How to reverse an array in javascript

Posted on April 23, 2019 | by Prashant Yadav

Posted in Algorithms, Arrays, Stack | Tagged Easy

An algorithm to reverse an array in javascript.

We will implement a simple algorithm to reverse an array in javascript, Everything will be written in ES6.

Example

```
Input:
[1, 2, 3, 4, 5]

Output:
[5, 4, 3, 2, 1]
```

Reversing array Inplace

We will reverse the array in place without using any extra space.

Using `Array.prototype.reverse()` method

```
let arr = [1, 2, 3, 4, 5];
arr.reverse();
console.log(arr);
//[5, 4, 3, 2, 1]
```

Time complexity: O(n).
Space complexity: O(1).

Using Brute force approach

Implementation

- We will loop through the half of the elements of the given array.

Practically prepare for your JavaScript interview

JavaScript Revision
JavaScript-Concept Based Problems
Data Structures
Algorithms
Machine Coding
Web Fundamentals

- And then we will swap the element at the first place with last place, second place with second last place and so on.
- We will use array destructuring for swapping the elements.

```
Copy
let reverseArray = (arr) => {
    for(let i = 0; i < arr.length / 2; i++){
       //Swap the elements
       [arr[i], arr[arr.length - i - 1]] = [arr[arr.length - i - 1], arr[i]];
    }

  return arr;
}
```

```
Copy
Input:
console.log(reverseArray([1, 2, 3, 4, 5]));

Output:
[5, 4, 3, 2, 1]
```

Time complexity: O(n).
Space complexity: O(1).

Time and Space complexity

- We are looping through the half of the array, so Time complexity is O(n).
- We are using constant space, so Space complexity is O(1).

Reversing array with extra variable

Using an extra array to reverse an array

Alternatively we can use an extra variable to reverse the array in javascript.

Implementation

- We will copy all the elements of the given array to the temporary array in reverse order and then return the temporary array.

Copy

```
let reverseArray = (arr) => {
    //Temp array
    let temp = [];

    for(let i = 0; i < arr.length; i++){
       //Copy all the values in reverse order
       temp[i] = arr[arr.length - i - 1];
    }

  return temp;
}
```

Copy

```
Input:
console.log(reverseArray([1, 2, 3, 4, 5]));

Output:
[5, 4, 3, 2, 1];
```

Time complexity: O(n).

Space complexity: O(n).

Using stack to reverse an array

Implementation

- We will add all the elements of the given array into the Stack.
- Then we will copy the elements from the Stack to the array. As Stack uses LIFO order (Last In First Out) elements will be copied in reverse direction.

Copy

```
let reverseArray = (arr) => {
    //Temp stack
    let stack = new Stack();

    for(let i = 0; i < arr.length; i++){
       //Copy all the values in stack
       stack.push(arr[i]);
    }

    for(let i = 0; i < arr.length; i++){
      //Copy elements back to the array
      arr[i] = stack.pop();
    }

  return arr;
}
```

Copy

```
Input:
console.log(reverseArray([1, 2, 3, 4, 5]));

Output:
[5, 4, 3, 2, 1];
```

Time complexity: O(n).
Space complexity: O(1).

Time and Space complexity

- We are first copying all the items of the array in stack which will take O(n) and then copying back all items to array from stack in O(n), so Time complexity is O(n) + O(n) = O(n).
- We are using stack to store the elements of the array, so Space complexity is O(n).

Reverse an array using recursion

Simple recursive function

Implementation

- We will create a function which will take the given array and its length as a input.
- If the length is empty then return empty array `[]`.
- Else we will call the same function recursively to return the last element of array concatenated with second last element and so on.

```
let reverseArray = (arr, n) => {
  //If the length is 0
  //then return an empty array
  if(n == 0){
     return [];
  }

  //Call the function recursively with one index less and so on.
  return [arr[n-1]].concat(reverseArray(arr, --n));
}
```

```
Input:
console.log(reverseArray([1, 2, 3, 4, 5], 5));

Output:
[5, 4, 3, 2, 1]
```

Time complexity: O(n).
Space complexity: O(n).

Time and Space complexity

- We are calling the same function recursively till the length of the array is not 0, so Time complexity is O(n).
- Recursive function will be stored in call stack, so Space complexity is O(n).

Using `Array.pop()`

Implementation

- We will use the same technique used in above example, but instead of using index to monitor the length we will remove the element itself.

Copy

```
let reverseArray = (arr) => {
  //If the length is 0
  //then return an empty array
  if(arr.length === 0){
    return [];
  }

  //Call the function recursively with one element less and so on.
  return [arr.pop()].concat(reverseArray(arr));
}
```

Copy

```
Input:
console.log(reverseArray([1, 2, 3, 4, 5]));

Output:
[5, 4, 3, 2, 1]
```

Time complexity: O(n).

Space complexity: O(n).

Recommended Posts:

Add two numbers represented by linked lists

Convert Fahrenheit to Celsius in JavaScript

javascript program to find largest of 2 numbers

Find least frequent number from an array

Diagonal traversal of binary tree

Learn how to reverse a linked list recursively

Rotate matrix 90 degrees clockwise and anti-clockwise

Find Least Common Ancestor (LCA) of binary tree

Program to check balanced parentheses

Print the last k nodes of the linked list in reverse.

Prev Next

Comments

Ashutosh Biswas says:

April 28, 2020 At 8:00 Pm

Thanks sharing! Useful info. In the brute force approach, for odd lengths the middle element is also taken into consideration. It could be skipped.

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Start typing...

Name*

Name

Email*

Email

POST COMMENT