`algorithms.js` shadowing

Tasks

1. Karp Rabin	🟧	2. Bellman Ford	🟦
3. Priority Queue	Scattered	4. Fibonacci	🟩
5. Binary Search	🟦	6. Dijkstra	🟥
7. Heap	Scattered	8. Insertion Sort	🟦
9. Merge Sort	🟦	10. Stack	Scattered
11. Counting Sort	🟩		

Files Ordering

1. algorithms/graph/bellmanford.js
2. algorithms/graph/dijkstra.js
3. algorithms/graph/SPFA.js
4. algorithms/graph/topological_sort.js
5. algorithms/math/extended_euclidian.js
6. algorithms/math/fibonacci.js
7. algorithms/math/fisher_yates.js
8. algorithms/math/gcd.js
9. algorithms/math/newton_sqrt.js
10. algorithms/searching/bfs.js
11. algorithms/searching/binarysearch.js
12. algorithms/searching/dfs.js
13. algorithms/sorting/bubble_sort.js
14. algorithms/sorting/counting_sort.js
15. algorithms/sorting/heap_sort.js
16. algorithms/sorting/insertion_sort.js
17. algorithms/sorting/merge_sort.js
18. algorithms/sorting/quicksort.js
19. algorithms/string/edit_distance.js
20. algorithms/string/karp_rabin.js
21. data_structures/bst.js
22. data_structures/graph.js
23. data_structures/heap.js
24. data_structures/linked_list.js
25. data_structures/priority_queue.js
26. data_structures/queue.js
27. data_structures/stack.js
28. main.js
29. util/comparator.js

Files shadowing

algorithms/graph/bellmanford.js

- 43 lines of code

```
'use strict';

var bellmanFord = function(graph, startNode) {
  var minDistance = {};
  var edges = [];
  var adjacencyListSize = 0;

  graph.vertices.forEach(function (s) {
    graph.neighbors(s).forEach(function(t) {
      edges.push({
        source: s,
        target: t,
        weight: graph.edge(s, t)
      });
    });

    minDistance[s] = Infinity;
    ++adjacencyListSize;
  });

  minDistance[startNode] = 0;

  var edgesSize = edges.length;
  var sourceDistance;
  var targetDistance;

  for (var i = 0; i < adjacencyListSize - 1; i++) {
    for (var j = 0; j < edgesSize; j++) {
      sourceDistance = minDistance[edges[j].source] + edges[j].weight;
      targetDistance = minDistance[edges[j].target];

      if (sourceDistance < targetDistance) {
        minDistance[edges[j].target] = sourceDistance;
      }
    }
  }

  for (i = 0; i < edgesSize; i++) {
    sourceDistance = minDistance[edges[i].source] + edges[i].weight;
    targetDistance = minDistance[edges[i].target];

    if (sourceDistance < targetDistance) {

      // Empty 'distance' object indicates Negative-Weighted Cycle
      return {
        distance: {}
      };
    }
  }

  return {
    distance: minDistance
  };
};

module.exports = bellmanFord;
```

algorithms/graph/dijkstra.js

- 33 lines of code

```
'use strict';

var PriorityQueue = require('../../data_structures/priority_queue');

function dijkstra(graph, s) {
  var distance = {};
  var previous = {};
  var q = new PriorityQueue();
  distance[s] = 0;
  graph.vertices.forEach(function (v) {
    if (v !== s) {
      distance[v] = Infinity;
    }
    q.insert(v, distance[v]);
  });

  var currNode;
  while (!q.isEmpty()) {
    currNode = q.extract();
    var neighbors = graph.neighbors(currNode);
    for (var i = 0; i < neighbors.length; i++) {
      var v = neighbors[i];
      var newDistance = distance[currNode] + graph.edge(currNode, v);
      if (newDistance < distance[v]) {
        distance[v] = newDistance;
        previous[v] = currNode;
        q.changePriority(v, distance[v]);
      }
    }
  }
  return {
    distance: distance,
    previous: previous
  };
}

module.exports = dijkstra;
```

algorithms/graph/SPFA.js

- **41 lines of code**

```
'use strict';

function SPFA(graph, s) {
  var distance = {};
  var previous = {};
  var queue    = {};
  var isInQue  = {};
  var head     = 0;
  var tail     = 1;
  // initialize
  distance[s] = 0;
  queue[0] = s;
  isInQue[s] = true;
  graph.vertices.forEach(function (v) {
    if (v !== s) {
      distance[v] = Infinity;
      isInQue[v] = false;
    }
  });

  var currNode;
  while (head != tail) {
    currNode = queue[head++];
    isInQue[currNode] = false;
    var neighbors = graph.neighbors(currNode);
    for (var i = 0; i < neighbors.length; i++) {
      var v = neighbors[i];
      // relaxation
      var newDistance = distance[currNode] + graph.edge(currNode, v);
      if (newDistance < distance[v]) {
        distance[v] = newDistance;
        previous[v] = currNode;
        if (!isInQue[v]){
          queue[tail++] = v;
          isInQue[v] = true;
        }
      }
    }
  }

  return {
    distance: distance,
    previous: previous
  };
}

module.exports = SPFA;
```

algorithms/graph/topological_sort.js

- 25 lines of code

```
'use strict';

var Stack = require('../../data_structures/stack');

var topologicalSort = function (graph) {
  var stack = new Stack();
  var firstHit = {};
  var secondHit = {};
  var time = 0;

var dagDFS = function (node) {
    if (firstHit[node]) return;
    var neighbors = graph.neighbors(node);
    firstHit[node] = ++time;
    for (var i = 0; i < neighbors.length; i++) {
      dagDFS(neighbors[i]);
    }
    secondHit[node] = ++time;
    stack.push(node);
  };

  graph.vertices.forEach(function (node) {
    if (!secondHit[node]) {
      dagDFS(node);
    }
  });

  return stack;
};

module.exports = topologicalSort;
```

algorithms/math/extended_euclidian.js

- 24 lines of code

```
'use strict';

var extEuclid = function (a, b) {
    var s = 0, oldS = 1;
    var t = 1, oldT = 0;
    var r = b, oldR = a;
    var quotient, temp;
    while (r !== 0) {
        quotient = Math.floor(oldR / r);

        temp = r;
        r = oldR - quotient * r;
        oldR = temp;

        temp = s;
        s = oldS - quotient * s;
        oldS = temp;

        temp = t;
        t = oldT - quotient * t;
        oldT = temp;
    }

    return {
        x: oldS,
        y: oldT
    };
};

module.exports = extEuclid;
```

algorithms/math/fibonacci.js

- 28 lines of code

```
'use strict';

var fibExponential = function (n) {
  return n < 2 ? n : fibExponential(n - 1) + fibExponential(n - 2);
};

var fibLinear = function (n) {
  var fibNMinus2 = 0,
      fibNMinus1 = 1,
      fib = n;
  for (var i = 1; i < n; i++) {
    fib = fibNMinus1 + fibNMinus2;
    fibNMinus2 = fibNMinus1;
    fibNMinus1 = fib;
  }
  return fib;
};

var fibWithMemoization = (function () {
  var cache = [0, 1];

  var fib = function (n) {
    if (cache[n] === undefined) {
      cache[n] = fib(n - 1) + fib(n - 2);
    }
    return cache[n];
  };

  return fib;
})();

// Use fibLinear as the default implementation
fibLinear.exponential = fibExponential;
fibLinear.withMemoization = fibWithMemoization;
module.exports = fibLinear;
```

algorithms/math/fisher_yates.js

- 10 lines of code

```
'use strict';

var fisherYates = function (a) {
  for (var i = a.length - 1; i > 0; i--) {
    var j = Math.floor(Math.random() * (i + 1));
    var tmp = a[i];
    a[i] = a[j];
    a[j] = tmp;
  }
};

module.exports = fisherYates;
```

algorithms/math/gcd.js

- 42 lines of code

```
'use strict';

var gcdDivisionBased = function (a, b) {
  var tmp = a;
  a = Math.max(a, b);
  b = Math.min(tmp, b);
  while (b !== 0) {
    tmp = b;
    b = a % b;
    a = tmp;
  }

  return a;
};

var gcdBinaryIterative = function (a, b) {

  if (a === 0) {
    return b;
  }

  if (b === 0) {
    return a;
  }

  for (var shift = 0; ((a | b) & 1) === 0; ++shift) {
    a >>= 1;
    b >>= 1;
  }

  while ((a & 1) === 0) {
    a >>= 1;
  }

  var tmp;

  do {
    while ((b & 1) === 0) {
      b >>= 1;
    }
    if (a > b) {
      tmp = b;
      b = a;
      a = tmp;
    }

    b -= a;  // Here b >= a
  } while (b !== 0);

  return a << shift;
};

gcdDivisionBased.binary = gcdBinaryIterative;
module.exports = gcdDivisionBased;
```

algorithms/math/newton_sqrt.js

- 20 lines of code

```
'use strict';

var sqrt = function (n, tolerance, maxIterations) {
  tolerance = tolerance || 1e-7;
  maxIterations = maxIterations || 1e7;

  var upperBound = n;
  var lowerBound = 0;

  var i = 0;
  var square, x;
  do {
    i++;
    x = (upperBound - lowerBound) / 2 + lowerBound;
    square = x * x;
    if (square < n) lowerBound = x;
    else upperBound = x;
  } while (Math.abs(square - n) > tolerance  && i < maxIterations);

  // Checks if the number is a perfect square to return the exact root
  var roundX = Math.round(x);
  if (roundX * roundX === n) x = roundX;

  return x;
};

module.exports = sqrt;
```

algorithms/searching/bfs.js

- 14 lines of code

```
'use strict';
var Queue = require('../../data_structures/queue.js');

var bfs = function (root, callback) {
  var q = new Queue();
  q.push(root);
  var node;
  while (!q.isEmpty()) {
    node = q.pop();
    callback(node.value);
    if (node.left) q.push(node.left);
    if (node.right) q.push(node.right);
  }
};

module.exports = bfs;
```

algorithms/searching/binarysearch.js

- 13 lines of code

```
'use strict';

var binarySearch = function (sortedArray, element) {
  var init = 0,
      end = sortedArray.length - 1;

  while (end >= init) {
    var m = ((end - init) >> 1) + init;
    if (sortedArray[m] === element) return true;

    if (sortedArray[m] < element) init = m + 1;
    else end = m - 1;
  }

  return false;
};

module.exports = binarySearch;
```

algorithms/searching/dfs.js

- 25 lines of code

```
'use strict';

var inOrder = function (node, callback) {
  if (node) {
    inOrder(node.left, callback);
    callback(node.value);
    inOrder(node.right, callback);
  }
};

var preOrder = function (node, callback) {
  if (node) {
    callback(node.value);
    preOrder(node.left, callback);
    preOrder(node.right, callback);
  }
};

var postOrder = function (node, callback) {
  if (node) {
    postOrder(node.left, callback);
    postOrder(node.right, callback);
    callback(node.value);
  }
};

inOrder.preOrder = preOrder;
inOrder.postOrder = postOrder;
module.exports = inOrder;
```

algorithms/sorting/bubble_sort.js

- 21 lines of code

```
'use strict';
var Comparator = require('../../util/comparator');

var bubbleSort = function(a, comparatorFn) {
  var comparator = new Comparator(comparatorFn),
    n = a.length,
    bound = n - 1;
  for (var i = 0; i < n - 1; i++) {
    var newbound = 0;
    for (var j = 0; j < bound; j++) {
      if (comparator.greaterThan(a[j], a[j + 1])) {
        var tmp = a[j];
        a[j] = a[j + 1];
        a[j + 1] = tmp;
        newbound = j;
      }
    }
    bound = newbound;
  }

  return a;
};

module.exports = bubbleSort;
```

algorithms/sorting/counting_sort.js

- 35 lines of code

```
'use strict';

var countingSort = function(array) {
  var max = maximumKey(array);
  var auxiliaryArray = [];
  var length = array.length;

  for (var i = 0; i < length; i++) {
    var position = array[i].key;

    if (auxiliaryArray[position] === undefined) {
      auxiliaryArray[position] = [];
    }

    auxiliaryArray[position].push(array[i]);
  }

  array = [];
  var pointer = 0;

  for (i = 0; i <= max; i++) {
    if (auxiliaryArray[i] !== undefined) {
      var localLength = auxiliaryArray[i].length;

      for (var j = 0; j < localLength; j++) {
        array[pointer++] = auxiliaryArray[i][j];
      }
    }
  }

  return array;
};


var maximumKey = function(array) {
  var max = array[0].key;
  var length = array.length;

  for (var i = 1; i < length; i++) {
    if (array[i].key > max) {
      max = array[i].key;
    }
  }

  return max;
};

module.exports = countingSort;
```

algorithms/sorting/heap_sort.js

- 11 lines of code

```
'use strict';
var MinHeap = require('../../data_structures/heap').MinHeap;

var heapsort = function (array, comparatorFn) {

  var minHeap = new MinHeap(comparatorFn);
  minHeap.heapify(array);

  var result = [];
  while (!minHeap.isEmpty())
    result.push(minHeap.extract());

  return result;
};

module.exports = heapsort;
```

algorithms/sorting/insertion_sort.js

- 16 lines of code

```
'use strict';
var Comparator = require('../../util/comparator');

var insertionSort = function(vector, comparatorFn) {
  var comparator = new Comparator(comparatorFn);

  for (var i=1, len=vector.length; i<len; i++) {
    var aux = vector[i],
      j = i;

    while (j > 0 && comparator.lessThan(aux, vector[j - 1])) {
      vector[j] = vector[j - 1];
      j--;
    }

    vector[j] = aux;
  }

  return vector;
};

module.exports = insertionSort;
```

algorithms/sorting/merge_sort.js

- 24 lines of code

```
'use strict';
var Comparator = require('../../util/comparator');

var mergeSortInit = function (a, compareFn) {
  var comparator = new Comparator(compareFn);

  return (function mergeSort(a) {
    if (a.length > 1) {
      var middle = a.length >> 1;
      var left = mergeSort(a.slice(0, middle));
      var right = mergeSort(a.slice(middle));
      a = merge(left, right, comparator);
    }

    return a;
  })(a);
};

var merge = function (a, b, comparator) {
  var i = 0,
      j = 0,
      result = [];

  while (i < a.length && j < b.length) {
    result.push(comparator.lessThan(a[i], b[j]) ? a[i++] : b[j++]);
  }

  return result.concat((i < a.length ? a.slice(i) : b.slice(j)));
};

module.exports = mergeSortInit;
```

algorithms/sorting/quicksort.js

- 32 lines of code

```
'use strict';
  var Comparator = require('../../util/comparator');

  var quicksortInit = function (array, comparatorFn) {

  var comparator = new Comparator(comparatorFn);

  return (function quicksort(array, lo, hi) {
    if (lo < hi) {
      var p = partition(array, comparator, lo, hi);
      quicksort(array, lo, p - 1);
      quicksort(array, p + 1, hi);
    }

    return array;
  })(array, 0, array.length - 1);
};

var partition = function (a, comparator, lo, hi) {
  swap(a, Math.floor(Math.random() * (hi - lo)) + lo, hi);
  var pivot = hi;

  var dividerPosition = lo;

  for (var i = lo; i < hi; i++) {
    if (comparator.lessThan(a[i], a[pivot])) {
      swap(a, i, dividerPosition);
      dividerPosition++;
    }
  }
  swap(a, dividerPosition, pivot);
  return dividerPosition;
};

var swap = function (array, x, y) {
  var tmp = array[y];
  array[y] = array[x];
  array[x] = tmp;
};

module.exports = quicksortInit;
```

algorithms/string/edit_distance.js

- 25 lines of code

```
'use strict';

var levenshtein = function (a, b) {
  var editDistance = [];
  var i, j;

  for (i = 0; i <= a.length; i++) {
    editDistance[i] = [];
    editDistance[i][0] = i;
  }

  for (j = 0; j <= b.length; j++) {
    editDistance[0][j] = j;
  }
  for (i = 1; i <= a.length; i++) {
    for (j = 1; j <= b.length; j++) {
      // Finds the minimum cost for keeping the two strings equal
      editDistance[i][j] =
        Math.min(
          editDistance[i - 1][j - 1], // if we replace a[i] by b[j]
          editDistance[i - 1][j], // if we delete the char from a
          editDistance[i][j - 1] // if we add the char from b
        ) +
        (a[i - 1] != b[j - 1] ? 1 : 0);
    }
  }

  return editDistance[a.length][b.length];
};

module.exports = levenshtein;
```

algorithms/string/karp_rabin.js

- 57 lines of code

```
'use strict';
var base = 997;
var karpRabin = function (a, b) {
  var aLength = a.length;
  var bLength = b.length;
  var rs = hashFunction(b);
  var newString = [];
  for (var i = 0; i < bLength; i++) {
    newString.push(a.charAt(i));
  }
  var rt = hashFunction(newString.join(''));
  if (rs === rt && checkEquality(b, newString.join(''))) {
    return true;
  }
  else {
    for (i = 1; i < aLength; i++) {
      var previousCharacter = newString[0];
      var nextCharacter = a.charAt(i);
      rt = reHash(
        bLength,
        rt,
        previousCharacter,
        nextCharacter
      );
      newString.shift();
      newString.push(nextCharacter);
      if (rs === rt && checkEquality(b, newString.join(''))) {
        return true;
      }
    }
    return false;
  }
};
var checkEquality = function (a, b) {
  var aLength = a.length;
  for (var i = 0; i < aLength; i++) {
    if (a.charAt(i) !== b.charAt(i)) {
      return false;
    }
  }
  return true;
};
var hashFunction = function (word) {
  var hash = 0;
  var wordLength = word.length;
  for (var i = 0, j = wordLength - 1; i < wordLength; i++, j--) {
    hash += word.charCodeAt(i) * Math.pow(base, j);
  }
  return hash;
};
var reHash = function (length, hash, previousCharacter, nextCharacter) {
  hash -= previousCharacter.charCodeAt(0) * Math.pow(base, length - 1);
  hash *= base;
  hash += nextCharacter.charCodeAt(0);
  return hash;
};
module.exports = karpRabin;
```

data_structures/bst.js

- 80 lines of code

```
'use strict';
var Comparator = require('../util/comparator');

function BST(compareFn) {
  this.root = null;
  this._size = 0;

  this._comparator = new Comparator(compareFn);

  Object.defineProperty(this, 'size', {
    get: function () { return this._size; }.bind(this)
  });
}

function Node(value, parent) {
  this.value = value;
  this.parent = parent;
  this.left = null;
  this.right = null;
}

BST.prototype.insert = function (value, parent) {
  if (!parent) {
    if (!this.root) {
      this.root = new Node(value);
      this._size++;
      return;
    }
    parent = this.root;
  }

  var child = this._comparator.lessThan(value, parent.value) ? 'left' : 'right';
  if (parent[child])
    this.insert(value, parent[child]);
  else {
    parent[child] = new Node(value, parent);
    this._size++;
  }
};

BST.prototype.contains = function (e) {
  return !!this._find(e);
};

BST.prototype._find = function (e, root) {

  if (!root) {
    if (this.root) root = this.root;
    else return false;
  }

  if (root.value === e)
    return root;

  if (this._comparator.lessThan(e, root.value))
    return root.left && this._find(e, root.left);

  if (this._comparator.greaterThan(e, root.value))
    return root.right && this._find(e, root.right);
};

BST.prototype._replaceNodeInParent = function (currNode, newNode) {
  var parent = currNode.parent;
  if (parent) {
    parent[currNode === parent.left ? 'left' : 'right'] = newNode;
    if (newNode)
      newNode.parent = parent;
  } else {
    this.root = newNode;
  }
};

BST.prototype._findMin = function (root) {
  var minNode = root;
  while (minNode.left) {
    minNode = minNode.left;
  }
  return minNode;
};

BST.prototype.remove = function (e) {
  var node = this._find(e);
  if (!node) {
    throw new Error('Item not found in the tree');
  }

  if (node.left && node.right) {
    var successor = this._findMin(node.right);
    this.remove(successor.value);
    node.value = successor.value;
  } else {
    this._replaceNodeInParent(node, node.left || node.right);
    this._size--;
  }
};

module.exports = BST;
```

data_structures/graph.js

- 26 lines of code

```
'use strict';

function Graph(directed) {
  this.directed = (directed === undefined ? true : !!directed);
  this.adjList = {};
  this.vertices = [];
}

Graph.prototype.addVertex = function (v) {
  this.vertices.push(v);
  this.adjList[v] = {};
};

Graph.prototype.addEdge = function (a, b, w) {
  w = (w === undefined ? 1 : w);

  if (!this.adjList[a]) this.addVertex(a);
  if (!this.adjList[b]) this.addVertex(b);

  this.adjList[a][b] = (this.adjList[a][b] || 0) + w;

  if (!this.directed) {
    this.adjList[b][a] = (this.adjList[b][a] || 0) + w;
  }
};

Graph.prototype.neighbors = function (v) {
  return Object.keys(this.adjList[v]);
};

Graph.prototype.edge = function (a, b) {
  return this.adjList[a][b];
};

module.exports = Graph;
```

data_structures/heap.js

- 71 lines of code

```
'use strict';
var Comparator = require('../util/comparator');

function MinHeap(compareFn) {
  this._elements = [null];
  this._comparator = new Comparator(compareFn);

  Object.defineProperty(this, 'n', {
    get: function () {
      return this._elements.length - 1;
    }.bind(this)
  });
}

MinHeap.prototype._swap = function (a, b) {
  var tmp = this._elements[a];
  this._elements[a] = this._elements[b];
  this._elements[b] = tmp;
};

MinHeap.prototype.isEmpty = function () {
  return this.n === 0;
};

MinHeap.prototype.insert = function (e) {
  this._elements.push(e);
  this._siftUp();
};
```

```

MinHeap.prototype.extract = function () {
  var element = this._elements[1];

var last = this._elements.pop();
  if (this.n) {
    this._elements[1] = last;
    this._siftDown();
  }

  return element;
};


MinHeap.prototype._siftUp = function () {
  var i, parent;

  for (i = this.n;
      i > 1 && (parent = i >> 1) && this._comparator.greaterThan(
        this._elements[parent], this._elements[i]);
      i = parent) {
    this._swap(parent, i);
  }
};

MinHeap.prototype._siftDown = function (i) {
  var c;
  for (i = i || 1; (c = i << 1) <= this.n; i = c) {
    if (c + 1 <= this.n && this._comparator.lessThan(
          this._elements[c + 1], this._elements[c]))
      c++;
    if (this._comparator.lessThan(this._elements[i],
          this._elements[c]))
      break;
    this._swap(i, c);
  }
};

MinHeap.prototype.heapify = function (a) {
  if (a) {
    this._elements = a;
    this._elements.unshift(null);
  }

  for (var i = this.n >> 1; i > 0; i--) {
    this._siftDown(i);
  }
};

function MaxHeap(compareFn) {

  MinHeap.call(this, compareFn);
  this._comparator.reverse();
}

MaxHeap.prototype = new MinHeap();

module.exports = {
  MinHeap: MinHeap,
  MaxHeap: MaxHeap
};
```

data_structures/linked_list.js

- 88 lines of code

```
'use strict';

function LinkedList() {

  this._length = 0;
  this.head = null;
  this.tail = null;

  Object.defineProperty(this, 'length', {
    get: function () {
      return this._length;
    }.bind(this)
  });
}

function Node(value) {
  this.value = value;
  this.prev = null;
  this.next = null;
}

LinkedList.prototype.isEmpty = function () {
  return this.length === 0;
};

LinkedList.prototype.add = function (n, index) {
  if (index > this.length || index < 0) {
    throw new Error('Index out of bounds');
  }

  var node = new Node(n);

  if (index !== undefined && index < this.length) {
    var prevNode,
        nextNode;

    if (index  === 0) {
      nextNode = this.head;
      this.head = node;
    } else {
      nextNode = this.getNode(index);
      prevNode = nextNode.prev;
      prevNode.next = node;
      node.prev = prevNode;
    }
    nextNode.prev = node;
    node.next = nextNode;
  } else {
    if (!this.head) this.head = node;

    if (this.tail) {
      this.tail.next = node;
      node.prev = this.tail;
    }
    this.tail = node;
  }

  this._length++;
};

LinkedList.prototype.get = function (index) {
  return this.getNode(index).value;
};

LinkedList.prototype.getNode = function (index) {
  if (index >= this.length || index < 0) {
    throw new Error('Index out of bounds');
  }

  var node = this.head;
  for (var i = 1; i <= index; i++) {
    node = node.next;
  }

  return node;
};

LinkedList.prototype.del = function (index) {
  if (index >= this.length || index < 0) {
    throw new Error('Index out of bounds');
  }
  var node = this.getNode(index);

  if (node === this.tail) {
    this.tail = node.prev;
  } else {
    node.next.prev = node.prev;
  }
  if (node === this.head) {
    this.head = node.next;
  } else {
    node.prev.next = node.next;
  }

  this._length--;
};

LinkedList.prototype.map = function (fn) {
  var node = this.head;
  while (node) {
    fn(node.value);
    node = node.next;
  }
};

module.exports = LinkedList;
```

data_structures/priority_queue.js

- **31 lines of code**

```
'use strict';

var MinHeap = require('./heap').MinHeap;

function PriorityQueue(initialItems) {
  MinHeap.call(this, function (a, b) {
    return a.priority < b.priority ? -1 : 1;
  });

  this._items = {};

  initialItems = initialItems || {};
  var self = this;
  Object.keys(initialItems).forEach(function (item) {
    self.insert(item, initialItems[item]);
  });
}

PriorityQueue.prototype = new MinHeap();

PriorityQueue.prototype.insert = function (item, priority) {
  var o = {
    item: item,
    priority: priority
  };

  this._items[item] = o;
  MinHeap.prototype.insert.call(this, o);
};

PriorityQueue.prototype.extract = function () {
  var min = MinHeap.prototype.extract.call(this);
  return min && min.item;
};

PriorityQueue.prototype.changePriority = function (item, priority) {
  this._items[item].priority = priority;
  this.heapify();
};

module.exports = PriorityQueue;
```

data_structures/queue.js

- **31 lines of code**

```
'use strict';

var LinkedList = require('./linked_list');

function Queue() {
  this._elements = new LinkedList();

  Object.defineProperty(this, 'length', {
    get: function () {
      return this._elements.length;
    }.bind(this)
  });
}

Queue.prototype.isEmpty = function () {
  return this._elements.isEmpty();
};

Queue.prototype.push = function (e) {
  this._elements.add(e);
};

Queue.prototype.pop = function () {
  if (this.isEmpty()) {
    throw new Error('Empty queue');
  }
  var e = this._elements.get(0);
  this._elements.del(0);
  return e;
};

Queue.prototype.peek = function () {
  if (this.isEmpty()) {
    throw new Error('Empty queue');
  }

  return this._elements.get(0);
};

module.exports = Queue;
```

data_structures/stack.js

- **10 lines of code**

```
'use strict';

var Queue = require('./queue');

function Stack() {
  Queue.call(this);
}

Stack.prototype = new Queue();

Stack.prototype.push = function (e) {
  this._elements.add(e, 0);
};

module.exports = Stack;
```

main.js

- **42 lines of code**

```
'use strict';

var lib = {
  Graph: {
    topologicalSort: require('./algorithms/graph/topological_sort'),
    dijkstra: require('./algorithms/graph/dijkstra'),
    SPFA: require('./algorithms/graph/SPFA'),
    bellmanFord: require('./algorithms/graph/bellman_ford')
  },
  Math: {
    fibonacci: require('./algorithms/math/fibonacci'),
    fisherYates: require('./algorithms/math/fisher_yates'),
    gcd: require('./algorithms/math/gcd'),
    extendedEuclidean: require('./algorithms/math/extended_euclidean'),
    newtonSqrt: require('./algorithms/math/newton_sqrt')
  },
  Search: {
    bfs: require('./algorithms/searching/bfs'),
    binarySearch: require('./algorithms/searching/binarysearch'),
    dfs: require('./algorithms/searching/dfs')
  },
  Sort: {
    bubbleSort: require('./algorithms/sorting/bubble_sort'),
    countingSort: require('./algorithms/sorting/counting_sort'),
    heapSort: require('./algorithms/sorting/heap_sort'),
    mergeSort: require('./algorithms/sorting/merge_sort'),
    quicksort: require('./algorithms/sorting/quicksort')
  },
  String: {
    editDistance: require('./algorithms/string/edit_distance'),
    karpRabin: require('./algorithms/string/karp_rabin')
  },
  DataStructure: {
    BST: require('./data_structures/bst'),
    Graph: require('./data_structures/graph'),
    Heap: require('./data_structures/heap'),
    LinkedList: require('./data_structures/linked_list'),
    PriorityQueue: require('./data_structures/priority_queue'),
    Queue: require('./data_structures/queue'),
    Stack: require('./data_structures/stack')
  }
};

module.exports = lib;
```

util/comparator.js

- **33 lines of code**

```
'use strict';

function Comparator(compareFn) {
  if (compareFn) {
    this.compare = compareFn;
  }
}

Comparator.prototype.compare = function (a, b) {
  if (a == b) return 0;
  return a < b ? -1 : 1;
};

Comparator.prototype.lessThan = function (a, b) {
  return this.compare(a, b) < 0;
};

Comparator.prototype.lessThanOrEqual = function (a, b) {
  return this.lessThan(a, b) || this.equal(a, b);
};

Comparator.prototype.greaterThan = function (a, b) {
  return this.compare(a, b) > 0;
};

Comparator.prototype.greaterThanOrEqual = function (a, b) {
  return this.greaterThan(a, b) || this.equal(a, b);
};

Comparator.prototype.equal = function (a, b) {
  return this.compare(a, b) === 0;
};

Comparator.prototype.reverse = function () {
  var originalCompareFn = this.compare;
  this.compare = function (a, b) {
    return originalCompareFn(b, a);
  };
};

module.exports = Comparator;
```