1. 求最大公约数和最小公倍数

// 递归法求最大公约数

public static int maxCommonDivisor(int m, int n) {

if (m < n) { // 保证m>n,若m<n,则进行数据交换

int temp = m;

m = n;

n = temp;

}

if (m % n == 0) { // 若余数为0,返回最大公约数

return n;

} else { // 否则,进行递归,把n赋给m,把余数赋给n

return maxCommonDivisor(n, m % n);

}

}

// 求最小公倍数

public static int minCommonMultiple(int m, int n) {

return m \* n / maxCommonDivisor(m, n);

}

1. 分解质因数

public static void primeFactor( int num) {// 该方法将一个正整数分解成质因数相乘的形式

if (num < 2) {// 若输入的数小于2,输出提示信息

System.out.println("必须输入不小于2的正整数！");

} else {

int primeNumber = 2;// 定义最小的质数

System.out.print(num + " = ");

while (primeNumber <= num) {// 在质数小于输入的数时，进行循环

if (primeNumber == num) {// 当质数等于输入的数时,直接输出

System.out.print(num);

break;// 跳出循环

} else if (num % primeNumber == 0) {// 当输入的数与质数的余数为0时,输出这个质数

System.out.print(primeNumber + " \* ");

num = num / primeNumber;// 把剩下的结果赋给num

} else {// 在余数不为0时,质数递增

primeNumber++;

}

}

}

}

1. 判断一个数是不是素数

public static boolean isPrime(int a) {

boolean flag = true;

if (a < 2) {// 素数不小于2

return false;

} else {

for (int i = 2; i <= Math.sqrt(a); i++) {

if (a % i == 0) {// 若能被整除，则说明不是素数，返回false

flag = false;

break;// 跳出循环

}

}

}

return flag;

}

1. 求1到1000以内的完数

public static void fun() {

for (int i = 1; i <= 1000; i++) {

int temp = 0; // 定义因子之和变量

for (int n = 1; n < i / 2 + 1; n++) {

if (i % n == 0) {

temp += n; // 能被整除的除数则被加到temp中

}

}

if (temp == i) { // 如果因子之和与原数相等的话，说明是完数

System.out.print(i + " "); // 输出完数

}

}

}

1. 快速排序

static void quicksort(int n[], int left, int right) {

int dp;

if (left < right) {

dp = partition(n, left, right);

quicksort(n, left, dp - 1);

quicksort(n, dp + 1, right);

}

}

static int partition(int n[], int left, int right) {

int pivot = n[left];

while (left < right) {

while (left < right && n[right] >= pivot)

right--;

if (left < right)

n[left++] = n[right];

while (left < right && n[left] <= pivot)

left++;

if (left < right)

n[right--] = n[left];

}

n[left] = pivot;

return left;

}

1. 二分查找

public static int search(int[] nums, int num) {

int low = 0;

int high = nums.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

//与中间值比较确定在左边还是右边区间,以调整区域

if (num > nums[mid]) {

low = mid + 1;

} else if (num < nums[mid]) {

high = mid - 1;

} else {

return mid;

}

}

return -1;

}

1. 二叉树先根遍历

public static void preOrder(BinaryTree root){ //先根遍历

if(root!=null){

System.out.print(root.data+"-");

preOrder(root.left);

preOrder(root.right);

}

}

public class BinaryTree {

int data; //根节点数据

BinaryTree left; //左子树

BinaryTree right; //右子树

public BinaryTree(int data) { //实例化二叉树类

this.data = data;

left = null;

right = null;

}

}