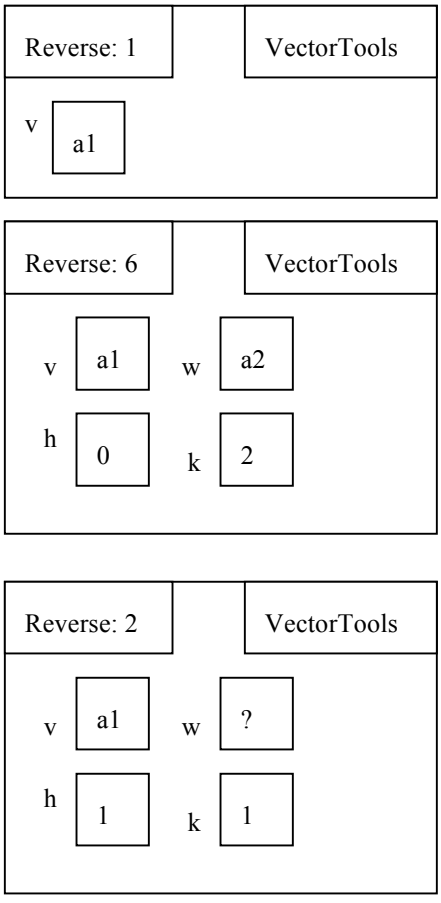
Question 1.

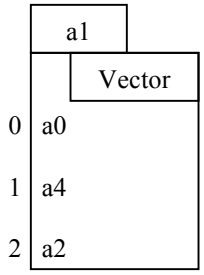
```
/** Let x be the value currently in b[h]. Permute
      b[h..k] and return an int j that satisfies
      b[h..j-1] <= b[j] = x < b[j+1..k]. */
public static int partition(int[] b, int h, int k) {
   int j= h;  int t= k;
   /* inv: b[h..j-1] <= b[j] = x <= b[t+1..k] */
   while (j < t) {
      if (b[j+1] <= b[j])
           { Swap b[j+1] and b[j];    j= j+1; }
      else { Swap b[j+1] and b[t];    t= t–1; }
   }
   return j;
}
```

Question 2.

```
int p= k-1;
int h= k-1;
// invariant: as on test
while (0 <= p) {
    if (b[p] != b[p+1]) {
         b[h]= b[p];
         h= h-1;
    }
    p= p-1;
}
```

Question 3





Question 4. (a) The throw statement is illegal because class ExException is not a subclass of class Throwable.

(b) 1: 5
5: 7
9: 7
3: 5
4: 5

Question 5.

```
/** Fill the elements of b[h..k][h..k] with integers n,
n+1, n+2, ..., in spiral fashion. */
public static void spiral(int[][] b, int h
                                        int k, int n) {
    if (h > k) { return; }

    if (h == k) {
       b[h][h]= n; return;
    }

    n= fillRow(b, h, h, k, true, n); // Fill first row
    n= fillCol(b, h+1, k, k, true, n); // Fill last col
    n= fillRow(b, k, h, k-1, false, n); // Fill last row
    n= fillCol(b, h+1, k-1, h, false, n); // Fill first col

    spiral(b, h+1, k-1, n);
}
```

Question 6.

```
Knicks vs Nets: 100 pts. Total: 100
Knicks vs Nets: 120 pts. Total: 220
Knicks vs Nets: 120 pts. Total: 340
g1 = g2: false
g2 = g3: false
Knicks vs Nets: 80 pts. Total: 300
Knicks vs Nets: 120 pts. Total: 300
Knicks vs Nets: 120 pts. Total: 300
Knicks vs Nets: 80 pts. Total: 300
g1 = g2: false
g1 = g4: true
```

Question 7.

```
public void makeInstructor(Instructor t) {
     if (t == person)
       return;
     if (person != null)
       removeInstructor();
```

```
    person= t;
    person.addCourse(this);
}

public void removeInstructor() {
    if (person == null)
       return;
    Instructor in= person;
    person= null;
    in.removeCourse(this);
}

public void addCourse(Course c) {
     if (courses.contains(c)) {
        return;
     }
     courses.add(c);
     c.makeInstructor(this);
  }

public void removeCourse(Course c) {
     if (!courses.contains(c)) {
        return;
     }
     courses.remove(c);
     c.removeInstructor();

  }
```

Question 8.

(a) The layout manager for a JFrame is a Border-Layout manager. With it, you can put up to five components in the JFrame, in the center, east, west, north, and south.

The layout manager for a Box is a BoxLayout manager. You get to say whether you want a horizontal box or a vertical box. The components appear in the row or column is the order in which they were added.

(b) 1. How does it start (how to initialize variables to make the invariant true)?

2. When does it stop (choose the loop condition B so that !b and the invariant implies the postcondition.

3. How does the repetend make progress toward termination?

4. How does the repetend keep the invariant true?

(c) For each primitive type in Java (e.g. **int**), there is a corresponding *wrapper class* (e.g. Integer). An object of the wrapper class contains (wraps) exactly one value of the primitive type. Wrapper classes serve two purposes. (1) They allow us to deal with primitive values as objects, and (2) they provide static fields and functions that deal with the type (e.g. field Integer.MAX_VALUE and function Integer.parseInt(v)).

(d) To execute the statement s= s + character; a new object of class String is created and the original String s along with c is copied into the new object. The time it takes to do this is proportional to the length of the initial String s. So, when revealing character after character, repeatedly, this statement takes time proportional to 1, 2, 3, 4, …, 4999; the sum of these values is about $5000^2 / 2$. So, reveal is taking time proportional to the square of the length of the message, and that is the problem.