SUMMER-HOMEWORK

暑假作业

等不是办法，干才有希望

不走心的努力都是敷衍自己

比特就业课假期作业- Java语法+数据结构作业

出题老师：

> Java选择题：高博（day01-day16）qq：1262913815
>
> Java代码题：刘益铭（day01-day16）qq：381348940

作业说明：

> 1、本次作业涵盖内容为Java基础，数据结构等相关知识点
>
> 2、如果对试卷上的题目，或者答案有问题，可以联系对应老师哦~~
>
> 3、同学们添加老师时备注：姓名+比特班级哦~

day01

一、选择题

1、JDK中提供的java、javac、jar等开发工具也是用Java编写的。（）

A. 对 B. 错

2、下面是java语言的基本特性是？【多选】（）

A. 封装 B. 多态 C. 继承 D. 高效

3、以下（）不是合法的标识符？（）

A. STRING B. x3x C. void D. deSf

4、关于Float，下列说法错误的是（）

A: Float是一个类

B: Float在java.lang包中

C: float a=1.0是正确的赋值方法

D: Float a= new Float(1.0)是正确的赋值方法

5、下面的java赋值语句哪些是有错误的（）

A: int i =1000;

B: float f = 45.0;

C: char s = '\u0639';

D: String s = "hello,world\0";

二、编程题

1、给定一个长度为 n 的字符串，请编写一个函数判断该字符串是否回文。如果是回文请返回true，否则返回false。

字符串回文指该字符串正序与其逆序逐字符一致。OJ链接 【难度：简单】

示例:
输入："absba"
输出：true

```
public class Solution {
    public boolean judge (String str) {
        public class Solution {
            public boolean judge (String str) {
                int left = 0;
                int right= str.length() - 1;
                while(left < right){
                    if(str.charAt(left) != str.charAt(right)){
                        return false;
                    }
                    left++;
                    right--;
                }
                return true;
            }
        }
    }
}
```

2、实现函数 int sqrt(int x).

计算并返回 x 的平方根（向下取整）。OJ链接 【难度：简单】

示例1:
输入：2
返回值：1

实例2:
输入：2143195649
返回值：46294

```
public class Solution {
    public int sqrt(int x) {
        为什么比较的时候都是用除而不是直接mid*mid <= x之类的，如果当x过大，很有可能mid*mid直接溢出
        import java.util.*;


        public class Solution {
            public int sqrt (int x) {
                int left = 1;
                int right = x;
                while(left <= right){
                    int mid = (left + right)/2;
                    if(mid <= x/mid && (mid+1)>x/(mid+1)){
                        return mid;
                    }
                    if(mid > x/mid){
                        right = mid - 1;
                    }else{
                        left = mid + 1;
                    }
                }
                return 0;//输入0的情况单独考虑下
            }
        }
```

```
}
```

day02

一、选择题

1、下面哪些选项是正确的【多选】（ ）AD

A: `>>`是算术右移操作符

B: `>>`是逻辑右移操作符

C: `>>>`是算术右移操作符 无符号右移

特别注意没有无符号左移这种运算符

D: `>>>`是逻辑右移操作符

2、定义如下程序：

```
public static void main(String[] args){
    Double x=1.2;
    long l = 1.2;
    float f =  x/l;
    System.out.println(f);
}
```

程序执行结果是？ （D）

A: `1` B: `1f` C: `运行报错` D: `编译报错`

3、有如下代码：请写出程序的输出结果。（B）

```
public class Test{
    public static void main(String[] args){
        int x = 0;
        int y = 0;
        int k = 0;
        for (int z = 0; z < 5; z++) {
            if ((++x > 2) && (++y > 2) && (k++ > 2)){
                x++;
                ++y;
                k++;
            }
        }
        System.out.println(x + "" +y + "" +k);
    }
}
```

++x => x=1

z=0 x=1

z=1 x=2

z=2 x=3 y=1

z=3 x=4, y=2

z=4 x=5, y=3 k=1

A: `432` B: `531` C: `421` D: `523`

4、下列语句序列执行后，输出结果是：（B）

```
public class Ex{
   public static void main(String[]args){
      int a=13;
      a=a/5;
      System.out.println(a);
   }
}
```

A: 1 B: 2 C: 3 D: 4

5、以下代码段执行后的输出结果为：（）

```
public class Test {
   public static void main(String args[]) {
      int i = -5;
      i =  ++(i++);
      System.out.println(i);
   }
}
```

A: -7 B: -3 C: 编译错误 D: -5

二、编程题

1、截断句子

句子 是一个单词列表，列表中的单词之间用单个空格隔开，且不存在前导或尾随空格。每个单词仅由大小写英文字母组成（不含标点符号）。例如，"Hello World"、"HELLO" 和 "hello world hello world" 都是句子。给你一个句子 s 和一个整数 k ，请你将 s 截断 ，使截断后的句子仅含 前 k 个单词。返回 截断 s 后得到的句子。OJ链接 【力扣-1816号问题】【难度：简单】

```
输入：s = "Hello how are you Contestant", k = 4
输出："Hello how are you"
解释：
s 中的单词为 ["Hello", "how" "are", "you", "Contestant"]
前 4 个单词为 ["Hello", "how", "are", "you"]
因此，应当返回 "Hello how are you"
```

```
class Solution {
   public String truncateSentence(String s, int k) {
```

```
class Solution {
    public String truncateSentence(String s, int k) {
        String[] str = s.split(" ");
        String ret = "";
        for(int i = 0;i < k;i++){
            if(i != k-1){
                ret += str[i] +" ";
            }else{
                ret += str[i];
            }
        }
        return ret;
    }
}
```

```
    }
}
```

2、删除有序数组中的重复项

给你一个有序数组 nums ， 请你 原地 删除重复出现的元素，使每个元素 只出现一次 ， 返回删除后数组的新长度。不要使用额外的数组空间，你必须在 原地 修改输入数组 并在使用 O(1) 额外空间的条件下完成。OJ链接 【力扣-26号问题】【难度：简单】

示例1:
输入：nums = [1,1,2]
输出：2, nums = [1,2]
解释：函数应该返回新的长度 2 ， 并且原数组 nums 的前两个元素被修改为 1, 2 。不需要考虑数组中超出新长度后面的元素。

示例2:
输入：nums = [0,0,1,1,1,2,2,3,3,4]
输出：5, nums = [0,1,2,3,4]
解释：函数应该返回新的长度 5 ， 并且原数组 nums 的前五个元素被修改为 0, 1, 2, 3, 4 。不需要考虑数组中超出新长度后面的元素。

```
class Solution {
    public int removeDuplicates(int[] nums) {
class Solution {
    public int removeDuplicates(int[] nums) {
        int head1 = 0;
        int head2 = 0;
        while(head2 < nums.length){
            if(nums[head1] != nums[head2]){
                head1++;//head1往前走一步
                nums[head1] = nums[head2];//把后面的值搬到前面来，或者说就是自己给自己赋值
            }
            head2++;//head1 head2所在值相等head2就往后走
        }
        return head1 + 1;
    }
}
    }
}
```

day03

一、选择题

1、以下JAVA程序的运行结果是什么 （ ）

```
public static void main(String[] args) {
    Double o1 = true ? new Integer(1) : new Double(2.0);
    Object o2;
    if(true){
    o2 = new Integer(1);
    }else{
        o2 = new Double(2.0);
    }
    System.out.print(o1);
    System.out.print(" ");
    System.out.print(o2);
}
```

A: `1 1` B: `1.0 1.0` C: `1 1.0` D: `1.0 1`

2、以下JAVA程序代码的输出是，定义如下程序：( )

```
public static void main(String args[]) {
    System.out.println(14^3);
}
```

A: `2744` B: `13` C: `17` D: `11`

3、已知 boolean result = false，则下面哪个选项是合法的：【多选】( )

A: `result=1`

B: `result=true`

C: `if(result!=0) {//so something...}`

D: `if(result) {//do something...}`

4、以下那些代码段能正确执行：【多选】( )

A:

```
public static void main(String args[]) {
    byte a = 3;
    byte b = 2;
    b = a + b; a+b会转换为int,这里要强制转换为byte
    System.out.println(b);
}
```

B:

```
public static void main(String args[]) {
    byte a = 127;
    byte b = 126;
    b = a + b;
    System.out.println(b);
}
```

C:

```
public static void main(String args[]) {
    byte a = 3;
    byte b = 2;
    a+=b; 自带类型转换
    System.out.println(b);
}
```

D:

```
public static void main(String args[]) {
    byte a = 127;
    byte b = 127;
    a+=b;
    System.out.println(b); 结果还是127，代码没问题，但是结果发生截断是不准确的
}
```

5、以下表达式的类型和值是什么？（注意整数除法）（ ）

```
-5 + 1/4 + 2*-3 + 5.0
```

A: int -3 B: int -4 C: double -5.5 D: double -6.0

二、编程题

1、移除元素

给你一个数组 nums 和一个值 val，你需要 原地 移除所有数值等于 val 的元素，并返回移除后数组的新长度。不要使用额外的数组空间，你必须仅使用 O(1) 额外空间并 原地 修改输入数组。元素的顺序可以改变。你不需要考虑数组中超出新长度后面的元素。OJ链接【力扣-27号问题】【难度：简单】

示例1:
输入：nums = [3,2,2,3], val = 3
输出：2, nums = [2,2]
解释：函数应该返回新的长度 2, 并且 nums 中的前两个元素均为 2。你不需要考虑数组中超出新长度后面的元素。例如，函数返回的新长度为 2 ，而 nums = [2,2,3,3] 或 nums = [2,2,0,0]，也会被视作正确答案。

示例2:
输入：nums = [0,1,2,2,3,0,4,2], val = 2
输出：5, nums = [0,1,4,0,3]
解释：函数应该返回新的长度 5, 并且 nums 中的前五个元素为 0, 1, 3, 0, 4。注意这五个元素可为任意顺序。你不需要考虑数组中超出新长度后面的元素。

```
class Solution {
    public int removeElement(int[] nums, int val) {
```

```
class Solution {
    public int removeElement(int[] nums, int val) {
        int head = 0;
        for(int i = 0;i < nums.length;i++){
            if(nums[i] != val){
                nums[head] = nums[i];
                head++;
            }
        }
        return head;
    }
}
```

```
    }
}
```

2、罗马数字转整数

罗马数字包含以下七种字符: I， V， X， L，C，D 和 M。

字符 数值 I 1 V 5 X 10 L 50 C 100 D 500 M 1000

例如， 罗马数字 2 写做 II ，即为两个并列的 1 。12 写做 XII ，即为 X + II 。 27 写做 XXVII, 即为 XX + V + II 。

通常情况下，罗马数字中小的数字在大的数字的右边。但也存在特例，例如 4 不写做 IIII，而是 IV。数字 1 在数字 5 的左边，所表示的数等于大数 5 减小数 1 得到的数值 4 。同样地，数字 9 表示为 IX。这个特殊的规则只适用于以下六种情况：

I 可以放在 V (5) 和 X (10) 的左边，来表示 4 和 9。 X 可以放在 L (50) 和 C (100) 的左边，来表示 40 和 90。 C 可以放在 D (500) 和 M (1000) 的左边，来表示 400 和 900。 给定一个罗马数字，将其转换成整数。

OJ链接【力扣-13号问题】【难度：简单】

示例1:
输入: s = "III"
输出: 3

示例2:

输入: s = "IV"

输出: 4

示例3:
输入: s = "LVIII"
输出: 58
解释: L = 50, V= 5, III = 3

示例4:
输入: s = "MCMXCIV"
输出: 1994
解释: M = 1000, CM = 900, XC = 90, IV

```
class Solution {
    public int romanToInt(String s) {
    }
}
```

```
class Solution {
    public int relevantInt(char c){
        int ret = 0;
        switch(c){
            case 'I':
                ret = 1;
                break;
            case 'V':
                ret = 5;
                break;
            case 'X':
                ret = 10;
                break;
            case 'L':
                ret = 50;
                break;
            case 'C':
                ret = 100;
                break;
            case 'D':
                ret = 500;
                break;
            case 'M':
                ret = 1000;
                break;
        }
        return ret;
    }
    public int romanToInt(String s) {
        int prenum = relevantInt(s.charAt(0));
        int sum = 0;
        for(int i = 1;i < s.length();i++){
            int curnum = relevantInt(s.charAt(i));
            if(prenum < curnum){//左边小就减
                sum -= prenum;
            }else{
                sum += prenum;//左边大就加
            }
            prenum = curnum;
        }
        sum += prenum;
        return sum;
    }
}
```

day04

一、选择题

1、在Java中，以下数据类型中,需要内存最多的是（ ）

A: byte B: long C: Object D: int

2、以下选项中，switch语句判断条件可以接受的数据类型有哪些【多选】（ ）

A: int B: byte C: char D: short

3、有如下代码：请写出程序的输出结果。（ ）

```
public class Test
{
   public static void main(String[] args) {
      int x = 0;
      for (int z = 0; z < 5; z++) {
         if (z>=2)
         {
            x++;
         }
      }
      System.out.println(x);
   }
}
```

A: 4 B: 3 C: 1 D: 2

4、下面结果输出是？ ()

```
public class IfTest{
   public static void main(String[]args){
      int x=3;
      int y=1;
      if(x=y)
         System.out.println("Not equal");
      else
         System.out.println("Equal");
    }
}
```

A: The output is “Equal”

B: The output in “Not Equal”

C: An error at line 5 causes compilation to fall.

D: The program executes but does not print a message.

5、下列循环语句序列执行完成后，i的值是（）

```
int i;
for(i=2;i<=10;i++){
   System.out.println(i);
}
```

A: 2 B: 10 C: 11 D: 不确定A

二、编程题

1、快乐数

编写一个算法来判断一个数 n 是不是快乐数。

「快乐数」定义为：

对于一个正整数，每一次将该数替换为它每个位置上的数字的平方和。然后重复这个过程直到这个数变为 1，也可能是 无限循环 但始终变不到 1。如果 可以变为 1，那么这个数就是快乐数。如果 n 是快乐数就返回 true ； 不是，则返回 false 。OJ链接【力扣-202号问题】【难度：简单】

示例1:
输入：n = 19
输出：true
解释：
12 + 92 = 82
82 + 22 = 68
62 + 82 = 100
12 + 02 + 02 = 1

示例2:
输入：n = 2
输出：false

【解题思路】：

通过反复调用 getNext(n) 得到的链是一个隐式的链表。隐式意味着我们没有实际的链表节点和指针，但数据仍然形成链表结构。起始数字是链表的头 “节点”，链中的所有其他数字都是节点。next 指针是通过调用 getNext(n) 函数获得。

意识到我们实际有个链表，那么这个问题就可以转换为检测一个链表是否有环。因此我们在这里可以使用弗洛伊德循环查找算法。这个算法是两个奔跑选手，一个跑的快，一个跑得慢。在龟兔赛跑的寓言中，跑的慢的称为 “乌龟”，跑得快的称为 “兔子”。

不管乌龟和兔子在循环中从哪里开始，它们最终都会相遇。这是因为兔子每走一步就向乌龟靠近一个节点（在它们的移动方向上）。

【代码实现】：

```
class Solution {
    public boolean isHappy(int n) {
    }
}
```

```
class Solution {    public int getSum(int n){        int sum = 0;
while(n != 0){            int div = n%10;            sum +=
div*div;            n /= 10;        }        return sum;    }
public boolean isHappy(int n) {        //如果不是快乐数，最终它会陷
入一个循环里面,所以set中就会有这个元素就说明陷入循环了        Set
set = new HashSet>();        int ret = getSum(n);        while(ret
!= 1 && !set.contains(ret)){            set.add(ret);
ret = getSum(ret);        }        //跳出循环两种情况，变为了1，或
者set中有了某个ret说名陷入了循环        return ret == 1;    }}
```

2、丑数

给你一个整数 n ，请你判断 n 是否为 丑数 。如果是，返回 true ； 否则，返回 false 。丑数 就是只包含质因数 2、3 和/或 5 的正整数。OJ链接【力扣-263号问题】【难度：简单】

主要思想还是说如果这个数它只含有2,3,5中的几个因子的话，当你把它有的这几个因子完全除掉之后数肯定是变为1的

示例1:
输入：n = 6
输出：true
解释：6 = 2 × 3

示例2:
输入：n = 8
输出：true
解释：8 = 2 × 2 × 2

示例3:
输入：n = 14
输出：false
解释：14 不是丑数，因为它包含了另外一个质因数 7 。

```
class Solution {
    public boolean isUgly(int n) {
class Solution {
    public boolean isUgly(int n) {
        if(n <= 0){
            return false;
        }
        int[] nums = new int[]{2,3,5};
        for(int i = 0;i < nums.length;i++){
            while(n % nums[i] == 0){
                n /= nums[i];//把这个因数给它除完
            }
        }
        return n == 1;
    }
}
    }
}
```

day05

一、选择题

1、以下程序的输出结果是（ ）

```
public class Print{
   static boolean out(char c){
      System.out.print(c);
      return true;
   }
   public static void main(String[] argv){
      int i = 0;
      for(out('A');out('B') && (i<2);out('C')){
         i++;
         out('D');
      }
   }
}
```

A: ABDCBDCB B: BCDABCD C: 编译错误 D: 运行错误

2、Java类Demo中存在方法func1、func2、func3和func4，请问该方法中，哪些是不合法的定义？【多选】

```
public class Demo{

      float func1(){
            int i=1;
            return;
      }
      float func2(){
            short i=2;
            return i;
      }
      float func3(){
            long i=3;
            return i;
      }
      float func4(){
            double i=4;
            return i;
      }
}
```

A: func1 B: func2 C: func3 D: func4

3、对于下面这段代码，以下说法正确的是（）

```
public class Test
{
   public int x;
   public static void main(String []args) 静态的方法里面不能直接使用非静态的内容
   {
      System. out. println("Value is" + x);
   }

}
```

A: 程序会打出 "Value is 0"

B: 程序会抛出NullPointerException

C: 非静态变量不能够被静态方法引用

D: 编译器会抛出 "possible reference before assignment"的错误

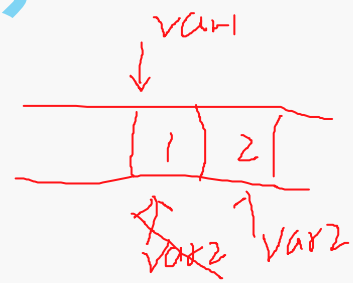
4、请输出最终内容（ ）

```
int i = 3;
String result = new String();
switch (i) {
    case 1:
        result = result + "him ";
    case 2:
        result = result + "her ";
    case 3:
        result = result + "it ";
    default:
        result = result + "me ";
}
System.out.println(result);
```

A: it me B: him her it me C: him her D: me

5、以下Java程序运行的结果是 （ ）

```
public class Tester{
    public static void main(String[] args){
        Integer var1=new Integer(1);
        Integer var2=var1;
        doSomething(var2);
        System.out.print(var1.intValue());
        System.out.print(var1==var2);
    }
    public static void doSomething(Integer integer){
        integer=new Integer(2);
    }
}
```



A: 1true B: 2true C: 1false D: 2false

二、编程题

1、各位相加

给定一个非负整数 num，反复将各个位上的数字相加，直到结果为一位数。OJ链接【力扣-258号问题】【难度：简单】

示例1:
输入: 38
输出: 2
解释: 各位相加的过程为：3 + 8 = 11, 1 + 1 = 2。 由于 2 是一位数，所以返回 2。

```
public int addDigits(int num) {
```

```
class Solution {
    public int getSum(int n){
        if(n/10 == 0){
            return n;
        }
        int sum = 0;
        while(n != 0){
            sum += n%10;
            n /= 10;
        }
        n = sum;
        return getSum(n);
    }
    public int addDigits(int num) {
        if(num/10 == 0){
            return num;
        }
        return getSum(num);
    }
}
```

2、猜数字大小

猜数字游戏的规则如下：

- 每轮游戏，我都会从 1 到 n 随机选择一个数字。 请你猜选出的是哪个数字。
- 如果你猜错了，我会告诉你，你猜测的数字比我选出的数字是大了还是小了。

你可以通过调用一个预先定义好的接口 int guess(int num) 来获取猜测结果，返回值一共有 3 种可能的情况（-1，1 或 0）：

- -1：我选出的数字比你猜的数字小 pick < num
- 1：我选出的数字比你猜的数字大 pick > num
- 0：我选出的数字和你猜的数字一样。恭喜！你猜对了！pick == num

  返回我选出的数字。

OJ链接【力扣-374号问题】【难度：简单】

示例1:
输入：n = 10, pick = 6
输出：6

示例2:
输入：n = 1, pick = 1
输出：1

```
public class Solution extends GuessGame {
    public int guessNumber(int n) {
        public class Solution extends GuessGame {
            public int guessNumber(int n) {
                //n这个n是你猜的范围的上界，也即是[1,n]
                //每次你从这个范围里面猜一个数，然后把这个数传到guess看你是猜大了还是小了
                //整个的意思就是去范围里面查找它设定的那个值
                int left = 1;
                int right = n;
                int mid = 0;
                while(left <= right){
                    mid = left + (right - left)/2;
                    if(guess(mid) > 0){
                        left = mid + 1;
                    }else if(guess(mid) < 0){
                        right = mid - 1;
                    }else{
                        break;
                    }
                }
                return mid;
            }
        }
    }
}
```

day06

一、选择题

1、类 ABC 定义如下：

```
1. public class ABC{
2.     public double max( double a, double b) { return a; }
3.
4. }
```

将以下哪个方法插入行 3 是不合法的（ ）

A: public float max(float a, float b, float c){ return a }

B: public double max (double c, double d){ return c }

C: public float max(float a, float b){ return a }

D: private int max(int a, int b, int c){return a }

2、下列关于Java类中方法的定义，正确的是（ ）

A: 若代码执行到return语句，则将当前值返回，而且继续执行return语句后面的语句。

B: 只需要对使用基本数据类型定义的属性使用getter和setter，体现类的封装性。

C: 方法的返回值只能是基本数据类型。

D: 在同一个类中定义的方法，允许方法名称相同而形参列表不同。

3、下面哪个方法是 public void example(){...} 的重载方法？（ ）

A: public void Example(int m){...}

B: public int example(){...}

C: public void example2(){...}

D: public int example (int m, float f){...}

4、检查程序，是否存在问题，如果存在指出问题所在，如果不存在，说明输出结果（ ）

```
package algorithms.com.guan.javajicu;
public class Inc {
    public static void main(String[] args) {
        Inc inc = new Inc();
        int i = 0;
        inc.fermin(i);
        i= i++;
        System.out.println(i);

    }
    void fermin(int i){
        i++;
    }
}
```

A: 0 B: 1 C: 2 D: 3

5、下列关于while循环、do-while循环和for循环说法错误的是【多选】（ ）

A: while循环先执行条件判断，do-while循环执行循环体

B: do-while循环结束的条件是关键字while后的条件表达式成立

C: for循环结构中的3个表达式缺一不可

D: while循环能够实现的操作，for循环也能实现

二、编程题

1、搜索插入位置

给定一个排序数组和一个目标值，在数组中找到目标值，并返回其索引。如果目标值不存在于数组中，返回它将会被按顺序插入的位置。

二分查找的一个简单应用

OJ链接【力扣-35号问题】【难度：简单】

示例1:
输入: nums = [1,3,5,6], target = 5
输出: 2

示例2:
输入: nums = [1,3,5,6], target = 2
输出: 1

【代码实现】：

```
class Solution {
    public int searchInsert(int[] nums, int target) {
        int left = 0;
        int right = nums.length - 1;
        int mid = 0;
        while(left <= right){
            mid = left + (right - left)/2;
            if(nums[mid] < target){
                left = mid+1;
            }else if(nums[mid] > target){
                right = mid-1;
            }else{
                break;
            }
        }
        if(nums[mid] == target){
            return mid;
        }else{
            return left;
        }

    }
}
```

2、第

你是产品经理，目前正在带领一个团队开发新的产品。不幸的是，你的产品的最新版本没有通过质量检测。由于每个版本都是基于之前的版本开发的，所以错误的版本之后的所有版本都是错的。假设你有 n 个版本 [1, 2, ..., n]，你想找出导致之后所有版本出错的第一个错误的版本。你可以通过调用 bool isBadVersion(version) 接口来判断版本号 version 是否在单元测试中出错。实现一个函数来查找第一个错误的版本。你应该尽量减少对调用 API 的次数。
OJ链接【力扣-278号问题】【难度：简单】

示例1:
输入：n = 5, bad = 4
输出：4
解释:
调用 isBadVersion(3) -> false
调用 isBadVersion(5) -> true
调用 isBadVersion(4) -> true
所以，4 是第一个错误的版本。

示例2:
输入：n = 1, bad = 1
输出：1

这个题是二分查找的变种，比较不同的就是left,right的变化不再是以mid的大小为准了。如果mid是坏的。那就说明后面肯定都是坏的，所以right = mid,再从这个区间去找。如果mid是好的，那么之前肯定都是好的，从mid+1的位置再去找第一个坏的

```
public class Solution extends VersionControl {
    public int firstBadVersion(int n) {
public class Solution extends VersionControl {
    public int firstBadVersion(int n) {
        int left = 1;
        int right = n;
        while(left < right){
            int mid = left + (right - left)/2;
            if(isBadVersion(mid) == true){
                right = mid;
            }else{
                left = mid+1;
            }
        }
        //出循环就是left，right相遇了，就是哪个第一个错误的
        return left;
    }
}
}
```

day07

一、选择题

1、扩展方法能访问被扩展对象的public成员（ ）

A: 能 B: 不能

扩展方法是只能声明静态的

2、下面关于静态方法说明正确的是（ ）

A: 在静态方法中可用this来调用本类的类方法

B: 在静态方法中调用本类的静态方法时可直接调用

C: 在静态方法中只能调用本类中的静态方法

静态内部类就是一个很好的例子

D: 在静态方法中绝对不能调用实例方法

3、如果类的方法的返回值为空，该方法的返回类型应是（ ）

A: void

B: null

C: abstract

D: default

4、Which of the following statements are valid array declaration? （ ）

A: int number();

B: float average[];

C: double[] marks;

D: counter int[];

5、关于下面代码 int[] x=new int[25]; 描述正确的是（ ）

A: x[25]存放了数据“0”

B: x[24] 存放了数据“0”

C: 若访问x[25]，程序将抛出异常

D: x[1]访问此数组的第一个元素

二、编程题

1、判定是否互为字符重排

给定两个字符串 s1 和 s2，请编写一个程序，确定其中一个字符串的字符重新排列后，能否变成另一个字符串。
OJ链接【力扣-面试题01.02号问题】【难度：简单】

```
示例1:
输入: s1 = "abc", s2 = "bca"
输出: true


示例2:
输入: s1 = "abc", s2 = "bad"
输出: false
```

```
class Solution {
    public boolean CheckPermutation(String s1, String s2) {
        class Solution {
            public boolean CheckPermutation(String s1, String s2) {
                //两个字符串里面的字符种类以及每个的个数都必须是一样的
                if(s1.length() != s2.length()){
                    return false;//长度都不一样，肯定不是重排的
                }
                int[] array = new int[26];
                for(int i = 0;i < s1.length();i++){
                    array[s1.charAt(i) - 97]++;
                }

                for(int i = 0;i < s2.length();i++){
                    array[s2.charAt(i) - 97]--;
                    if(array[s2.charAt(i) - 97] < 0){//如果两个里面字符种类个数不一样，那么就会出现有的位置减出值是负数的
                        return false;
                    }
                }
                //如果是重排的，最后数组肯定是全0
                return true;

            }
        }
}
```

2、最

有一堆石头，每块石头的重量都是正整数。每一回合，从中选出两块 最重的 石头，然后将它们一起粉碎。假设石头的重量分别为 x 和 y，且 x <= y。那么粉碎的可能结果如下：

- 如果 x == y，那么两块石头都会被完全粉碎；
- 如果 x != y，那么重量为 x 的石头将会完全粉碎，而重量为 y 的石头新重量为 y-x。
- 最后，最多只会剩下一块石头。返回此石头的重量。如果没有石头剩下，就返回 0。

OJ链接【力扣-1046号问题】【难度：简单】

示例:
输入：[2,7,4,1,8,1]
输出：1
解释：
先选出 7 和 8，得到 1，所以数组转换为 [2,4,1,1,1]，
再选出 2 和 4，得到 2，所以数组转换为 [2,1,1,1]，
接着是 2 和 1，得到 1，所以数组转换为 [1,1,1]，
最后选出 1 和 1，得到 0，最终数组转换为 [1]，这就是最后剩下那块石头的重量。

```
class Solution {
    public int lastStoneWeight(int[] stones
    }
}
```

```
class Solution {
    public int lastStoneWeight(int[] stones) {
        PriorityQueue<Integer> priorityQueue = new
PriorityQueue<>(new Comparator<Integer>() {
            @Override
            public int compare(Integer o1, Integer o2) {
                return o2.compareTo(o1);//建立大根堆
            }
        });
        for(int i = 0;i < stones.length;i++){
            priorityQueue.offer(stones[i]);
        }

        while(priorityQueue.size() > 1){//最起码有两个元素才能
连续出堆两次，只有一个元素就没必要了
            int num1 = priorityQueue.poll();
            int num2 = priorityQueue.poll();
            int sub = num1 - num2;
            if(sub != 0){
                priorityQueue.offer(sub);
            }
        }
        //这里就要两种情况，堆是空的，或者刚好剩一个元素
        if(priorityQueue.isEmpty()){
            return 0;
        }
        return priorityQueue.poll();

    }
}
```

day08

一、选择题

1、关于java中的数组，下面的一些描述，哪些描述是准确的（ ）

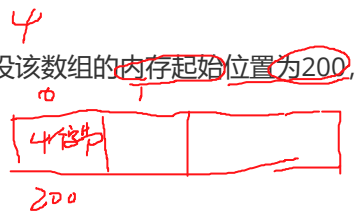
A: 数组是一个对象，不同类型的数组具有不同的类

B: 数组长度是可以动态调整的

C: 数组是一个连续的存储结构

D: 一个固定长度的数组可类似这样定义：int array[100]

2、若声明一个浮点数数组如下： float average[]=new float[30]；假设该数组的内存起始位置为200，average[15]的内存地址是（ ）



A: 214 B: 215 C: 260 D: 256

3、下列哪个类的声明是正确的（ ）

抽象类就是用来继承的

A: abstract final class HI{}

B: abstract private move{}

C: protected private number{}

D: public abstract class Car{}

4、与算法的时间复杂度有关的是（ ）

A. 问题规模 B. 计算机硬件性能 C. 编译程序质量 D. 程序设计语言

5、某算法的时间复杂度为

$$O(n^2)$$

，表明该算法的（ ）

A. 问题规模是n^2 B. 执行时间等于n^2 C. 执行时间与n^2成正比 D. 问题规模与n^2成正比

二、编程题

1、拼写单词

给你一份『词汇表』（字符串数组） words 和一张『字母表』（字符串） chars。

假如你可以用 chars 中的『字母』（字符）拼写出 words 中的某个『单词』（字符串），那么我们就认为你掌握了这个单词。

注意：每次拼写（指拼写词汇表中的一个单词）时，chars 中的每个字母都只能用一次。返回词汇表words 中你掌握的所有单词的长度之和。OJ链接【力扣-1160号问题】【难度：简单】

示例1:
输入：words = ["cat","bt","hat","tree"], chars = "atach"
输出：6
解释：
可以形成字符串 "cat" 和 "hat"，所以答案是 3 + 3 = 6。

示例2:
输入：words = ["hello","world","leetcode"], chars = "welldonehoneyr"
输出：10
解释：
可以形成字符串 "hello" 和 "world"，所以答案是 5 + 5 = 10。

也还是一个哈希表的思想

```
class Solution {
    public int countCharacters(String[] words, String chars) {
        int count = 0;
        for(String s:words){
            int[] arr = new int[26];
            //把chars中的各个字符统计好
            for(int i = 0;i < chars.length();i++){
                arr[chars.charAt(i) - 97]++;
            }

            int flg = 1;
            for(int i = 0;i < s.length();i++){
                if(arr[s.charAt(i) - 97] > 0){
                    arr[s.charAt(i) - 97]--;
                }else{
                    flg = 0;
                    break;
                }
            }
            if(flg == 1){
                count += s.length();
            }
        }
        return count;
    }
}
```

```
    }
}
```

2、一年中的第几天

给你一个按 YYYY-MM-DD 格式表示日期的字符串 date，请你计算并返回该日期是当年的第几天。通常情况下，我们认为 1 月 1 日是每年的第 1 天，1 月 2 日是每年的第 2 天，依此类推。每个月的天数与现行公元纪年法（格里高利历）一致。OJ链接【力扣-1160号问题】【难度：简单】

```
示例1:
输入：date = "2019-01-09"
输出：9

示例2:
输入：date = "2019-02-10"
输出：41
```

```
class Solution {
    public int dayOfYear(String date) {
        class Solution {
            public boolean isLeapYear(int year){
                if((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)){
                    return true;
                }
                return false;
            }
            public int dayOfYear(String date) {
                String[] arr = date.split("-");
                int[] dates = new int[]{31,28,31,30,31,30,31,31,30,31,30,31};
                if(isLeapYear(Integer.parseInt(arr[0]))){
                    dates[1]++;//如果是闰年是不一样的
                }
                int month = Integer.parseInt(arr[1]);
                int data = Integer.parseInt(arr[2]);
                int ret = 0;
                for(int i = 0;i < month - 1;i++){
                    ret += dates[i];
                }
                ret += data;
                return ret;
            }
        }
    }
}
```


day09

一、选择题

1、请指出冒泡排序平均时间复杂度（ ）

A. n^2 B. nlogn C. n D. logn

2、确定如下关于求n!算法的时间复杂度是（ ）

```
long fac(int n) {
    return (n > 1 ? n * fac(n - 1) : 1);
}
```

A. O(n) B. O(nlog(n)) C. O(n^2) D. O(n^3)

3、判断对错。List，Set，HashMap都继承自Collection接口 （ ）

A. 对 B. 错

4、下面哪些类实现或者继承了Collection接口？【多选】（ ）

A. HashMap B. ArrayList C. Vector D. Iterator

5、关于容器下面说法正确的是 （ ）

A. 列表(List)和集合(Set)存放的元素都是可重复的。

B. 列表(List)和集合(Set)存放的元素都是不可重复的。

C. 映射(Map)<key,value>中key是可以重复的。

D. 映射(Map)<key,value>中value是可以重复的。

二、编程题

1、从尾到头打印链表

输入一个链表的头节点，从尾到头反过来返回每个节点的值（用数组返回）。OJ链接【难度：简单】

```
示例：
输入：head = [1,3,2]
输出：[2,3,1]
```

```
class Solution {
    public int[] reversePrint(ListNode head) {
class Solution {
    public int[] reversePrint(ListNode head) {
        //递归，栈都可
        Stack<Integer> stack = new Stack<>();
        ListNode cur = head;
        while(cur != null){
            stack.push(cur.val);
            cur = cur.next;
        }
        int[] arr = new int[stack.size()];
        int k = 0;
        while(!stack.empty()){
            arr[k] = stack.pop();
            k++;
        }
        return arr;
    }
}
}
```

2、移除未排序链表重复节点

编写代码，移除未排序链表中的重复节点。保留最开始出现的节点。OJ链接【难度：简单】

示例1：
输入：[1, 2, 3, 3, 2, 1]
输出：[1, 2, 3]

实例2:
输入：[1, 1, 1, 1, 2]
输出：[1, 2]

```
class Solution {
    public ListNode removeDuplicateNodes(ListNode head) {
        class Solution {
            public ListNode removeDuplicateNodes(ListNode head) {
                if(head == null){
                    return null;
                }
                ListNode virHead = new ListNode(-1);
                ListNode tmp = virHead;
                Set<Integer> set = new HashSet<>();
                ListNode cur = head;
                while(cur != null){
                    if(!set.contains(cur.val)){
                        set.add(cur.val);
                        tmp.next = cur;
                        tmp = tmp.next;
                    }
                    cur = cur.next;
                }
                tmp.next = null;
                return virHead.next;

            }
        }
    }
}
```

看元素是不是已经在set里面有了，没有就把节点连接起来，否则就跳过

day10

一、选择题

1、数组常用的两种基本操作是 （ ）

数组对于涉及到随机访问的操作比较友好

A. 建立与删除 B. 删除与查找 C. 查找与修改 D. 插入与索引

2、线性表的逻辑顺序与物理顺序总是一致的 （ ）

A. 是 B. 不

3、定义了一维 int 型数组 a[10] 后，下面错误的引用是（ ）

A. a[0] = 1; B. a[0] = 5*2; C. a[10] = 2; D. a[1] = a[2] * a[0];

4、从一个长度为n的数组中删除第i个元素(1≤i≤n)时,需向前移动（）个元素 （ ）

A. n-i B. n-i+1 C. n-i-1 D. i

5、下列关于链表的描述中正确的是【多选】（ ）

A. 链表由头指针唯一确定，单链表可以用头指针的名字来命名

B. 线性链表的存储空间不一定连续，并且各元素的存储顺序是任意的

C. 链表的插入删除不需要移动元素，可以只改变指针

D. 链表可随机访问任一元素

二、编程题

1、从链表中删去总和值为零的连续节点

给你一个链表的头节点 head，请你编写代码，反复删去链表中由 总和 值为 0 的连续节点组成的序列，直到不存在这样的序列为止。删除完毕后，请你返回最终结果链表的头节点。OJ链接【难度：中等】

示例1:
输入：head = [1,2,-3,3,1]
输出：[3,1]
提示：答案 [1,2,1] 也是正确的。

示例2:
输入：head = [1,2,3,-3,4]
输出：[1,2,4]

两次遍历，其实关键在于一个点，就是如果存在和为0的序列，map中的key又是唯一的，那么一会出现后面有个key与前面某个重复，因为中间序列和为0嘛，所以就会把原先的value值更新掉，然后我们第二次遍历的时候，这个key值的value值就是后面的，它链接的时候就直接把中间那部分和为0的序列删掉了

```
class Solution {
    public ListNode removeZeroSumSublists(ListNode head) {
        ListNode virHead = new ListNode(0);
        virHead.next = head;
        HashMap<Integer,ListNode> map = new HashMap<>();
        //第一次遍历，把和与节点关系对应起来
        int sum = 0;
        for(ListNode tmp = virHead;tmp != null;tmp = tmp.next){
            sum += tmp.val;
            map.put(sum,tmp);
        }

        //第二次遍历，删除掉总和值为0的序列
        sum = 0;
        for(ListNode tmp = virHead;tmp != null;tmp = tmp.next){
            sum += tmp.val;
            tmp.next = map.get(sum).next;
        }
        return virHead.next;
    }
}
```

2、两数相加

给你两个 非空 的链表，表示两个非负的整数。它们每位数字都是按照 逆序 的方式存储的，并且每个节点只能存储 一位 数字。请你将两个数相加，并以相同形式返回一个表示和的链表。你可以假设除了数字 0 之外，这两个数都不会以 0 开头。OJ链接【难度：中等】

示例1:
输入：l1 = [2,4,3], l2 = [5,6,4]
输出：[7,0,8]
解释：342 + 465 = 807.

示例2:
输入：l1 = [0], l2 = [0]
输出：[0]

```
class Solution {
    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
    }
}
```

```
class Solution {
    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        ListNode virHead = new ListNode(-1);
        ListNode tmp = virHead;
        int carry = 0;//用来记录进位

        while(l1 != null || l2 != null){
            //看看是不是空的，如果是空，值就是0，不是空值就是节点值
            int num1 = (l1 == null)? 0 : l1.val;
            int num2 = (l2 == null)? 0 : l2.val;

            int sum = num1 + num2 + carry;
            //更新carry
            carry = sum/10;//先求进位
            sum = sum % 10;//如果sum>10值是不一样的

            ListNode node = new ListNode(sum);//把这个节点连起来
            tmp.next = node;
            tmp = tmp.next;

            if(l1 != null){
                l1 = l1.next;//不为空就往后走
            }
            if(l2 != null){
                l2 = l2.next;
            }
        }

        //这里走完了，但是可能最后一个和还进了一位，所以这个要单独加上
        if(carry == 1){
            tmp.next = new ListNode(carry);//把最后这个1补上
        }

        return virHead.next;
    }
}
```

day

一、

1、线性结构的是【多选】 (A)BC

A. 数组 B. 链式存储栈 C. 顺序存储栈 D. 顺序存储二叉树

2、有一个单向链表，头指针和尾指针分别为p，q，以下哪项操作的复杂度不受链表长度的影响【多选】ACD

A. 删除头部元素 B. 删除尾部元素 C. 头部元素之前插入一个元素 D. 尾部元素之后插入一个元素

3、在一个单链表中，p、q分别指向表中两个相邻的结点，且q所指结点是p所指结点的直接后继，现要删除q所指结点，可用的语句是 （ ）C

A. p=q.next B. p.next=q C. p.next=q.next D. q.next=NULL

4、在一个具有 n 个结点的有序单链表中插入一个新结点并仍然保持有序的时间复杂度是 （ ）B

只要找到位置插入就行，因为本身就是有序的

A. O（1） B. O（n） C. O（n2） D. O（nlog2n）

5、对链表进行插入和删除操作时不必移动链表中结点。 （ ）A

A. 正确 B. 错误

二、编程题

1、排序链表

给你链表的头结点 head ，请将其按 **升序** 排列并返回 **排序后的链表** 。OJ链接【难度：中等】

示例1:
输入：head = [4,2,1,3]
输出：[1,2,3,4]

示例2:
输入：head = [-1,5,3,4,0]
输出：[-1,0,3,4,5]

```
class Solution {
    public ListNode sortList(ListNode head) {

    }
}
```

```
class Solution {
    public ListNode sortList(ListNode head) {
        if(head == null){
            return null;
        }
        PriorityQueue<ListNode> priorityQueue = new
PriorityQueue<>(new Comparator<ListNode>(){//创建小根堆
            public int compare(ListNode o1,ListNode o2){
                return o1.val - o2.val;
            }
        });

        //把节点入堆
        ListNode cur = head;
        while(cur != null){
            priorityQueue.offer(cur);
            cur = cur.next;
        }

        ListNode virHead = new ListNode(-1);
        ListNode tmp = virHead;
        while(!priorityQueue.isEmpty()){
            tmp.next = priorityQueue.poll();
            tmp = tmp.next;
        }
        tmp.next = null;//最后这里一定要置为空，因为最后的这个节点next不一定是null
        return virHead.next;

    }
}
```

2、奇偶链表

给定一个单链表，把所有的奇数节点和偶数节点分别排在一起。请注意，这里的奇数节点和偶数节点指的是节点编号的奇偶性，而不是节点的值的奇偶性。

OJ链接【难度：中等】

示例1:
输入: 1->2->3->4->5->NULL
输出: 1->3->5->2->4->NULL

示例2:
输入: 2->1->3->5->6->4->7->NULL
输出: 2->3->6->7->1->5->4->NULL

```
class Solution {
    public ListNode oddEvenList(ListNode h
    }
}
```

```
class Solution {
    public ListNode oddEvenList(ListNode head) {
        if(head == null){
            return null;
        }
        ListNode oddHead = head;//奇数链表的头节点，当然也是最终合并后的头节点
        ListNode evenHead = head.next;//偶数链表的头节点

        ListNode odd = oddHead;//遍历奇数链表
        ListNode even = evenHead;//遍历偶数链表

        while(even != null && even.next != null){//先连接奇数索引节点再是偶数索引节点，偶数汕尾
            odd.next = even.next;
            odd = odd.next;
            even.next = odd.next;
            even = even.next;
        }
        //现在把两个部分链接起来
        odd.next = evenHead;
        return head;
    }
}
```

day12

一、选择题

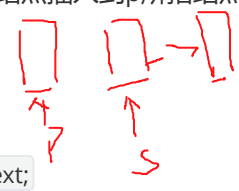
1、在单链表中，要将s所指结点插入到p所指结点之后，其语句应为（）

A. s.next=p+1; p.next=s;

B. p.next=s; s.next=p.next

C. s.next=p.next; p.next=s.next;

D. s.next=p.next; p.next=s;



2、下述有关栈和队列的区别，说法错误的是（）

A. 栈是限定只能在表的一端进行插入和删除操作。

B. 队列是限定只能在表的一端进行插入和在另一端进行删除操作。

C. 栈和队列都属于线性表

D. 栈的插入操作时间复杂度都是o(1)，队列的插入操作时间复杂度是o(n)

3、输入序列是ABC，输出序列变为BCA时，经过的栈操作为（）

A. push,push,push,pop,pop,pop

B. push,push,pop,push,pop,pop

C. push,pop,push,push,pop,pop

D. push,push,pop,pop,push,pop

4、栈的特点是先入先出，这种说法（ ）

A. 正确 B. 错误

5、一个队列的入队序列为 1234，则队列可能的输出序列是（ ）

A. 4321 B. 1234 C. 1432 D. 3241

二、编程题

1、合并链表

将两个升序链表合并为一个新的 **升序** 链表并返回。新链表是通过拼接给定的两个链表的所有节点组成的。OJ链接
【难度：简单】

示例1:
输入：l1 = [1,2,4], l2 = [1,3,4]
输出：[1,1,2,3,4,4]

示例2:
输入：l1 = [], l2 = [0]
输出：[0]

```
class Solution {
    public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
```

```
class Solution {
    public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
        ListNode newNode = new ListNode(-1);//虚拟节点
        ListNode tmp = newNode;
        while(list1 != null && list2 != null){
            if(list1.val < list2.val){
                tmp.next = list1;
                tmp = list1;
                list1 = list1.next;
            }else{
                tmp.next = list2;
                tmp = list2;
                list2 = list2.next;
            }
        }
        if(list1 == null){//也包含初始条件下某一个就为null的情况
            tmp.next = list2;
        }
        if(list2 == null){
            tmp.next = list1;
        }
        return newNode.next;
    }
}
```

```
    }
}
```

2、删除链表中重复的结点

在一个排序的链表中，存在重复的结点，请删除该链表中重复的结点，重复的结点不保留，返回链表头指针。 例如，链表 1->2->3->3->4->4->5 处理后为 1->2->5 OJ链接【难度：中等】

示例1:
输入：{1,2,3,3,4,4,5}
返回值：{1,2,5}
示例2:
输入：{1,1,1,8}
输出：{8}

```
public class Solution {
    public ListNode delete
```

```
import java.util.*;

public class Solution {
    public ListNode deleteDuplication(ListNode pHead) {
        if(pHead == null){
            return null;
        }
        Map<Integer,Integer> map = new HashMap<>();
        ListNode node = pHead;
        //统计下每个值出现的次数
        while(node != null){
            if(map.get(node.val) == null){
                map.put(node.val,1);
            }else{
                int oldVal = map.get(node.val);
                map.put(node.val,oldVal+1);
            }
            node = node.next;
        }

        ListNode virHead = new ListNode(-1);
        ListNode tmp = virHead;
        ListNode cur = pHead;
        while(cur != null){
            if(map.get(cur.val) == 1){//只出现一次的就连起来
                tmp.next = cur;
                tmp = tmp.next;
            }
            cur = cur.next;
        }
        tmp.next = null;
        return virHead.next;
    }
}
```

```
    }
}
```

day13

一、选择题

1、单链表实现的栈，栈顶指针为Top(只是一个指针)，入栈一个P节点时，其操作步骤为（ ）

A. Top.next=p;

入栈，可能是尾插入栈，也可能是头插入栈

B. p.next=Top.next;Top.next=p;

C. p.next=Top;Top=p.next;

D. p.next=Top;Top=Top.next;

2、下列与队列结构有关联的是 （）

A. 函数的递归调用

B. 数组元素的引用

C. 多重循环的执行

D. 先到先服务的作业调度

3、以下 （）不是队列的基本运算

A. 从队尾插入一个新元素 B. 从队列中删除第i个元素 C. 判断一个队列是否为空 D. 读取队头元素的值

4、字符 A 、 B 、 C 依次进入一个栈，按出栈的先后顺序组成不同的字符串，至多可以组成 ( ) 个不同的字符串（）

A. 14 B. 5 C. 6 D. 8

5、判断一个循环队列Q（最多元素为m）为满队列的条件是（）

A. rear+1=front B. (rear+1)%m=front C. rear=front D. (rear+1)%m=null

二、编程题

1、反转链表

给你单链表的头节点 head ，请你反转链表，并返回反转后的链表。OJ链接【难度：简单】

示例1:
输入：head = [1,2,3,4,5]
输出：[5,4,3,2,1]
示例2:
输入：head = [1,2]
输出：[2,1]

【代码实现】：

```
class Solution {

    public ListNode reverseList(ListNode head) {
```

```
class Solution {
    public ListNode reverseList(ListNode head) {
        if(head == null){
            return null;
        }

        ListNode prev = head;
        ListNode cur = head.next;
        head.next = null;

        while(cur != null){
            ListNode curNext = cur.next;
            cur.next = prev;
            prev = cur;
            cur = curNext;
        }
        head = prev;
        return head;
    }
}
```

```
    }
}
```

2、找到链表环的入口

给定一个链表，返回链表开始入环的第一个节点。 如果链表无环，则返回 null。如果链表中有某个节点，可以通过连续跟踪 next 指针再次到达，则链表中存在环。 为了表示给定链表中的环，评测系统内部使用整数 pos 来表示链表尾连接到链表中的位置（索引从 0 开始）。如果 pos 是 -1，则在该链表中没有环。注意：pos 不作为参数进行传递，仅仅是为了标识链表的实际情况。不允许修改链表。

OJ链接【难度：中等】

示例1:
输入：head = [3,2,0,-4], pos = 1
输出：返回索引为 1 的链表节点
解释：链表中有一个环，其尾部连接到第二个节点。

示例2:
输入：head = [1,2], pos = 0
输出：返回索引为 0 的链表节点
解释：链表中有一个环，其尾部连接到第一个节点。

示例2:
输入：head = [1], pos = -1
输出：返回 null
解释：链表中没有环。

```
public class Solution {
    public ListNode detectCycle(ListNode head) {
```

```
public class Solution {
    public ListNode detectCycle(ListNode head) {
        if(head == null){
            return null;
        }
        ListNode fast = head;
        ListNode slow = head;
        while(fast != null && fast.next != null){//奇数个节点或者偶数个节点没环的终止条件
            fast = fast.next.next;
            slow = slow.next;
            if(fast == slow){
                break;
            }
        }
        if(fast == null || fast.next == null){
            //说明没有环
            return null;
        }
        //上面没有return掉，就说明有环，并且fast与slow相遇了
        fast = head;
        while(fast != slow){
            fast = fast.next;
            slow = slow.next;
        }
        return fast;
    }
}
```

```
    }
}
```

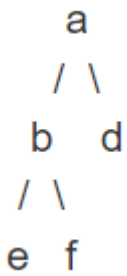
day14

一、选择题

1、递归算法一般需要利用哪种数据结构实现（ ）

A. 数组 B. 链表 C. 队列 D. 栈

2、以下二叉树前序遍历的顺序是（ ）



A. abefd B. abdef C. ebfad D. efbda

3、已知一棵完全二叉树的第6层（设根为第1层）有8个叶结点，则该完全二叉树的结点个数最多是（ ）

A. 39 B. 52 C. 111 D. 119

4、具有 12 个结点的完全二叉树有（ ）

A. 5个叶子结点 B. 5个度为2的结点 C. 7个分支结点 D. 2个度为1的结点

5、深度为 7 的二叉树共有 127 个结点，则下列说法中错误的是（ ）

A. 该二叉树有一个度为1的结点

B. 该二叉树是满二叉树

C. 该二叉树是完全二叉树

D. 该二叉树有64个叶子结点

二、编程题

1、用栈操作构建数组

给你一个目标数组 target 和一个整数 n。每次迭代，需要从 list = {1,2,3..., n} 中依序读取一个数字。

请使用下述操作来构建目标数组 target ：

不需要用到栈，反而多此一举

- Push：从 list 中读取一个新元素， 并将其推入数组中。
- Pop：删除数组中的最后一个元素。
- 如果目标数组构建完成，就停止读取更多元素。

题目数据保证目标数组严格递增，并且只包含 1 到 n 之间的数字。请返回构建目标数组所用的操作序列。题目数据保证答案是唯一的。OJ链接【难度：简单】

示例1:
输入：target = [1,3], n = 3
输出：["Push","Push","Pop","Push"]
解释：
读取 1 并自动推入数组 -> [1]
读取 2 并自动推入数组，然后删除它 -> [1]
读取 3 并自动推入数组 -> [1,3]

示例2:
输入：target = [1,2,3], n = 3
输出：["Push","Push","Push"]

【代码实现】：

```
class Solution {
    public List<String> buildArray(int[] target, int n) {
class Solution {
    public List<String> buildArray(int[] target, int n) {
        List<String> ret = new ArrayList<>();
        int index = 0;//用来记录target数组的下标
        int i = 1;//用来产生1~n的数字
        while(i <= n && index < target.length){
            if(i == target[index]){
                //如果匹配上，就push
                ret.add("Push");
                index++;//这是实打实的放了一个元素所以index++
            }else{
                //不匹配，按照题目意思就是先push再pop
                ret.add("Push");
                ret.add("Pop");
            }
            i++;
        }
        return ret;
    }
}

    }
}
```

2、下一个更大元素

给你两个 没有重复元素 的数组 nums1 和 nums2 ， 其中nums1 是 nums2 的子集。请你找出 nums1 中每个元素在 nums2 中的下一个比其大的值。nums1 中数字 x 的下一个更大元素是指 x 在 nums2 中对应位置的右边的第一个比 x 大的元素。如果不存在，对应位置输出 -1 。

OJ链接【难度：简单】

示例1:
输入: nums1 = [4,1,2], nums2 = [1,3,4,2].
输出: [-1,3,-1]
解释:
对于 num1 中的数字 4 ，你无法在第二个数组中找到下一个更大的数字，因此输出 -1 。
对于 num1 中的数字 1 ，第二个数组中数字1右边的下一个较大数字是 3 。
对于 num1 中的数字 2 ，第二个数组中没有下一个更大的数字，因此输出 -1 。

```java
class Solution {
    public int[] nextGreaterElement(int[] nums1, int[] nums2) {
        //暴力解法
        // int[] ret = new int[nums1.length];
        // int index = 0;
        // for(int i = 0;i < nums1.length;i++){
        //     int j = 0;
        //     for(j = 0;j < nums2.length;j++){
        //         if(nums2[j] == nums1[i]){
        //             break;
        //         }
        //     }
        //     int flg = 0;
        //     for(j = j+1;j < nums2.length;j++){
        //         if(nums2[j] > nums1[i]){
        //             ret[index++] = nums2[j];
        //             flg = 1;
        //             break;
        //         }
        //     }
        //     if(flg == 0){
        //         ret[index++] = -1;
        //     }
        // }
        // return ret;

        //哈希表加单调栈
        Stack<Integer> stack = new Stack<>();
        Map<Integer,Integer> map = new HashMap<>();
        //倒序遍历nums2
        for(int i = nums2.length - 1;i >= 0;i--){
            while(!stack.empty() && nums2[i] >= stack.peek()){
                stack.pop();//连续出栈
            }
            //现在看栈顶元素
            if(stack.empty()){
                map.put(nums2[i],-1);//如果栈是空的，说明这个元素后面没有比他大的
            }else{
                map.put(nums2[i],stack.peek());//不是空那就是栈顶元素
            }
            //把该元素放入栈中
            stack.push(nums2[i]);
        }

        int[] ret = new int[nums1.length];
        for(int i = 0;i < nums1.length;i++){
            ret[i] = map.get(nums1[i]);
        }
        return ret;
    }
}
```

1、一棵有15个节点的完全二叉树和一棵同样有15个节点的普通二叉树，叶子节点的个数最多会差多少个？（）

A. 3 B. 5 C. 7 D. 9

2、在一棵二叉树中，假定每个结点只有左孩子，没有右孩子，对它分别进行前序遍历和后序遍历，则具有相同的遍历结果。（）

A. 正确 B. 错误

3、一棵满二叉树同时又是一棵平衡树。这种说法（）

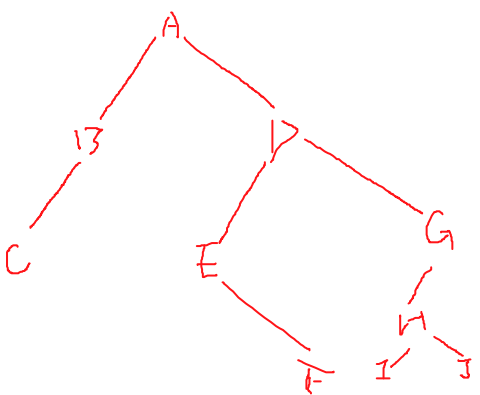
A. 正确 B. 错误

4、已知一棵二叉树的先序和中序遍历序列如下：先序：A、B、C、D、E、F、G、H、I、J中序：C、B、A、E、F、D、I、H、J、G其后序遍历序列为（）

A. C、B、D、E、A、G、I、H、J、F

B. C、B、D、A、E、G、I、H、J、F

C. C、E、D、B、I、J、H、G、F、A



D. C、E、D、B、I、H、J、G、F、A

E. C、B、F、E、I、J、H、G、D、A

5、完全二叉树中的叶子结点只可能在最后两层中出现 ()

A. 正确 B. 错误

二、编程题

1、最小栈

设计一个支持 push ， pop ， top 操作，并能在常数时间内检索到最小元素的栈。

- push(x) —— 将元素 x 推入栈中。
- pop() —— 删除栈顶的元素。
- top() —— 获取栈顶元素。
- getMin() —— 检索栈中的最小元素。

OJ链接【难度：简单】

输入：
["MinStack","push","push","push","getMin","pop","top","get
[[],[-2],[0],[-3],[],[],[],[]]
输出：
[null,null,null,null,-3,null,0,-2]
解释：
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin(); --> 返回 -3.
minStack.pop();
minStack.top(); --> 返回 0.
minStack.getMin(); --> 返回 -2.

```
class MinStack {

    public MinStack() {
    }

    public void push(int val) {

    }

    public void pop() {



    }
```

```
class MinStack {

    public Stack<Integer> stack;
    public Stack<Integer> minStack;
    public MinStack() {
        stack = new Stack<>();
        minStack = new Stack<>();
    }

    public void push(int val) {
        if(stack.empty() && minStack.empty()){
            //两个都为空，两边都压入元素
            stack.push(val);
            minStack.push(val);
            return;
        }

        stack.push(val);
        if(val <= minStack.peek()){
            minStack.push(val);
        }
    }

    public void pop() {
        if(!stack.empty()){
            int top = stack.pop();
            if(top == minStack.peek() && !minStack.empty
()){
                minStack.pop();//相等的话minStack中也要
pop掉
            }
        }

    }

    public int top() {
        if(stack.empty()){
            return -1;
        }
        return stack.peek();
    }

    public int getMin() {
        if(minStack.empty()){
            return -1;
        }
        return minStack.peek();
    }
}
```

```
    public int top() {

    }

    public int getMin() {

    }
}
```

2、有效的括号

给定一个只包括 '('，')'，'{'，'}'，'['，']' 的字符串 s ，判断字符串是否有效。有效字符串需满足：

1. 左括号必须用相同类型的右括号闭合。
2. 左括号必须以正确的顺序闭合。

OJ链接【难度：简单】

```
示例1:
输入：s = "()"
输出：true

示例2:
输入：s = "()[]{}"
输出：true

示例3:
输入：s = "(]"
输出：false
```

```
class Solution {
    public boolean isValid(String s) {
```

```
class Solution {
    public boolean isValid(String s) {
        Stack<Character> stack = new Stack<>();

        for(int i = 0;i < s.length();i++){
            char ch = s.charAt(i);
            if(ch == '(' || ch == '[' || ch == '{'){
                stack.push(ch);//左括号就入栈
            }else{//现在是右括号
                //则可能有右括号多的情况
                if(stack.empty()){
                    return false;//栈空，没有和这个右括号匹配的
                }else{
                    char top = stack.peek();//现在看一下栈顶元素
                    if(top == '(' && ch == ')' || top == '[' &&
ch == ']' || top == '{' && ch == '}'){
                        //匹配上了，就把栈顶元素弹出
                        stack.pop();
                    }else{
                        return false;
                    }
                }

            }
        }

        //i遍历完了，就看栈空不空
        if(stack.empty()){
            //如果栈空，就说明全部匹配掉了
            return true;
        }else{
            //如果栈不是空,就说明栈里面还有左括号没有匹配弹出掉
            return false;
        }

    }
}
```

```
    }
}
```

day16

一、选择题

1、二叉树是非线性数据结构,所以（ ）

A. 它不能用顺序存储结构存储;

B. 它不能用链式存储结构存储;

C. 顺序存储结构和链式存储结构都能存储;

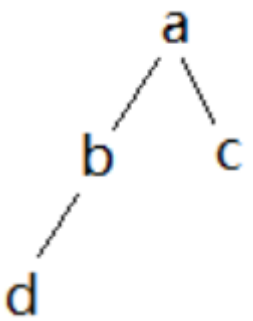
D. 顺序存储结构和链式存储结构都不能使用

2、执行（ ）操作时，需要使用队列作为辅助存储空间。

A. 查找哈希（hash）表 B. 层序遍历 C. 先序（根）遍历二叉树 D. 深度优先搜索图

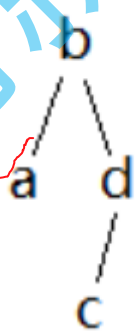
3、中序遍历为abcd的二叉树可能是下面的哪棵【多选】（ ）

A.

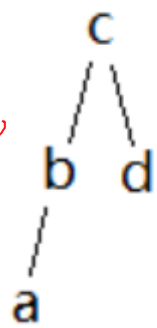


B.

C.



D.



4、关于堆数据结构，下面描述中不恰当的一项是：（ ）

A. 用堆可以实现优先队列（priority_queue）

B. 使用堆可以实现排序算法，复杂度为NlogN

C. 堆是一棵完全二叉树

D. 在大顶堆的二叉树中，第N层中的所有元素比第N+1层中的所有元素都要大

5、下列关键字序列为堆的是（ ）

A. 100，60，70，50，32，65

B. 60，70，65，50，32，100

C. 65，100，70，32，50，60

D. `70, 65, 100, 32, 50, 60`

二、编程题

1、用队列实现栈

请你仅使用两个队列实现一
empty）。实现 MyStack 类

- void push(int x) 将元
- int pop() 移除并返回
- int top() 返回栈顶元素
- boolean empty() 如果

OJ链接【难度：简单】

示例:
输入:
["MyStack", "push", "push",
[[], [1], [2], [], [], []]
输出:
[null, null, null, 2, 2, false]
解释:
MyStack myStack = new My
myStack.push(1);
myStack.push(2);
myStack.top(); // 返回 2
myStack.pop(); // 返回 2
myStack.empty(); // 返回 Fa

```
class MyStack {

    public MyStack() {


    }

    public void push(int x) {




    }

    public int pop() {
```

```
class MyStack {

    //用两个队列模拟栈
    public Queue<Integer> qu1;
    public Queue<Integer> qu2;

    public MyStack() {
        qu1 = new LinkedList<Integer>();
        qu2 = new LinkedList<Integer>();
    }

    public void push(int x) {//压栈操作
        if(qu1.isEmpty() && qu2.isEmpty()){//两个都为空的时候随便入一个队
            qu1.offer(x);
            return;//入了就直接return掉
        }
        //否则就是入队到不为空的队列
        if(qu1.isEmpty()){
            qu2.offer(x);
        }else{
            qu1.offer(x);
        }

    }

    public int pop() {
        if(empty()){
            //说明栈空
            return -1;//不要写抛异常，做题就返回-1
        }
        int ret = 0;
        if(qu1.isEmpty()){
            int queueSize = qu2.size();//先保存下来，因为动了之后size会变
化，影响for循环
            for(int i = 0;i < queueSize - 1;i++ ){
                qu1.offer(qu2.poll());
            }
            ret = qu2.poll();
        }else{
            int queueSize = qu1.size();
            for(int i = 0;i <queueSize - 1;i++ ){
                qu2.offer(qu1.poll());
            }
            ret = qu1.poll();
        }
        return ret;
    }

    public int top() {//peek一下
        if(empty()){
            //说明栈空
            return -1;//不要写抛异常，做题就返回-1
        }
        int ret = 0;
        if(qu1.isEmpty()){
            int queueSize = qu2.size();//先保存下来，因为动了之后size会变
化，影响for循环
            for(int i = 0;i < queueSize ;i++ ){
                ret = qu2.poll();//把qu2中的元素按照之前pop的逻辑全部弹到qu1
中，最后一次就是栈顶
                qu1.offer(ret);
            }
        }else{
            int queueSize = qu1.size();
            for(int i = 0;i < queueSize ;i++ ){
                ret = qu1.poll();
                qu2.offer(ret);
            }
        }
        return ret;
    }

    public boolean empty() {
        return qu1.isEmpty() && qu2.isEmpty();
    }
}
```

```
    }

    public int top() {


    }

    public boolean empty() {


    }
}
```

2、用栈实现队列

请你仅使用两个栈实现先入先出队列。队列应当支持一般队列支持的所有操作（push、pop、peek、empty）：
实现 MyQueue 类：

- void push(int x) 将元素 x 推到队列的末尾
- int pop() 从队列的开头移除并返
- int peek() 返回队列开头的元素
- boolean empty() 如果队列为空

OJ链接【难度：简单】

```
示例:
输入:
["MyQueue", "push", "push", "peek",
[[], [1], [2], [], [], []]
输出:
[null, null, null, 1, 1, false]
解释:
MyQueue myQueue = new MyQueue
myQueue.push(1); // queue is: [1]
myQueue.push(2); // queue is: [1, 2]
myQueue.peek(); // return 1
myQueue.pop(); // return 1, queue is
myQueue.empty(); // return false
```

```
class MyQueue {

    public MyQueue() {



    }

    public void push(int x) {
```

```
class MyQueue {

    public Stack<Integer> pushStack;//入队列的元素全部压入这个栈里面
    public Stack<Integer> popStack;//出队列都是出的这个栈的栈顶元素
    public MyQueue() {
        pushStack = new Stack<>();
        popStack = new Stack<>();
    }

    public void push(int x) {//入队 offer
        pushStack.push(x);
    }

    public int pop() {//出队 poll
        if(empty()){
            return -1;//如果队列是空，就不能出队了
        }

        if(popStack.empty()){
            while(!pushStack.empty()){//只要pushStack不是空，就把元素
转到popStack
                popStack.push(pushStack.pop());
            }
            return popStack.pop();
        }else{
            return popStack.pop();
        }

    }

    public int peek() {//查看队头元素
        if(empty()){
            return -1;
        }
        if(popStack.empty()){
            while(!pushStack.empty()){//只要pushStack不是空，就把元素
转到popStack
                popStack.push(pushStack.pop());
            }
            return popStack.peek();
        }else{
            return popStack.peek();
        }
    }

    public boolean empty() {//判断队列是否为空
        return pushStack.empty() && popStack.empty();//两个栈都为空，
队列就是空
    }
}
```

```
    }
    public int pop() {




    }

    public int peek() {




    }

    public boolean empty() {




    }
}
```