1.

```
public static void b(int n, int k) {
  if (n == k || k == 0) {   // base
    return 1;              // case
  }
  else {                         // recursive
    return n / k * b(n-1, k-1);      // case
  }
}
```

```
b(5, 3) -> answer is 10
|
|-- return 5/3*b(4, 2) -> becomes 5/3*6 -> 10
             |
             |--return 4/2*b(3, 1) -> becomes 4/2*3 -> 6
                         |
                         |--return 3/1*b(2, 0) -> becomes 3/1*1 -> 3
                                     |
                                     |--return 1
```

(d) There are 4 total calls.

2.

```
public static void f(int a, int b) {
  if (a <= b) {               // base
    return a + b;              // case
  }
  else {                         // recursive
          return (a - b) + f(a-2, b-1);    // case
  }
}
```

```
f(6, 3) -> answer is 6
|
|-- return (6-3)+f(4, 2) -> becomes (6-3)+3 -> 6
               |
               |--return (4-2)+f(2, 1) -> becomes (4-2)+1 -> 3
                             |
                             |--return (2-1)+f(0, 0) -> becomes (2-1)+0 -> 1
                                           |
                                           |--return 0+0 = 0
```

(d) There are 4 total calls.

(e) If we change the recursive definition, infinite recursion occurs when calling f(6,3).

(f) The only legal values are cases where a <= b, which correspond exactly to the base case of the function. Any recursive cases will lead to infinite recursion.

3.

```
makePal(0) -> "A"
makePal(1) -> "B"
makePal(2) -> mp(1) + mp(0) + mp(1) -> "BAB"
makePal(3) -> mp(2) + mp(1) + mp(2) -> "BAB" + "B" + "BAB" -> BABBBAB
makePal(4) -> something really long but let's just look at the length of
  whatever string makePal(4) is going to be:

len of mp(4) -> len of (mp(3) + mp(2) + mp(3))
        -> len of mp(3) + len of mp(2) + len of mp(3)
            [because we can break up the length of a string into pieces]
        -> 7 + 3 + 7 -> 17
len of mp(5) -> len of (mp(4) + mp(3) + mp(4))
        -> len of mp(4) + len of mp(3) + len of mp(4)
        -> 17 + 7 + 17 -> 41
```

4.

```java
public static int countUpper(String str) {
  if (str.length() == 0) {
    return 0;
  }
  else {
    char letter = str.charAt(0);
    String rest = str.substring(1, str.length());
    if (isUpper(letter)) {
      return countUpper(rest) + 1;
    }
    else {
      return countUpper(rest);
    }
  }
}
```

5.

```
binsearch(array, 28, 0, 6)
|
|-- mid = (0+6)/2 = 3
|-- return binsearch(array, 28, 4, 6) -> 4
      |
      |-- mid = (4+6)/2 = 5
      |-- return binsearch(array, 28, 4, 4) -> 4
             |
             |-- mid = (4+4)/2 = 4
             |-- return 4
```

6.

```
public class Sundae {
  protected int numScoops;

  public Sundae(int newScoops) {
    numScoops = newScoops;
  }

  public int getCalories() {
    return 137 * numScoops;
  }
}

public class BSplit extends Sundae
{
  private int numBananas;

  public BSplit(int newScoops, int newBananas) {
    super(newScoops);
    numScoops = newScoops;   // either this line or the previous should be here
    numBananas = newBananas;
  }

  public int getCalories() {
    return super.getCalories() + 90 * numBananas;
  }
}
```

7.

Line 3 is an error. After commenting out that line and running the others, f has 1 unit of fuel and g has 10 units of fuel.

8.

1: “Menu: Chicken, fries, burgers, cake”
2: ”Cost: 2000”
3. ERROR
4. “Menu: Chicken, rice, tofu, pizza, scones”
5. ”Cost: 2050”
6. “Competition is rumbling…”
   ”Cost: 2000”
7. ”Cost: 2350”
8. ”Cost: 2000”
9. “Competition is rumbling…”
   ”Cost: 2350”