

Leetcode

Leetcode

-
- HardHard
-
-
- Offer

```
function quickSort(arr) {  
  if (arr.length <= 1) { // 1  
    return arr;  
  }  
  const pivotIndex = Math.floor(arr.length / 2); //  
  const pivot = arr.splice(pivotIndex, 1)[0]; //  
  const left = [], right = [];  
  for (let i = 0; i < arr.length; i++) {  
    if (arr[i] < pivot) { //  
      left.push(arr[i]);  
    } else { //  
      right.push(arr[i]);  
    }  
  }  
  return quickSort(left).concat([pivot], quickSort(right)); //  
}  
  
//  
const arr = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5];  
console.log(quickSort(arr)); // [1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]
```

```
function binarySearch(arr, target) {  
  let left = 0, right = arr.length - 1;  
  while (left <= right) { //  
    const mid = Math.floor((left + right) / 2); //  
    if (arr[mid] === target) { //  
      return mid;  
    } else if (arr[mid] < target) { //  
      left = mid + 1;  
    } else { //  
      right = mid - 1;  
    }  
  }  
  return -1; // -1  
}  
  
//  
const arr = [1, 2, 3, 4, 5, 6, 7, 8, 9];  
console.log(binarySearch(arr, 5)); // 4  
console.log(binarySearch(arr, 10)); // -1
```

LRU

LRUVuekeep-aliveLRU

```

class LRUCache {
  constructor(capacity) {
    this.capacity = capacity; //
    this.map = new Map(); // Map0(1)
  }

  get(key) {
    const value = this.map.get(key); //
    if (value === undefined) { // -1
      return -1;
    } else { // MapMap
      this.map.delete(key);
      this.map.set(key, value);
      return value;
    }
  }

  put(key, value) {
    if (this.map.has(key)) { // Map
      this.map.delete(key);
    }
    this.map.set(key, value); // Map
    if (this.map.size > this.capacity) { //
      const oldestKey = this.map.keys().next().value; // Map
      this.map.delete(oldestKey);
    }
  }
}

//
const cache = new LRUCache(2); // 2LRU
cache.put(1, 1);
cache.put(2, 2);
console.log(cache.get(1)); // 11
cache.put(3, 3); // 2
console.log(cache.get(2)); // -12
cache.put(4, 4); // 1
console.log(cache.get(1)); // -11
console.log(cache.get(3)); // 33
console.log(cache.get(4)); // 44

```

```
function climbStairs(n) {  
  if (n <= 2) { // n2n  
    return n;  
  }  
  let a = 1, b = 2, c;  
  for (let i = 3; i <= n; i++) { // n  
    c = a + b;  
    a = b;  
    b = c;  
  }  
  return c;  
}  
  
//  
console.log(climbStairs(2)); // 212  
console.log(climbStairs(3)); // 31212  
console.log(climbStairs(4)); // 5121222112
```

```
function fibonacci(n) {  
  if (n === 0 || n === 1) { // n01n  
    return n;  
  }  
  let a = 0, b = 1, c;  
  for (let i = 2; i <= n; i++) { // n  
    c = a + b;  
    a = b;  
    b = c;  
  }  
  return c;  
}  
  
//  
console.log(fibonacci(0)); // 0  
console.log(fibonacci(1)); // 1  
console.log(fibonacci(2)); // 1  
console.log(fibonacci(3)); // 2  
console.log(fibonacci(4)); // 3
```

```
//  
class TreeNode {  
  constructor(val) {  
    this.val = val;  
    this.left = null;  
    this.right = null;  
  }  
}
```

```
//  
function preorderTraversal(root) {  
  const res = []; //  
  function preorder(root) {  
    if (!root) {  
      return;  
    }  
    res.push(root.val);  
    preorder(root.left);  
    preorder(root.right);  
  }  
  preorder(root);  
  return res;  
}  
//  
const root = new TreeNode(1);  
root.left = new TreeNode(2);  
root.right = new TreeNode(3);  
root.left.left = new TreeNode(4);  
root.left.right = new TreeNode(5);  
  
console.log(preorderTraversal(root)); // [1, 2, 4, 5, 3]
```

```
//
function inorderTraversal(root) {
  const res = []; //
  function inorder(root) {
    if (!root) {
      return;
    }
    inorder(root.left);
    res.push(root.val);
    inorder(root.right);
  }
  inorder(root);
  return res;
}

//
const root = new TreeNode(1);
root.left = new TreeNode(2);
root.right = new TreeNode(3);
root.left.left = new TreeNode(4);
root.left.right = new TreeNode(5);

console.log(inorderTraversal(root)); // [4, 2, 5, 1, 3]
```

```
//
function postorderTraversal(root) {
  const res = []; //
  function postorder(root) {
    if (!root) {
      return;
    }
    postorder(root.left);
    postorder(root.right);
    res.push(root.val);
  }
  postorder(root);
  return res;
}

//
const root = new TreeNode(1);
root.left = new TreeNode(2);
root.right = new TreeNode(3);
root.left.left = new TreeNode(4);
root.left.right = new TreeNode(5);

console.log(postorderTraversal(root)); // [4, 5, 2, 3, 1]
```