David Pick
CM 2403

I spent 4 hours on this assignment.

```
//@ requires l <= r && r < a.length;
//@ assignable a[l..r];
//@ ensures a[r] == (\max int k; l <= k && k <= r; a[k]);
public void bubble(int[] a, int l, int r);


//@ requires i <= j && j < a.length;
//@ assignable a[i], a[j];
//@ ensures a[i] == \old(a[j]) && a[j] == \old(a[i]);
public void exchange(int[] a, int i, int j);


//@ requires n >= 0;
//@ assignable \nothing;
//@ ensures \result == (\product int k; 1 <= k && k <= n; k);
public int fact(n);

//-------------------------------------------------------------------
------

//@ assert true && y >= 0;
x = fact(y);
//@ assert x == (\product int k; 1 <= k && k <= y; k);

//-------------------------------------------------------------------
------

//@ assert true && 5 >= 0;
x = fact(5);
//@ assert x == (\product int k; 1 <= k && k <= 5; k);
// (\product int k; 1 <= k && k <= 5; k) ==> 120
//@ assert x == 120

//-------------------------------------------------------------------
------

//@ requires lt <= rt && rt < a.length;
//@ assignable a[lt..rt];
//@ ensures a[rt] == (\max int k; lt <= k && k <= rt; a[k]);

//@ assert lt <= rt && rt <a.length
//@ assert true && lt <= rt;
//@ assert a[lt] == (\max int k; lt <= k && k <= lt; a[k]) && lt <=
rt;
public void bubble(int[] a, int lt, int rt) {
//@ assert a[lt] == (\max int k; lt <= k && k <= lt; a[k]) && lt <= rt;
```

```
  int i = lt;
//@ assert  a[i] == (\max int k; lt <= k && k <= i; a[k]) && i <= rt;
    /*@
      @ maintaining a[i] == (\max int k; lt <= k && k <= i; a[k]);
      @ maintaining i <= rt;
      @ decreasing -i;
      @*/

//@ assert  a[i] == (\max int k; lt <= k && k <= i; a[k]) && i <= rt;
  while (i < rt) {
 //@ assert a[i] == (\max int k; lt <= k && k <= i; a[k]) && i <= rt &&
i < rt;

 //@ assert a[i] == (\max int k; lt <= k && k <= i; a[k]) && i <= rt &&
i < rt;
    if(a[i] > a[i+1]) {
 //@ assert a[i] == (\max int k; lt <= k && k <= i; a[k]) && i <= rt
&& i < rt && (a[i] > a[i + 1]);
 //@ assert a0 == a[i + 1] && a1 == a[i] && a[i] == (\max int k; lt <=
k && k <= i; a[k]) &&
 //  i < rt && i <= rt && (a[i] > a[i + 1]) && i <= j && j < a.length;
      exchange(a, i, i+1);
 //@ a[i] == a0 && a[i + 1] == a1 && a[i] == (\max int k; lt <= k && k
<= i; a[k]) && i < rt && i <= rt;
 //@ assert a[i] == (\max int k; lt <= k && k <= i; a[k]) && i < rt &&
i <= rt && a[i] <= a[i + 1];
    }
 //@ assert a[i] == (\max int k; lt <= k && k <= i; a[k]) && i < rt &&
a[i] <= a[i + 1];
 //@ assert a[i + 1] == (\max int k; lt <= k && k <= i + 1; a[k]) && i
+ 1 <= rt;
    i' = i + 1;
 //@ assert a[i'] == (\max int k; lt <= k && k <= i'; a[k]) && i' <=
rt;
  }
 //@ assert a[i] == (\max int k; lt <= k && k <= i; a[k]) && i <= rt
&& !(i < rt);
 //@ assert a[i] == (\max int k; lt <= k && k <= i; a[k]) && i == rt;
 //@ assert a[rt] == (\max int k; lt <= k && k <= rt; a[k]);
//@ assert a[rt] == (\max int k; lt <= k && k <= rt; a[k]);
}

//-------------------------------------------------------------------
------

The bubble method is totally correct because when it makes the call to
exchange it switches a[i] and a[i + 1] if a[i] is
larger than a[i + 1]. Since there are only a finite number of times it
can make this exchange, we know that
the bubble method must at some point terminate.

//-------------------------------------------------------------------
```

```
------

 //@ assert 1 <= a.length;
/*@ assert a.length - 1 >= 0 &&
@  (\forall int k; a.length - 1 < k && k < a.length; a[k-1] <= a[k]) &&
@ (\forall int j; 0 <= j && j <= a.length - 1;
@ a[j] <= (\min int k; a.length - 1 < k && k < a.length;a[k]));
@*/

i = a.length - 1;
  /*@
    @ maintaining i >= 0;
    @ maintaining (\forall int k; i < k && k < a.length; a[k-1] <=
a[k]);
    @ maintaining (\forall int j; 0 <= j && j <= i;
    @                     a[j] <= (\min int k; i < k && k < a.length;
a[k]));
    @ decreasing i;
    @*/

/*@ assert i >= 0 &&
@  (\forall int k; i < k && k < a.length; a[k-1] <= a[k]) &&
@ (\forall int j; 0 <= j && j <= i;
@ a[j] <= (\min int k; i < k && k < a.length;a[k]));
@*/
while (i > 0) {
/*@ assert i >= 0 &&
@  (\forall int k; i < k && k < a.length; a[k-1] <= a[k]) &&
@ (\forall int j; 0 <= j && j <= i;
@ a[j] <= (\min int k; i < k && k < a.length;a[k])) && i > 0;
@*/
/*@ assert i >= 0 &&
@  (\forall int k; i < k && k < a.length; a[k-1] <= a[k]) &&
@ (\forall int j; 0 <= j && j <= i;
@ a[j] <= (\min int k; i < k && k < a.length;a[k])) && i > 0 &&
@  a[i] == (\max int k; 1 <= k && k <= r; a[k]);
@*/
  bubble(a, 0, i);

//The forall expressions say that a[] must be ordered, therefore
//the max expression is captured in the forall's

/*@ assert i - 1 >= 0 &&
@  (\forall int k; i - 1 < k && k < a.length; a[k-1] <= a[k]) &&
@ (\forall int j; 0 <= j && j <= i - 1;
@ a[j] <= (\min int k; i - 1 < k && k < a.length;a[k]));
@*/
  i' = i - 1;
/*@ assert i' >= 0 &&
@  (\forall int k; i' < k && k < a.length; a[k-1] <= a[k]) &&
@ (\forall int j; 0 <= j && j <= i';
@ a[j] <= (\min int k; i' < k && k < a.length;a[k]));
```

```
@*/

}
/*@ assert i >= 0 &&
@  (\forall int k; i < k && k < a.length; a[k-1] <= a[k]) &&
@ (\forall int j; 0 <= j && j <= i;
@ a[j] <= (\min int k; i < k && k < a.length;a[k])) &&i <= 0;
@*/

  //@ assert (\forall int k; 0 < k && k < a.length; a[k-1] <= a[k]);



//-----------------------------------------------------------------------
------

//@ requires n >= 0;
//@ assignable \nothing;
//@ ensures \result == (\product int k; 1 <= k && k <= n; k);
public int fact(n) {
//@ assert n >= 0;
  int r = 0;
  if (n == 0) {
 //@ assert n >= 0 && n == 0;
 //@ assert n == 0;
 //@ assert 1 == (\product int k; 1 <= k && k <= n; k) && n == 0;
 //@ assert 1 == (\product int k; 1 <= k && k <= n; k);
    r = 1;
 //@ assert r == (\product int k; 1 <= k && k <= n; k);
  } else {
 //@ assert n >= 0 && n != 0;
 //@ assert n > 0;
    r = fact(n - 1);
 //@ assert r * n == (\product int k; 1 <= k && k <= n - 1; k) * n;
    r' = r * n;
 //@ assert r' == (\product int k; 1 <= k && k <= n - 1; k) * n;
 //@ assert r' == (\product int k; 1 <= k && k <= n; k);
  }
//@ assert r == (\product int k; 1 <= k && k <= n; k);
  return r;
}

//-----------------------------------------------------------------------
------
The fact method must terminate at some point because we know that the
argument
passed to the method decreases with each call. At some point it must
become
zero meaning the method will terminate
//-----------------------------------------------------------------------
------
```