- For $n>0$, the if conditional and the first return statement are executed. So each recursive call with $n>0$ adds two to the step count.
- Since there are n such function calls and these are followed by one with $n=0$, the step count for the function is $2n+2$.
- Surprisingly, the recursive function has a lower step count than its iterative counterpart. But, the step count only tells us how many steps are executed, it does not tell us how much time each step takes.
- The recursive version, on avg, run slowly compared to the iterative version.

Ex-1.11 We want to determine the step-count for a function that adds two-dimensional arrays.

```
void add(int a[][MAX_SIZE], int b[][MAX_SIZE], int c[][MAX_SIZE]
, int rows, int cols)
{
int i,j;
for(i=0; i<rows; i++) {
  count++; /* for i for loop */
  for(j=0; j<cols; j++) {
   count++; /* for j for loop */
   c[i][j] = a[i][j] + b[i][j];
   count++; /* for assignment statement */
  }
  count++; /* last time of j for loop */
}
count++; /* last time of i for loop */
}
```

- If count = 0 initially, on termination, count = rows + rows.cols + rows.cols + rows + 1
  = $\boxed{2.rows.(cols+1)+1}$ ✓

- Another way to obtain step counts is to use a tabular method. To construct a step count table we first determine the step count for each statement. We call this the **steps/execution** or s/e for short. Next we figure out