

Solutions

Each answer shows the steps, then the **result**.

2.1

```
public static int rowTotal(int[][] g, int r)
{
    int total = 0;
    for (int v : g[r]) total += v;
    return total;
}
```

- total starts at 0
- loops over the elements of row **r** only
- returns the total

2.2

- totals 3, 5, 5; a later equal total does not replace it: **1**

2.3

- $r + 1 < g.length \ \&\& \ g[r + 1][c] == 0$

2.4

- top edge, not a corner: **3**

3.1 B

- row 0: only $3 > 1$; row 1: only $6 > 2$; $1 + 1 = 2$

3.2 B

- the bound is checked **first**, against the number of **columns**

3.3 B

- each $g[r][c]$ goes to $t[c][r]$: the rows become columns (the transpose)

3.4

(a)

```
public static int emptyPlots(int[][] crops)
{
    int count = 0;
    for (int[] row : crops)
        for (int y : row)
            if (y == 0) count++;
    return count;
}
```

- traverses every element

- tests $== 0$
- counts and returns

(b)

```
public static boolean isolated(int[][] crops, int r, int c)
{
    if (crops[r][c] == 0) return false;
    if (r > 0 && crops[r - 1][c] != 0) return false;
    if (r < crops.length - 1 && crops[r + 1][c] != 0) return false;
    if (c > 0 && crops[r][c - 1] != 0) return false;
    if (c < crops[0].length - 1 && crops[r][c + 1] != 0) return false;
    return true;
}
```

- returns **false** for an empty plot
- checks up and down with bounds first
- checks left and right with bounds first
- returns **false** as soon as one neighbour is not empty
- returns **true** only after all checks