Big O Introduction

```
// O(1) or O(c)
function addItems(n) {
   return n + n;
}

// O(log n)
function printItems(n) {
   for (let i = 0; i < n; i *=2) {
      console.log(i + " ");
   }
}

// O(n log n)
function printItems(n) {
   for (let i = 0; i < n; i++) {
      for (let j = 1; j < n; j *=2) {
         console.log(i +" "+j);
      }
   }
}

// O(n)
function printItems(n) {
   for (let i = 0; i < n; i++) {
      console.log(i);
   }
}

// O(m + n)
function printItems(m, n) {
   for (let i = 0; i < m; i++) {
      console.log(i);
   }
   for (let j = 0; j < n; j++) {
         console.log(j);
      }
}

// O(m * n)
function printItems(m, n) {
   for (let i = 0; i < m; i++) {
      for (let j = 0; j < n; j++) {
         console.log(i +", "+j);
      }
```

```
   }
}

// O(n^2)
function printItems(n) {
   for (let i = 0; i < n; i++) {
      for (let j = 0; j < n; j++) {
         console.log(i +", "+j);
      }
   }
}

// O(2^n)
function printItems(n) {
   if (n < 1) {
      return n;
   }
   return printItems(n - 1);
}
```

Note:
In Computer Science contexts, logarithms are considered to be base 2, because many algorithms, such as binary search, binary trees, quick sort, and merge sort naturally operate in base 2. But the logarithm in Big O notation does not significantly affect the complexity classification. Base two is commonly used, but the base can technically be any positive number greater than one and the Big O notation will still be valid.

In Computer Science 2^n is an important function (1, 2, 4, 8, 16, 32, 64, 128, 512, 1024...)