Benchmark

```
// Require dependencies.
var generate = require("../shared/generate");

// Generate array of given length.
var length = 14;
var stuff = generate(length);
var randomValue = stuff[Math.floor(Math.random() * length)];

for (var i = 0; i < stuff.length; i++) {
  if (stuff[i] === randomValue) {
    console.log(i + " : " + randomValue);
  }
}
```

```
// Benchmark is a library that times
var Benchmark = require("benchmark");
var generate = require("../shared/generate");

// Generate an array of the given length.
var length = 100000000;
var stuff = generate(length);
var randomValue = stuff[Math.ceil(Math.random() * length)];

// A "suite" is a series of code snippets you
//   want to execute and time.
var suite = new Benchmark.Suite();

suite
// Add the function 'linearSearch' to the suite.
  .add("Linear Search", function linearSearch() {
    for (var i = 0; i < stuff.length; i += 1) {
      if (stuff[i] === randomValue) {
        return stuff[i];
      }
    }
    return false;
  })

  // On 'start', run the 'start' function.
  .on("start", function start() {
    console.log("Beginning benchmark...");
  })

  // On the 'complete' event, run the 'report' function.
  .on("complete", function report() {
    // Get successful benchmark.
    var benchmark = Benchmark.filter(this, "successful")[0];
    console.log("On average, " + benchmark.name + " took " + benchmark.stats.mean + " seconds to complete.");
  })

  // Run the test!
  .run();
```

```
// Create an array of length "length" filled with pseudo-random values
function generate(length) {
  var arr = [];

  for (var i = 0; i < length; i += 1) {
    arr.push(Math.ceil(Math.random() * length));
  }
  return arr;
}
module.exports = generate;
```

Binary Search

```
var result = binarySearch([1, 23, 43, 56, 77, 89, 211, 212, 789, 972, 1001, 4567, 4599, 83784], 77);
console.log(result);

function binarySearch(numbersArr, searchElement) {
  // Set some starting values.
  var currentElement;
  var currentIndex;
  var maxIndex = numbersArr.length - 1;
  var minIndex = 0;

  // This is the main loop.
  while (minIndex <= maxIndex) {

    // Get a position near the middle.
    currentIndex = Math.floor((minIndex + maxIndex) / 2);

    // Get that element.
    currentElement = numbersArr[currentIndex];

    // Test it.
    if (currentElement < searchElement) {

      // if it's less than we are looking for, look *above* this value.
      minIndex = currentIndex + 1;
    }
    else if (currentElement > searchElement) {

      // If it's more than we are looking for, look *below* this value.
      maxIndex = currentIndex - 1;
    }
    else {
      // We found it; return the index.
      return currentIndex;
    }
  }
  return false;
}
```

Selection Sort

```
// ===============================================
// RUN THIS USING NODE
// ===============================================
// ================================================
// TEST CASES
// ================================================
// Case 1 - Small Set of Numbers
var arraySize = 40;
// // Case 2 - Large set of Numbers
// var arraySize = 400000;
var array = [];
for (var index = 0; index < arraySize; index++) {
  var randomNumber = Math.round(Math.random() * arraySize);
  array.push(randomNumber);
}
// ================================================
// SOLUTION - Selection Sort
// ================================================
function swap(items, firstIndex, secondIndex) {
  var temp = items[firstIndex];
  items[firstIndex] = items[secondIndex];
  items[secondIndex] = temp;
}

function selectionSort(items) {
  // FILL IN YOUR CODE HERE
  // Use the above swap function when you are ready to start swapping elements in the array.
}
// ================================================
// FUNCTION CALL
// ================================================
console.log("PRE-SORT");
console.log(array.join(" "));
console.log("---------------------------");
console.log("POST-SORT");
console.log(selectionSort(array).join(" "));
```

```
// ===============================================
// RUN THIS USING NODE
// ===============================================
// ================================================
// TEST CASES
// ================================================
// Case 1 - Small Set of Numbers
var arraySize = 40;
// // Case 2 - Large set of Numbers
// var arraySize = 400000;
var array = [];

for (var index = 0; index < arraySize; index++) {
  var randomNumber = Math.round(Math.random() * arraySize);
  array.push(randomNumber);
}
// ================================================
// SOLUTION - Selection Sort
// ================================================
function swap(items, firstIndex, secondIndex) {
  var temp = items[firstIndex];
  items[firstIndex] = items[secondIndex];
  items[secondIndex] = temp;
}

function selectionSort(items) {
  var len = items.length;
  var min;
  for (var i = 0; i < len; i++) {
    // set index of minimum to this position
    min = i;
    // check the rest of the array to see if anything is smaller
    for (var j = i + 1; j < len; j++) {
      if (items[j] < items[min]) {
        min = j;
      }
    }
    // if the current position isn't the minimum, swap it and the minimum
    if (i !== min) {
      swap(items, i, min);
    }
  }
  return items;
}
// ================================================
// FUNCTION CALL
// ================================================
console.log("PRE-SORT");
console.log(array.join(" "));
console.log("---------------------------");
console.log("POST-SORT");
console.log(selectionSort(array).join(" "));
```

Insertion Sort

```
// ============================================
// RUN THIS USING NODE
// ============================================
// ============================================
// TEST CASES
// ============================================
// Case 1 - Small Set of Numbers
var arraySize = 40;
// // Case 2 - Large set of Numbers
// var arraySize = 400000;
var array = [];
for (var index = 0; index < arraySize; index++) {
  var randomNumber = Math.round(Math.random() * arraySize);
  array.push(randomNumber);
}
// ============================================
// SOLUTION - Insertion Sort
// ============================================

// ============================================
// FUNCTION CALL
// ============================================
console.log("PRE-SORT");
console.log(array.join(" "));
console.log("---------------------------");
console.log("POST-SORT");
console.log(insertionSort(array).join(" "));
```

```
// ==================================================
// RUN THIS USING NODE
// ==================================================
// ==================================================
// TEST CASES
// ==================================================
// Case 1 - Small Set of Numbers
var arraySize = 40;
// // Case 2 - Large set of Numbers
// var arraySize = 400000;
var array = [];

for (var index = 0; index < arraySize; index++) {
  var randomNumber = Math.round(Math.random() * arraySize);
  array.push(randomNumber);
}
// ==================================================
// SOLUTION - Insertion Sort
// ==================================================
function insertionSort(items) {

  // index into unsorted section, moving right
  var i;
  // index into sorted section, moving left
  var j;

  for (i = 0; i < items.length; i++) {
    // store the current value to insert later (this will be overwritten by the shift)
    var value = items[i];
    // Starting at the element (items[i - 1]) before the current value (value, items[i]), move left
    // through the array (decrementing j) and shift each value to the right (move to items[j + 1]) if it is larger
    // than the current value. Stop when you reach a value which is less than or equal to the current value.
    for (j = i - 1; j > -1 && items[j] > value; j--) {
      items[j + 1] = items[j];
    }
    // insert the value once you've reached the location where items[j] <= value
    items[j + 1] = value;
  }
  return items;
}

// ==================================================
// FUNCTION CALL
// ==================================================
console.log("PRE-SORT");
console.log(array.join(" "));
console.log("---------------------------");
console.log("POST-SORT");
console.log(insertionSort(array).join(" "));
```

Quicksort

```
// Case 1 - Small Set of Numbers
var arraySize = 40;
// // Case 2 - Large set of Numbers
// var arraySize = 400000;
var array = [];
for (var index = 0; index < arraySize; index++) {
  var randomNumber = Math.round(Math.random() * arraySize);
  array.push(randomNumber);
}

// SOLUTION - Selection Sort
function quickSort(items) {
  items.sort();
  return items;
}

// FUNCTION CALL
console.log("PRE-SORT");
console.log(array.join(" "));
console.log("---------------------------");
console.log("POST-SORT");
console.log(quickSort(array).join(" "));
```

~~~
```
define function quicksort (list)
// Select a pivot value
pivot = select_pivot_from list
// Create array of values less than/greater than pivot
left  = [element in list where element < pivot]
right = [element in list where element > pivot]
// Recursively sort left/right, and insert pivot in its final position
return quicksort (left) + pivot + quicksort (right)
```
~~~

```
// modified from https://gist.github.com/ttezel/3124434
var unsorted = [];
for (var index = 0, t = 400000; index < t; index++) {
  unsorted.push(Math.round(Math.random() * t));
}
function quickSort(array) {
  if (array.length <= 1) {
    return array;
  }
  // get random pivot element (and remove from array to add back in later)
  var pivot = array.splice(Math.floor(Math.random() * array.length), 1);
  // create left array (elements <= pivot), and right array (elements > pivot)
  var left = [];
  var right = [];
  // loop through array and create left/right
  array.forEach(function(el) {
    if (el <= pivot) {
      left.push(el);
    }
    else {
      right.push(el);
    }
  });
  // get result of recursively sorting the left array (using quicksort),
              then join that with the pivot and the
  // result of recursively sorting the right array (using quicksort).
  // equivalent of `return quicksort(left) + pivot + quicksort (right);` in the pseudocode
  return quickSort(left).concat(pivot, quickSort(right));
}
console.log("Pre Sort:", unsorted.join(" "));
var sorted = quickSort(unsorted);
console.log("Post Sort:", sorted.join(" "));
console.log("DONE!");
```

```
var arr = [];
for (var index = 0, t = 400; index < t; index++) {
  arr.push(Math.round(Math.random() * t));
}

// var arr = [5, 3, 1, 6, 4, 2, 3, 7];
function swap(items, firstIndex, secondIndex) {
  var temp = items[firstIndex];
  items[firstIndex] = items[secondIndex];
  items[secondIndex] = temp;
}

function partition(items, left, right) {
  var pivot = items[left]; // items[Math.floor((right + left) / 2)];
  var i = left - 1;
  var j = right + 1;
  while (i < j) {
    i++;
    j--;
    while (items[i] < pivot) {
      i++;
    }
    while (items[j] > pivot) {
      j--;
    }
    if (i < j) {
      swap(items, i, j);
    }
  }
  return j;
}

function quickSort(items, left, right) {
  // console.log('calling quickSort(items, ', left, ', ', right, ')');
  var index;
  if (right > left) {
    index = partition(items, left, right);
    // console.log('index: ', index);
    quickSort(items, left, index);
    quickSort(items, index + 1, right);
  }
  return items;
}

console.log("Pre Sort:", arr.join(" "));
var result = quickSort(arr, 0, arr.length - 1);
console.log("Post Sort:", result.join(" "));
console.log("DONE!");
```

Lodash

```
var _ = require("lodash");
// _.times
// Works like a loop. Takes in the number of iterations,
                and a callback function to execute each time
// with the current iteratee optionally passed in as an argument
console.log("--------------------------");
console.log("_.times");
_.times(10, function(iteratee) {
  console.log("Iteration number:", iteratee + 1);
});
console.log("--------------------------");

// _.random
// Works like Math.random, but gives us a number from 0 to whatever value we pass in
console.log("_.random");
console.log("Between 0 and 10:", _.random(10));
// If we pass in 2 values, then it will give us a number from the first number
                and the second number inclusive
console.log("Between 1 and 20:", _.random(1, 20));
console.log("--------------------------");

// _.each
// Works just like each in jQuery
console.log("_.each");
var arr = [1, 2, 3, 4, 5];
_.each(arr, function(item, index) {
  console.log("Index", index, "multiplied by 2 is", item * 2);
});
console.log("--------------------------");

// _.uniq
// Returns a duplicate free version of a given array
console.log("_.uniq");
var dupeArray = [1, 2, 2, 1, 4, 6, 2, 9, 10, 10, 1, 6];
console.log("Original array:", dupeArray);
var uniqueArray = _.uniq(dupeArray);
console.log("Original array:", uniqueArray);
console.log("--------------------------");

// _.shuffle
// Returns a shuffled array
console.log("_.shuffle");
console.log("Original array:", uniqueArray);
var shuffledArray = _.shuffle(uniqueArray);
console.log("Shuffled Array:", shuffledArray);
console.log("--------------------------");

// _.sum, _.multiply, _.mean
// Useful math functions for getting the sum, max, and mean of elements in an array
console.log("Array to use:", dupeArray);
console.log("Array sum:", _.sum(dupeArray));
console.log("Array max:", _.max(dupeArray));
console.log("Array mean:", _.mean(dupeArray));
console.log("--------------------------");
```

```
// _.clone
// Often times we want to copy an objects values onto another object to manipulate
                but don't want to alter
// the original. Since JavaScript objects are pass-by-reference,
                simply setting an object equal to another doesn't copy it
// as you would expect. Instead it just creates another variable
                that's pointing to the same object in memory.
// Any changes made to one object affects another
// _.clone attempts to solve this

// Before clone
console.log("Without using _.clone:");

var originalPerson = { name: "Sarah", age: 22 };
var samePerson = originalPerson;
samePerson.age = 25;

console.log("Original Person:", originalPerson);
console.log("Modified copy of original person:", samePerson);
console.log("Using _.clone:");

var clonedPerson = _.clone(originalPerson);
clonedPerson.name = "Mike";
console.log("Original Person:", originalPerson);
console.log("Modified cloned Person:", clonedPerson);
```

```
var _ = require("lodash");
// RUN THIS USING NODE
// TEST CASES
// Case 1 - Small Set of Numbers
// Creates a 40 element array with random numbers ranging from 0
// to 40
var arr1 = _.times(40, _.constant(_.random(40)));
// Case 2 - Large set of Numbers
// Creates a 400000 element array with random numbers ranging from 0
// to 400000
var arr2 = _.times(400000, _.constant(_.random(400000)));

// FUNCTION CALL
console.log("PRE-SORT");
console.log("Array 1:", arr1.join(" "));
console.log("Array 2:", arr2.join(" "));
console.log("---------------------------");
console.log("POST-SORT");
console.log("Array 1:", _.sortBy(arr1).join(" "));
console.log("Array 2:", _.sortBy(arr2).join(" "));
```

# Big O

# Big O Analysis
Express the running times of the following algorithms in Big O notation. Justify your responses.
Some of these are just review. A few apply Big O to algorithms we saw before.
Assume the _worst case_ running time—i.e., consider only the _maximum_ number of instructions the algorithm could take.

What is the running time of...
* Selection sort?
* Insertion sort?
* Linear search?
* Binary search?
* Finding duplicates in an array?

### BONUSES
If you're mathematically inclined, you might find these interesting. If not, feel free to take a stab anyway, but don't worry too much about proving your solution.

What is the running time for an algorithm that—
* Finds all triplets `(x, y, z)` such that `x + y + z = n`, where `n` is specified by the user?
  * E.g.: `threeSum(list, search)` will find all possible triplets of numbers in `list` that sum to `search`.
* Same question, but for doubles?
  * In general, what is the running time for finding n-tuples in `list` that sum to `search`?

# Solutions
The running times are...
* Selection sort: O(n<sup>2</sup>)
* Insertion sort: O(n<sup>2</sup>)
* Linear search: O(n)
* Binary search: O(lg n)
* Finding duplicates in an array: O(n<sup>2</sup>)

* `three_sum`: O(n<sup>3</sup>). In general, `n_sum` runs in O(n<sup>n</sup>).

Data Structures

```
var array = [1, 2, 3, 4];
console.log(array);

// Adding to beginning
array.unshift(1);
console.log(array);

// Adding to beginning
for (var i = array.length; i >= 0; i--) {
  array[i] = array[i - 1];
}
array[0] = -1;
```

```
// Creates the Queue Class for use later
class Queue {
  constructor() {
    this.items = [];
  }
  // Push, Pop, Peek
  enqueue(element) {
    this.items.push(element);
  }
  dequeue() {
    this.items.shift();
  }
  get first() {
    return this.items[0];
  }
  isEmpty() {
    return this.items.length === 0;
  }
  size() {
    return this.items.length;
  }
}

// Creates an instance of the Queue
var newQueue = new Queue();
// Starts running methods
newQueue.enqueue("Ahmed");
newQueue.enqueue("Roger");
newQueue.enqueue("John");
console.log(newQueue.first);
```

```
var myPets = {
  cat: "Mr. Hyena",
  lizard: "Mr. Big Big",
  goat: "Wolf Who Ate Wall Street",
  pigeon: "Joan"
};

var myPetAnimals = ["cat", "lizard", "goat", "pigeon"];
var myPetNames = ["Mr. Hyena", "Mr. Big Big", "Wolf Who Ate Wall Street", "Joan"];
```

```
// Creates the Stack Class for use later
class Stack {
  constructor() {
    this.items = [];
  }
  // Push, Pop, Peek
  push(element) {
    this.items.push(element);
  }
  pop(element) {
    this.items.pop(element);
  }
  peek() {
    return this.items[this.items.length - 1];
  }
  isEmpty() {
    return this.items.length === 0;
  }
  clear() {
    this.items = [];
  }
}

// Creates an instance of the Stack
var newStack = new Stack();
// Starts running methods
newStack.push(1);
newStack.push(2);
newStack.push(4);
console.log(newStack.peek());
```

```
var map = new Map();

map.set("cat", "Mr. Hyena");
map.set("lizard", "Mr. Big Big");
map.set("goat", "Wolf Who Ate Wall Street");
map.set("pigeon", "Joan");

console.log(map.keys());
console.log(map.values());
console.log(map.get("pigeon"));
```

```
class ListNode {
  constructor(data) {
    this.data = data;
    this.next = null;
  }
}

class LinkedList {
  constructor(head) {
    this.head = head;
  }
  getCount() {
    let count = 1;
    let currentNode = this.head;
    while (currentNode.next) {
      currentNode = currentNode.next;
      count++;
    }
    return count;
  }
  getFirst() {
    return this.head
  }
  getLast() {
    let lastNode = this.head;
    while (lastNode.next) {
      lastNode = lastNode.next;
    }
    return lastNode.data;
  }
  addNode(node) {
    let lastNode = this.head;
    while (lastNode.next) {
      lastNode = lastNode.next;
    }
    lastNode.next = node;
  }
}

let node1 = new ListNode(1);
let node2 = new ListNode(2);

let list = new LinkedList(node1);
let node3 = new ListNode(3);

list.addNode(node3);
list.addNode(new ListNode(4));

console.log("count: ", list.getCount());
console.log(list.getLast());
console.log(list.getFirst());
```