"lo" element is swapped with itself. Finally, the recursive call in each case takes the same time.

14.5

On an 850Mhz Pentium III, insertion sort takes 0.010 seconds when the array is sorted in ascending order, and 0.541 seconds when sorted in reverse order, The reason is that when the array is in order, inserting the rightmost element into the sorted portion to the left takes a single step, with no recursive call to `insertInOrder`. However, when the array is sorted in reverse, the `insertInOrder` into a partition of size k requires k recursive calls.

14.6

On an 850Mhz Pentium III, quicksort takes 0.020 seconds when the array is sorted in ascending order, and 0.010 seconds when sorted in reverse order. (This computer is so fast that it would take an array of nearly a million elements to require 5 seconds; however, stack overflow occurs with such a large array.) The reason that the two versions don't take the same amount of time is that the partitioning algorithm happens to work well on one, but not as well on the other. Detailed analysis of the "best case" and "worst case" arrangements is difficult.

14.7

```
int sum (IntList list) {
  int s = 0;
  while (list != null) {
    s = s + list.getValue();
    list = list.getTail();
  }
}
```

14.8

Here's a direct client version of the previous `sum` method. This is not an instance method of the class `IntList`, but would be present in a separate client.

```
int sum (IntList list) {
  if (list == null)
    return list.getValue();
  else
    return list.getValue() + sum(list.getTail());
  }
}
```

Here's an instance method (of `IntList` to sum recursively. An important difference is that this method can only be applied to an `IntList` object that is not `null`, so the `IntList` contains at least one element. The previous versions of `sum` could be applied to `null IntLists`.

```
int sum () {
  if (tail == null)
    return value;
  else
    return value + tail.sum();
  }
}
```

14.9

```
public void addAfterNth_mut (int n, int x) {
  if (n==0) {
    IntList l = new IntList(x, tail);
    tail = l;
  } else if (tail != null)
    tail.addAfterNth_mul(n-1, x);
}
```

14.10

```
public IntList addNth_mut (int n, int x) {
  if (n == 0)
    return new IntList(x, this);
  else if (n == 1) {
    IntList l = new IntList(x, tail);
    tail = l;
    return this;
  } else if (tail != null) {
    tail.addNth_mut(n-1, x);
    return this;
  }
}
```

14.11

```
1  public class Binom {
2
3    static int cost (int m, int n) {
4      if (n > 0 && m > 0)
5        return 1 + cost(m-1, n) + cost(m-1, n-1);
6      else
7        return 0;
8    }
9
10   public static void main (String[] args) {
11     System.out.println(cost(30, 15));
12   }
13 }
```

and

```
public class BinomCache {

  static int[][] cache;

  private static int costCache (int m, int n) {
    if (cache[m][n] == -1) {
      int ans;
      if (n > 0 && m > 0)
        ans = 1 + costCache(m-1, n) + costCache(m-1, n-1);
      else
        ans = 0;
      cache[m][n] = ans;
    }
    return cache[m][n];
  }

  public static int cost (int m, int n) {
    cache = new int[m+1][n+1];
    for (int i=0; i<m+1; i++)
      for (int j=0; j<n+1; j++)
        cache[i][j] = -1;
    return costCache(m, n);
  }

  public static void main (String[] args) {
    System.out.println(cost(30, 15));
  }
}
```