of the preceding elements
(from 0 to i-1). Make sure what they say is clear, if not deduct points.

Base case (3 points):
i = 0; L=[]; total=0
i <= N -> 0 <= N
total = a[0]+…a[i-1] -> total = 0 an empty range has no sum and total is
set to 0.
L = [a[j] :: (forall j :: 0 <= j < i ➔ a[j] > a[0]+…+a[j-1])] ->
      L = [] because i = 0 and range has no elements
All three parts of loop invariant hold, so base case holds.
They can express parts of this in words as long as logic is correct and
clear. If the loop invariant from the previous step is incorrect, give
points if the logic fits their stated invariant.

Induction (5 points):
Assume invariant hold for step k
i = k
k <= N && total(k) = a[0]+…a[k-1] &&
    L(k) = [a[j] :: (forall j :: 0 <= j < k ➔ a[j] > a[0]+…+a[j-1])]
total(k) is the total at step k, L(k) is the list at step k

Step k+1
i = k+1
0 <= k+1 <= N if k == N, we would have exited the loop, so at k+1 either
k+1 < N or k+1 == N -> k <= N.

total(k+1) = total(k) + a[k] = a[0]+…a[k-1] + a[k]
if a[k] > total(k)
      L(k+1) = [L(k), a[k]] i.e. a[k] is appended to list
      L(k+1) = [a[j] :: (forall j :: 0 <= j < k+1
                    ➔ a[j] > a[0]+…+a[j-1])]
else
      L(k) == L(k+1) = [a[j] :: (forall j :: 0 <= j < k+1 ➔ a[j] >
a[0]+…+a[j-1])]
i.e, list hasn't changed so L(k+1) holds
All three parts hold so invariant holds
They can express parts of this in words as long as logic is correct.
If they state the LI incorrectly, but have correct logic for induction
give points.
They must somehow state both parts. Take off 2 points if they leave off
the else part.

Decrement function (3 points): D = N-i
At iteration k.
D(k) = N-k
```

```
D(k+1) = N-(k+1)=D(k)-1 < D(k)  D decreases at each iteration
D = 0 -> N == i which is loop exit condition
1 point off if they don’t argue that it decreases.
```

Take points off depending on how far off the proof is.

2\. 15 points

Pseudocode (8)

```
m = 0
k = 0
while (m < N)
        if a[m] is red
                swap(a,k,m)
                k = k +1
        m = m + 1
```

There can be variations of the above. They don’t have to use swap, but the method should make sense. Don’t worry about efficiency. There are may ways to solve this.

Postcondition: a[0..k-1] is all red && a [k..N-1] is all blue (3 pts)

Loop invariant: a[0..k-1] is all red && a [k..m-1] is all blue && m <= N (4 pts)

Give partial credit for good attempts. The loop invariant must imply the post condition on exit. The loop invariant should match the pseudocode.

3\. 20 points

Dafny code: 9 points, 3 point each invariant or assertion

```
function Factorial(n: int): int
  requires n >= 0
{
  if n == 0 then 1 else n * Factorial(n-1)
}

method LoopyFactorial(n: int) returns (u: int)
  requires n >= 0
  ensures u == Factorial(n)
  {
    u := 1;
    var r := 0;
    while (r < n)
      invariant r <= n && u == Factorial(r)
    {
      var v := u;
      var s := 1;
      while (s<=r)
        invariant s <= r+1 && u == Factorial(r)*s
      {
        u:=u+v;
        s:=s+1;
      }
      r:=r+1;
      assert (u == Factorial(r));
```

```
    }
  }
```

Proof:
There are two loops, prove the inner loop first. Assume the outer loop invariant to prove the inner loop and then use the result of inner loop to prove the outer.

Inner Loop
Invariant of inner loop: s <= r+1 && u == r! * s

Base case of inner loop: (3 points)
s = 1, u=r! from outer loop invariant
s <= r + 1, r's minimum value at first iteration is 0 -> 1 <= 1.
r increases with each iteration, s <= r + 1 so 1 <= r + 1 for every outer iteration.
u = r! * s, u = r! * 1
Both conditions hold, base of inner loop case holds.

Induction: (3 points)
Assume: s(k) <= r(k) + 1 && u(k) == r! * s(k)
s(k) < r or we would have exited loop, s(k+1) <= r
u(k+1) = u(k) + v = u(k) + r! (from outer loop invariant)
s(k+1) = s(k) + 1
u(k+1) = u(k) + r! = r! * s(k) + r! = r!*(s(k)+1) = r!*s(k+1)
Both conditions hold, so inner loop invariant holds, given outer loop invariant

Outer loop:
Invariant: r <= n && u == r!

Base case: (2 points)
r = 0; n >= 0 -> r <= n
u = 1; r = 0; r! = 0! = 1 = u
Both conditions hold, base case holds

Induction: (3 points)
Assume: r(k) <= n && u(k) == r(k)!
r(k+1) = r(k)+1
r(k) < n otherwise loop would have exited -> r(k+1) = r(k) + 1 <= n
u(k+1) = r(k)! * s from inner loop invariant
s = r(k)+1 loop exit condition from inner loop
u(k+1) = r(k)! * s = r(k)! * (r(k)+1) = (r(k)+1)! = r(k+1)!
Both conditions hold, inductive case holds

Grading rubric: Correct invariants, 3pts each and correct assertion, 3pts. If their invariants missed the bound on the induction variable (r <= n), give just 2pts.
If they followed the proof structure even if they got the invariants wrong in the Dafny part, then give points if logic is correct. They don't have to be rigorous as above, if logic is correct.

If in doubt about the Dafny code, plug it in at http://rise4fun.com/Dafny/tutorial/Guide If it verifies, it's ok even if there are variations in the invariant and assert statements from what is shown above, as long as the basic code remains the same.

Collaboration and Reflection (0.5 points each)
Points for any reasonable answers.