```
#include <iostream>
#include <vector>

int main()
{
    int iterations = 0;
    std::cout << "How many iterations? ";
    std::cin >> iterations;

    //Get the amount of bits for the binary number
    std::vector<int> ans(iterations);
    std::srand(static_cast<unsigned int>(std::time(nullptr)));


    // Creates 5 random 32 bit binary strings with a coin flip for each bit
     std::vector<std::bitset<32> > binary_strings;
     std::bitset<32> b;
     for(int i = 0; i < 5; i++)
     {
        for (int j = 0; j < 32; j++)
        {
            b[j] = std::rand() % 2;
        }
        binary_strings.push_back(b);
     }

    //Create variables in order to count the amount of 1', 0's, biggest run by
     both, and the amount of runs of 4, 5, 6 for both 0's and 1's
    int ones_count = 0;
    int zeros_count = 0;
    int prev = 0;
    int curr_run = 1;
    int biggest_run_1 = 0;
    int biggest_run_0 = 0;
    int run_4_1 = 0;
    int run_5_1 = 0;
    int run_6_1 = 0;
    int run_4_0 = 0;
    int run_5_0 = 0;
    int run_6_0 = 0;

    //Create an xor vector to hold the xor values for the feedback functions
     of all binary strings
    std::vector<int> b_xor;
    b_xor.reserve(5);
    //Preprocess 1st iteration so that we can do runs and everything in the
     same loop
    //Different feedback functions for all of the binary strings
    b_xor[0] = binary_strings[0][15] ^ binary_strings[0][28] ^
     binary_strings[0][3];
```

```
b_xor[1] = binary_strings[1][30] ^ binary_strings[1][16] ^
 binary_strings[1][5];
b_xor[2] = binary_strings[2][12] ^ binary_strings[2][17] ^
 binary_strings[2][2];
b_xor[3] = binary_strings[3][31] ^ binary_strings[3][23] ^
 binary_strings[3][8];
b_xor[4] = binary_strings[4][7] ^ binary_strings[4][1] ^
 binary_strings[4][10];

//Preprocess 1st iteration so that we can do runs and everything in the
 same loop
ans[ans.size() - 1] = b_xor[0] ^ b_xor[1] ^ b_xor[2] ^ b_xor[3] ^ b_xor[4];

//Preprocess 1st iteration so that we can do runs and everything in the
 same loop
if (ans[ans.size() - 1] == 1)
{
    ones_count += 1;
}
else
{
    zeros_count += 1;
}

//Preprocess 1st iteration so that we can do runs and everything in the
 same loop
//Move the binary string over to the left one and set the rightmost bit to
 their respective feedback function
for (int i = 0; i < 5; i++)
{
    binary_strings[i] <<= 1;
    binary_strings[i][0] = b_xor[i];
}

for (int i = ans.size() - 2; i >= 0; i--)
{
    //Feedback functions for all of the binary strings
    b_xor[0] = binary_strings[0][15] ^ binary_strings[0][28] ^
     binary_strings[0][3];
    b_xor[1] = binary_strings[1][30] ^ binary_strings[1][16] ^
     binary_strings[1][5];
    b_xor[2] = binary_strings[2][12] ^ binary_strings[2][17] ^
     binary_strings[2][2];
    b_xor[3] = binary_strings[3][31] ^ binary_strings[3][23] ^
     binary_strings[3][8];
    b_xor[4] = binary_strings[4][7] ^ binary_strings[4][1] ^
     binary_strings[4][10];

    // Set the bit in the binary sequence we are generating to the xor of
     all of the feedback functions
    ans[i] = b_xor[0] ^ b_xor[1] ^ b_xor[2] ^ b_xor[3] ^ b_xor[4];
```

```
// This counts 1's and 0's for test 1
if (ans[i] == 1)
{
    ones_count += 1;
}
else
{
    zeros_count += 1;
}

//This counts the runs for test 2 and 3
//If the current bit matches the bit processed before then we are on a
 run
if (ans[i] == ans[i + 1])
{
    curr_run += 1;
}
else
{
    //Checks to see if this was the biggest run of 1's or 0's so far
    //When we are in this part of the if statement we know that the
     run just ended and the previous bit that was processed was the
     last bit in that run so we know if the run was of 1's or 0's
    if (ans[i + 1] == 1 && curr_run > biggest_run_1)
    {
        biggest_run_1 = curr_run;
    }
    else if(ans[i + 1] == 0 && curr_run > biggest_run_0)
    {
        biggest_run_0 = curr_run;
    }

    //Checks to see if the run was length 4, 5, 6 and if it was a 0 or
     a 1
    if (curr_run == 4 && ans[i + 1] == 1)
    {
        run_4_1 += 1;
    }
    else if (curr_run == 4 && ans[i + 1] == 0)
    {
        run_4_0 += 1;
    }
    else if (curr_run == 5 && ans[i + 1] == 1)
    {
        run_5_1 += 1;
    }
    else if (curr_run == 5 && ans[i + 1] == 0)
    {
        run_5_0 += 1;
    }
```

```
        else if (curr_run == 6 && ans[i + 1] == 1)
        {
            run_6_1 += 1;
        }
        else if (curr_run == 6 && ans[i + 1] == 0)
        {
            run_6_0 += 1;
        }

        //resets run to 1 as the current bit was different than the previous and is starting a new run
        curr_run = 1;
    }

    //Pushes all of the binary strings 1 to the left and sets the last index to their respective xor statement (feedback functions)
    for (int i = 0; i < 5; i++)
    {
        binary_strings[i] <<= 1;
        binary_strings[i][0] = b_xor[i];
    }
}

//Checks to see if the run never ended from the last iteration and sees if its the biggest
if (ans[0] == 1 && curr_run > biggest_run_1)
{
    biggest_run_1 = curr_run;
}
else if(ans[0] == 0 && curr_run > biggest_run_0)
{
    biggest_run_0 = curr_run;
}

//Prints out total number of 1's and 0's in the binary string
std::cout << "Number of 0's: " << zeros_count << std::endl << "Number of 1's: " << ones_count << std::endl;

//Prints out the longest run of 0's and longest run of 1's
std::cout << "Longest run of 0's: " << biggest_run_0 << std::endl << "Longest run of 1's: " << biggest_run_1 << std::endl;

//Prints out number of runs of 4 for 0's and 1's
std::cout << "Number of 4 runs of 0's: " << run_4_0 << std::endl << "Number of 4 runs of 1's: " << run_4_1 << std::endl;

//Prints out number of runs of 5 for 0's and 1's
std::cout << "Number of 5 runs of 0's: " << run_5_0 << std::endl << "Number of 5 runs of 1's: " << run_5_1 << std::endl;

//Prints out number of runs of 6 for 0's and 1's
```

```
    std::cout << "Number of 6 runs of 0's: " << run_6_0 << std::endl <<
     "Number of 6 runs of 1's: " << run_6_1 << std::endl;

    return 0;
}
```