Problem 1: Functionality:

- The main function takes in a string from the command line that is set by the user.
- It prints out the input and the resulting hash value which is determined by the HashF1 function.
- The hashF1 function defines ret which will be our return value i.e. the hash value, also the int i that is used for the for loop, the integer array of hashA which will be the basis on how we compute the hash value, filler is just a string of letters to add characters to sIn which is the string hashF1 works on which as its equal to s + filler.
- It checks to make sure the string being passed in is between 1 and 64 characters.
- Then there is a loop that loops the same amount of times as the length of sIn.
- Within this loop is the variable byPos which is sIn.charAt(i) this basically loops through each character in sIn one at a time.
- This is followed by making each integer in our array the value of byPos multiplied by a certain number, thus filling our array with a bunch of different numbers.

```
for (i = 0; i < sIn.length(); i++){
                char byPos = sIn.charAt(i);
                hashA[0] += (byPos * 17);
                hashA[1] += (byPos * 31);
                hashA[2] += (byPos * 101);
                hashA[3] += (byPos * 79);
```

- Then once that is done the function uses the modulus operation on each of the numbers in our array, which divides them by 255 and gives us the remainder.

```
hashA[0] %= 255;
hashA[1] %= 255;
hashA[2] %= 255;
hashA[3] %= 255;
```

.

- Finally, our hash value(ret) is the sum of each number in our array multiplied by 256,

```
ret = hashA[0] + (hashA[1] * 256) + (hashA[2] * 256 * 256) +
(hashA[3] * 256 * 256 * 256);
            if (ret < 0) ret *= -1
```

if this value is negative we multiply it by -1 to make it positive and then return ret.

Problem 2 collisions:

```
initial  value: KV3l
Number of Loops: 3098
hash = 2.03477388E8
counter = 0

initial  value: xNz
Number of Loops: 10740
hash = 2.03477388E8
counter = 5

initial  value: HQMZ
Number of Loops: 20369
hash = 2.03477388E8
counter = 10

initial  value: HuM6
Number of Loops: 3723
hash = 2.03477388E8
counter = 1

initial  value: H9Tk
Number of Loops: 12763
hash = 2.03477388E8
counter = 6

initial  value: jJ6V
Number of Loops: 22059
hash = 2.03477388E8
counter = 11

initial  value: igp
Number of Loops: 4610
hash = 2.03477388E8
counter = 2

initial  value: S8Pe
Number of Loops: 13790
hash = 2.03477388E8
counter = 7

initial  value: TK7j
Number of Loops: 22324
hash = 2.03477388E8
counter = 12

initial  value: Ec4d
Number of Loops: 6155
hash = 2.03477388E8
counter = 3

initial  value: Yny
Number of Loops: 14187
hash = 2.03477388E8
counter = 8

initial  value: Quz
Number of Loops: 24741
hash = 2.03477388E8
counter = 13

initial  value: NIDe
Number of Loops: 10505
hash = 2.03477388E8
counter = 4

initial  value: jtb
Number of Loops: 15348
hash = 2.03477388E8
counter = 9

initial  value: 1jdA
Number of Loops: 30447
hash = 2.03477388E8
counter = 14
```

```
res = hashF1(args[0]); // call hash function with <input>
if (res < 0) { // Error
    System.out.println("Error: <input> must be 1 to 64 characters long.");
} else {
    System.out.println("input = " + args[0] + " : Hash = " + res);
    System.out.println("Start searching for collisions \n");
    // Your code starts here!
    String str1 = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";//base of our random string
    int counter = 1;// counter declaration
    int loopNum = 0;//number of loops counter declaration

    while (loopNum < 50000) {//a while loop that runs a variable amount of times.
        StringBuilder strTest = new StringBuilder();// a new string builder object for our test string
        for (int i = 0; i < (getRandomInt( upper: 64) + 1); ++i) {//
            strTest.append(str1.charAt(getRandomInt( upper: 62)));/*appending our test string to random characters from str1
            producing a random string*/
        }
        if (hashF1(strTest.toString()) == res) {/*if statement to check if our hash value for our random string
            is the same as the hash value of the predefined string we passed in*/
            System.out.flush();
            System.out.println( "initial  value: " + strTest.toString());// print statements to print values
            System.out.println("Number of Loops: " + loopNum);
            System.out.println("hash = " + hashF1(strTest.toString()));
            System.out.println("counter = " + counter+ "\n");

            ++counter;// increment counter which will also be the total number of collisions we have
        }
        loopNum++;// increment the number of loops counter until we hit 50000 then break
    }

}
private static int getRandomInt(int upper) {// random int generator function using the java.util.Random library
    Random r = new Random();
    return r.nextInt(upper);
}
```

Problem 3 hashF1:

```
  private static double hashF1(String s){
    double ret;
    int i;
    int[] hashA = new int[]{1,1,1,1,1,1,1,1,1,1};// increased the size of the
array hashA by 7 to make it more robust.

    String filler, sIn;

    filler = new
String("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789ABCDEFGHIJKL
MNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdef
ghijklmnopqrstuvwxyz0123456789" );// changed the filler to have more variation and
make it longer.

    if ((s.length() > 64) || (s.length() < 1)) { // String does not have required
length
        ret = -1;
    }
    else {
```

```
        sIn = s + filler; // Add characters, now have "<input>HABCDEF..."
       //sIn = sIn.substring(0, 150); // when i dont limit the string to just 64
characters, there are much less collisions.
        for (i = 0; i < sIn.length(); i++){
            char byPos = sIn.charAt(i); // get i'th character
            hashA[0] += (byPos * 177362101); // increased the size we multiply
byPos by to create a larger hash value.
            hashA[1] += (byPos * 301536575);
            hashA[2] += (byPos * 104149219);
            hashA[3] += (byPos * 179342922);
            hashA[4] += (byPos * 167931525);
            hashA[5] += (byPos * 196733183);
            hashA[6] += (byPos * 153939759);
            hashA[7] += (byPos * 514341921);
            hashA[8] += (byPos * 130937707);
            hashA[9] += (byPos * 434343153);
            hashA[10] +=(byPos * 324667247);

        }

        hashA[0] %=  18013 ;  //increased the size of the modulus considerably to
make it more random
        hashA[1] %=  18013 ;
        hashA[2] %=  18013 ;
        hashA[3] %=  18013 ;
        hashA[4] %=  18013 ;
        hashA[5] %=  18013 ;
        hashA[6] %=  18013 ;
        hashA[7] %=  18013 ;
        hashA[8] %=  18013 ;
        hashA[9] %=  18013 ;
        hashA[10] %= 18013 ;


        ret = hashA[0] + (hashA[1] * 256) + (hashA[2] * 256 * 256) + (hashA[3] *
256 * 256 * 256) + (hashA[4] * 256 * 256 * 256 * 256) + (hashA[5] * 256 * 256 *
256 * 256 * 256 ) + (hashA[6] * 256 * 256 * 256 * 256 * 256 * 256) + (hashA[7] *
256 * 256 * 256 * 256 * 256 * 256 * 256) + (hashA[8] * 256 * 256 * 256 * 256 * 256
* 256 * 256 * 256 ) + (hashA[9] * 256 * 256 * 256 * 256 * 256 * 256 * 256 * 256 *
256 ) + (hashA[10] * 256 * 256 * 256 * 256 * 256 * 256 * 256 * 256 * 256 * 256 ) ;
        if (ret < 0) ret *= -1;
    }
    return ret;
}
//this reduced the collision for Bamb0 from the 250 range to around 30 - 40 which
is a significant improvement that could probably improved by making the hash value
more unique, longer strings have collisions as low as 10 - 20.
```