Recursion Day 1 — Worked Solutions

APCS MC Practice — Answer Key with Traces

Problem: f1

```
public int f1(int a) {
    if (a <= 1) { return a; }
    else { return a + f1(a - 2); }
}
```

This function adds every other integer going down by 2 until reaching 1 or below.

f1(1) = `1`

→ a=1 ≤ 1, so return 1

f1(4) = `6`

→ f1(4) = 4 + f1(2)
→ f1(2) = 2 + f1(0)
→ f1(0) = 0 (base case: 0 ≤ 1)
→ Unwind: 2 + 0 = 2, then 4 + 2 = 6

f1(8) = `20`

→ f1(8) = 8 + f1(6) = 8 + 6 + f1(4) = 8 + 6 + 4 + f1(2) = 8 + 6 + 4 + 2 + f1(0)
→ f1(0) = 0
→ Sum: 8 + 6 + 4 + 2 + 0 = 20

Problem: f2

```
public void f2(int a) {
    if (a == 0) { return; }
    else {
        System.out.println(a);
        f2(a-1);
    }
}
```

Print happens BEFORE the recursive call, so we print in descending order.

f2(5) outputs: `5, 4, 3, 2, 1`

→ Print 5, call f2(4)
→ Print 4, call f2(3)
→ Print 3, call f2(2)
→ Print 2, call f2(1)
→ Print 1, call f2(0)
→ Base case reached, return

Problem: f3

```
public void f3(int a) {
    if (a == 0) { return; }
    else {
        f3(a-1);
        System.out.println(a);
    }
}
```

Print happens AFTER the recursive call, so prints occur on the way back up (ascending order).

f3(5) outputs: `1, 2, 3, 4, 5`

→ f3(5) calls f3(4) calls f3(3) calls f3(2) calls f3(1) calls f3(0)
→ f3(0) returns (base case)
→ f3(1) prints 1, returns
→ f3(2) prints 2, returns
→ f3(3) prints 3, returns
→ f3(4) prints 4, returns
→ f3(5) prints 5, returns

Problem 8: something(4, 6)

```
public int something(int a, int b) {
    if (b <= 1) { return a; }
    else { return something(a, b-1); }
}
```

This function just decrements b until b ≤ 1, then returns a. It never modifies a.

Answer: `(A) 4`

→ something(4,6) → something(4,5) → something(4,4) → something(4,3) → something(4,2) → something(4,1)
→ b=1 ≤ 1, return a = 4

Problem 11: mystery1

```
public int mystery1(int x, int y) {
    if (x < y) { return x; }
    else { return mystery1(x - y, y); }
}
```

This computes x % y (modulo). It repeatedly subtracts y from x until x < y.

mystery1(6, 13) = `6`

→ 6 < 13 is true, return 6 immediately

mystery1(8, 2) = `0`

→ mystery1(8,2) → mystery1(6,2) → mystery1(4,2) → mystery1(2,2) → mystery1(0,2)
→ 0 < 2 is true, return 0

mystery1(14, 10) = `4`

→ mystery1(14,10) → mystery1(4,10)
→ 4 < 10 is true, return 4

Problem 13: mystery3

```
public int mystery3(int n) {
    if (n < 0) { return -mystery3(-n); }
    else if (n < 10) { return n; }
    else { return mystery3(n / 10 + n % 10); }
}
```

This computes the digital root: repeatedly sums digits until single digit. Handles negatives by preserving sign.

mystery3(6) = 6

→ 6 < 10, return 6

mystery3(17) = 8

→ mystery3(17) = mystery3(17/10 + 17%10) = mystery3(1 + 7) = mystery3(8)
→ 8 < 10, return 8

mystery3(-479) = -2

→ mystery3(-479) = -mystery3(479)
→ mystery3(479) = mystery3(47 + 9) = mystery3(56)
→ mystery3(56) = mystery3(5 + 6) = mystery3(11)
→ mystery3(11) = mystery3(1 + 1) = mystery3(2)
→ 2 < 10, return 2
→ Final: -2

Problem 14: mystery4

```
public int mystery4(int n) {
    if (n < 0) { return mystery4(-n); }
    else if (n < 10) { return n; }
    else { return n % 10 + mystery4(n / 10); }
}
```

This sums all digits of n. For negatives, it converts to positive first (no sign preservation).

mystery4(8) = 8

→ 8 < 10, return 8

mystery4(-52) = 7

→ mystery4(-52) = mystery4(52)
→ mystery4(52) = 52%10 + mystery4(52/10) = 2 + mystery4(5)
→ mystery4(5) = 5
→ Final: 2 + 5 = 7

mystery4(3052) = 10

→ mystery4(3052) = 2 + mystery4(305)
→ mystery4(305) = 5 + mystery4(30)
→ mystery4(30) = 0 + mystery4(3)
→ mystery4(3) = 3
→ Unwind: 0 + 3 = 3, 5 + 3 = 8, 2 + 8 = 10

Problem 19: mystery8

```
public void mystery8(int n) {
    if (n > 100) { System.out.print(n); }
    else {
        mystery8(2 * n);
        System.out.print(", " + n);
    }
}
```

Doubles n until > 100, prints that value, then prints on the way back with commas.

mystery8(113) = `113`

→ 113 > 100, print 113 immediately

mystery8(70) = `140, 70`

→ 70 ≤ 100, call mystery8(140)
→ 140 > 100, print 140
→ Return to mystery8(70), print ", 70"

mystery8(42) = `168, 84, 42`

→ 42 → call mystery8(84)
→ 84 → call mystery8(168)
→ 168 > 100, print 168
→ Return: print ", 84"
→ Return: print ", 42"

Problem 30: printStars(4)

```
public static void printStars(int k) {
    if (k > 0) {
        printStars(k-1);
        for (int j=1; j<=k; j++)
            System.out.print("*");
        System.out.println();
    }
}
```

Recurses down first (to k=0), then prints k stars per line on the way back up.

Answer: `(B)`

→ printStars(4) calls printStars(3) calls printStars(2) calls printStars(1) calls printStars(0)
→ printStars(0): k=0 not > 0, do nothing
→ printStars(1): print 1 star, newline → *
→ printStars(2): print 2 stars, newline → **
→ printStars(3): print 3 stars, newline → ***
→ printStars(4): print 4 stars, newline → ****

Output: *, **, ***, **** (ascending triangle)

Problem 31: mystery(16)

```
public int mystery(int k) {
    if (k == 1) return 0;
    else return 1 + mystery(k/2);
}
```

This counts how many times you can divide by 2 before reaching 1 (i.e., floor of $\log_2(k)$).

Answer: (C) 4

→ mystery(16) = 1 + mystery(8)
→ mystery(8) = 1 + mystery(4)
→ mystery(4) = 1 + mystery(2)
→ mystery(2) = 1 + mystery(1)
→ mystery(1) = 0
→ Unwind: 1+0=1, 1+1=2, 1+2=3, 1+3=4

Problem 33: compute(1, 5)

```
public static int compute(int x, int y) {
    if (x == y) return x;
    else return compute(x+1, y-1);
}
```

x increases by 1 while y decreases by 1 until they meet. They meet at the average.

Answer: (C) 3

→ compute(1,5): x≠y, call compute(2,4)
→ compute(2,4): x≠y, call compute(3,3)
→ compute(3,3): x==y, return 3

Note: (1+5)/2 = 3

Problem 34: Which causes infinite recursion?

For x and y to meet, they must have the same parity (both even or both odd) since x+1 and y-1 preserve parity difference.

I. compute(2, 8): Both even. 2→3→4→5, 8→7→6→5. Meet at 5. FINITE.

II. compute(8, 2): x=8 > y=2. x increases, y decreases. They diverge forever. INFINITE.

III. compute(2, 5): Different parity! 2(even), 5(odd). They cross but never equal. INFINITE.

Answer: (E) II and III

Problem 36: mystery(0, 16)

```
public void mystery(int a, int b) {
    System.out.print(a + " ");
    if (a <= b)
        mystery(a + 5, b - 1);
}
```

Prints a, then recurses with a+5 and b-1 while a ≤ b.

Answer: (D) 0 5 10 15

→ mystery(0,16): print 0, 0≤16, call mystery(5,15)
→ mystery(5,15): print 5, 5≤15, call mystery(10,14)
→ mystery(10,14): print 10, 10≤14, call mystery(15,13)
→ mystery(15,13): print 15, 15≤13 is FALSE, stop

Problem 37: smile(4)

```
public static void smile(int n) {
    if (n == 0) return;
    for (int k=1; k<=n; k++)
        System.out.print("smile!");
    smile(n-1);
}
```

Prints n smiles, then n-1 smiles, then n-2, etc. Total = n + (n-1) + ... + 1 = n(n+1)/2

Answer: (E) 10 smiles

→ smile(4): print 4 smiles, call smile(3)
→ smile(3): print 3 smiles, call smile(2)
→ smile(2): print 2 smiles, call smile(1)
→ smile(1): print 1 smile, call smile(0)
→ smile(0): return
→ Total: 4 + 3 + 2 + 1 = 10 smiles

Problem 42: getSomething(4)

```
public int getSomething(int value) {
    if (value < 1) return 0;
    else return 1 + getSomething(value-1) + getSomething(value-2);
}
```

Note: The original has "value--1" which is likely a typo for "value-1". This is similar to Fibonacci counting: counts leaf nodes in a Fibonacci-like tree.

Answer: (E) 7

→ getSomething(4) = 1 + gS(3) + gS(2)
→ getSomething(3) = 1 + gS(2) + gS(1)
→ getSomething(2) = 1 + gS(1) + gS(0) = 1 + 1 + 0 = 2
→ getSomething(1) = 1 + gS(0) + gS(-1) = 1 + 0 + 0 = 1
→ getSomething(0) = 0
→ gS(3) = 1 + 2 + 1 = 4
→ gS(4) = 1 + 4 + 2 = 7

Problem 44: fun(3)

```
public void fun(int x) {
```

```
    if (x >= 1) {
        System.out.print(x);
        fun(x-1);
    }
}
```

Prints x BEFORE recursive call, counting down until x < 1.

Answer: (A) 3 2 1

→ fun(3): 3≥1, print 3, call fun(2)
→ fun(2): 2≥1, print 2, call fun(1)
→ fun(1): 1≥1, print 1, call fun(0)
→ fun(0): 0≥1 is FALSE, done

Problem 49: fun(3)

```
public void fun(int x) {
    if (x < 1) {
        System.out.print(x);
    }
    else {
        System.out.print(x);
        fun(x-1);
    }
}
```

Prints x in both branches. When x < 1, prints and stops. Otherwise prints and recurses.

Answer: (B) 3 2 1 0

→ fun(3): 3≥1, print 3, call fun(2)
→ fun(2): 2≥1, print 2, call fun(1)
→ fun(1): 1≥1, print 1, call fun(0)
→ fun(0): 0<1, print 0, done

Problem 56: mystery (binary conversion)

```
// precondition: num >= 0
public static void mystery(int num) {
    if (num > 1)
        mystery(num / 2);
    System.out.print(num % 2);
}
```

This recursively divides by 2, then prints remainders on the way back. This produces binary representation, most significant bit first.

Answer: (E) Prints the binary representation of num

→ Example: mystery(13)
→ mystery(13): 13>1, call mystery(6)
→ mystery(6): 6>1, call mystery(3)
→ mystery(3): 3>1, call mystery(1)

```
→ mystery(1): 1>1 FALSE, print 1%2 = 1
→ Return to mystery(3): print 3%2 = 1
→ Return to mystery(6): print 6%2 = 0
→ Return to mystery(13): print 13%2 = 1
→ Output: 1101 (which is 13 in binary: 8+4+1)
```

Key Recursion Patterns to Remember

1. Print BEFORE recursive call → outputs in forward/descending order
2. Print AFTER recursive call → outputs in reverse/ascending order
3. return x when x < y, else return f(x-y, y) → computes x % y (modulo)
4. return 1 + f(k/2) → computes floor($\log_2(k)$)
5. f(n/10) + n%10 → sums all digits
6. Recurse with f(n/10), then print n%2 → converts to binary
7. For convergence: check if parameters approach the base case!