Homework 1

1. In each of the following situations, indicate wherether $f = O(g)$, or $f = \Omega(g)$, or both (in which case $f = \Theta(g)$)

	$f(n)$	$g(n)$	O, Ω, or Θ?
(a)	$n - 100$	$n - 200$	$f = \Theta(g)$
(b)	$n^{1/2}$	$n^{2/3}$	$f = O(g)$
(c)	$100n + \log n$	$n + (\log n)^2$	$f = \Theta(g)$
(d)	$n \log n$	$10n \log 10n$	$f = \Theta(g)$
(e)	$\log 2n$	$\log 3n$	$f = \Theta(g)$
(f)	$10 \log n$	$\log(n^2)$	$f = \Theta(g)$
(g)	$n^{1.01}$	$n \log^2 n$	$f = \Omega(g)$
(h)	$n^2 / \log n$	$n(\log n)^2$	$f = \Omega(g)$
(i)	$n^{0.1}$	$(\log n)^{10}$	$f = \Omega(g)$
(j)	$(\log n)^{\log n}$	$n / \log n$	$f = \Omega(g)$
(k)	$\sqrt(n)$	$(\log n)^3$	$f = O(g)$
(l)	$n^{1/2}$	$5^{\log_2 n}$	$f = O(g)$
(m)	$n2^n$	$3n$	$f = \Omega(g)$
(n)	2^n	2^{n+1}	$f = \Theta(g)$
(o)	$n!$	2^n	$f = \Omega(g)$
(p)	$(\log n)^{\log n}$	$2^{(\log_2 n)^2}$	$f = O(g)$
(q)	$\sum_{i=1}^{n} i^k$	n^{k+1}	$f = O(g)$

2. Dasgupta Problem 1.13

   Is the difference of $5^{30,000}$ and $6^{123,456}$ a multiple of 31?

   We know that $5^3 = 125$ and $31 * 4 = 124$ so it follows that.

$$5^{30,000} \equiv (5^3)^{10,000} \equiv (125)^{10,000} \equiv 1^{10,000} \equiv 1 (\text{mod } 31)$$

   We also know that $6^6 = 46,656$, $31 \times 1,505 = 46,655$, and $123,456/6 = 20,576$. So it follows that

$$6^{123,456} \equiv (6^6)^{20,576} \equiv (46,655)^{20,576} \equiv 1^{20,576} \equiv 1 (\text{mod } 31)$$

   So it follows that

$$5^{30,000} - 6^{123,456} \equiv 1 - 1 \equiv 0 (\text{mod } 31)$$

   So the difference is divisible by 31.

3. Levitin Problem 2.1.5b

   Prove the alternative formula for the number of bits in the binary representation of a positive integer n:

$$b = \lceil \log_2(n+1) \rceil$$

   First, note that integers are sequential, meaning that given $a \in \mathbb{Z}$, there are no integers between a and $a+1$. This implies that given $a < b \in \mathbb{Z}$, $a + 1 \leq b$ (if $a < b$, then a is, at most, $b - 1$, and $b - 1 + 1 = b$).

Suppose $0 < n \in \mathbb{Z}$. I assert that n is bounded above and below by powers of two, which is to say:

$$\forall n, \exists x \in \mathbb{Z} \quad | \quad 2^x \leq n < 2^{x+1}$$

Note that for $n = 1$, $2^0 \leq 1 < 2^1$. Suppose $\forall n, \exists x \in \mathbb{Z} \quad | \quad 2^x \leq n < 2^{x+1}$. Consider $n + 1$. n is, at most $2^{x+1} - 1$. Since integers are sequential (as noted at the beginning of the proof) $n + 1$ is, at most, 2^{x+1}, in which case let $x = x + 1$ ($2^{x+1} \leq n + 1 < 2^{x+2}$). In all other cases, $n + 1 < 2^{x+1}$, and since $2^x \leq n$, it must be true that $2^x < n + 1$. So, by induction, the assertion must be true.

So, for some $n \in \mathbb{Z}$, $2^x \leq n < 2^{x+1}$ for some $x \in \mathbb{Z}$. At its smallest, $n = 2^x$ which implies that (noting that $\lceil x \rceil = x$ since $x \in \mathbb{Z}$)

$$\log_2(2^x + 1) > \log_2(2^x) = x$$

which implies (since $log_2(2^x)$ produces an integer) that

$$\lceil \log_2(2^x + 1) \rceil > \log_2(2^x) = x$$

At it's largest, $n = 2^{x+1} - 1$, which implies that, at its largest

$$\lceil \log_2(2^{x+1} - 1 + 1) \rceil = x + 1$$

So $x < \lceil \log_2(n + 1) \rceil \leq x + 1 \implies x + 1 = \lceil \log_2(n + 1) \rceil$. Let $b = x + 1$.

4. Levatin Problem 2.1.10 Invention of Chess

   (a) According to a well-known legend, the game of chess was invented many centuries ago in northwestern India by a certain sage. When he took his invention to his king, the king liked the game so much that he offered the inventor any reward he wanted. The inventor asked for some grain to be obtained as follows: just a single grain of wheat was to be placed on the first square of the chessboard, two on the second, four on the third, eight on the fourth, and so on, until all 64 squares had been filled. If it took just 1 second to count each grain, how long would it take to count all the grain due to him?

   To find the time necessary to count all the grain, we must find the amount of grain. This quantity is

   $$\sum_{i=0}^{63} 2^i$$

   This happens to be $2^{64} - 1$. A binary number with n 1 digits is $\sum_{i=0}^{n-1} 2^i$. Naturally, adding 1 to this gives us a 1 followed by n zeros, or 2^{n+1}.

   (b) How long would it take if instead of doubling the number of grains for each square of the chessboard, the inventor asked for adding two grains?

   Let $g = \{1, 3, 5, ...1 + (63)2\}$, representing the grain on each of the squares. Let $x = \sum g$. If we subtract 1 from every element of g, we get $g' = \{0, 2, 4, ...(63)2\}$.

Since there are 64 elements in g, it follows that $x-64 = \sum g'$. If we divide both sides by two, we divide every element of g' by 2, producing $g'' = \{0, 1, 2, ...63\}$. So $\frac{x-64}{2} = \sum g''$. But we know that the sum of all integers from a to b is $\frac{b-a+1}{2}(a+b)$, so $\sum g = \frac{63-0+1}{2}(63+0) = \frac{64(63)}{2}$. Thus

$$\frac{x-64}{2} = \frac{64(63)}{2}$$

or

$$x = 64^2 = 2^{12} = 4096$$

5. Evaluate a polynomial a value x using Horner's rule. Represent polynomials with dictionaries mapping exponents to coefficients, e.g. $3x^5 2x^3 + 5x3$ would be

```
{5:3,3:-2,1:5,0:-3}
```

```
'use strict';

class Polynomial {
    constructor(coefficients) {
        this.coefficients = coefficients;
        this.max = Math.max(...Object.keys(coefficients));

        // If a coefficient isn't provided, it's 0
        // We know we have the coefficient for the maximum one, so -1
        for( let i = this.max - 1; i >= 0; --i) {
            if( !coefficients[i] ){
                coefficients[i] = 0;
            }
        }
    }
    evaluate(x) {
        let i = this.max;
        let result = 0;
        while(i >= 0) {
            // Get the coefficient
            let a = this.coefficients[i--];
            // Multiply x by what we got last time
            let b = result * x;
            // The result is the sum
            result = a + b;
        }
        return result;
    }
}

let solutions = [
    [
        {0:1},  // coefficients
        1,      // x
        1       // result
    ],
    [
        {0:1},  // coefficients
        2,      // x
        1       // result
    ],
    [
        {3:1, 2:-6, 1:11, 0:-6},// coefficients
        2,                      // x
        0                       // result
    ],
    [
        {3:2, 2:-6, 1:2, 0:-1}, // coefficients
        3,                      // x
        5                       // result
    ],
    [
        {4:1, 3:3, 2:5, 1:7, 0:9},  // coefficients
        2,                          // x
        83                          // result
    ],
    [
        {3:2, 1:-3, 0:-5},  // coefficients
        -2,                 // x
        -15                 // result
    ]
];

for( let i in solutions) {
    let soln = solutions[i]
    let poly = new Polynomial(soln[0]);
    let result = poly.evaluate(soln[1]);
    if( result !== soln[2]) {
        console.log(`${i}: Got ${result} expected ${soln[2]}`);
    }
}
```

6. Suppose you were given a list of size $3n$ integers. Write a function to determine whether or not you can partition the list into n triples such that the sum of each triple was the same. For example, given:

   ```
   [2,4,8,12,15,2,0,6,3,2,9,1]
   ```

   the answer is *true* because you could partition this list as follows:

```
[(4,3,9),(15,0,1),(8,2,6),(2,2,12)]
```

while the answer would be *false* for

```
[6, -1, 8, 3455, 11, 7]
```

The answer is also false if the length of the input list is not a multiple of 3. Make sure to correctly handle the empty list and the list with exactly three elements!

I'm pretty sure this is NP-Hard, by the way. As such, this problem was very hard to do in a non-painfully-inefficient way. Thus, I'm sure there is a better solution, but this suffices.

```
'use strict';

let threePartition = (numbers, number) => {
  let length = numbers.length;

  if( length === 0) {
    return true;
  }
  if( length % 3 !== 0){
    return false;
  }

  for( let i = 0; i < length; ++i ) {
    for( let j = i + 1; j < length; ++j) {
      for( let k = j + 1; k < length; ++k) {
        let sum = numbers[i] + numbers[j] + numbers[k];

        // Don't want to overwrite number
        let ournum = number;
        // If we're not looking for a specific number, we
        // can try this sum as the specific one
        if(number === undefined) {
          ournum = sum;
        }
        // If this number matches the one we need (or there
        // is no one we need) and the subset that doesn't
        // include these numbers is divisible into groups
        // that add up to this number, we win
        if( ournum === sum && threePartition(
              // Filter out all elements that aren't ours
              numbers.filter(
                  (e,ind) => [i, j, k].indexOf(ind) === -1
              ),
              ournum
          ) ) {
```

```
          return true;
        }
      }
    }
  }
  return false;
};

let solutions = [
    [ [2,4,8,12,15,2,0,6,3,2,9,1],  true ],
    [ [2,4,8,12,17,2,0,6,3,2,9,1],  false],
    [ [6, -1, 8, 3455, 11, 7],      false],
    [ [6, -1, 8, -3, 11, 7],        true ],
    [ [],                           true ],
    [ [1, 2, 3],                    true ],
    [ [1],                          false],
    [ [1, 2],                       false],
    [ [1, 2, 3, 4],                 false],
    [ [1, 2, 3, 4, 5, 6],           false],
    [ [1, 2, 3, 4, 5, 7],           true ],
    [ [1, 2, 3, 4, 5, 6, 7, 8, 9],  true],
    [ [1, 2, 3, 4, 5, 6, 7, 8, 10], false],
];

for( let i in solutions) {
    let soln = solutions[i]
    let result = threePartition(soln[0]);
    if( result ^ soln[1]) {
        console.log(‘${i}: Got ${result} expected ${soln[1]}‘);
    }
}
```

7. Consider the function

$$C(n,k) = \begin{cases} 1 & \text{if } k = 0 \\ 1 & \text{if } k = n \\ C(n-1,k) + C(n-1,k-1) & \text{if } 1 \le k \le n-1 \end{cases}$$