1. Solution and criteria

Matching answer or –2

A. 4

B. wheels: 6, 2 wheel drive  (missing : or , okay)

C. wheels: 4 wheels: 6, 6 wheel drive (missing : or , okay)

D. 20

E. 55

F. Infinite Recursion or Runtime Error or Stack Overflow Error or Out of Memory Error or Exception

G. 12,502,500 or 5000(5001)/2 or evidence of attempting to compute 5000(5001)/2

H. N or O(N)

I. N log N or O( N log N ) (if base 2 included on log okay)

J. 24

2. Model solution

The biggest problems were with people not tracking cells they had already visited. Because water can flow to equal elevations in this problem not tracking this and not moving to a cell if it has already been visited leads to infinite recursion. A number of people also did not check elevations to see if a move was legal.

```java
public boolean drains(int[][] world, int row, int column)
{
    public boolean[][] visited = new boolean[world.length][world[0].length];
    // all set to false when created so no need to initialize
    return recursiveDrains(world, row, column, visited, world[row][column]);
}

public boolean recursiveDrains(int[][] world, int row, int col, boolean[][] visited, int previousElevation)
{
    boolean result = false;
    if(row == 0 || row == world.length – 1 || col == 0 || col == world[0].length – 1)
        result = true;
    else if( visited[row][col] || world[row][col] > previousElevation )
        result = false;
    else
    {
        visited[row][col] = true;
        currentElevation = world[row][col];
        result = recursiveDrains(world, row – 1, col, visited, currentElevation)
            || recursiveDrains(world, row, col + 1, visited, currentElevation)
            || recursiveDrains(world, row, col - 1, visited, currentElevation);
        visited[row][col] = false; // not really necessary
    }
}"
```
3. Model solution

A large number of people altered the original image while calculating. This does not meet the post condition. Also a large number of people treated org and the result as matrices of integers rather than as matrices of Pixel objects.

```java
public Pixel[][] filter(Pixel[][] org) {
    Pixel[][] result = new Pixel[org.length][org[0].length];
    int total = 0;
    int numNeighbours = 0;
    for(int r = 0; r < org.length; r++)
        for(int c = 0; c < org[0].length; c++)
            for(int rc = r - 1; rc < r + 2; rc++)
                for(int cc = cc - 1; cc < c + 2; cc++)
                    if( rc >= 0 && rc < world.length && cc >= 0 && cc < world[0].length && !(rc == r && cc == c) )
                        total += org[rc][cc].getColor();
                        numNeighbours++;
    result[r][c] = new Pixel( total / numNeighbours );
    return result;
}
```

4. Model solution

Most students who lost points did so because they did not consider all four cases or forgot to update mySize.

```java
public void insert( Comparable val ) {
    SortableNode temp = new SortableNode();
    temp.data = val; //prev and next initialized go null
    if( mySize == 0 ) //first node in list
        head = temp;
    tail = temp;
}
else if( val.compareTo( head.data ) < 0 ) // new head
    temp.next = head;
    head.prev = temp;
    head = temp;
}
else if( val.compareTo( tail.data ) >= 0 ) // new tail
    temp.prev = tail;
    tail.next = temp;
    tail = temp;
}
else // general case
    {
        SortableNode spot = head.next;
        // start at second element and find correct spot
        while( val.compareTo(spot.data) >= 0)
        {
            spot = spot.next;
            temp.next = spot;
            temp.prev = spot.prev;
            spot.prev.next = temp;
            spot.prev = temp;
        }
        mySize++;
    }

5. Model solution

Most people who lost points did so because they tried to Sort swimmers instead of times or because they sorted Times and assumed the Swimmers were automatically sorted as well. There were also many people who did not handle the case when there are multiple swimmers with the same time and the problems this can cause.

Overview: Find the swimmers with the fastest four 50 free times and move them to the front of the swimmers list. Then copy them to the resulting list.

Steps:

1. finding the fastest swimmer
2. consider the first element of the list the fastest
3. go to the next element in the list to see if this swimmer has a better time.
   3.a if so that swimmer becomes the swimmer with the fastest time
4. continue to look at the rest of the swimmers in the list until all have been considered.
5. When all swimmers have been considered swap the fastest swimmer and the swimmer in the first spot of the list. This moves the fastest swimmer to the front of the list.
6. repeat steps 2 – 4 starting with the second, then third, then fourth swimmers. (This will move the swimmers with the four fastest times in the 50 free to the front of swimmers.)
7. Copy the first four elements of the swimmer list (now the fastest four) to the results list.

public ArrayList getRelay(ArrayList swimmers)
{
    Swimmer temp;
    int indexOfFastest = 0;
    final int SWIMMERS_PER_RELAY = 4;
    Time bestTime, candidate;

    for(int i = 0; i < SWIMMERS_PER_RELAY; i++)
    {
        indexOfFastest = i;
        for(int j = 0; j < swimmers.size(); j++)
        {
            bestTime = ((Swimmer)swimmers.get(indexOfFastest)).get50FreeTime();
            candidate = ((Swimmer)swimmers.get(j)).get50FreeTime();
            if( candidate.compareTo(best) < 0 )
            {
                indexOfFastest = j;
            }
        }
        temp = (Swimmer)swimmers.get(i);
swimmers.set(i, swimmers.get(indexOfFastest));
swimmers.set(indexOfFastest, temp);

ArrayList result = new ArrayList();
for(int i = 0; i < SWIMMERS_PER_RELAY; i++)
    result.add( swimmers.get(i) );

return result;