  
 **TEKNİK BİLİMLER MESLEK YÜKSEK OKULU  
BİLGİSAYAR PROGRAMCILIĞI 1.SINIF  
WEB EDİTÖRÜ**  
  
  
**PROJE ADI  
Web Editörü Final Sınavı**

**BÜLENT VURAL  
20191132054  
  
  
  
**

**BÖLÜM 1  
  
Versiyon Kontrol Sistemi Nedir ?**

Versiyon Kontrol Sistemi (VCS), revizyon kontrol (revision control) veya kaynak kontrol (source control) diye de geçip, değişiklik yönetim sistemi anlamına gelmektedir. Bir ya da daha fazla dosya üzerinde yapılan değişiklikleri kaydeden ve daha sonra belirli bir sürüme geri dönebilmenizi sağlayan bir sistemdir.

“Sürüm kontrolü” terimi genellikle programcılarla ilişkilendirilirken, yazarlar, gazeteciler ve hatta üniversite öğrencileri için de aynı derecede önemlidir. Belge revizyonlarını ve sürümlerini otomatik olarak takip eden yaygın hizmetlere örnekler arasında Google Docs ve DropBox bulunur.

**Amaçları Nelerdir ?**

Geliştiricilerin, kod değişikliklerini takip etmelerini sağlamak.

Geliştiricilerin, kod değişiklik geçmişini görmelerini sağlamak.

Geliştiricilerin, aynı kod dosyalarında aynı anda çalışmasına izin vermek.

Geliştiricilerin, kodlarını dallanma yoluyla ayırmalarına izin vermek.

Farklı dallardan (branch) kodu birleştirmek.

Geliştiricilerin, çakışmalarını gösterir ve bunları çözmelerine izin vermek.

Geliştiricilerin, değişikliklerini önceki bir duruma döndürmelerine izin vermek.

**Ne için Kullanırız ?**

Bir dosyanın değişik sürümlerini korumak istiyorsanız, Sürüm Kontrol Sistemi (VCS) kullanmanız çok akıllıca olacaktır. VCS, dosyaların ya da bütün projenin geçmişteki belirli bir sürümüne erişmenizi, zaman içinde yapılan değişiklikleri karşılaştırmanızı, soruna neden olan şeyde en son kimin değişiklik yaptığını, belirli bir hatayı kimin, ne zaman sisteme dahil ettiğini ve daha başka pek çok şeyi görebilmenizi sağlar. Öte yandan, VCS kullanmak, bir hata yaptığınızda ya da bazı dosyaları yanlışlıkla sildiğinizde durumu kolayca telâfi etmenize yardımcı olur. Üstelik, bütün bunlar size kayda değer bir ek yük de getirmez.  
  
 **Visual Studio ile GitHub Kullanımı**

Ön Hazırlık

https://gitextensions.github.io/ sayfasına gidilir.

Download Lates Version başlığından son sürüme gidilir.

Sayfanın en altından GitExtensions-{versiyon\_numarası}-SetupComplete.msi dosyası indirilir.

**Kurulum**

Kurulumu tüm kullanıcılar için yapılır. (All User seçili olacak.)

Komut satırı olarak Windows Command Prompt seçilir.

Git Windows 32bit ve KDiff3 Windows 32bit seçenekleri işaretlenerek ikisini de kurulur.

Checkout Windows Style olacak. (Projede Mac ya da Linux kullanıcısı varsa as-is kalacak.)

Sonraki adımda yine Windows seçerek devam edilir.

KDiff3 kurulumuna gelince full kurulur.

**Clone Oluşturma**

C sürücüsünün içerisine Projects adında bir klasör oluşturulur.

GitHub'a login olunur.

Clone oluşturulacak proje bulunur ve Fork oluşturulur.

Fork tamamlanınca Clone or Download seçeneğine tıklanır ve .git uzantlılı magnet linki kopyalanır.

Projects klasörüne gidip, boşlukta sağ tuşa basarak context menüden GitClone seçeneği seçilir.

Destination kısmından Projects klasörü seçilir.

Repository Clone kısmına aldığını magnet linki yapıştırılır.

Clone butonuna basıp tamamlanması beklenir.

**İlk Sefer Ayarları**

SQL Server üzerinden Authentication'ı Windows yerine SQL olarak ayarlanıp veritabanına yeniden bağlanılır.

Web.config dosyanızdan Data Source, initial catalog, user id ve password alanları güncellenir.

Build alınır.

**Commit & Push**Değişiklikler yapıldıktan sonra,

Visual Studio üst menüsünden GITEXT tıklanır.

Açılan context menüden Commit seçeneğini seçilir.

Gelen pencereden değişiklik yapılan dosyalar ve DIFF seçeneğinden kodlar görülebilir.

Push edilmek istenmeyen dosyanın üzerine sağ tuşa basılır ve Assume Unchanged seçeneğini işaretlenir.

Değişiklikler tamamlanınca yorum ekleyip Commit & Push butonuna basılır.

İşlemin tamamlanması beklenir.

**Remote & Pull**

Aynı projedeki kullanıcıları teker teker remote eklemek gerekir.

Visual Studio üst menüsünden GITEXT tıklanır.

Açılan context menüden Manage Remote seçeneğini seçilir.

New seçeneğine basılır ve uzak kullanıcıya bir isim verilir.

Projedeki magnet linki yapıştırılır.

Remote kaydedilir.

İstenilen kullanıcıların repository'leri eklendikten sonra Pull işlemi yapılabilir.

Visual Studio üst menüsünden GITEXT tıklanır.

Açılan context menüden Manage Remote seçeneği seçilir.

Remote seçerek pull edilecek kullanıcının repository'si işaretlenir.

Merge Options kısmından Merge Remote Branch to Current Branch'ı işaretlenerek yapılan güncellemelerin aktif olması sağlanır.

Pull butonuna basılır ve güncellemeler beklenir.

**PROJECT EULER**

**4. Largest palindrome product**

A palindromic number reads the same both ways. The largest palindrome made from the product of two 2-digit numbers is 9009 = 91 × 99.

Find the largest palindrome made from the product of two 3-digit numbers.  
  
**Solution:**  
var palindromes = [];

function reverseString(str) {

for (var i = str.length - 1, o = ''; i >= 0; o += str[i--]) { }

return o;

}

for(i=999; i>99; i--) {

for(j=999; j>99; j--) {

possiblePalindrome = i \* j;

possiblePalindromeStr = possiblePalindrome.toString();

if (possiblePalindromeStr == reverseString(possiblePalindromeStr)) {

palindromes.push(possiblePalindrome)

}

}

}

largestPalindrome = Math.max.apply(Math, palindromes);

console.log(largestPalindrome);  
  
**Solution: 906609**

**14. Longest Collatz sequence**

The following iterative sequence is defined for the set of positive integers:

n → n/2 (n is even)

n → 3n + 1 (n is odd)

Using the rule above and starting with 13, we generate the following sequence:

13 → 40 → 20 → 10 → 5 → 16 → 8 → 4 → 2 → 1

It can be seen that this sequence (starting at 13 and finishing at 1) contains 10 terms. Although it has not been proved yet (Collatz Problem), it is thought that all starting numbers finish at 1.

Which starting number, under one million, produces the longest chain?

NOTE: Once the chain starts the terms are allowed to go above one million.

**Solutions:**

// https://projecteuler.net/problem=14

// The following iterative sequence is defined for the set of positive integers:

// n → n/2 (n is even)

// n → 3n + 1 (n is odd)

// Although it has not been proved yet (Collatz Problem), it is thought that all starting numbers finish at 1.

// Which starting number, under one million, produces the longest chain?

// NOTE: Once the chain starts the terms are allowed to go above one million.

function collatz\_num(num) {

if(num % 2 == 0) {

return num / 2;

}

else {

return (num \* 3) + 1;

}

}

function collatz\_sequence(num) {

var counter = 1;

while(num != 1) {

num = collatz\_num(num);

counter++;

}

return counter;

}

function longest\_chain(num) {

var max = 0;

var max\_index;

for(var i=1; i<num; i++) {

var sequence = collatz\_sequence(i);

if(sequence > max) {

max = sequence;

max\_index = i;

}

}

return max\_index;

}

console.log(longest\_chain(1000000));

**Solution: 837799**

**24. Lexicographic permutations**

A permutation is an ordered arrangement of objects. For example, 3124 is one possible permutation of the digits 1, 2, 3 and 4. If all of the permutations are listed numerically or alphabetically, we call it lexicographic order. The lexicographic permutations of 0, 1 and 2 are:

012 021 102 120 201 210

What is the millionth lexicographic permutation of the digits 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9?

**Solutions:**  
function permutate(n, array) {

const al = array.length

for (let i = 0; i < n - 1; i++) {

let k, l

for (let j = 0; j < al - 1; j++) {

if (array[j] < array[j + 1]) {

k = j

}

}

for (let j = k; j < al; j++) {

if (array[k] < array[j]) {

l = j

}

}

let tmp = array[k]

array[k] = array[l]

array[l] = tmp

let begin = k + 1

let end = al - 1

while (begin < end) {

tmp = array[begin]

array[begin] = array[end]

array[end] = tmp

begin += 1

end -= 1

}

}

return array

}

console.log(permutate(1000000, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]).join(""))

**Solution: 2783915460**

**54. Poker hands**In the card game poker, a hand consists of five cards and are ranked, from lowest to highest, in the following way:

* **High Card**: Highest value card.
* **One Pair**: Two cards of the same value.
* **Two Pairs**: Two different pairs.
* **Three of a Kind**: Three cards of the same value.
* **Straight**: All cards are consecutive values.
* **Flush**: All cards of the same suit.
* **Full House**: Three of a kind and a pair.
* **Four of a Kind**: Four cards of the same value.
* **Straight Flush**: All cards are consecutive values of same suit.
* **Royal Flush**: Ten, Jack, Queen, King, Ace, in same suit.

The cards are valued in the order:  
2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King, Ace.

If two players have the same ranked hands then the rank made up of the highest value wins; for example, a pair of eights beats a pair of fives (see example 1 below). But if two ranks tie, for example, both players have a pair of queens, then highest cards in each hand are compared (see example 4 below); if the highest cards tie then the next highest cards are compared, and so on.

Consider the following five hands dealt to two players:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Hand** |  | **Player 1** |  | **Player 2** |  | **Winner** |
| **1** |  | 5H 5C 6S 7S KD  Pair of Fives |  | 2C 3S 8S 8D TD  Pair of Eights |  | Player 2 |
| **2** |  | 5D 8C 9S JS AC  Highest card Ace |  | 2C 5C 7D 8S QH  Highest card Queen |  | Player 1 |
| **3** |  | 2D 9C AS AH AC  Three Aces |  | 3D 6D 7D TD QD  Flush with Diamonds |  | Player 2 |
| **4** |  | 4D 6S 9H QH QC  Pair of Queens Highest card Nine |  | 3D 6D 7H QD QS  Pair of Queens Highest card Seven |  | Player 1 |
| **5** |  | 2H 2D 4C 4D 4S  Full House With Three Fours |  | 3C 3D 3S 9S 9D  Full House with Three Threes |  | Player 1 |

The file, [poker.txt](https://projecteuler.net/project/resources/p054_poker.txt), contains one-thousand random hands dealt to two players. Each line of the file contains ten cards (separated by a single space): the first five are Player 1's cards and the last five are Player 2's cards. You can assume that all hands are valid (no invalid characters or repeated cards), each player's hand is in no specific order, and in each hand there is a clear winner.

How many hands does Player 1 win?

**Solutions:**var hands = require( '../resources/0054-poker.js' );

/\* 1 High Card : Highest value card.

\* 2 One Pair : Two cards of the same value.

\* 3 Two Pairs : Two different pairs.

\* 4 Three of a Kind : Three cards of the same value.

\* 5 Straight : All cards are consecutive values.

\* 6 Flush : All cards of the same suit.

\* 7 Full House : Three of a kind and a pair.

\* 8 Four of a Kind : Four cards of the same value.

\* 9 Straight Flush : All cards are consecutive values of same suit.

\* 10 Royal Flush : Ten, Jack, Queen, King, Ace, in same suit.

\*/

function handValue( cards ){

var nums = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

, suits = [0,0,0,0]

;

for( var i=0, l=cards.length; i<l; i++ ){

nums[ {T:10, J:11, Q:12, K:13, A:14}[ cards[i][0] ] || +cards[i][0] ]++;

suits[ {C:0, S:1, D:2, H:3}[ cards[i][1] ] ]++;

}

var numsStr = nums.join( '' )

, suitsStr = suits.join( '' )

;

return (

/1{5}$/ .test( numsStr ) ? [ 10 ]

: (

/4/ .test( suitsStr ) &&

/1{5}/ .test( numsStr )

) ? [ 9, numsStr.lastIndexOf( 1 ) ]

: /4/ .test( numsStr ) ? [ 8, numsStr.lastIndexOf( 4 ), numsStr.lastIndexOf( 1 ) ]

: /(?=.\*3)(?=.\*2)/.test( numsStr ) ? [ 7, numsStr.lastIndexOf( 3 ), numsStr.lastIndexOf( 2 ) ]

: /5/ .test( suitsStr ) ? [ 6, numsStr.split( '' ).reverse().join( '' ) ]

: /1{5}/ .test( numsStr ) ? [ 5, numsStr.lastIndexOf( 1 ) ]

: /3/ .test( numsStr ) ? [ 4, numsStr.lastIndexOf( 3 ), numsStr.split( '' ).reverse().join( '' ) ]

: /2.\*2/ .test( numsStr ) ? [ 3, numsStr.lastIndexOf( 2 ), numsStr.indexOf( 2 ), numsStr.indexOf( 1 ) ]

: /2/ .test( numsStr ) ? [ 2, numsStr.indexOf( 2 ), numsStr.split( '' ).reverse().join( '' ) ]

: numsStr.split( '' ).reverse().join( '' )

);

}

module.exports = function(){

var wins = 0;

outer:

for( var i=0, l=hands.length; i<l; i++ ){

var p1Value = handValue( hands[i].slice( 0, 5 ) )

, p2Value = handValue( hands[i].slice( 5 ) )

;

for( var j=0, m=p1Value.length; j<m; j++ ){

if( p1Value[j] > p2Value[j] ){

wins++;

}

if( p1Value[j] !== p2Value[j] ){

continue outer;

}

}

console.log( 'Error, should not get here', hands[i] );

}

return wins;

};

|  |
| --- |
|  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Solution: 376**

**44. Pentagon numbers**Pentagonal numbers are generated by the formula, Pn=n(3n−1)/2. The first ten pentagonal numbers are:

1, 5, 12, 22, 35, 51, 70, 92, 117, 145, ...

It can be seen that P4 + P7 = 22 + 70 = 92 = P8. However, their difference, 70 − 22 = 48, is not pentagonal.

Find the pair of pentagonal numbers, Pj and Pk, for which their sum and difference are pentagonal and D = |Pk − Pj| is minimised; what is the value of D?

**Solutions:**

module.exports = function(){

var pentagonals = []

, pHash = {}

, limit = 10000

, dMin = Infinity

, num

;

// generate the first 100 numbers

for( var i=0; i<limit; i++ ){

num = i \* ( 3\*i - 1 ) / 2;

pentagonals.push( num );

pHash[ num ] = 1;

}

for( i=1; i<limit; i++ ){

for( j=i+1; j<limit; j++ ){

if( pHash[ pentagonals[ i ] + pentagonals[ j ] ] && pHash[ pentagonals[ j ] - pentagonals[ i ] ] ){

dMin = Math.min( dMin, pentagonals[ j ] - pentagonals[ i ] );

}

}

}

return dMin;

}

**Solution: 5482660**

**LEETCODE**   
**114. Flatten Binary Tree to Linked List**  
Given a binary tree, flatten it to a linked list in-place.  
For example, given the following tree:

1

/ \

2 5

/ \ \

3 4 6

The flattened tree should look like:

1

\

2

\

3

\

4

\

5

\

6

**Code:**

|  |
| --- |
| /\*\* |
|  | \* Definition for a binary tree node. |
|  | \* function TreeNode(val) { |
|  | \* this.val = val; |
|  | \* this.left = this.right = null; |
|  | \* } |
|  | \*/ |
|  | /\*\* |
|  | \* @param {TreeNode} root |
|  | \* @return {void} Do not return anything, modify root in-place instead. |
|  | \*/ |
|  | var flatten = function(root) { |
|  | var stack = []; |
|  | var p = root; |
|  |  |
|  | while (p !== null || stack.length !== 0) { |
|  | if (p.right !== null) { |
|  | stack.push(p.right); |
|  | } |
|  |  |
|  | if (p.left !== null) { |
|  | // [!!!]point of confusing, if null then pop stack |
|  | p.right = p.left; |
|  | p.left = null; |
|  | } else if (stack.length !== 0) { |
|  | var node = stack.pop(); |
|  | p.right = node; |
|  | } |
|  |  |
|  | p = p.right; |
|  | } |
|  | }; |
|  |  |
|  | // Recursive solution |
|  |  |
|  | var flatten = function(root) { |
|  | if (root === null || (root.left === null && root.right === null)) { |
|  | return; |
|  | } |
|  |  |
|  | var rootLeft = root.left; |
|  | var rootRight = root.right; |
|  | root.left = null; |
|  | root.right = null; |
|  |  |
|  | flatten(rootLeft); |
|  | flatten(rootRight); |
|  |  |
|  | root.right = rootLeft; |
|  |  |
|  | var aux = root; |
|  | while (aux !== null && aux.right !== null) { |
|  | aux = aux.right; |
|  | } |
|  | aux.right = rootRight; |
|  | }; |

Accepted Runtime: 48 ms

Your input [1,2,5,3,4,null,6]

Output [1,null,2,null,3,null,4,null,5,null,6]

Expected [1,null,2,null,3,null,4,null,5,null,6]

**204. Count Primes**Count the number of prime numbers less than a non-negative number, *n*.

**Example:  
Input:** 10  
**Output:** 4  
**Explanation:** There are 4 prime numbers less than 10, they are 2, 3, 5, 7.

**Code:**

|  |
| --- |
| // Description: |
|  |  |
|  | // Count the number of prime numbers less than a non-negative number, n. |
|  |  |
|  | /\*\* |
|  | \* @param {number} n |
|  | \* @return {number} |
|  | \*/ |
|  | // Ever so slightly better...156ms faster than ~76.20% and 128.5MB less than ~13.63% |
|  | var countPrimes = function(n) { |
|  | if (n <= 2) return 0; |
|  |  |
|  | let memo = []; |
|  |  |
|  | for (var i = 2; i < n; i++) { |
|  | memo[i] = true; |
|  | } |
|  |  |
|  | let sqRt = Math.sqrt(n - 1); |
|  |  |
|  | for (i = 2; i <= sqRt; i++) { |
|  | if (memo[i]) { |
|  | for (var j = 2 \* i; j < memo.length; j += i) { |
|  | memo[j] = false; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | let count = 0; |
|  |  |
|  | for (i = 2; i < memo.length; i++) { |
|  | if (memo[i]) count += 1; |
|  | } |
|  |  |
|  | return count; |
|  | }; |
|  |  |
|  | /\*\* |
|  | \* @param {number} n |
|  | \* @return {number} |
|  | \*/ |
|  | // Brute force 164ms faster than ~71.03% and 130.1MB less than ~10.60%...too much memory usage. |
|  | var countPrimes = function(n) { |
|  | let seen = []; |
|  | let count = 0; |
|  |  |
|  | for (let i = 2; i < n; i++) { |
|  | if (seen[i] === undefined) { |
|  | count += 1; |
|  | for (let j = 1; j \* i < n; j++) seen[i \* j] = 1; |
|  | } |
|  | } |
|  |  |
|  | return count; |
|  | }; |

Accepted Runtime: 80 ms  
Your input 10  
Output 4  
Expected 4  
  
 **94. Binary Tree Inorder Traversal**  
Given a binary tree, return the inorder traversal of its nodes' values.

**Example:  
Input:** [1,null,2,3]

1

\

2

/

3

**Output:** [1,3,2]

**Follow up:** Recursive solution is trivial, could you do it iteratively? **Code:**

|  |
| --- |
| /\*\* |
|  | \* Definition for a binary tree node. |
|  | \* function TreeNode(val) { |
|  | \* this.val = val; |
|  | \* this.left = this.right = null; |
|  | \* } |
|  | \*/ |
|  | /\*\* |
|  | \* @param {TreeNode} root |
|  | \* @return {number[]} |
|  | \*/ |
|  | // Recursive approach 56ms faster than 100% and 33.9MB less than ~10.33% |
|  | var inorderTraversal = function(root) { |
|  | let result = []; |
|  |  |
|  | const traverse = node => { |
|  | if (node === null) return node; |
|  | if (node.left) traverse(node.left); |
|  | result.push(node.val); |
|  | if (node.right) traverse(node.right); |
|  | }; |
|  |  |
|  | traverse(root); |
|  | return result; |
|  | }; |
|  | // Iterative approach in 56ms faster than 100% and 33.8MB less than ~32.06% |
|  | var inorderTraversal = function(root) { |
|  | let result = []; |
|  | let stack = []; |
|  | let current = root; |
|  |  |
|  | while (current != null) { |
|  | stack.push(current); |
|  | current = current.left; |
|  | } |
|  |  |
|  | while (stack.length > 0) { |
|  | let node = stack.pop(); |
|  | result.push(node.val); |
|  |  |
|  | node = node.right; |
|  | while (node != null) { |
|  | stack.push(node); |
|  | node = node.left; |
|  | } |
|  | } |
|  |  |
|  | return result; |
|  | }; |

Accepted Runtime: 52 ms  
Your input [1,null,2,3]  
Output [1,3,2]  
Expected [1,3,2]

**104. Maximum Depth of Binary Tree**Given a binary tree, find its maximum depth.

The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

**Note:** A leaf is a node with no children.

**Example:**

Given binary tree [3,9,20,null,null,15,7],

3

/ \

9 20

/ \

15 7

return its depth = 3.

**Code:**

|  |
| --- |
| // Given a binary tree, find its maximum depth. |
|  |  |
|  | // The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node. |
|  |  |
|  | // Note: A leaf is a node with no children. |
|  |  |
|  | // Given binary tree [3,9,20,null,null,15,7], |
|  | // 3 |
|  | // / \ |
|  | // 9 20 |
|  | // / \ |
|  | // 15 7 |
|  |  |
|  | /\*\* |
|  | \* Definition for a binary tree node. |
|  | \* function TreeNode(val) { |
|  | \* this.val = val; |
|  | \* this.left = this.right = null; |
|  | \* } |
|  | \*/ |
|  | /\*\* |
|  | \* @param {TreeNode} root |
|  | \* @return {number} |
|  | \*/ |
|  | // 68ms faster than ~71.35% and 36.9mb less than ~49.53% |
|  | var maxDepth = function(root) { |
|  | if (root === null) return 0; |
|  | else { |
|  | let leftDepth = maxDepth(root.left); |
|  | let rightDepth = maxDepth(root.right); |
|  | return Math.max(leftDepth, rightDepth) + 1; |
|  | } |
|  | }; |

Accepted Runtime: 60 ms

Your input [3,9,20,null,null,15,7]

Output 3

Expected 3

**144. Binary Tree Preorder Traversal**Given a binary tree, return the preorder traversal of its nodes' values.

**Example:**

**Input:** [1,null,2,3]

1

\

2

/

3

**Output:** [1,2,3]

**Follow up:** Recursive solution is trivial, could you do it iteratively?

**Code:**

|  |
| --- |
| /\*\* |
|  | \* Definition for a binary tree node. |
|  | \* function TreeNode(val) { |
|  | \* this.val = val; |
|  | \* this.left = this.right = null; |
|  | \* } |
|  | \*/ |
|  | /\*\* |
|  | \* @param {TreeNode} root |
|  | \* @return {number[]} |
|  | \*/ |
|  | // Recursive approach 60ms faster than ~54.97% and 33.9MB less than ~14.96% |
|  | var preorderTraversal = function(root) { |
|  | let result = []; |
|  |  |
|  | const traverse = node => { |
|  | if (node === null) return node; |
|  | result.push(node.val); |
|  | if (node.left != null) traverse(node.left); |
|  | if (node.right != null) traverse(node.right); |
|  | }; |
|  |  |
|  | traverse(root); |
|  | return result; |
|  | }; |
|  |  |
|  | // Iterative approach 56ms faster than 100% and 33.9MB less than ~14.96% |
|  | var preorderTraversal = function(root) { |
|  | let result = []; |
|  | let stack = []; |
|  | stack.push(root); |
|  |  |
|  | while (stack.length > 0) { |
|  | let node = stack.pop(); |
|  | if (node === null) continue; |
|  | result.push(node.val); |
|  | if (node.right) stack.push(node.right); |
|  | if (node.left) stack.push(node.left); |
|  | } |
|  |  |
|  | return result; |
|  | }; |

Accepted Runtime: 72 ms

Your input [1,null,2,3]

Output [1,2,3]

Expected [1,2,3]