

Solutions

1.1 Base Cases

Reference example

```
class Solver {  
    int sum(int n) {  
        if (n == 0) return 0;  
        return n + sum(n - 1);  
    }  
}  
System.out.println(new Solver().sum(4));
```

1.1.1 - Explain

What goes wrong with $\text{sum}(n) = n + \text{sum}(n - 1)$ if it has no base case?

There is no stopping condition, so the computation continues making calls until the runtime cannot grow the call stack further.

1.1.2 - Write code

Write a recursive factorial for nonnegative n and show $\text{factorial}(0)$ and $\text{factorial}(5)$.

Use 1 as the empty-product base result and decrease n .

```
class Solver {  
    long fact(int n) {  
        if (n == 0) return 1;  
        return n * fact(n - 1);  
    }  
}  
Solver s = new Solver();  
System.out.println(s.fact(0));  
System.out.println(s.fact(5));
```

Expected output

```
1  
120
```

Solutions

1.2 The Call Stack

Reference example

```
class Solver {  
    void visit(int n) {  
        if (n == 0) return;  
        System.out.print(n + " ");  
        visit(n - 1);  
        System.out.print(n + " ");  
    }  
}  
new Solver().visit(2);
```

Recursive calls and returns



Print on descent: 2, 1 Print on return: 1, 2

Each call retains its own local state until its child returns.

1.2.1 - Explain

For visit(3) in the example, give the printed sequence.

The sequence is 3 2 1 1 2 3. Each call prints before descending and again after its child returns.

Solutions

1.2.2 - Write code

Write a recursive method that prints 1 through 3 in increasing order by making the recursive call before printing.

The deeper call completes the smaller prefix first.

```
class Solver {  
    void up(int n) {  
        if (n == 0) return;  
        up(n - 1);  
        System.out.print(n + " ");  
    }  
}  
new Solver().up(3);
```

Expected output

1 2 3

Solutions

1.3 Recursive Correctness

Reference example

```
class Solver {  
    int count(int[] a, int i) {  
        if (i == a.length) return 0;  
        return (a[i] > 0 ? 1 : 0) + count(a, i + 1);  
    }  
}  
System.out.println(new Solver().count(new int[]{-1, 3, 4}, 0));
```

1.3.1 - Explain

State a decreasing measure for `count(a, i)`.

The number of unprocessed elements, `a.length - i`, decreases by one on each recursive call until it is zero.

1.3.2 - Explain

Why is checking the base case alone not a full correctness argument?

It covers only the smallest input. Also show the recursive subproblem is smaller and that combining its correct result produces the required result for the current input.

Solutions

1.4 Recursion Versus Iteration

Reference example

```
int sum = 0;
for (int n = 4; n > 0; n--) sum += n;
System.out.println(sum);
```

1.4.1 - Explain

Compare auxiliary space for a simple depth- n recursive factorial and a loop-based factorial.

The direct recursion uses $O(n)$ call-stack space. The loop uses $O(1)$ auxiliary space under a fixed-width arithmetic model.

1.4.2 - Write code

Implement an iterative factorial for $n = 5$.

Accumulate a product starting at 1.

```
long product = 1;
for (int n = 2; n <= 5; n++) product *= n;
System.out.println(product);
```

Expected output

120