Javascript Notes: Concepts, Algorithms, and Data Structures

Samuel Duval

August 8, 2022

Welcome to my notes on coding. In this document I'll keep track of basic Javascript concepts, algorithms, and perhaps some data structures as well. These notes are to help me formalize this information in my own brain. Pretty much everything is from the internet, attribution will be given in some cases, but generally for the sake of brevity and since all concepts are basic it will be omitted.

Contents

1 Basic Javascript Concepts

1.1 Array.prototype.slice()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/slice

- slice() returns a shallow copy of a portion of an array
- array.slice(startIndex, endIndex)
- slice() does not modify the array
- without arguments slice() will automatically be slice(0, endIndex)

Example slice()

```
let fruits = ['Banana', 'Orange', 'Lemon', 'Apple', 'Mango']

let citrus = fruits.slice(1, 3)
// citrus contains ['Orange','Lemon']

// fruits STILL contains ['Banana', 'Orange', 'Lemon', 'Apple', 'Mango']
```

1.2 Array.prototype.sort()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort

- sort() returns the same array in sorted order
- default order is ascending, done by converting everything to a string then comparing the UTF-16 code unit values. So it does not naturally work for numbers as expected.
- sort() modifies the the array

Example sort() for integers

```
1 const numbers = [4, 2, 5, 1, 3];
2
3 numbers.sort((a, b) => a - b);
4
5 console.log(numbers);
6 // [1, 2, 3, 4, 5]
```

1.3 Array.prototype.join()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/join

- join() returns a new STRING by concatenating all elements in an ARRAY
- items should be seperated by commas (,) (or specified seperator string), if only one item then that item will be returned without using the seperator.

Example join()

```
1  const elements = ['Fire', 'Air', 'Water'];
2
3  console.log(elements.join());
4  // expected output: "Fire,Air,Water"
5
6  console.log(elements.join(''));
7  // expected output: "FireAirWater"
8
9  console.log(elements.join('-'));
10 // expected output: "Fire-Air-Water"
```

1.4 Array.prototype.indexOf()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/indexOf

- indexOf() returns the first index at which a given element can be found in an array or -1 if not found
- good for finding the index of first appearance of something. If you want last use lastIndexOf()

Example indexOf()

```
1 const names = ['sam', 'bill', 'steve', 'matt']
2
3 const idx = names.indexOf('sam')
4 names[idx] = 'replaced-sam'
5
6 console.log(names)
7 //[ 'replaced-sam', 'bill', 'steve', 'matt' ]
```

1.5 Array.prototype.reduce()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce

- reduce() passes a callback function on each item in the array and has a second parameter for storing current value
- reduce((callback, currentValue) → callback, currentValue)

Example reduce()

```
1 const array = [1,2,3,4,5]
2
3 let reduceExample = array.reduce((callback,
    currentValue) => callback + currentValue)
4
5 console.log(reduceExample)
6 //output: 15
```

1.6 String.prototype.split()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/split

- takes a string and splits it into parts and returns it as an array
- split(), split(separator), split(separator, limit)
- can pass in a value(separator) and it will decide where the string will be split
- can also take regular expressions ex. too many commas split(/,+/) may help resolve

Example split()

```
1 const splitString = 'Hello,my,name,is,Sam'
2
3 let split = splitString.split(',')
4
5 console.log(split)
6 //[ 'Hello', 'my', 'name', 'is', 'Sam' ]
```

1.7 Array.prototype.fill()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/fill

- Modifies an array with the value given in fill()
- fill(0) by default, fill(value, start, end)
- useful for filling an array with numbers (0 seems common followed by other operations)

Example fill()

```
1 const fillArray = [0,1,2,3,4,5]
2 const fillArray2 = [0,1,2,3,4,5]
3
4 fillArray.fill(0)
5 //[ 0, 0, 0, 0, 0, 0 ]
6
7 fillArray2.fill(0,3,5)
8 //[ 0, 1, 2, 0, 0, 5 ]
```

1.8 map()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map

- Executes a function on all elements of an array and returns a new array
- Does not change original array
- map(callback)
- Can even return new array of objects
- Useful for pulling specific parts out of an array or for manipulating an entire array (ex. to find total value of objects within)

Example simple map() multiplying all elements by two

```
const numbers = [1,2,3,4,5];

let mapNumbers = numbers.map(nums => (nums * 2))
//[ 2, 4, 6, 8, 10 ]
```

Example map() Number constructor turning string numbers into regular integers

```
const stringNumbers = ['1', '2', '3', '4', '5']

const regNumbers = stringNumbers.map(Number)
// [ 1, 2, 3, 4, 5 ]
```

Example simple map() to manipulate objects within array

```
const products = [
    {
        name: 'laptop',
        price: 1500,
        count: 5
    },
    {
        name: 'desktop',
        price: 2000,
        count: 10
    },
    {
        name: 'phone',
        price: 1000,
        count: 100
```

```
    }
];

const totalProductValue = products.map(item => item.
    price * item.count);
//[ 7500, 20000, 100000 ]

const totalProductValueObject = products.map(item =>
    ({
    name: item.name,
    totalValue: item.price * item.count
}));
// [
//     { name: 'laptop', totalValue: 7500 },
//     { name: 'desktop', totalValue: 20000 },
//     { name: 'phone', totalValue: 100000 }
// ]
```

1.9 Array.prototype.push()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/push

- push() add ones or more elements to the end of an array
- push(x,y,z,1,2,3,etc.) can have as many parameters as desired
- push returns the new array length (for updated original array simply return that iteself)

Example push()

```
1  const numbers = [1,2,3,4,5]
2
3  numbers.push(6)
4  //[ 1, 2, 3, 4, 5, 6 ]
5
6  numbers.push(6,7,8,9,10)
7  //[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
8
9  //note: if done in order above it would be
10 //[1, 2, 3, 4, 5, 6, 6, 7, 8, 9, 10]
11
12 const total = numbers.push(6,7,8,9,10)
13 //10
14 //note: would be 16 if done as above
```

1.10 Array.prototype.splice()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/splice

- splice() modifies an array by removing and/or replacing existing elements from it
- splice(startIndex, removalCount, newElementsAdded)

Example splice()

```
1  const numbers = [1,2,3,4,5];
2
3  numbers.splice(2,3)
4  //[ 1, 2 ]
5
6  let deleted = numbers.splice(2,3)
7  //[ 3, 4, 5 ]
8
9  numbers.splice(2,3,6,9)
10 //[ 1, 2, 6, 9 ]
11
12 numbers.splice(2,0,6,9)
13 //[ 1, 2, 6, 9, 3, 4, 5 ]
```

1.11 Bitwise XOR (^)

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Bitwise_XOR

- XOR (^) returns a 1 if the other corresponding bit is different

XOR (^) rules

```
1 //0 & 0 = 0
2 //0 & 1 = 1
3 //1 & 1 = 0
```

Example XOR (^)

```
1 const a = 5;
2 // 00000000000000000000000000000101
3 const b = 3;
4 // 00000000000000000000000000000011
5
6 console.log(a ^ b);
7 // 00000000000000000000000000000110
8 // expected output: 6
```

1.12 String.prototype.charCodeAt()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/charCodeAt

- Returns the unicode integer of the character in a string representing the UTF-16 code unit of the given index.

Example charCodeAt()

```
1  const sentence = 'My name is Sam';
2
3  const index = 4;
4  const otherLocation = 1;
5
6  console.log(`The character code ${sentence.charCodeAt(
       index)} is equal to ${sentence.charAt(index)}`);
7  // "The character code 97 is equal to a"
8
9  console.log(`The character code ${sentence.charCodeAt(
       otherLocation)} is equal to ${sentence.charAt(
       otherLocation)}`);
10 //"The character code 121 is equal to y"
```

1.13 String.fromCharCode()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/fromCharCode

- The static String.fromCharCode() method returns a string created from the specified sequence of UTF-16 code units.

Example fromCharCode()

```
1 String.fromCharCode(65, 66, 67);
2 // returns "ABC"
```

1.14 Array.prototype.some()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/some

- Executes a given function and returns true if at least one element in array provides a truthy value.
- If no item in array is true, it will provide false

Example some()

```
1 const numbers = [1,2,3,4,5]
2
3 const res = numbers.some(greaterThanThree)
4 //true
5
6 function greaterThanThree(num) {
7     return num > 3;
8 }
```

1.15 Array.prototype.every()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/every

- Executes a given function on all items in an array and return true only if ALL items are truthy
- If any item in the array returns a falsy value, it will automatically return false

Example every()

```
1 const numbers = [1,2,3,4,5,-1]
2
3 const res = numbers.every(isPositive)
4 //false
5
6 function isPositive(num) {
7     return num > 0;
8 }
```

1.16 Array.prototype.includes()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/includes

- Checks in an element is included in an array
- If the item is included it will return true
- If the item is not included it will return false

Example includes()

```
1 const names = ['Sam', 'Gabe', 'Troy']
2
3 const res = names.includes('Sam')
4 // true
5 const res = names.includes('Bill')
6 // false
```

1.17 Array.prototype.find()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find

- Returns the first element in an array for which the callback has a truthy value
- If object will return the whole object (or you can return a part of an object)
- if nothing is found it will return undefined

Example find()

```
const names = ['Sam', 'Gabe', 'Troy']

const namedPerson = names.find(findName)
//Gabe

function findName(name) {
    return name === 'Gabe'
}
```

Example find() with objects

```
const inventory = [
    {name: 'apples', quantity: 2},
    {name: 'bananas', quantity: 0},
    {name: 'cherries', quantity: 5}
  ];

  const result = inventory.find( ({ name }) => name
   === 'cherries' );
  //{ name: 'cherries', quantity: 5 }
  const result = inventory.find( ({ name }) => name
   === 'cherries' ).quantity;
  //5
```

1.18 Math.max(), Math.min(), Math.abs()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/max

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/min

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/abs

- Math.max()
- Returns highest value in list
- Math.min()
- Returns lowest value in list
- Math.abs()
- Returns absolute value of a number (distance from zero)

Example Math.max()

```
console.log(Math.max(1, 3, 2));
// expected output: 3

console.log(Math.max(-1, -3, -2));
// expected output: -1
```

Example Math.min()

```
console.log(Math.min(2, 3, 1));
// expected output: 1

console.log(Math.min(-2, -3, -1));
// expected output: -3
```

Example Math.abs()

```
function difference(a, b) {
  return Math.abs(a - b);
}

console.log(difference(3, 5));
// expected output: 2

console.log(difference(5, 3));
// expected output: 2
```

1.19 parseInt() & parseFloat()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/parseInt

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/parseFloat

- parseInt() take a float or string and parses the integer out of it
- Will go from start to end of the first integer it sees (if it sees words, space, or any non-number it will stop)
- parseFloat() will do the same except it will run through the first period (.) and stop at the next period / non-number
- These don't work if the int / float isn't at the start of the string
- Note: parseInt(string, radix) careful of radix when passing values to parseInt

Example parseInt()

```
let testString = '1337demo'
let parseIntDemo = parseInt(testString)
//1337
```

Example parseFloat()

```
let testStringTwo = '1337.37.37demo'
let parseFloatDemo = parseFloat(testStringTwo)
//1337.37
```

Example parseInt() without numbers at start

```
let testString = 'doesnotwork1337.37.37demo'
let parseIntDemo = parseInt(testString)
//NaN
```

Example parseInt() with radix 16

```
console.log(roughScale(' 0xF', 16));
// expected output: 1500
```

1.20 Logical (and &&) and (or ||)

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Logical_OR

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Logical_AND

- The logical AND (&&) operator (logical conjunction) for a set of boolean operands will be true if and only if all the operands are true. Otherwise it will be false.
- More generally, the operator returns the value of the first falsy operand encountered when evaluating from left to right, or the value of the last operand if they are all truthy.
- The logical OR (||) operator (logical disjunction) for a set of operands is true if and only if one or more of its operands is true.
- the || operator actually returns the value of one of the specified operands, so if this operator is used with non-Boolean values, it will return a non-Boolean value.

Example Logical AND &&

```
1 const a = 3;
2 const b = -2;
3
4 console.log(a > 0 && b > 0);
5 // expected output: false
```

Example Logical AND &&

```
1 result = 2 && 0;
2 // result is assigned 0
3 result = 'foo' && 4;
4 // result is assigned 4
```

Example Logical OR ||

```
1 const a = 3;
2 const b = -2;
3
4 console.log(a > 0 || b > 0);
5 // expected output: true
```

Example Logical OR ||

```
o3 = true  || false
// t || f returns true
o4 = false || (3 == 4)
// f || f returns false
o5 = 'Cat' || 'Dog'
// t || t returns "Cat"
```

1.21 Set() constructor

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/Set

- The Set constructor lets you create Set objects that store unique values of any type, whether primitive values or object references.

Example set1

```
const set1 = new Set([1, 2, 3, 4, 5]);

console.log(set1.has(1));
// expected output: true

console.log(set1.has(5));
// expected output: true

console.log(set1.has(6));
// expected output: false
```

Example using Set() object

```
let mySet = new Set()

mySet.add(1)
// Set [ 1 ]
mySet.add(5)
// Set [ 1, 5 ]
mySet.add(5)
// Set [ 1, 5 ]
mySet.add('some text') // Set [ 1, 5, 'some text' ]
let o = {a: 1, b: 2}
mySet.add(o)
```

1.22 String.prototype.charAt()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/charAt

- The String object's charAt() method returns a new string consisting of the single UTF-16 code unit located at the specified offset into the string.
- charAt(index)

Example charAt()

```
1 const sentence = 'The quick brown fox jumps over the
    lazy dog.';
2
3 const index = 4;
4
5 console.log(`The character at index ${index} is ${
    sentence.charAt(index)}`);
6 // expected output: "The character at index 4 is q"
```

1.23 for...of

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...of

- The for...of statement creates a loop iterating over iterable objects, including: built-in String, Array, array-like objects (e.g., arguments or NodeList), TypedArray, Map, Set, and user-defined iterables. It invokes a custom iteration hook with statements to be executed for the value of each distinct property of the object.

Example for...of

```
const array1 = ['a', 'b', 'c'];

for (const element of array1) {
  console.log(element);
}

// expected output: "a"
// expected output: "b"
// expected output: "c"
```

Example for...of

```
for (variable of iterable) {
  statement
}
```

1.24 for...in

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...in

- The for...in statement iterates over all enumerable properties of an object that are keyed by strings (ignoring ones keyed by Symbols), including inherited enumerable properties.

Example for...in

```
const object = { a: 1, b: 2, c: 3 };

for (const property in object) {
  console.log(`${property}: ${object[property]}`);
}

// expected output:
// "a: 1"
// "b: 2"
// "c: 3"
```

Example for...in

```
for (const variable in object) {
  statement
}
```

1.25 Conditional (ternary) operator

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Conditional_Operator

- The conditional (ternary) operator is the only JavaScript operator that takes three operands
- A condition followed by a question mark (?), then an expression to execute if the condition is truthy followed by a colon (:), and finally the expression to execute if the condition is falsy.
- This operator is frequently used as an alternative to an if...else statement.
- condition ? exprIfTrue : exprIfFalse

Example ternary operator

```
function getFee(isMember) {
  return (isMember ? '$2.00' : '$10.00');
}

console.log(getFee(true));
// expected output: "$2.00"

console.log(getFee(false));
// expected output: "$10.00"

console.log(getFee(null));
// expected output: "$10.00"
```

Example ternary operator

```
const age = 26;
const beverage = age >= 21 ? "Beer" : "Juice";
console.log(beverage); // "Beer"
```

1.26 Map.prototype.get()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/get

- The get() method returns a specified element from a Map object.
- If the value that is associated to the provided key is an object, then you will get a reference to that object and any change made to that object will effectively modify it inside the Map object.
- get(key)

Example Map.prototype.get()

```
const map1 = new Map();
map1.set('bar', 'foo');

console.log(map1.get('bar'));
// expected output: "foo"

console.log(map1.get('baz'));
// expected output: undefined
```

Example get()

```
const arr = [];
const myMap = new Map();

myMap.set('bar', arr);
myMap.get('bar').push('foo');

console.log(arr);
// ["foo"]
console.log(myMap.get('bar'));
// ["foo"]
```

1.27 Map.prototype.has()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/has

- The has() method returns a boolean indicating whether an element with the specified key exists or not.
- has(key)

Example Map.prototype.has()

```
const map1 = new Map();
map1.set('bar', 'foo');

console.log(map1.has('bar'));
// expected output: true

console.log(map1.has('baz'));
// expected output: false
```

Example has()

```
let myMap = new Map()
myMap.set('bar', "foo")

myMap.has('bar')
// returns true
myMap.has('baz')
// returns false
```

1.28 Set

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set

- The Set object lets you store unique values of any type, whether primitive values or object references.
- Set objects are collections of values. A value in the Set may only occur once; it is unique in the Set's collection.
- You can iterate through the elements of a set in insertion order. The insertion order corresponds to the order in which each element was inserted into the set by the add()

Example Set 0bject

```
const mySet1 = new Set()

mySet1.add(1)           // Set [ 1 ]
mySet1.add(5)           // Set [ 1, 5 ]
mySet1.add(5)           // Set [ 1, 5 ]
mySet1.add('some text') // Set [ 1, 5, 'some text' ]
const o = {a: 1, b: 2}
mySet1.add(o)

mySet1.add({a: 1, b: 2})   // o is referencing a
   different object, so this is okay

mySet1.has(1)              // true
mySet1.has(3)              // false, since 3 has not
   been added to the set
mySet1.has(5)              // true
mySet1.has(Math.sqrt(25))  // true
mySet1.has('Some Text'.toLowerCase()) // true
mySet1.has(o)       // true

mySet1.size         // 5

mySet1.delete(5)    // removes 5 from the set
mySet1.has(5)       // false, 5 has been removed

mySet1.size         // 4, since we just removed one
   value

mySet1.add(5)       // Set [1, 'some text', {...},
   {...}, 5] - a previously deleted item will be added
```

```
    as a new item, it will not retain its original
    position before deletion

console.log(mySet1)
```

Example remove duplicate elements from array

```
// Use to remove duplicate elements from the array

const numbers =
    [2,3,4,4,2,3,3,4,4,5,5,6,6,7,5,32,3,4,5]

console.log([...new Set(numbers)])

// [2, 3, 4, 5, 6, 7, 32]
```

Example relationship with array

```
const myArray = ['value1', 'value2', 'value3'];

// Use the regular Set constructor to transform an
    Array into a Set
const mySet = new Set(myArray);

mySet.has('value1')     // returns true

// Use the spread operator to transform a set into an
    Array.
console.log([...mySet]); // Will show you exactly the
    same Array as myArray
```

Example relationship with string

```
const text = 'India';

const mySet = new Set(text);  // Set(5) {'I', 'n', 'd
    ', 'i', 'a'}
mySet.size  // 5

//case sensitive & duplicate omission
new Set("Firefox")  // Set(7) { "F", "i", "r", "e", "f
    ", "o", "x" }
new Set("firefox")  // Set(6) { "f", "i", "r", "e", "o
    ", "x" }
```

1.29 Object.prototype.toString()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/toString

- The toString() method returns a string representing the object.
- toString()

Example toString()

```
function Dog(name) {
  this.name = name;
}

const dog1 = new Dog('Gabby');

Dog.prototype.toString = function dogToString() {
  return `${this.name}`;
};

console.log(dog1.toString());
// expected output: "Gabby"
```

Example toString()

```
const arr = [1, 2, 3];

arr.toString()
// "1,2,3"
Object.prototype.toString.call(arr)
// "[object Array]"
```

1.30 String.prototype.replace()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/replace

- The replace() method returns a new string with some or all matches of a pattern replaced by a replacement.
- The pattern can be a string or a RegExp, and the replacement can be a string or a function called for each match.
- If pattern is a string, only the first occurrence will be replaced.
- The original string is left unchanged.
- replace(substr, newSubstr)

Example replace()

```
1  const p = 'The quick brown fox jumps over the lazy dog
       . If the dog reacted, was it really lazy?';
2
3  console.log(p.replace('dog', 'monkey'));
4  // expected output: "The quick brown fox jumps over
       the lazy monkey. If the dog reacted, was it really
       lazy?"
5
6  const regex = /Dog/gi;
7  console.log(p.replace(regex, 'ferret'));
8  // expected output: "The quick brown fox jumps over
       the lazy ferret. If the ferret reacted, was it
       really lazy?"
9
10 const regex = /Dog/i;
11 console.log(p.replace(regex, 'ferret'));
12 // expected output: "The quick brown fox jumps over
       the lazy ferret. If the dog reacted, was it really
       lazy?"
```

1.31 Set.prototype.add()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/add

- The add() method inserts a new element with a specified value in to a Set object
- Only works if if there isn't an element with the same value already in the Set.
- add(value)

Example add()

```
const set1 = new Set();

set1.add(42);
set1.add(42);
set1.add(13);

for (const item of set1) {
  console.log(item);
  // expected output: 42
  // expected output: 13
}
```

Example add()

```
const mySet = new Set();

mySet.add(1);
mySet.add(5).add('some text'); // chainable

console.log(mySet);
// Set [1, 5, "some text"]
```

1.32 Array.prototype.unshift()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/unshift

- The unshift() method adds one or more elements to the beginning of an array and returns the new length of the array.
- unshift(element0)
- unshift(element0, element1)

Examples using unshift()

```
const arr = [1, 2]

arr.unshift(0)
// result of the call is 3, which is the new array length
// arr is [0, 1, 2]

arr.unshift(-2, -1)
// the new array length is 5
// arr is [-2, -1, 0, 1, 2]

arr.unshift([-4, -3])
// the new array length is 6
// arr is [[-4, -3], -2, -1, 0, 1, 2]

arr.unshift([-7, -6], [-5])
// the new array length is 8
// arr is [ [-7, -6], [-5], [-4, -3], -2, -1, 0, 1, 2 ]
```

1.33 Set.prototype.size

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/size

- The size accessor property returns the number of (unique) elements in a Set object.
- The value of size is an integer representing how many entries the Set object has.
- A set accessor function for size is undefined; you cannot change this property.
- set.size()

Examples using size()

```
1 const mySet = new Set();
2 mySet.add(1);
3 mySet.add(5);
4 mySet.add('some text')
5
6 mySet.size;
7 // 3
```

1.34 Set.prototype.delete()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/delete

- The delete() method removes a specified value from a Set object, if it is in the set.
- delete(value)
- Returns true if value was already in Set; otherwise false.

Examples using delete()

```
const mySet = new Set();
mySet.add('foo');

mySet.delete('bar');
// Returns false. No "bar" element found to be deleted.
mySet.delete('foo');
// Returns true. Successfully removed.

mySet.has('foo');
// Returns false. The "foo" element is no longer present.
```

Examples using delete()

```
const setObj = new Set();
// Create a new set.

setObj.add({x: 10, y: 20});
// Add object in the set.

setObj.add({x: 20, y: 30});
// Add object in the set.

// Delete any point with `x > 10`.
setObj.forEach(function(point){
  if (point.x > 10){
    setObj.delete(point)
  }
})
```

1.35 delete operator

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/delete

- The JavaScript delete operator removes a property from an object; if no more references to the same property are held, it is eventually released automatically.
- delete expression
- true for all cases except when the property is an Object.hasOwn non-configurable property, in which case, false is returned in non-strict mode.

Example using delete operator

```
const Employee = {
  firstname: 'John',
  lastname: 'Doe'
};

console.log(Employee.firstname);
// expected output: "John"

delete Employee.firstname;

console.log(Employee.firstname);
// expected output: undefined
```

Example using delete operator

```
const trees = ['redwood', 'bay', 'cedar', 'oak', '
    maple'];
delete trees[3];
console.log(3 in trees); // false
```

1.36 Array.prototype.reverse()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reverse

- The reverse() method reverses an array in place. The first array element becomes the last, and the last array element becomes the first.
- reverse()
- return the reversed array

Example using reverse()

```
const array1 = ['one', 'two', 'three'];
console.log('array1:', array1);
// expected output: "array1:" Array ["one", "two", "three"]

const reversed = array1.reverse();
console.log('reversed:', reversed);
// expected output: "reversed:" Array ["three", "two", "one"]

// Careful: reverse is destructive -- it changes the original array.
console.log('array1:', array1);
// expected output: "array1:" Array ["three", "two", "one"]
```

Example using reverse() with array

```
const items = [1, 2, 3];
console.log(items);
// [1, 2, 3]

items.reverse();
console.log(items); ]
// [3, 2, 1]
```

Example using reverse() in array like object

```
const obj = {0: 1, 1: 2, 2: 3, length: 3};
console.log(obj);
// {0: 1, 1: 2, 2: 3, length: 3}

Array.prototype.reverse.call(obj);
//same syntax for using apply()
```

```
7 console.log(obj);
8 // {0: 3, 1: 2, 2: 1, length: 3}
```

1.37 Array() constructor

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/Array

- The Array() constructor is used to create Array objects.

Example Array() constructor

```
// literal constructor
[element0, element1, ..., elementN]

// construct from elements
new Array(element0, element1, ..., elementN)

// construct from array length
new Array(arrayLength)
```

Example using literal notation

```
let fruits = ['Apple', 'Banana'];

console.log(fruits.length);
// 2
console.log(fruits[0]);
// "Apple"
```

1.38 Math.trunc()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/Array

- The Math.trunc() function returns the integer part of a number by removing any fractional digits.
- Math.trunc(x)

Example Math.trunc()

```
console.log(Math.trunc(13.37));
// expected output: 13

console.log(Math.trunc(42.84));
// expected output: 42

console.log(Math.trunc(0.123));
// expected output: 0

console.log(Math.trunc(-0.123));
// expected output: -0
```

Example Math.trunc()

```
Math.trunc(13.37);     // 13
Math.trunc(42.84);     // 42
Math.trunc(0.123);     //  0
Math.trunc(-0.123);    // -0
Math.trunc('-1.123'); // -1
Math.trunc(NaN);       // NaN
Math.trunc('foo');     // NaN
Math.trunc();          // NaN
```

1.39 continue

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/continue

- The continue statement terminates execution of the statements in the current iteration of the current or labeled loop, and continues execution of the loop with the next iteration.
- continue [label];
- In a while loop, it jumps back to the condition.
- In a for loop, it jumps to the update expression.

Example continue

```
let text = '';

for (let i = 0; i < 10; i++) {
  if (i === 3) {
    continue;
  }
  text = text + i;
}

console.log(text);
// expected output: "012456789"
```

Example continue and label

```
let i = 0;
let j = 8;

checkiandj: while (i < 4) {
  console.log('i: ' + i);
  i += 1;

  checkj: while (j > 4) {
    console.log('j: ' + j);
    j -= 1;

    if ((j % 2) == 0)
      continue checkj;
    console.log(j + ' is odd.');
  }
  console.log('i = ' + i);
  console.log('j = ' + j);
}
```

Example output continue and label

```
i: 0

// start checkj
j: 8
7 is odd.
j: 7
j: 6
5 is odd.
j: 5
// end checkj

i = 1
j = 4

i: 1
i = 2
j = 4

i: 2
i = 3
j = 4

i: 3
i = 4
j = 4
```

1.40 Map.prototype.set()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/set

- The set() method adds or updates an entry in a Map object with a specified key and a value.
- set(key, value)
- The key of the element to add to the Map object. The key may be any JavaScript type (any primitive value or any type of JavaScript object).
- The value of the element to add to the Map object. The value may be any JavaScript type (any primitive value or any type of JavaScript object).
- The Map object.

Example map.set()

```
const map1 = new Map();
map1.set('bar', 'foo');

console.log(map1.get('bar'));
// expected output: "foo"

console.log(map1.get('baz'));
// expected output: undefined
```

1.41 String.prototype.substring()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/substring

- The substring() method returns the part of the string between the start and end indexes, or to the end of the string.
- substring(indexStart)
- substring(indexStart, indexEnd)

Example substring()

```
const str = 'Mozilla';

console.log(str.substring(1, 3));
// expected output: "oz"

console.log(str.substring(2));
// expected output: "zilla"
```

Example substring()

```
const anyString = 'Mozilla';

// Displays 'M'
console.log(anyString.substring(0, 1));
console.log(anyString.substring(1, 0));

// Displays 'Mozill'
console.log(anyString.substring(0, 6));

// Displays 'lla'
console.log(anyString.substring(4));
console.log(anyString.substring(4, 7));
console.log(anyString.substring(7, 4));

// Displays 'Mozilla'
console.log(anyString.substring(0, 7));
console.log(anyString.substring(0, 10));
```

1.42 String.prototype.toLowerCase() String.prototype.toUpperCase()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/toLowerCase https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/toUpperCase

- The toLowerCase() method returns the calling string value converted to lower case.
- toLowerCase()
- The toUpperCase() method returns the calling string value converted to uppercase (the value will be converted to a string if it isn't one).
- toUpperCase()

Example toLowerCase()

```
1 const sentence = 'The quick brown fox jumps over the
    lazy dog.';
2
3 console.log(sentence.toLowerCase());
4 // expected output: "the quick brown fox jumps over
    the lazy dog."
```

Example toUpperCase()

```
1 const sentence = 'The quick brown fox jumps over the
    lazy dog.';
2
3 console.log(sentence.toUpperCase());
4 // expected output: "THE QUICK BROWN FOX JUMPS OVER
    THE LAZY DOG."
```

1.43 this

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this

- In most cases, the value of this is determined by how a function is called (runtime binding). It can't be set by assignment during execution, and it may be different each time the function is called

Example this

```
const test = {
  prop: 42,
  func: function() {
    return this.prop;
  },
};

console.log(test.func());
// expected output: 42
```

1.44 Object.values()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_objects/Object/values

- The Object.values() method returns an array of a given object's own enumerable property values, in the same order as that provided by a for...in loop.
- (The only difference is that a for...in loop enumerates properties in the prototype chain as well.)
- Object.values(obj)

Example Object.values()

```
const object1 = {
  a: 'somestring',
  b: 42,
  c: false
};

console.log(Object.values(object1));
// expected output: Array ["somestring", 42, false]
```

Example values()

```
const obj = { foo: 'bar', baz: 42 };
console.log(Object.values(obj)); // ['bar', 42]

// Array-like object
const arrayLikeObj1 = { 0: 'a', 1: 'b', 2: 'c' };
console.log(Object.values(arrayLikeObj1 )); // ['a', '
    b', 'c']

// Array-like object with random key ordering
// When using numeric keys, the values are returned in
     the keys' numerical order
const arrayLikeObj2 = { 100: 'a', 2: 'b', 7: 'c' };
console.log(Object.values(arrayLikeObj2 )); // ['b', '
    c', 'a']

// getFoo is property which isn't enumerable
const my_obj = Object.create({}, { getFoo: { value() {
     return this.foo; } } });
my_obj.foo = 'bar';
console.log(Object.values(my_obj)); // ['bar']

```

```
// non-object argument will be coerced to an object
console.log(Object.values('foo')); // ['f', 'o', 'o']
```

1.45 constructor

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes/constructor

- The constructor method is a special method of a class for creating and initializing an object instance of that class.
- constructor() /* ... */
- constructor(argument0, argument1, /* ... ,*/ argumentN) /* ... */

Example Classes Constructor

```
class Polygon {
  constructor() {
    this.name = 'Polygon';
  }
}

const poly1 = new Polygon();

console.log(poly1.name);
// expected output: "Polygon"
```

1.46 Classes

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes

- Classes are a template for creating objects. They encapsulate data with code to work on that data.
- Classes in JS are built on prototypes but also have some syntax and semantics that are not shared with ES5 class-like semantics.

Example Classes

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
  // Getter
  get area() {
    return this.calcArea();
  }
  // Method
  calcArea() {
    return this.height * this.width;
  }
}

const square = new Rectangle(10, 10);

console.log(square.area); // 100
```

1.47 Array.from()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/from

- The Array.from() static method creates a new, shallow-copied Array instance from an iterable or array-like object.

Example from()

```
1 console.log(Array.from('foo'));
2 // expected output: Array ["f", "o", "o"]
3
4 console.log(Array.from([1, 2, 3], x => x + x));
5 // expected output: Array [2, 4, 6]
```

Examples from()

```
1  // Arrow function
2  Array.from(arrayLike, (element) => {stuff})
3  Array.from(arrayLike, (element, index) => {stuff})
4
5  // Mapping function
6  Array.from(arrayLike, mapFn)
7  Array.from(arrayLike, mapFn, thisArg)
8
9  // Inline mapping function
10 Array.from(arrayLike, function mapFn(element) {stuff})
11 Array.from(arrayLike, function mapFn(element, index) {
       stuff})
12 Array.from(arrayLike, function mapFn(element) {stuff},
       thisArg)
13 Array.from(arrayLike, function mapFn(element, index) {
       stuff}, thisArg)
```

1.48 Array.prototype.forEach()

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach

- The forEach() method executes a provided function once for each array element.

Example forEach()

```
const array1 = ['a', 'b', 'c'];

array1.forEach(element => console.log(element));

// expected output: "a"
// expected output: "b"
// expected output: "c"
```

Examples forEach()

```
const words = ['one', 'two', 'three', 'four'];
words.forEach((word) => {
  console.log(word);
  if (word === 'two') {
    words.shift(); //'one' will delete from array
  }
}); // one // two // four

console.log(words); // ['two', 'three', 'four']
```

1.49 Map() constructor

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/Map

- The Map() constructor creates Map objects.
- new Map()
- new Map(iterable)

Example new Map()

```
let myMap = new Map([
  [1, 'one'],
  [2, 'two'],
  [3, 'three'],
])
```

2 Array Algorithm Problems

2.1 Build Array from Permutation

https://leetcode.com/problems/build-array-from-permutation/

1. initialize result array
2. loop through nums
3. push nums at each i to the answer
4. return result

Solution with for loop

```
const buildArray = (nums) => {

    let result = [];

    for (let i = 0; i<nums.length; i++) {
        result.push(nums[nums[i]]);
    }

    return result;
};
```

Solution with map()

```
const buildArray = (nums) => {

    return nums.map(result => nums[result]);

};
```

2.2 Count Items Matching a Rule

https://leetcode.com/problems/count-items-matching-a-rule/

1. create rule index object
2. reduce(ans, items) so that ruleIndex[ruleKey] is equal to ruleValue
3. if equal increase ans by 1
4. if not equal answer is 0

Solution countMatches

```
1  const countMatches = (items, ruleKey, ruleValue) => {
2
3      let ruleIndex = {
4          'type': 0,
5          'color': 1,
6          'name': 2
7      };
8
9      return items.reduce((ans,item) =>
10     item[ruleIndex[ruleKey]] === ruleValue ? ans + 1 :
       ans, 0);
11 };
```

2.3 Create Target Array in the Given Order

https://leetcode.com/problems/create-target-array-in-the-given-order/

1. initialize empty target array
2. loop through array
3. loop though each index of the previous loop
4. if array[i] is less than or equal to inner loop, increment inner loop
5. loop in the nums array giving target the value: target[array[j]] = nums[i]
6. return target

Solution createTargetArray

```
const createTargetArray = (nums, array) => {

    let result = [];

    for ( let i=0; i<array.length; i++) {
        for (let j=0; j<i; j++) {
            if (array[i] <= array[j]) {
                array[j]++;
            }
        }
    }

    for (let i in nums) {
        result[array[i]] = nums[i];
    }

    return result;
};
```

Solution createTargetArray with splice()

```
const createTargetArray = (nums, index) => {

    let target = []

    for(let i in nums){
        target.splice(index[i], 0, nums[i])
    }

    return target
};
```

2.4 Decode XORed Array

https://leetcode.com/problems/decode-xored-array/

1. initialize result with first in array
2. for loop through encoded
3. push result through result[i] ^ encoded[i]
4. return result

Solution decode

```
const decode = (encoded, first) => {

    let result = [first];

    for (let i = 0; i<encoded.length; i++) {
        result.push(result[i] ^ encoded[i]);
    }

    return result;
};
```

2.5 Decompress Run-Length Encoded List

https://leetcode.com/problems/decompress-run-length-encoded-list/

1. initialize empty result array
2. loop through nums, starting from 1, increasing by 2 each time
3. push result into a new array nums[i-1]
4. fill new array with nums[i]
5. return result

Solution decompressRLEList

```
const decompressRLEList = (nums) => {

    let result = [];

    for (let i=1; i< nums.length; i+=2) {
        result.push(...new Array(nums[i-1]).fill(nums[
   i]));
    }

    return result;
};
```

2.6 Final Value of Variable After Performing Operations

https://leetcode.com/problems/final-value-of-variable-after-performing-operations/

1. initialize count at 0
2. loop through the values of operations (not the index)
3. if i is equal to 'X++' or '++X' increment
4. else decrement
5. return final count

Solution finalValueAfterOperations

```
const finalValueAfterOperations = (operations) => {

    let result = 0;

    for (let i of operations) {
        if (i === 'X++' || i === '++X') {
            result++;
        } else result --;
    }

    return result;
};
```

2.7 Concatenation of Array

https://leetcode.com/problems/concatenation-of-array/

1. take nums and split it using ... operator
2. return nums twice in a single [ ]

Solution getConcatenation

```
const getConcatenation = (nums) => {

    return [...nums, ...nums];

}
```

2.8 Kids With the Greatest Number of Candies

https://leetcode.com/problems/kids-with-the-greatest-number-of-candies/

1. initialize results array
2. initialize max value tracker
3. loop through candies array
4. create logic for max value
5. loop through candies[i]
6. test each candies with extraCandies to see if value is true
7. return result

Solution kidsWithCandies with for-of loop

```
const kidsWithCandies = (candies, extraCandies) => {

    let result = [];
    let max = 0;

    for (const val of candies) {
        val > max && (max = val);
    }
    for (let i = 0; i< candies.length; ++i) {
        result.push(candies[i] + extraCandies >= max);
    }

    return result;
};
```

Solution kidsWithCandies with for loop

```
var kidsWithCandies = function(candies, extraCandies)
   {

    let answer = []
    let max = 0

    for(let i=0; i<candies.length; i++){
        candies[i] > max && (max = candies[i])
    }

    for(let j=0; j<candies.length; j++){
        answer.push(candies[j] + extraCandies >= max)
```

```
    }
    return answer
};
```

2.9 Maximum Number of Words Found in Sentences

https://leetcode.com/problems/maximum-number-of-words-found-in-sentences/

1. Initialize max and temp variables at zero
2. Loop through sentence.length
3. Split sentences based on parentheses and add the length of each to temp variable
4. Find which value of temp or max is biggest and pass to max
5. Return max

Solution mostWordsFound

```
const mostWordsFound = (sentence) => {

    let result = 0;
    let temp = 0;

    for (let i=0; i < sentence.length; i++) {
        temp = sentence[i].split(" ").length;
        if(temp>result){
            result = temp;
        }
    }

    return result;
};
```

Solution mostWordsFound with for-in loop and Math.max

```
const mostWordsFound = (sentence) => {

    let max = 0;
    let temp = 0;

    for (let i in sentence) {
        temp = sentence[i].split(" ").length;
    }

    return Math.max(max, temp)
}
```

2.10 Number of Good Pairs

https://leetcode.com/problems/number-of-good-pairs/

1. initialize counter
2. loop through nums array
3. loop again starting at i+1
4. logic if(nums[i] === nums[j] increase count
5. return count

Solution numIdenticalPairs

```
const numIdenticalPairs =  (nums) => {

    let result = 0;

    for (let i=0; i<nums.length; i++) {
        for (let j=i+1; j<nums.length; j++) {
            if (nums[i] === nums[j]) {
                result++;
            }
        }
    }

    return result;
};
```

2.11 Shuffle String

https://leetcode.com/problems/shuffle-string/

1. initialize empty results array
2. loop through idices array
3. tie the indices to the s parameter
4. return the result and join() it together since we are looking for a string

Solution restoreString

```
const restoreString = (s, indices) => {

    let result = [];

    for(let i = 0; i<indices.length; i++){
        result[indices[i]] = s[i];
    }

    return result.join('');
};
```

2.12 Richest Customer Wealth

https://leetcode.com/problems/richest-customer-wealth/

1. initialize an answer
2. loop through the people, accounts
3. initialize temp variable
4. loop through the banks accounts[i]
5. add all their wealth to temp
6. use math.max to find the highest number
7. return the account with most wealth

Solution maximumWealth

```
const maximumWealth = (accounts) => {

    let result = 0;

    for(let i=0; i<accounts.length; i++){
        let temp = 0;
        for(let j = 0; j< accounts[i].length; j++){
            temp += accounts[i][j];
        }
        result = Math.max(result, temp);
    }

    return result;
};
```

2.13 Running Sum of 1d Array

https://leetcode.com/problems/running-sum-of-1d-array/

1. create loop from 1(i) to end of array
2. add nums[i] and nums[i-1] to array
3. return nums array

Solution runningSum

```
const runningSum = (nums) => {

    for (let i = 1; i < nums.length; i++) {
        nums[i] += nums [i-1];
    }

    return nums;
};
```

2.14 Shuffle the Array

https://leetcode.com/problems/shuffle-the-array/

1. create empty result array
2. loop through n with i¡ n
3. push into nums[i]
4. push into nums[i+n]
5. return result

Solution shuffle

```
const shuffle =  (nums, n) => {

    let result = [];

    for (let i = 0; i < n; i++) {
        result.push(nums[i]);
        result.push(nums[i+n]);
    }

    return result;
};
```

2.15 How Many Numbers Are Smaller Than the Current Number

leetcode.com/problems/how-many-numbers-are-smaller-than-the-current-number/

1. Initialize empty results array
2. Initialize sorted array
3. Slice() sorted array to duplicated
4. Sort() sorted array using ((a, b) → a - b) so it works for numbers
5. For loop through nums
6. Compare the value of sorted array with nums array and push into results the indexOf at which sorted is in nums[i]
7. Return results

Solution smallerNumbersThanCurrent

```
const smallerNumbersThanCurrent = (nums) => {

    let result = [];
    let sorted = nums.slice().sort((a,b)=>a-b);

    for (let i = 0; i<nums.length; i++) {
        result.push(sorted.indexOf(nums[i]));
    }

    return result;
};
```

2.16 Sum of All Odd Length Subarrays

https://leetcode.com/problems/sum-of-all-odd-length-subarrays/

1. Initialize result at 0
2. For loop through arr starting at 1
3. Start from second element and add to itself the first element and so on. So each index has the value of all previous indexes added to it
4. loop through all start indices
5. End is initially start and look for all odd (+=2) subarrays
6. Sum up += the query in result
7. Return result
8. Setup helper sumBetween function that takes sum from start to end

Solution sumOddLengthSubarrays with helper function

```
const sumOddLengthSubarrays = (arr) => {

    let result = 0;

    for(let i = 1; i<arr.length; i++){
        arr[i] += arr[i - 1];
    }

    for(let start = 0; start<arr.length; start++){
        for(let end = start; end<arr.length; end +=2){
            result += sumBetween(start, end);
        }
    }

    return result;

    function sumBetween(start, end){
        return arr[end] - (arr[start -1] || 0);
    };
};
```

Solution sumOddLengthSubarrays with parseInt

```
const sumOddLengthSubarrays = (arr) => {

    let result = 0;
```

```
    let n = arr.length;

    for(let i = 0; i< n; ++i) {
        result += parseInt(((i + 1) * (n - i) + 1) /
   2) * arr[i];
    }

    return result;
};
```

Solution sumOddLengthSubarrays with for loops and modulo

```
const sumOddLengthSubarrays = (arr) => {

    let result = 0;

    for(let i=0; i<arr.length; i++){
        for(let j=i; j<arr.length; j++){
            if((i-j)%2==0){
                for(let k=i;k<=j;k++){
                    result+=arr[k];
                }
            }
        }
    }

    return result;
}
```

Solution sumOddLengthSubarrays with while loop

```
var sumOddLengthSubarrays = function(arr) {

    let i=1;
    let sum=0;

     while(i<=arr.length){
         for(let j=0;j<=arr.length-i;j++){
              for(let k=j;k<i+j;k++){
                 sum+=arr[k]
              }
         }
         i+=2
     }

     return sum
```

```
16 };
```

2.17 Count Number of Pairs With Absolute Difference K

https://leetcode.com/problems/count-number-of-pairs-with-absolute-difference-k/

1. Create empty object
2. Initialize answer at zero
3. Loop through nums giving values to the object
4. Loop through nums if obj[nums[i]-k] += that value to answer
5. Return Answer

Solution countKDifference

```
const countKDifference = (nums, k) => {

    let obj = {};
    let result = 0;

    for (let i of nums) {
        obj[i] ? obj[i]++ : obj[i] = 1;
    }

    for (let i=0; i<nums.length; i++) {
        if(obj[nums[i] -k]){
            result += obj[nums[i] - k];
        }
    }

    return result;
};
```

2.18 Minimum Number of Moves to Seat Everyone

https://leetcode.com/problems/minimum-number-of-moves-to-seat-everyone/

1. Initialize answer variable
2. Sort seats
3. Sort students
4. Loop through arrays finding the absolute value of the number and += to answer
5. Return answer

Solution minMovesToSeat

```
const minMovesToSeat = (seats, students) => {

    let result = 0;

    seats = seats.sort((a,b) => a-b);
    students = students.sort((a,b) => a-b);

    for (let i in seats) {
        result += Math.abs(seats[i] - students[i]);
    }

    return result;
};
```

2.19 Minimum Amount of Time to Fill Cups

https://leetcode.com/problems/minimum-amount-of-time-to-fill-cups/

1. Initalize answer to zero
2. Sort the amount array
3. Whie loop amount 1 and 2 decrement them until zero while increasing answer
4. Sort in the while loop to keep the biggest amounts furthest away
5. Add the remaining amount to the answer in second (only 1 amount so will always be 1 per second)
6. Return answer

Solution fillCups

```
const fillCups = (amount) => {

    let result = 0;

    amount.sort((a, b) => a - b);

    while (amount[1] && amount[2]) {
        answer++;
        amount[1]--;
        amount[2]--;
        amount.sort((a, b) => a - b);
    }

    result += amount[2];
    return result;
};
```

2.20 Assign Cookies

https://leetcode.com/problems/assign-cookies/

1. Initalize empty answer
2. Sort (g) children and (s) cookies
3. Whie loop amount 1 and 2 decrement them until zero while increasing answer
4. Loop with forEach through (s) and if child is greater than g[answer] increment answer (this needs work not sure exactly what is happening here)
5. Return answer

Solution findContentChildren

```
const findContentChildren = (g,s) => {

    let result = 0;

    g.sort((a,b) => a-b);
    s.sort((a,b) => a-b);

    s.forEach((child) => {
        if(child >= g[result]){
            result++;
        }
    })

    return result;
}
```

3 String Algorithm Problems

3.1 Defanging an IP Address

https://leetcode.com/problems/defanging-an-ip-address/

1. split and join the address
2. return return address

Solution defangIPaddr

```
const defangIPaddr = (address) => {

    return address.split('.').join('[.]');

};
```

3.2 Jewels and Stones

https://leetcode.com/problems/jewels-and-stones/

1. Initialize count to zero
2. Loop through jewels string
3. Loop through stones string
4. Logic to check if each part of stones is equal to jewels
5. If equal add to count
6. Return count

Solution numJewelsInStones

```
const numJewelsInStones = (jewels, stones) => {

    let result = 0;

    for(let i = 0; i<jewels.length; i++){
        for(let j=0; j<stones.length; j++){
            if(stones[j] === jewels[i]){
                result ++;
            }
        }
    }

    return result;
};
```

3.3 Goal Parser Interpretation

https://leetcode.com/problems/goal-parser-interpretation/

1. Split command by () join o in place
2. Split command by (al) join al in place
3. Return command

Solution interpret

```
const interpret = (command) => {

    return command.split('()').join('o').split('(al)')
   .join('al');

};
```

3.4 Cells in a Range on an Excel Sheet

https://leetcode.com/problems/cells-in-a-range-on-an-excel-sheet/

1. Initialize empty answer array
2. Initialize array with string s [col1, row1, EMPTY SPACE, col2, row2] = s
3. Loop through columns - c with charCodeAt
4. Loop through rows - r with +row
5. Push answer to a string using fromCharCode(c) + r
6. Return answer

Solution cellsInRange

```
const cellsInRange = (s) => {

    let result = [];
    let [col1, row1, , col2, row2] = s;

    for(let c = col1.charCodeAt(0), ce = col2.
   charCodeAt(0); c<= ce; c++){
        for(let r = +row1, re = +row2; r<=re; r++){
            result.push(String.fromCharCode(c) + r);
        }
    }

    return result;
};
```

3.5 Split a String in Balanced Strings

https://leetcode.com/problems/split-a-string-in-balanced-strings/

1. Initialize empty matches variable
2. Initalize empty balance variable
3. Loop through s.length
4. If each S[i] is equal to L add +1 to balance, else reduce it by one
5. If balance is equal to zero add +1 to matches
6. Return matches

Solution balancedStringSplit with balance counter

```
const balancedStringSplit = (s) => {

    let result = 0;
    let balance = 0;

    for (let i = 0; i < s.length; i++) {
        s[i] === 'L' ? balance++ : balance--;
        if (balance === 0) {
            result++;
        }
    }

    return result;
};
```

Solution balancedStringSplit with stack

```
const balancedStringSplitStack = (s) => {

    let result = 0;
    let stack = [];

    stack.push(s[0])

    for (let i = 1; i < s.length; i++) {
        let top = stack[stack.length - 1];
        if (top !== undefined && top !== s[i]) {
          stack.pop();
        } else {
            stack.push(s[i]);
```

```
14          }
15          if (stack.length === 0) {
16              result += 1;
17          }
18      }
19
20      return result;
21  };
```

3.6 Sorting the Sentence

https://leetcode.com/problems/sorting-the-sentence/

1. Return s.split to put into parts
2. Sort by number orders and use x[x.length-1] to find last item (the number)
3. Map to remove the last item (the number) from the words
4. Join(' ') to turn it back into a string as opposed to array

Solution sortSentence

```
const sortSentence = (s) => {

    return s.split(' ')
        .sort((a,b) => a[a.length-1] - b[b.length-1])
        .map((word) => word.slice(0, word.length-1))
        .join(' ');

};
```

3.7 Decode the Message

https://leetcode.com/problems/decode-the-message/

1. Create map variable of empty map()
2. Set start at 97
3. Initialize empty result string
4. For of loop through key if map has i or is empty continue
5. If not set i as a string fromCharCode(start++)
6. map.set(' ', ' ')
7. For of loop through message and add map.get(m) to result
8. Return result

Solution decodeMessage

```
const decodeMessage = (key, message) => {

    let map = new Map();
    let start=97;
    let result="";

    for(let i=0; i<key.length;i++){
        if(map.has(key[i])||key[i]===" "){
            continue;
        } else {
        map.set(key[i],String.fromCharCode(start++))
        }
    }

    map.set(" "," ");

    for(let m of message){
        result+=map.get(m);
    }

    return result;
};
```

3.8 Maximum Nesting Depth of the Parentheses

https://leetcode.com/problems/maximum-nesting-depth-of-the-parentheses/

1. Initialize answer int at zero
2. Initialize counter int at zero
3. For loop through s
4. If s[i] is equal to (answer.math.max(answer, ++counter)
5. Else decrement counter
6. Return answer

Solution maxDepth

```
const maxDepth = (s) => {

    let result = 0;
    let count = 0;

    for (let i = 0; i < s.length;  i++) {
        if (s[i] === '(') {
            result = Math.max(result, ++count);
        } else if (s[i] === ')') {
            count--;
        }
    }

    return result
};
```

Solution maxDepth with split() and reduce()

```
const maxDepth = (s) => {

    let l = 0;
    let r = 0;

    return s.split('').reduce((depth, c) => {
        if (c === '(') l++;
        if (c === ')') r++;

        return Math.max(l - r, depth);
    }, 0)
}
```

3.9 Fizz Buzz

https://leetcode.com/problems/fizz-buzz/

1. Initialize empty answer array
2. Loop through n starting at 1 and ending when ¡= n
3. If i modulo 15 is equal to zero push string 'FizzBuzz'
4. Else if i modulo 5 is equal to zero push string 'Buzz'
5. Else if i modulo 3 is equal to zero push string 'Fizz'
6. Else push string i
7. Return snswer

Solution fizzBuzz with for loop

```
const fizzBuzz = (n) => {

    const result = [];

    for (let i=1; i<=n; i++) {
        if (i % 15 === 0 ) {
            result.push('FizzBuzz');
        } else if (i % 3 === 0) {
            result.push('Fizz');
        } else if (i % 5 === 0) {
            result.push('Buzz');
        } else {
            result.push(i.toString());
        }
    }

    return result;
};
```

Solution fizzBuzz ES6

```
var fizzBuzz = function(n) {
    return new Array(n).fill(0).map((a, i) => (++i % 3
    ? '' : 'Fizz') + (i % 5 ? '' : 'Buzz') || '' + i)
};
```

3.10 Check If Two String Arrays are Equivalent

https://leetcode.com/problems/check-if-two-string-arrays-are-equivalent/

1. Join(") word1 and word2 so they become strings
2. Compare the two strings so they are equal
3. Return true if equal
4. Return false if not equal

Solution arrayStringsAreEqual

```
const arrayStringsAreEqual = (word1, word2) => {

    let w1 = word1.join('');
    let w2 = word2.join('');

    if (w1 === w2) {
        return true;
    } else {
        return false;
    }
};
```

Solution arrayStringsAreEqual ES6

```
var arrayStringsAreEqual = function(word1, word2) {
    return word1.join('') === word2.join('')
};
```

3.11 Count the Number of Consistent Strings

https://leetcode.com/problems/count-the-number-of-consistent-strings/

1. Initialize empty answer array
2. For of loop through i of words
3. Set word equals to i
4. Loop through word.length and set wordChar as word[j]
5. If !allowed includes wordChar break
6. Else if j===word.length -1 push word.length to answer
7. Return answer.length

Solution countConsistentStrings with loops

```
const countConsistentStrings = (allowed, words) => {

    let result = [];

    for(let i of words){
        let word = i;
        for(let j = 0; j<word.length; j++){
            let wordChar = word[j];
            if(!allowed.includes(wordChar)){
              break;
            } else if(j === word.length-1){
                result.push(word.length);
            }
        }
    }

    return result.length;
};
```

Solution countConsistentStrings ES6

```
const countConsistentStrings = (allowed, words) => {

    let set = new Set(allowed)

    return words.reduce((a, w) => {
      return w.split('').every(l => set.has(l)) ? ++a
    : a
    }, 0)
```

```
8
9     }
```

3.12 To Lower Case

https://leetcode.com/problems/to-lower-case/

1. Return s.toLowerCase()

Solution toLowerCase

```
const toLowerCase = (s) => {

    return s.toLowerCase()

};
```

Solution toLowerCase with hex code

```
//hex code for letter 'A' is 0x41 and for letter 'a'
   is 0x61. That's a diff of 0x20.
const toLowerCase = (str) => {

    let i = 0;
    let result = "";

    while (i < str.length) {
        result += String.fromCharCode(str.charCodeAt(i
   ) | 0x20);
        i++;
    }

    return result;
};
```

3.13 Rings and Rods

https://leetcode.com/problems/rings-and-rods/

1. Initialize new Map() called map
2. Initialize answer int to zero
3. Loop through rings.length
4. Set color to rings[i]
5. Set rod to +rings[i+1]
6. If !map has rod Map.set rod, new Set()
7. Map.get rod.add(color)
8. Loop through map with [rod, setColors]
9. If set color size is equal to 3 answer ++
10. Return answer

Solution countPoints

```
const countPoints = (rings) => {

    let map = new Map;
    let result = 0;

    for(let i = 0; i<rings.length; i += 2){
        let color = rings[i];
        let rod = +rings[i + 1];
        if(!map.has(rod))
            map.set(rod, new Set());
            map.get(rod).add(color);
    }

    for(let [rod, setColors] of map){
        if(setColors.size === 3){
            result++;
        }
    }

    return result;
};
```

Solution countPoints is this slower?

```
var countPoints = function(rings) {

    let rods = Array(10).fill("");

    for(let i = 0; i < rings.length; i += 2){
        if(!(rods[rings[i+1]].includes(rings[i])))
   rods[rings[i+1]] += rings[i]
    }

    return rods.filter(rod => rod.length > 2).length
};
```

3.14 Truncate Sentence

https://leetcode.com/problems/truncate-sentence/

1. Split string so that it becomes array
2. Slice the array from index 0 to index k (desired length)
3. Join the array into a string (' ') removing commas
4. Return the string

Solution truncateSentence

```
const truncateSentence = (s, k) => {

    return s.split(' ').slice(0,k).join(' ');

 };
```

3.15 Check if the Sentence Is Pangram

https://leetcode.com/problems/check-if-the-sentence-is-pangram/

1. Make new Set (what does this mean?)
2. Split(") the sentence
3. Set size to equal 26
4. Return

Solution checkIfPangram

```
const checkIfPangram = (sentence) => {

    return new Set(sentence.split("")).size === 26;

};
```

3.16 Count Asterisks

https://leetcode.com/problems/count-asterisks/

1. Set count and result counters
2. For loop through s.length
3. If s[i] === — increment count
4. If count is multiple of two and s[i] = * then increment result
5. Return result

Solution countAsterisks

```
const countAsterisks = (s) => {

    let count = 0;
    let result = 0;

    for (let i = 0; i < s.length; i++) {
        if (s[i] === '|') {
           count ++
        }
        if (count % 2 === 0 && s[i] === '*') {
            result ++
        }
    }

    return result;
};
```

My messy solution while learning

```
var countAsterisks = function(s) {

    let count = 0
    let  s2 = s.split('|')
    let temp = []

     for(let i=0; i<s2.length; i+=2){
         temp = s2[i].split('').join('')
         for(let j=0; j<temp.length;j++){
             if(temp[j] === '*'){
                 count++
             }
         }
```

```
14      }
15
16      return count
17  };
```

3.17 Unique Morse Code Words

https://leetcode.com/problems/unique-morse-code-words/

1. Set const morse to morse code from questions
2. Initialize answer to new Set
3. For of loop through words with word
4. Initialize transform as empty string
5. For loop through word to seperate each character
6. Set i euqal to char.charCodeAt(0) - 97
7. Add morse[i] to transform
8. Add transform to answer Set
9. Return size of answer set

Solution uniqueMorseRepresentations

```
let morse = [".-","-...","-.-.","-..",".","..-.","--."
    ,"....","..",".---","-.-",".-..","--","-.","---","
    .--.","--.-",".-.","...","-","..-","...-",".--","
    -..-","-.--","--.."];

const uniqueMorseRepresentations = (words) => {

    let result = new Set;

    for (let word of words) {
        let transform = '';
        for (let char of word) {
            let i = char.charCodeAt(0) - 97;
            transform += morse[i];
        }
        result.add(transform);
    }

    return result.size;
};
```

Solution uniqueMorseRepresentations with object

```
const alphabet = {
```

```
    a: '.-', b: '-...',   c: '-.-.', d: '-..', e: '.',
    f: '..-.', g: '--.', h: '....', i: '..',  j: '.---
    ',  k: '-.-',  l: '.-..', m: '--',
    n: '-.',  o: '---', p: '.--.',  q: '--.-',  r: '
    .-.', s: '...', t: '-', u: '..-', v: '...-', w: '
    .--', x: '-..-',  y: '-.--', z: '--..'
}

const uniqueMorseRepresentations = words => {

    return new Set(words.map(word => word.split('').
    map(letter => alphabet[letter]).join(''))).size
}
```

3.18 Number of Strings That Appear as Substrings in Word

leetcode.com/problems/number-of-strings-that-appear-as-substrings-in-word/

1. Initialize answer to zero
2. Loop through patterns
3. If word includes(patter[i]) increment answer
4. Return answer

Solution numOfStrings

```
const numOfStrings = (patterns, word) => {

    let result = 0;

    for (let i = 0; i<patterns.length ; i++) {
        if (word.includes(patterns[i])) {
            result++;
        }
    }

return result;
};
```

3.19 Remove Outermost Parentheses

https://leetcode.com/problems/remove-outermost-parentheses/

1. Initialize empty answer string
2. Initialize counter to zero
3. Loop through s
4. If s[i] is equal to (if counter is greater than zero add (to answer string and increment counter
5. Else if s[i] is equal to) and counter greater than one add) to answer string and decrement counter
6. Return answer

Solution removeOuterParentheses

```
const removeOuterParentheses = (s) => {

    let result = '';
    let counter = 0;

    for (let i = 0; i<s.length; i++) {
        if (s[i] === '(') {
            if (counter > 0) {
                result += '(';
            }
            counter++;
        } else if(s[i] === ')') {
            if (counter > 1) {
                result += ')';
            }
            counter--;
        }
    }

    return result;
};
```

3.20 Replace All Digits with Characters

https://leetcode.com/problems/replace-all-digits-with-characters/

1. loop through s starting at 1 += 2
2. Initialize value to String fromCharCode(s[i-1].charCodeAt()+Number(s[i]))
3. Update s to equal s.replace(s[i], value)
4. Return s

Solution replaceDigits

```
const replaceDigits = (s) => {

    for (let i = 1; i<s.length; i+=2) {
        let value = String.fromCharCode(s[i-1].
   charCodeAt()+Number(s[i]));
        s = s.replace(s[i], value);
    }

    return s;
};
```

4 Hash Tables

4.1 Destination City

https://leetcode.com/problems/destination-city/

1. Create new set named departure
2. Loop through paths and add path[0] to departure
3. Loop through paths again and check if subarray has second index value (if it does not it is unique)
4. If no second index value, return that value (should be destination)

Solution destCity

```
const destCity = (paths) => {

    let set = new Set();

    for (let path of paths) {
        set.add(path[0]);
    }
    for (let path of paths) {
        if (!set.has(path[1])) {
            return path[1];
        }
    }
};
```

4.2 Maximum Number of Pairs in Array

https://leetcode.com/problems/maximum-number-of-pairs-in-array/

1. Initialize new map and count
2. Loop through nums
3. If map[nums[i]] then go ahead and delete map[nums[i]] and increment count
4. Else replace map[nums[i]] with one
5. Return [count, Object.values(map).length]

Solution numberOfPairs

```
const numberOfPairs = (nums) => {

    let map = new Map();
    let result = 0;


    for (let i=0;i<nums.length;i++) {
        if (map[nums[i]]) {
            delete(map[nums[i]]);
            result++;
        } else {
            map[nums[i]] = 1;
        }
    }

    return [result, Object.values(map).length];
};
```

4.3 Design an Ordered Stream

https://leetcode.com/problems/design-an-ordered-stream/

1. Created class OrderedStream
2. Setup constructor
3. Set pointed to zero and initialize empty list
4. Insert (id, value)
5. Initialize empty chunk array
6. Set this.list[id-1] to value
7. While loop through (this.list[this.pointer])
8. Push (this.list[this.pointer]) into chunk
9. Increment this.pointer
10. Return chunk

Solution orderedStream

```
class OrderedStream {
    constructor(n) {
      this.pointer = 0;
      this.list = [];
    }

    insert(id, value) {
      let chunk = [];
      this.list[id - 1] = value;
      while(this.list[this.pointer]) {
        chunk.push(this.list[this.pointer]);
        this.pointer++;
      }

      return chunk;
    }
  }
```

4.4 Increasing Decreasing String

https://leetcode.com/problems/increasing-decreasing-string/

1. Initialize arr with Array.from(s) (splitting s into an array of its parts)
2. Initialize empty answer string
3. Sort arr
4. While loop through arr.length
5. Filter arr so that if i === 0 or x !== answer.length-1 increase answer by x and return false
6. Else return true
7. Reverse arr
8. Return answer

Solution sortString

```
const sortString = (s) => {

    let arr = Array.from(s);
    let result = '';

    arr.sort();

    while (arr.length) {
      arr = arr.filter((x, i) => {
        if (i === 0 || x !== result[result.length -
   1]) {
          result += x;
          return false;
        }
        return true;
      });
      arr.reverse();
    }

    return result;
};
```

4.5 Check if All Characters Have Equal Number of Occurrences

https://leetcode.com/problems/check-if-all-characters-have-equal-number-of-occurrences/

1. Initialize empty Map
2. Loop through letters of s
3. Map[letter] is equal to itself or zero + 1 (setup count of each letter)
4. Initialize occurences equal to the set of Object.values(map)
5. If occurences.size === 1 (all letter groups are equal) return true
6. Else return false

Solution areOccurrencesEqual with Map

```
const areOccurrencesEqual = (s) => {

    let map = new Map();

    for (let letter of s) {
        map[letter] = (map[letter] || 0) + 1;
    }

    let count = new Set(Object.values(map));
    if(count.size === 1) return true;
    return false;
};
```

Solution areOccurrencesEqual with reduce()

```
const areOccurrencesEqualReduce = (s) => {

    let count = s.split('').reduce((obj, cur)=>{
      obj.hasOwnProperty(cur) ? obj[cur] += 1 : obj[
   cur] = 1
      return obj
    },{})

    return new Set(Object.values(count)).size === 1
  };
```

4.6 Divide Array Into Equal Pairs

https://leetcode.com/problems/divide-array-into-equal-pairs/

1. Create empty map ()
2. Loop through nums and if map has number delete it else set the number true
3. Return true if map size is empty else return false

Solution divideArray

```
const divideArray = (nums) => {

    let map = new Map();

    for (const number of nums) {
      map.has(number) ? map.delete(number) : map.set(
   number, true);
    }

    return map.size === 0;
  };
```

4.7 N-Repeated Element in Size 2N Array

https://leetcode.com/problems/n-repeated-element-in-size-2n-array/

1. Initialize empty Map()
2. Loop through values of nums
3. If value is in map return value
4. Else return zero

Solution repeatedNTimes with Map()

```
const repeatedNTimes = (nums) => {

    let map = new Map();

    for ( let num of nums) {
        if (num in map) {
            return num;
        } else {
            map[num] = 0;
        }
    }
};
```

Solution repeatedNTimes with ES6

```
var repeatedNTimes = function(nums) {
    return nums.find((a, index, array) => array.
   indexOf(a) !== index)
};
```

Solution repeatedNTimes with Set()

```
const repeatedNTimes = (A) => {

    let set = new Set();

    for(let a of A) {
        if(set.has(a))
            return a;

        set.add(a);
    }
};
```

4.8 Sum of Unique Elements

https://leetcode.com/problems/sum-of-unique-elements/

1. Create empty Map()
2. Initialize sum at zero
3. Loop through nums if map[n] is undefined increase sum by n and set map[n] to true
4. Else if decrement n from sum and set map[n] to false
5. Return sum

Solution sumOfUnique with Map()

```
const sumOfUnique = (nums) => {

    let map = new Map();
    let result = 0;

    for (let n of nums) {
        if (map[n] === undefined) {
            result += n;
            map[n] = true;
        } else if(map[n]){
            result -= n;
            map[n] = false;
        }
    }

    return result;
};
```

Solution sumOfUnique with forEach

```
var sumOfUnique = function(nums) {
    let result = []
    nums.forEach(function(e){
        if(nums.indexOf(e)  == nums.lastIndexOf(e)){
            result.push(e)
        }
    })
    return result.reduce((a,b) => a + b , 0)
}
```

4.9 Maximum Number of Balls in a Box

https://leetcode.com/problems/maximum-number-of-balls-in-a-box/

1. Create empty Map()
2. Loop from low to high limit incrementing i and setting sum to zero
3. Loop through i with j using Math.trunc for the numbers that are more digits
4. Map.set the numbers so the map is counting the values
5. Return with math.max the map of split map with the highest values

Solution countBalls

```
const countBalls = (lowLimit, highLimit) => {

    let map = new Map();

    for (let i = lowLimit, sum = 0; i <= highLimit; i
   ++, sum = 0) {
       for (let j = i; j; j = Math.trunc(j / 10)) {
           sum += j % 10;
       }
       map.set(sum, (map.get(sum) || 0) + 1);
    }

    return Math.max(...map.values());
};
```

4.10 Check if Number Has Equal Digit Count and Digit Value

https://leetcode.com/problems/check-if-number-has-equal-digit-count-and-digit-value/

1. Create array with num length and fill it with zero
2. For of Loop through num and increase arr[Number(i)++]
3. return arr.join(") if it is equal to num

Solution digitCount

```
const digitCount = (num) => {

    let arr= Array(num.length).fill(0);

    for (let i of num) {
        arr[Number(i)]++;
    }

    return arr.join('') === num;
};
```

4.11 Keep Multiplying Found Values by Two

https://leetcode.com/problems/keep-multiplying-found-values-by-two/submissions/

1. Loop through nums
2. If nums includes original value multiply original by two
3. Return original

Solution findFinalValue with while loop

```
const findFinalValue = (nums, original) => {

    while (nums.includes(original)) {
        original = original * 2;
    }

    return original;
};
```

Solution findFinalValue with for loop

```
const findFinalValue = (nums, original) => {

    for(let i = 0; i<nums.length; i++){
        if(nums.includes(original)){
           original = original * 2
        }
    }

    return original
};
```

Solution findFinalValue with set

```
var findFinalValueSet = function(nums, original) {

    const set = new Set(nums)
    let result = original

    while (set.has(result))
        result *= 2

    return result
};
```

4.12 Two Out of Three

https://leetcode.com/problems/two-out-of-three/

1. Initialize arr with three split... Set() for nums1,2,3
2. Initialize map object
3. Initialize empty results array
4. For of loop through arr and build map to have count of each value
5. For in loop through map and push any values ¿= 2 into the results array
6. Return results

Solution twoOutOfThree O(n) time/space

```
const twoOutOfThree = (nums1, nums2, nums3) => {

    let arr = [...new Set(nums1), ...new Set(nums2),
   ...new Set(nums3)];
    let map = new Map;
    let result = [];

    for(let n of arr) {
        if(map[n]) {
            map[n] += 1;
        } else {
            map[n] = 1;
        }
    }

    for(let i in map) {
        if(map[i] >= 2) {
            result.push(i);
        }
    }

  return result;
};
```

Solution twoOutOfThree - clunky first answer

```
const twoOutOfThree = (nums1, nums2, nums3) => {

    let result = [];

```

```
    for(let n of nums1){
        for(let n2 of nums2){
            if(n === n2){
                result.push(n);
            }
            for(let n3 of nums3){
                if (n2 === n3){
                    result.push(n2);
                }
                if (n === n3){
                    result.push(n3);
                }
            }
        }
    }

    let finalResult = [...new Set(result)];
    return finalResult;
};
```

4.13 Kth Distinct String in an Array

https://leetcode.com/problems/kth-distinct-string-in-an-array/

1. Initialize empty map
2. Initialize answer array
3. For each through the array to build the map
4. For loop through [key, val] of map Object
5. Return the values === 1 and push their key to the answer
6. Return answer [k-1] since it starts at zero index —— empty string if nothing

Solution kthDistinct

```
const kthDistinct = (arr, k) => {

    let map = new Map;
    let result = [];

    arr.forEach(i => map[i] = map[i] + 1 || 1);

    for (let [key, val] of Object.entries(map))
        if (val === 1) result.push(key);

    return result[k-1] || "";
};
```

4.14 Maximum Number of Words You Can Type

https://leetcode.com/problems/maximum-number-of-words-you-can-type/

1. Initialize empty answer variable
2. Set t = true
3. Create new Set with brokenLetter split into characters
4. Loop through text and set letter to charAt(i)
5. If set has letter set t to false
6. If letter is empty or letter is there increase by one, set t to true again
7. If t is there, increase answer by 1
8. Return answer

Solution canBeTypedWords

```
const canBeTypedWords = (text, brokenLetters) => {

    let result = 0;
    let t = true;
    let set = new Set(brokenLetters.split(''));

    for(let i=0;i<text.length;i++){
        let letter = text.charAt(i);
        if(set.has(letter)) t = false;
        if(letter==' ') {
            if(t) result += 1;
            t = true;
        }
    }

    if(t) result += 1;

    return result;
};
```

4.15 Substrings of Size Three with Distinct Characters

https://leetcode.com/problems/substrings-of-size-three-with-distinct-characters/

1. Initialize answer at zero
2. For loop through s.length - 1
3. Set str to a slice of s that starts at i and goes till i+3
4. Make a new Set(str)
5. If the size of the set is === 3, += 1 to answer
6. Return answer

Solution countGoodSubstrings

```
const countGoodSubstrings = (s) => {

    let result = 0;

    for (let i = 0; i < s.length - 2; i++) {
        let str = s.slice(i, i + 3);
        let set = new Set(str);
        if (set.size === 3) result += 1;
    }

    return result;
};
```

4.16 Find the Difference of Two Arrays

https://leetcode.com/problems/find-the-difference-of-two-arrays/submissions/

1. Set set1 to a new Set of nums1
2. Set set2 to a new Set of nums2
3. ForEach loop through nums2 where it will delete any values that are also in set1
4. ForEach loop through nums1 where it will delete any values that are also in set2
5. Return and array of split arrays set1 and set2

Solution findDifference

```
const findDifference = (nums1, nums2) => {

    let set1 = new Set(nums1);
    let set2 = new Set(nums2);

    nums2.forEach(i => {set1.delete(i)});
    nums1.forEach(i => {set2.delete(i)});

    return ([[...set1], [...set2]]);
};
```

4.17 Isomorphic Strings

https://leetcode.com/problems/isomorphic-strings/

1. Initialize new Map()
2. Loop through s.length
3. If there is no map['s' +s[i]] then add it and set to t[i]
4. If there is no map['t' +t[i]] then add it and set to s[i]
5. If t[i] is not equal to the s map or if s[i] is not equal to the t map return false
6. Return true

Solution isIsomorphic

```
const isIsomorphic = (s, t) => {

    let map = new Map();

    for (let i = 0; i<s.length; i++){
        if(!map['s' + s[i]]) {
            map['s' + s[i]] = t[i];
        }
        if(!map['t' + t[i]]) {
            map['t' + t[i]] = s[i];
        }
        if(t[i] != map['s' + s[i]] || s[i] != map['t' + t[i]]){
            return false;
        }
    }

    return true;
};
```

4.18 Word Pattern

https://leetcode.com/problems/word-pattern/

1. Set words to s.split(' ')
2. Set map to new Map()
3. If words.length is not equal to pattern return false (catches diff length)
4. If words.size is not equal to set(patter).size return false (catches equal word)
5. For loop through pattern length
6. If map pattern has pattern [i] and it is not equal to words[i] return false
7. Map.set pattern[i] and words[i]
8. Return true

Solution wordPattern

```
const wordPattern = (pattern, s) => {

    let words = s.split(' ');
    let map = new Map();

    if(words.length !== pattern.length){
        return false;
    }
    if(new Set(words).size !== new Set(pattern).size){
        return false;
    }

    for(let i = 0; i < pattern.length; i++) {
        if(map.has(pattern[i]) &&
           map.get(pattern[i]) !== words[i]) {
            return false;
        }
        map.set(pattern[i], words[i]);
    }

    return true;
};
```

4.19 Longest Harmonious Subsequence

https://leetcode.com/problems/longest-harmonious-subsequence/

1. Initialize new Map()
2. Set result at zero
3. For of loop through nums map[v] = map[v] + 1 or 1 (build map)
4. Loop through the key of map and parse int key + 1
5. Set result to math max result, map[key] + map[parseInt(key) + 1]
6. Return result

Solution findLHS

```
1  const findLHS = (nums) => {
2
3      let map = new Map();
4      let result = 0;
5
6      for(let v of nums) {
7          map[v] = map[v] + 1 || 1;
8      }
9
10     for (let key in map) {
11         if(map[parseInt(key) + 1]) {
12             result = Math.max(result, map[key] + map[
    parseInt(key) + 1]);
13         }
14     }
15
16     return result;
17 };
```

4.20 Next Greater Element I

https://leetcode.com/problems/next-greater-element-i/

1. Initialize new Map()
2. Initialize empty array called stack
3. Loop through nums2 with forEach and nest a while loop
4. While stack.length 0 && it is within stack push stack n
5. Return nums1.map(n map[n] or -1)

Solution nextGreaterElement

```
const nextGreaterElement = (nums1, nums2) => {

    let map = {};
    let stack = [];

    nums2.forEach(n => {
      while (stack.length > 0 && stack[stack.length -
   1] < n) {
        map[stack.pop()] = n;
      }
      stack.push(n);
    })

    return nums1.map(n => map[n] || -1);
  };
```

5 Dynamic Programming

5.1 Counting Bits

https://leetcode.com/problems/counting-bits/

1. Initialize result array at [0]
2. Initialize offset at 1
3. For loop through n starting at 1, going till n+1
4. If offset * 2 === i then set offset to i
5. result[i] is equal to 1 + result [i - offset]
6. Return result

Solution countBits

```
const countBits = (n) => {

    let result = [0];
    let offset = 1;

    for (let i = 1; i < n + 1; i++) {
      if (offset * 2 === i) {
        offset = i;
      }
      result[i] = 1 + result[i - offset];
    }

    return result;
  };

```

5.2 Fibonacci Number

https://leetcode.com/problems/fibonacci-number/

1. Initialize result to [0,1]
2. For loop starting from 2 till i ¡= n
3. Result push result[i-2] plus result[i-1]
4. Return result[n]

Solution fib

```
const fib = (n) => {

    let result = [0,1];

    for(let i = 2; i<= n; i++){
        result.push(result[i-1] + result[i-2]);
    }

    return result[n];
};
```

5.3 Pascal's Triangle

https://leetcode.com/problems/pascals-triangle/

1. Initialize empty results array
2. For loop through n
3. Set result[i] to empty array
4. Set result[i][0] equal to result[i][i] = 1
5. For loop through i starting at 1
6. Set result[i][j] equal to result[i-1][j] + result[i-1][j-1]
7. Return result

Solution generate

```
const generate = (n) => {

    let result = [];

    for (let i = 0; i<n; i++) {
        result[i] = [];
        result[i][0] = result[i][i] = 1;
        for (let j = 1; j < i; j++) {
            result[i][j] = result[i-1][j] + result[i
   -1][j-1];
        }
    }

    return result;
};
```

Solution generate with less mess

```
const generate = (n) => {

    let r = [];

    for (let i =0;i<n; i++) {
        r[i] = [];
        r[i][0] = r[i][i] = 1;
        for (let j = 1; j<i; j++) {
            r[i][j] = r[i-1][j]+r[i-1][j-1];
        }
    }
```

```
12
13      return r;
14  };
```

5.4 Divisor Game

https://leetcode.com/problems/divisor-game/

1. return n % 2 === 0

Solution divisorGame

```
const divisorGame = (n) => {

    return n % 2 === 0;

};
```

Solution divisorGame

```
const divisorGame = (n) => {

    let result = Array(n + 1).fill(false);

    for (let i = 2; i <= n; ++i){
        for (let j = Math.floor(Math.sqrt(i)); 1 <= j;
    --j){
            if (!(i % j) && !result[i - j]){
                result[i] = true;
            }
        }
    }

    return result[n];
};
```

5.5 N-th Tribonacci Number

https://leetcode.com/problems/n-th-tribonacci-number/

1. return Initialize result array to 0,1,1
2. For loop through n starting from 3
3. Set result to result -1,-2-3 for initial values
4. Return result[n]

Solution tribonacci

```
const tribonacci = (n) => {

    let result = [0,1,1];

    for(let i = 3; i<=n; i++) {
        result[i] = result[i-1] + result[i-2] + result
    [i-3];
     }

    return result[n];
};
```

5.6 Min Cost Climbing Stairs

https://leetcode.com/problems/min-cost-climbing-stairs/

1. Initialize L array at cost[0]
2. Initialize R array at cost[1]
3. Initialize current as empty array
4. if cost.length === 1 return zero
5. If cost.length === 2 return math.min(L, R)
6. Loop through cost starting from 2
7. Set current to cost[i] + math.min(L, R)
8. Set L to R
9. Set R to current
10. Return Math.min(L, R)

Solution minCostClimbingStairs

```
const minCostClimbingStairs = (cost) => {

    let L = cost[0];
    let R = cost[1];
    let current = [];

    if (cost.length === 1) return 0;
    if (cost.length === 2) return Math.min(L, R);

    for (let i = 2; i < cost.length; i++) {
        current = cost[i] + Math.min(L, R);
        L = R;
        R = current;
    }

    return Math.min(L, R);
};
```

5.7 Pascal's Triangle II

https://leetcode.com/problems/pascals-triangle-ii/

1. Initialize result as empty array
2. Loop through row index + 1
3. Set result[i] to empty array
4. Result [i][0] = result [i][i] = 1
5. Loop through i starting from 1
6. Set result [i][j] to result[i-1][j] + result[i-1][j-1]
7. Return result[rowIndex]

Solution getRow

```
const getRow = (rowIndex) => {

    let result = [];

    for(let i = 0; i<rowIndex+1; i++){
        result[i] = [];
        result[i][0] = result[i][i] = 1;
        for(let j = 1; j < i; j++){
            result[i][j] = result[i-1][j] + result[i
   -1][j-1];
        }
    }

    return result[rowIndex];
};
```

5.8 Best Time to Buy and Sell Stock

https://leetcode.com/problems/best-time-to-buy-and-sell-stock/

1. Initialize profits at zero
2. Initialize min at prices[0]
3. Loop through prices starting from index 1
4. If min greater than prices[i] set min=prices[i]
5. Else if prices[i] - min greater than profit set profit = prices[i] - min
6. Return profit

Solution maxProfit

```
const maxProfit = (prices) => {

    let profit = 0;
    let min = prices[0];

    for(let i = 1; i < prices.length; i++) {
        if(min > prices[i]) {
            min = prices[i];
        } else if(prices[i] - min > profit) {
            profit = prices[i] - min;
        }
    }

    return profit;
};
```

5.9 Climbing Stairs

https://leetcode.com/problems/climbing-stairs/

1. Initialize result to [0,1,2,3]
2. For loop through n starting from 4
3. Set result[i] to result[i-1] + result[i-2]
4. Return result[n]

Solution climbStairs

```
const climbStairs = (n) => {

    let result = [0,1,2,3];

    for(let i = 4; i <= n; i++){
        result[i] = result[ i - 1] + result[i - 2];
    }

    return result[n];
};
```

5.10 Is Subsequence

https://leetcode.com/problems/is-subsequence/

1. Initialize results to zero
2. If s.length is greater than t.length return false
3. Loop through t.length and if s[result] === t[i] increment result
4. Return check if result is equal to s.length

Solution isSubsequence

```
const isSubsequence = (s, t) => {

    let result = 0;

    if (s.length > t.length){
        return false;
    }

    for (let i = 0; i < t.length; i++) {
        if (s[result] === t[i]) {
            result++;
        }
    }

    return result === s.length;
};
```

6 Depth First Search Problems

6.1 Maximum Depth of Binary Tree

https://leetcode.com/problems/maximum-depth-of-binary-tree/

1. Check for empty tress with if root is undefined or null and return zero
2. Return math max of recursive call maxDepth for root left and root right plus one

Solution maxDepth

```
const maxDepth = (root) => {

    if (root === undefined || root === null) {
        return 0;
    }

    return Math.max(maxDepth(root.left), maxDepth(root
   .right)) + 1;
};
```

Solution maxDepth bfs

```
const maxDepth = (root) => {

    let result = 0;

    const bfs = (node, level) => {

        if (!node) return;
        if(level > result) result = level;

        bfs(node.left, level +1);
        bfs(node.right, level +1);
    };

    bfs(root, 1);
    return result;
};
```

6.2 Find a Corresponding Node of a Binary Tree in a Clone of That Tree

https://leetcode.com/problems/find-a-corresponding-node-of-a-binary-tree-in-a-clone-of-that-tree/

1. If !original AND !cloned return false
2. If original === target return cloned
3. Return recursive getTargetCopy(original.left, cloned.left, target)
4. Or recursive getTargetCopy(original.right, cloned.right, target)

Solution getTargetCopy

```
1 const getTargetCopy = (original, cloned, target) => {
2
3     if(!original && !cloned) return false;
4     if(original === target) return cloned;
5
6     return getTargetCopy(original.left, cloned.left,
    target) || getTargetCopy(original.right, cloned.
    right, target);
7 };
```

6.3 Evaluate Boolean Binary Tree

https://leetcode.com/problems/evaluate-boolean-binary-tree/

1. Create new function dfs (node)
2. If no node return
3. If node.val is 0 return false
4. If node.val is 1 return true
5. If node.val is 2 result dfs(node.left) —— dfs(node.right)
6. If node.val is 3 result dfs(node.left) AND dfs(node.right)
7. Return dfs[root]

Solution evaluateTree

```
const evaluateTree = (root) => {

    const dfs = (node) => {

        if (!node) return false;
        if (node.val === 0) return false;
        if (node.val === 1) return true;
        if (node.val === 2) {
            return dfs(node.left) || dfs(node.right);
        } else if(node.val === 3){
            return dfs(node.left) && dfs(node.right);
        }
    };

    return dfs(root);
};
```

6.4 Find All The Lonely Nodes

https://leetcode.com/problems/find-all-the-lonely-nodes/

1. Create new function dfs with (root)
2. If no root return results
3. If no right root and yes left root result.push value of left root
4. If no left root and yes right root result.push value of right root
5. Recursive dfs(root.right)
6. Recursive dfs(root.left)
7. Call dfs on root
8. Return result

Solution getLonelyNodes

```
const getLonelyNodes = (root) => {

    let result = [];

    const dfs = (root) => {

        if (!root) return result;
        if (!root.right && root.left) {
            result.push(root.left.val);
        }
        if (!root.left && root.right) {
            result.push(root.right.val);
        }
        dfs(root.right);
        dfs(root.left);
    };

    dfs(root);
    return result;
};
```

6.5 Range Sum of BST

https://leetcode.com/problems/range-sum-of-bst/

1. Initialize result as zero
2. If no root return result
3. If root.val is greater than or equal high AND root.val is greater than or equal to low
4. Result += root val
5. If root.val is greater than low add recursive root.left, low, high
6. If root.val is lower than high add recursive root.right, low, high
7. Return result

Solution rangeSumBST

```
const rangeSumBST = (root, low, high) => {

    let result = 0;

    if(!root) return result;
    if(root.val <= high && root.val >= low){
        result += (root.val);
    }
    if(root.val > low){
        result += rangeSumBST(root.left, low, high);
    }
    if(root.val < high){
        result += rangeSumBST(root.right, low, high);
    }

    return result;
};
```

6.6 Sum of Root To Leaf Binary Numbers

https://leetcode.com/problems/sum-of-root-to-leaf-binary-numbers/

1. Start bfs function with (root, string)
2. If no root return zero
3. String += root.val
4. If no right and left root return parseInt(string, 2)
5. Return dfs(root.right, string) + dfs(root.left, string)
6. Return dfs(root, '')

Solution sumRootToLeaf

```
const sumRootToLeaf = (root) => {

    const dfs = (root, string) => {

        if (!root) return 0;
        string += root.val;
        if (!root.right && !root.left) {
            return parseInt(string, 2);
        }

        return dfs(root.right, string) + dfs(root.left
    , string);
    };

    return dfs(root, ' ');
};
```

6.7 N-ary Tree Preorder Traversal

https://leetcode.com/problems/n-ary-tree-preorder-traversal/

1. Initialize result as [] in function arguments
2. If !root return results
3. Push root.val into results array
4. For of loop through child of root.children
5. Recursive call to preorder(child, result) in loop
6. Return results

Solution preorder recursive

```
const preorder = (root, result = []) => {

    if (!root) return result;
    result.push(root.val);

    for (let child of root.children){
        preorder(child, result);
    }

    return result;
};
```

Solution preorderIterative

```
const preorderIterative = (root) => {

    let stack = [];
    let result = [];

    if (!root) {
        return result;
    }
    stack.push(root);

    while(stack.length) {
        let node = stack.pop();
        result.push(node.val);
        for(let i = node.children.length - 1; i >= 0;
   i--) {
            stack.push(node.children[i]);
```

```
16          }
17      }
18
19      return result;
20  };
```

6.8 N-ary Tree Postorder Traversal

https://leetcode.com/problems/n-ary-tree-postorder-traversal/

1. Initialize result as empty array
2. Recurse(root) and return result
3. Create recurse function with (node)
4. If no node return result
5. Loop through child of node.children
6. Recurse child
7. Push node.val into results

Solution postorder

```
const postorder = (root) => {

    let result = [];

    recurse(root);
    return result;

    function recurse(node) {

        if (!node) return result;
        for (let child of node.children) {
            recurse(child);
        }
        result.push(node.val);
    }
};
```

6.9 Increasing Order Search Tree

https://leetcode.com/problems/increasing-order-search-tree/

1. Initialize null result variable
2. Initialize null newTree variable
3. Make new recurse function with node variable
4. If !node return result
5. If no result result = newTree = node
6. Else newTree.left = null, newTree.right = node, newTree = newTree.right;
7. Call recurse on (node.right)
8. Call recurse on (root)
9. Return result

Solution increasingBST

```
const increasingBST = (root) => {

    let result = null;
    let newTree = null;

    const recurse = (node) => {

        if (!node) return result;
        recurse (node.left);
        if (!result) {
            result = newTree = node;
        } else {
            node.left = null;
            newTree.right = node;
            newTree = newTree.right;
        }
        recurse (node.right);
    };

    recurse (root);
    return result;
};
```

6.10 Merge Two Binary Trees

https://leetcode.com/problems/merge-two-binary-trees/

1. If no root1 return root2, if no root2 return root1
2. Root1.val += root2.val
3. Root1.left = recusive call mergeTrees(root1.left, root2.left)
4. Root1.right = mergeTrees(root1.right, root2.right)
5. Return root1

Solution mergeTrees

```
const mergeTrees = (root1, root2) => {

    if (!root1){
        return root2;
    }
    if(!root2){
        return root1;
    }

    root1.val += root2.val;
    root1.left = mergeTrees(root1.left, root2.left);
    root1.right = mergeTrees(root1.right, root2.right)
   ;

    return root1;
};
```

7 Breadth First Search

7.1 Invert Binary Tree

https://leetcode.com/problems/invert-binary-tree/

1. Initialize queue as [root]
2. While queue.length let n = queue.shift()
3. If n != null [n.left, n.right] = [n.right, n.left]
4. Queue.push(n.left, n.right);
5. Return root

Solution mergeTrees

```
const invertTree = (root) => {

    let queue = [root];

    while (queue.length) {

        let n = queue.shift();

        if(n !== null){
            [n.left, n.right] = [n.right, n.left];
            queue.push(n.left, n.right);
        }
    };

    return root;
};
```

7.2 Maximum Depth of N-ary Tree

https://leetcode.com/problems/maximum-depth-of-n-ary-tree/

1. Initialize result at 0
2. Initialize queue as empty array
3. If !root return result
4. Queue.push(root)
5. While queue.length let n = queue.length
6. For loop where i less than n
7. Let node = queue.shift()
8. Push ...node.children into queue
9. Result++
10. Return result

Solution maxDepth

```
const maxDepth = (root) => {

    let result = 0;
    let queue = [];

    if(!root) return result;
    queue.push(root);

    while(queue.length){

        let n = queue.length;

        for(let i = 0; i<n; i++){
            let node = queue.shift();
            queue.push(...node.children);
        }
        result++;
    }

    return result;
};
```

7.3 Island Perimeter

https://leetcode.com/problems/island-perimeter/submissions/

1. Initialize result as zero. Initialize rows as grid.length. Initialize cols as grid[0].length
2. For loop through rows with row. For loop through cols with col
3. If !grid[row][col] continue
4. Result += 4
5. If row GREATERTHAN 0 AND grid[row-1][col] result –
6. If col GREATERTHAN 0 AND grid[row][col - 1] result –
7. If row LESSTHAN rows - 1 AND grid[row+1][col] result –
8. if col LESSTHAN cols -1 AND grid[row][col+1] result –
9. Return result

Solution islandPerimeter

```
const islandPerimeter = (grid) => {

    let result = 0;
    let rows = grid.length;
    let cols = grid[0].length;

    for(let row = 0; row < rows; row++){
        for(let col = 0; col < cols; col++){

            if(!grid[row][col]) continue;
            result += 4;

            if(row > 0 && grid[row - 1][col]) result
   --;
            if(col > 0 && grid[row][col - 1]) result
   --;
            if(row < rows - 1 && grid[row + 1][col])
   result--;
            if(col < cols - 1 && grid[row][col + 1])
   result--;
        }
    }

    return result
};
```

7.4 Average of Levels in Binary Tree

https://leetcode.com/problems/average-of-levels-in-binary-tree/

1. Initialize results array
2. Initialize node at [root]
3. While node.length
4. Let nodeLength = node.length and row = 0
5. For loop through nodeLength
6. Let n = node.shift(); row+= n.val
7. If n.left node.push(n.left)
8. If n.right node.push(n.right)
9. Result.push row/nodeLength
10. Return result

Solution averageOfLevels

```
const averageOfLevels = (root) => {

    let result = [];
    let node = [root];

    while (node.length) {
        let nodeLength = node.length;
        let row = 0;
        for (let i = 0; i < nodeLength; i++) {
            let n = node.shift();
            row += n.val;
            if (n.left) node.push(n.left);
            if (n.right) node.push(n.right);
        }
        result.push(row/nodeLength);
    }

    return result;
};
```

7.5 Univalued Binary Tree

https://leetcode.com/problems/univalued-binary-tree/

1. Initialize queue as empty array
2. Queue.push(root)
3. While queue.length !== 0
4. Let n = queue.shift()
5. If n.val not equal to root.val return false
6. If n.left queue.push n.left
7. If n.right queue.push n.right
8. Return true

Solution isUnivalTree

```
const isUnivalTree = (root) => {

    let queue = [];
    queue.push(root);

    while(queue.length !== 0) {
        let n = queue.shift();

        if(n.val !== root.val) return false;
        if(n.left) queue.push(n.left);
        if(n.right) queue.push(n.right);
    };

    return true;
};
```

7.6 Flood Fill

https://leetcode.com/problems/flood-fill/

1. Initialize currColor as image[sr][sc]
2. Initialize queue as [[sr,sc]];
3. if currColor === color return image
4. While queue.length let [row,col] = queue.shift()
5. If image[row][col] === currColor update them to color
6. UP If row - greaterthan or equal to 0 queue.push row-1,col
7. DOWN If row + 1 less than image.length queue push row + 1, col
8. RIGHT If col + 1 less than image[0].length queue push row, col +1
9. LEFT if col - 1 greater than or equal to zero queue.push row, col-1
10. Return image

Solution floodFill

```
const floodFill = (image, sr, sc, color) => {

    let currColor = image[sr][sc];
    let queue = [[sr, sc]];

    if(currColor === color) return image;

    while(queue.length !== 0){

        let [row, col] = queue.shift();

        if(image[row][col] === currColor){
            image[row][col] = color;
            if(row - 1 >= 0) queue.push([row-1, col]);
            //up
            if(row + 1 < image.length) queue.push([row
    + 1, col]);
            //down
            if(col + 1 < image[0].length) queue.push([
   row, col+1]);
            //right
            if(col -1 >=0) queue.push([row, col-1]);
            //left
        }
```

```
23      }
24
25      return image;
26  };
```

7.7 Two Sum IV - Input is a BST

https://leetcode.com/problems/two-sum-iv-input-is-a-bst/

1. Initialize set as a new set
2. Initialize queue as [root]
3. While queue.length !== 0
4. let n = queue.shift()
5. If set has(k-n.val) return true
6. Set.add(n.val)
7. If(n.left) queue push n.left
8. If n.right queue.push n.right
9. Return false

Solution findTarget

```
const findTarget = (root, k) => {

    let set = new Set();
    let queue = [root];

    while(queue.length) {
        let n = queue.shift();
        if(set.has(k - n.val)) return true;
        set.add(n.val);
        if(n.left) queue.push(n.left);
        if(n.right) queue.push(n.right);
    }

    return false;
};
```

7.8 Minimum Absolute Difference in BST

https://leetcode.com/problems/minimum-absolute-difference-in-bst/

1. Initialize result as Infinity
2. Initialize prev as -Infinity
3. Build recursive with (node) and return if node is empty
4. Recursive call on node.left
5. Result = Math.min(result, node.val -prev)
6. Prev = node.val
7. Recursive call on node.right
8. Recursive call on (root)
9. Return result

Solution getMinimumDifference

```
const getMinimumDifference = (root) => {

    let result = Infinity;
    let prev = -Infinity;

    const recursive = (node) => {

        if(!node) return result;

        recursive(node.left);
        result = Math.min(result, node.val - prev);
        prev = node.val;
        recursive(node.right);
    };

    recursive(root);
    return result;
};
```

7.9 Minimum Distance Between BST Nodes

https://leetcode.com/problems/minimum-distance-between-bst-nodes/submissions/

Solution minDiffInBST

```
const minDiffInBST = (root) => {
    let prev = null;
    let result = Infinity;

    const recursive = (node) => {
        if (!node) return;
        recursive(node.left);
        if (prev) {
            result = Math.min(result, Math.abs(node.
   val - prev.val));
        }
        prev = node;
        recursive(node.right);
    };

    recursive(root);
    return result;
};
```

Solution minDiffInBST Iterative

```
const minDiffInBST = root => {

    const stack = [];
    let curr = root, prev = null, min = Infinity;

    while (stack.length || curr) {
        if (curr) {
            stack.push(curr);
            curr = curr.left;
        } else {
            curr = stack.pop();
            if (prev) {
                min = Math.min(min, Math.abs(curr.val
   - prev.val))
            }
            prev = curr;
            curr = curr.right;
        }
    }

```

```
20      return min;
21  };
```

7.10 Same Tree

https://leetcode.com/problems/same-tree/

1. If no p and q return true
2. if no p or queue, or p.val is not equal to q.val return false
3. Return recursive isSameTree(p.left and q.left) AND isSameTree(p.right and q.right)

Solution isSameTree

```
1 const isSameTree = (p, q) => {
2
3     if (!p && !q) return true;
4     if (!p || !q || p.val !== q.val) {
5         return false;
6     }
7
8     return isSameTree(p.left, q.left) && isSameTree(p.
    right, q.right);
9 };
```

8 Binary Search Tree

8.1 Search in a Binary Search Tree

https://leetcode.com/problems/search-in-a-binary-search-tree/

1. Initialize result as null
2. Build recursive const with (node) as argument
3. If no node return result
4. If node.val is equal to val, result = node and return result
5. If val lessthan node.val recursive node.left
6. if val greaterthan node.val recursive node.right
7. Recursive through root and return result

Solution searchBST

```
const searchBST = (root, val) => {

    let result = null;

    const recursive = (node) => {

        if(!node) return result;
        if(node.val === val){
            result = node;
            return result;
        }
        if(val < node.val) recursive(node.left);
        if(val > node.val) recursive(node.right);
    };

    recursive(root);
    return result;
};
```

8.2 Convert Sorted Array to Binary Search Tree

https://leetcode.com/problems/convert-sorted-array-to-binary-search-tree/

1. If nums is empty return null
2. Build recursive with low, high as arguments
3. Let mid equal Math.floor low + high divided by 2
4. Let root equal new TreeNode nums[mid]
5. If low is greater than high return null
6. Root.left = recursive(low, mid - 1)
7. Root.right = recursive(mid + 1, high)
8. Return root
9. Return recursive(0, nums.length - 1)

Solution sortedArrayToBST

```
const sortedArrayToBST = (nums) => {

    if (!nums) return null;

    const recursive = (low, high) => {

        let mid = Math.floor((low + high) / 2);
        let root = new TreeNode(nums[mid]);

        if(low > high) return null;

        root.left = recursive(low, mid - 1);
        root.right = recursive(mid + 1, high);

        return root;
    };

    return recursive(0, nums.length - 1);
};
```

8.3 Unique Binary Search Trees

https://leetcode.com/problems/unique-binary-search-trees/

1. Initialize result as new array n + 1 and fill with 0
2. Set result[0] to 1 and result[1] to 1
3. For loop starting from 2 less then or equal to n
4. For loop stating from 1 less then or equal to i
5. Set result[i] to be result[j - 1] times result[i - j]
6. Return result[n]

Solution numTrees

```
1  const numTrees = (n) => {
2
3      let result = new Array(n + 1).fill(0);
4      result[0] = 1;
5      result[1] = 1;
6
7      for (let i = 2; i <= n; i++){
8          for (let j = 1; j <= i; j++){
9              result[i] += result[j - 1] * result[i - j
       ];
10         }
11     }
12
13     return result[n];
14 };
```

8.4 Lowest Common Ancestor of a Binary Search Tree

https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-search-tree/

1. If root.val is greater than p.val AND q.val recursive root.left, p, q
2. If root.val is less than p.val AND q.val recursive root.right, p, q
3. Return root

Solution lowestCommonAncestor

```
const lowestCommonAncestor = (root, p, q) => {

    if(root.val > p.val && root.val > q.val) {
        return lowestCommonAncestor(root.left, p, q);
    }

    if(root.val < p.val && root.val < q.val) {
        return lowestCommonAncestor(root.right, p, q);
    }

    return root;
};
```

8.5 Closest Binary Search Tree Value

https://leetcode.com/problems/closest-binary-search-tree-value/

1. Initialize child as root.left if target less than root.val else as root.right
2. If no child return root.val
3. Set closest as recursive(child, target)
4. Return closest if Math.abs(closest - target) less than Math.abs(root.val - target) else return root.val

Solution closestValue

```
const closestValue = (root, target) => {

    let child = target < root.val ? root.left : root.
   right;
    if (!child) return root.val;
    let closest = closestValue(child, target);

    return Math.abs(closest - target) < Math.abs(root.
   val - target) ? closest : root.val;
};
```

8.6 Binary Search Tree to Greater Sum Tree

https://leetcode.com/problems/binary-search-tree-to-greater-sum-tree/

1. Build getSum function with node and sum
2. If no node return sum
3. Set node.val += getSum(node.right, sum)
4. Return getSum(node.left, node.val)
5. Recursive call on getSum(root, 0)
6. Return root

Solution bstToGst

```
const bstToGst = (root) => {

    const getSum = (node, sum) => {

        if (!node) return sum;
        node.val += getSum(node.right, sum);

        return getSum(node.left, node.val);
    };

    getSum(root, 0);
    return root;
};
```

8.7 Construct Binary Search Tree from Preorder Traversal

https://leetcode.com/problems/construct-binary-search-tree-from-preorder-traversal/

1. Construct recursive function with lower, upper
2. If preorder[0] less than lower or greaterthan upper return null
3. If preorder.length === 0 return null
4. Initialize root as new TreeNode and shift first item of preorder into it
5. Set root.left as recursive(lower, root.val)
6. Set root.right as recursive(root.val, upper)
7. Return root
8. Return recursive(-Infinity, Infinity)

Solution bstFromPreorder

```
const bstFromPreorder = (preorder) => {

    const recursive = (lower, upper) => {

        if (preorder[0] < lower || preorder[0] > upper
   ) return null;
        if (preorder.length === 0) return null;

        let root = new TreeNode(preorder.shift());

        root.left = recursive(lower, root.val);
        root.right = recursive(root.val, upper);

        return root;
    };

    return recursive(-Infinity, Infinity);
};
```

8.8 Balance a Binary Search Tree

https://leetcode.com/problems/balance-a-binary-search-tree/

1. Initialize builtArray as arrayBuilder(root)
2. Return bstBuilder(builtArray)
3. Construct helper arrayBuilder function
4. Construct helper bstBuilder function

Solution balanceBST

```
const balanceBST = (root) => {

    let builtArray = arrayBuilder(root);

    return bstBuilder(builtArray);
};

const arrayBuilder = (node) => {

    if (!node) return [];

    return [...arrayBuilder(node.left), node.val, ...
   arrayBuilder(node.right)];
};

const bstBuilder = (arr) => {

    if (arr.length === 0) return null;
    if (arr.length === 1) return new TreeNode(arr);

    let mid = Math.floor(arr.length / 2);
    let left = bstBuilder(arr.slice(0, mid));
    let right = bstBuilder(arr.slice(mid + 1));

    return new TreeNode(arr[mid], left, right);
};
```

8.9 Kth Smallest Element in a BST

https://leetcode.com/problems/kth-smallest-element-in-a-bst/

1. Initialize stack, counter, and node
2. While true if node stack.push and set node to left
3. Else if stack is empty continue and increase counter
4. If counter equal k return node.val otherwise move to node.right

Solution kthSmallest

```
const kthSmallest = (root, k) => {

    let stack = [];
    let counter = 0;
    let node = root;

    while (true) {
        if (node) {
            stack.push(node);
            node = node.left;
        } else {
            if (stack.length === 0) break;
            node = stack.pop();
            counter += 1;
            if (counter === k) return node.val;
            node = node.right;
        }
    }
};
```

8.10 Delete Node in a BST

https://leetcode.com/problems/delete-node-in-a-bst/

1. Build callDFS function with (node) if no node return null
2. If node.val equal key and no node.left, return right if no node.right return left
3. Initialize curr as node.right
4. While curr.left set curr as curr.left
5. Set curr.left as node.left and return node.right
6. If key is greater than node.val node.right = recursive(node.right) else node.left is recursive(node.left)
7. Return node and return recursive call on root

Solution deleteNode

```
const deleteNode = (root, key) => {

    const callDFS = (node) => {

        if (!node) return null;
        if (node.val === key) {
            if (!node.left) return node.right;
            if (!node.right) return node.left;
            let curr = node.right;
            while (curr.left) {
                curr = curr.left;
            }
            curr.left = node.left;
            return node.right;
        }
        if (key > node.val) {
            node.right = callDFS(node.right);
        } else {
            node.left = callDFS(node.left);
        }

        return node;
    };

    return callDFS(root);
};
```

9 Linked List

9.1 Convert Binary Number in a Linked List to Integer

https://leetcode.com/problems/convert-binary-number-in-a-linked-list-to-integer/

1. Initialize result as zero
2. While loop as long as head is not null
3. Set result to result bitwise shift 1 or head.val
4. Set head to head.next
5. Return result

Solution getDecimalValue

```
const getDecimalValue = (head) => {

    let result = 0;

    while(head !== null) {
        result = (result << 1) | head.val;
        head = head.next;
    }

    return result;
};
```

9.2 Delete N Nodes After M Nodes of a Linked List

https://leetcode.com/problems/delete-n-nodes-after-m-nodes-of-a-linked-list/

1. Initialize current as head.next, prev as head, and count to one
2. While loop as long as current not null, if count is equal to m + n set count to zero
3. If count is less than m, prev = current and current = current.next
4. Else if count is less than m + n, current = current.next, prev.next = current
5. Increment count
6. Return head

Solution deleteNodes

```
const deleteNodes = (head, m, n) => {

    let current = head.next;
    let prev = head;
    let count = 1;

    while (current !== null) {
        if (count === m + n) count = 0;
        if (count < m) {
            prev = current;
            current = current.next;
        } else if (count < m + n) {
            current = current.next;
            prev.next = current;
        }
        count++;
    }

    return head;
};
```

9.3 Reverse Linked List

https://leetcode.com/problems/reverse-linked-list/

1. Initialize result as null
2. While loop as long as head is not null
3. Set next as head.next
4. Set head.next as result
5. Set result as head
6. Set head as next
7. Return result

Solution reverseList

```
const reverseList = (head) => {

    let result = null;

    while (head !== null) {
        let next = head.next;
        head.next = result;
        result = head;
        head = next;
    }

    return result;
};
```

9.4 Delete Node in a Linked List

https://leetcode.com/problems/delete-node-in-a-linked-list/

1. Set node.val to node.next.val
2. Set node.next as node.next.next

Solution deleteNode

```
const deleteNode = (node) => {

    node.val = node.next.val;
    node.next = node.next.next;
};
```