我将我在面试中遇到的各种算法题总结如下，希望对找工作的同学有帮助！*表示遇到的次数

*1，用递归的方式判断某个字符串是不是回文字符串(正反读都一样)，比如（“abcdcba”）

```
public static boolean is回文(String s) {
        if (s.length() == 0 || s.length() == 1) {
            return true;
        }
        if (s.charAt(0) == s.charAt(s.length()-1)) {
            return is回文(s.substring(1, s.length()-1));
        }
        return false;
}
```

2，加法运算得到一个整型数字比如说是 10

有如下几种情况：

1+9=10

2+8=10

1+2+7=10

1+2+3+4=10

1+1+1+1+1+5=10

。。。。。。。。

。。。。。。。。

很多种情况(4+6 与 6+4 相同)

```
public static void split(int num, int result, int begin, StringBuffer sbf) {
        if(result==num){//如果结果为指定的结果，则直接打印并退出
            System.out.println(num+"="+sbf.toString());
            return;
        }
        for(int i=begin;i<=num-result;i++){
            String str="";
            if(sbf.length()==0){
                str=""+i;
            }else{
                str="+"+i;
            }
            sbf.append(str);
            split(num,result+i,i,sbf);
            sbf.delete(sbf.length()-str.length(), sbf.length());
        }
```

```
    }
    public static void main(String[] args) {
          int num = 10;
          int begin = 1;
            int result = 0;
            StringBuffer sbf = new StringBuffer();
            split(num, result, begin, sbf);
    }
```

*3,递归几个题

```
//求阶乘，如4!=24
    public static long factorial(int i) {
        if (i < 0)
            return -1;
        else if (i == 0 || i == 1)
            return 1;
        else
            return i * factorial(i - 1);
    }

    //求和运算，如1 + 2 + 3 +... = ?
    public static int sum(int i) {
        if (i == 0)
            return 0;
        else
            return i + sum(i - 1);
    }

    //求斐波那契数列，如1,1,2,3,5,8,13....
    public static int fibonacci(int i) {
        if (i == 0 || i == 1)
            return 1;
        else
            return fibonacci(i - 1) + fibonacci(i - 2);
    }

//非递归求斐波那契数列
    public static void fibonacci(int m) {
        long x = 1, y = 1;
        for (int i = 1; i <= m; i++) {
```

```
            y = x + y;
            x = y - x;
        }
    }
```

*4,一个农夫养了一头牛，三年后，这头牛每年会生出 1 头牛，生出来的牛三年后，又可以每年生出一头牛.....问农夫 10 年后有多少头牛?n 年呢？（用 JAVA 实现）

方法 1：

```
public class Cow {
    private static int count = 1;
    public  void feedCow(int year, int age) {
        year++;
        age++;
        if (year <= 10) {
            if (age >= 3) {
                count++;
                feedCow(year, 0);
            }
            feedCow(year, age);
        }
    }
    public static void main(String[] args) {
        new Cow().feedCow(0, 0);
        System.out.println(count);
    }
}
```

方法二：

```
public class Cow {
    public static int count = 0;
    public Cow(int year) {
        count++;
        for (int i = 3 + year; i <= 10; i++) {
            new Cow(i);
        }
    }
    public static void main(String[] args) {
        new Cow(0);
        System.out.println(count);
    }
}
```

方法三：

```
public class Cow {
    private int age;
    public Cow() {
        age = 0;
    }
    public Cow play() {
        age++;
        return age > 3 ? new Cow() : null;
    }
    public static void main(String[] args) {
        List<Cow> list = new ArrayList<Cow>();
        list.add(new Cow());
        for (int i = 0; i <= 10; i++) {
            for (int j = 0; j < list.size(); j++) {
                Cow cow = list.get(j).play();
                if (cow != null)
                    list.add(cow);
            }
        }
        System.out.println("10年后，共有：" + list.size());
    }
}
```

**5，一个整数数组，包含正数和负数，从该数组中找一个子数组，该子数组所有元素的和最大。例如数组：6，-1，9，3，-2，0，-8，4，7，1 它和最大为 19，就是这个数组本身。写出代码。

```
public class Test {
    private static int calculate(int[] array) {
        int start = 0; //起始索引
        int end = 0; //结束索引
        int max = array[0]; // 子数组的和
        int remax = array[0]; // 子数组和的最大值
        for (int i = 1; i < array.length; i++) {
            if (max > 0) { //若子数组的和的值大于0，则继续加
                max += array[i];
            } else { //否则舍去
                max = array[i];
                start = i;
            }
```

```
            if (max > remax) {
                remax = max;
                end = i;
            }
        }
        /*当输入数组中所有整数都是负数时，子数组和的最大值就是数组中的最大元素。*/
        if (remax < 0) {
            remax = array[0];
            start = end = 0;
            for (int i = 1; i < array.length; i++) {
                if (remax < array[i]) {
                    remax = array[i];
                    start = i;
                    end = i;
                }
            }
        }
        System.out.println("从" + start + "到" + end + "最大，最大为：" + remax);
        return remax;
    }
    public static void main(String[] args) {
        int[] array = {6, -1, 9, 3, -2, 0, -8, 4, 7, 1};
        calculate(array);
    }
}
```

*6，实现一个二叉排序树

```
class Node {
    int data;
    Node left;
    Node right;
    public Node(int data) {
        this.data = data;
    }
    void add(Node node) {
//添加的节点数据大于当前节点的数据时,放在右子树这边
        if (node.data < data) {
            if (left == null)
                left = node;
            else
```

```
                left.add(node);
        } else {
            if (right == null)
                right = node;
            else
                right.add(node);
        }
    }
    void printBiTree() {
        if (left != null) {
            left.printBiTree();
        }
        System.out.print(data + " ");
        if (right != null) {
            right.printBiTree();
        }
    }
}
class BiTree {
    Node root;//根结点
    BiTree addNode(Node node) {
        if (root == null)
            root = node;
        else
            root.add(node);
        return this;
    }
    BiTree addNode(int... datas) {
        for (int data : datas) {
            Node node = new Node(data);
            if (root == null)
                root = node;
            else
                root.add(node);
        }
        return this;
    }
    void printBiTree() {
        root.printBiTree();
    }
```

```
}
public class TreeTest {
    public static void main(String[] args) {
        BiTree tree = new BiTree();
        tree.addNode(8, 9, 12, 6, 3, 7);
        tree.printBiTree();
    }
}
```

*7，折半查找的非递归算法

```
class BinarySearch {
    static int search(int key, int... datas) {
        int start = 0;
        int end = datas.length - 1;
        int pos;
        while (start <= end) {
            pos = (start + end) / 2;
            if (datas[pos] == key)
                return pos;
            else if (datas[pos] > key)
                end = pos - 1;
            else
                start = pos + 1;
        }
        return -start;
    }
}
public class BinarySearchTest {
    public static void main(String[] args) {
        int [] datas = {3, 7, 8, 9, 12};
        System.out.println(BinarySearch.search(10, datas));
    }
}
```

8， 折半查找的递归算法

```
class BinarySearch {
    static int search(int key, int start , int end, int... datas) {
        if (start > end)
            return -start;
        int pos = (start + end) / 2;
```

```
        if (datas[pos] == key)
            return pos;
        else if (datas[pos] > key)
            return search(key, start, pos - 1, datas);
        else
            return search(key, pos + 1, end, datas);
    }
}
public class BinarySearchTest {
    public static void main(String[] args) {
        int [] datas = {3, 7, 8, 9, 12};
        System.out.println(BinarySearch.search(9, 0, datas.length - 1,
datas));
    }
}
```

*8，快速排序

```
class QuickSort {
    /*调用此方法将确定start所指的值的最后排好序的位置*/
    private static int sort(int start, int end, int... datas) {
        int key = datas[start];
        while (start < end) {
            while (start < end && key <= datas[end]) {
                end--;
            }
            datas[start] = datas[end];
            while (start < end && key >= datas[start]) {
                start++;
            }
            datas[end] = datas[start];
        }
        datas[start] = key;
        return start;
    }
    static void qSort(int start, int end, int... datas) {
        if (start < end) {
            int pos = sort(start, end, datas);
            qSort(start, pos - 1, datas);  //递归调用
            qSort(pos + 1, end, datas);
        }
```

```
    }
}
```

冒泡排序

```
class BubbleSort {
    /* 从大到小*/
    public static void sort(int... datas) {
        boolean flag = true;
        for (int i = datas.length - 1; i > 0 && flag; i--) {
            flag = false;
            for (int j = 0; j < i; j++) {
                if (datas[j] < datas[j+1]) { //改成>则从小到大排序
                    int temp = datas[j];
                    datas[j] = datas[j+1];
                    datas[j+1] = temp;
                    flag = true;
                }
            }
        }
    }
}
```

9，简单选择排序

```
class SelectSort {
    public static void sort(int... datas) {
        for (int i = 0; i<datas.length - 1; i++) {
            int k = i;
            for (int j = i + 1; j < datas.length; j++) {
                if (datas[k] < datas[j]) //若改成>则表示从小到大排序
                    k = j;
            }
            if (k != i) {
                int temp = datas[k];
                datas[k] = datas[i];
                datas[i] = temp;
            }
        }
    }
}
```

10，插入排序

```
class InsertSort {
    public static void sort(int... datas) {
        for (int i = 1; i < datas.length; i++) {
            int temp = datas[i];
            int j;
            for (j = i; j > 0; j--) {
                if (datas[j-1] > temp) { //改成<则表示从大到小排序
                    datas[j] = datas[j-1];
                } else
                    break;
            }
            datas[j] = temp;
        }
    }
}
```