Problem 1

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

Open brackets must be closed by the same type of brackets.

Open brackets must be closed in the correct order.

Note that an empty string is also considered valid.

Example 1:

Input: "()"

Output: true

Example 2:

Input: "()[]{}"

Output: true

Example 3:

Input: "(]"

Output: false

Example 4:

Input: "([)]"

Output: false

Example 5:

Input: "{[]}"

Output: true

Rules to follow:

1. Each close character must be accompanied by a corresponding open character and vice versa
2. No more close characters than open characters at any given index
3. An expression contained within a set of characters ( “()” “[]” “{}”) must be valid

Dictionary: { “[“ -> “]” , “(“ -> “)” , “{“ -> “}” }

Str = [ ( {} [ ( ) ] ) ] ] ]

I = 0, stack []

stack = [ “[“ ]

I = 3, stack = [ “[“, “(“, “{“ ]

stack.pop() = “{“

Dictionary[“{“] = “}”

Str[3] === dictionary[“{“] -> we are good to move on

I = 6, stack = [ “[“, “(“, “[“, “(“ ]

stack.pop() = “(”

[ ( **[ )** ] ]

I = 7, stack = [ “[“, “(“, “[“ ]

stack.pop() = “[”

Dictionary[“[“] = “]”

Str[7] = “(“ -> not equal so return false

I = 8, stack = [ “[“, “(“ ]

stack.pop() = “(”

I = 9, stack = [ “[“ ]

stack.pop() = “[”

Pseudocode:

1. Going to create an object to keep track of open and close characters { “[“ -> “]” ....}
2. Create a stack to keep track of open characters
3. For loop through the entire string
   1. If the character is an open character (a key of the object ^)
      1. Add to the stack
   2. If the character is a close character (not a key of the object ^)
      1. Pop the stack and if the close character that corresponds the the popped value in the object ^ is not the current value, return false
4. Return if stack is empty

function strValid (str) {

let dict = {

“(“ : “)”,

“[“ : “]”,

“{“ : “}”

};

let stack = [];

for (let i = 0; i < str.length; i ++ ){

if(dict[str[i]] != undefined){

stack.push(str[i]);

} else {

let lastOpen = stack.pop();

if(str[i] == dict[lastOpen]){

return false;

}

}

}

if(stack.length == 0){

return true;

} else {

return false;

}

}

Problem 2

You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have n versions [1, 2, ..., n] and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API bool isBadVersion(version) which will return whether version is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

Example:

Given n = 5, and version = 4 is the first bad version.

call isBadVersion(3) -> false

call isBadVersion(5) -> true

call isBadVersion(4) -> true

Then 4 is the first bad version.