CS 61B Fall 2014 Regex, Bits & Algorithmic Analysis

1 Bit Manipulation

1. Write 22 in binary.
   10110

2. Assuming $x_1, x_2, ..., x_n$ are integers. What is $(x_1 \hat{} x_2 \hat{} ... \hat{} x_n) \hat{} (x_1 \hat{} x_2 \hat{} ... \hat{} x_n)$?
   0

3. Write an expression to check whether a 32-bit integer is less than 0 using only == and the bit operators.
   `(x >>> 31) == 1`

4. What does the following code do?

```
public static int mysteryBit(int n) {
    return n & (n - 1);
}
```

   Return `n` with the rightmost 1 bit set to 0.

5. Write a program to count the number of 1 bits in an integer. You can use the function in part 5 as a hint.

```
public static int countBits(int n) {
    int count = 0;
    while (n != 0) {
        n &= (n - 1);
        count += 1;
    }
    return count;
}
```

2 Algorithmic Analysis

1. For each of the following function, find the Big-Theta expression for:

   a) The number of `i += 1` or `i *= 2` operations

   b) The number of `j += 1` operations

   c) The number of print operations

   d) The runtime of the function

```
public static void printIndices(int n) {
    for (int i = 0; i < n; i += 1) {
        for (int j = 0; j < i; j += 1) {
            System.out.println(i + j);
        }
    }
}

public static void printIndices2(int n) {
    for (int i = 1; i < n; i *= 2) {
        for (int j = 0; j < i; j += 1) {
            System.out.println(j);
        }
    }
}
```

   a) $\Theta(n)$ for `printIndices` and $\Theta(\log(n))$ for `printIndices2`

   b) $\Theta(n^2)$ for `printIndices` and $\Theta(n)$ for `printIndices2`

   c) Same as b)

   d) Same as b)

2. What is the big-Theta running time of the following functions?

```
public int weirdFib(int n) {
    if (n <= 1) {
        return n;
    }
    return weirdFib(n - 1) + weirdFib(n - 1);
}


public static void mystery(int n) {
    if (n == 1) {
        return;
    }
    for (int i = 0; i < n; i += 1) {
        mystery(n-1);
    }
}
```

   $\Theta(2^n)$ for `fib` and $\Theta(n!)$ for `mystery`

3 Regex

Write a Java regular expression to match each of the following sets of binary strings. You may only use the following characters: `()|01*`

1) All binary strings
2) Binary strings that begins and ends with `1`
3) Binary strings that contains at least three `1`s
4) Binary string that contains at least three consecutive `1`s
5) Binary string that doesn't contain the substring `110`.

1) `(0|1)*`
2) `1(0|1)*1|1`
3) `0*10*10*1(0|1)*`
4) `(0|1)*111(0|1)*`
5) `(0|10)*1*`