Day 1

1.如果 `int x=20, y=5`，则语句 `System.out.println(x+y + ""+(x+y)+y);` 的输出结果是（ D ）。

A. `2530`　　B. `55`　　C. `2052055`　　D. `25255`

解：

先计算 () 里的：x + y = 25.

从左向右计算：`x + y` 为 25，遇到字符串将数字转为字符串，然后将后边的续接到前边：25255. ♢

2.在 Java 中，在同一包内，类 Cat 里面有个公共方法 sleep()，该方法前有 static 修饰，则可以直接用 Cat.sleep()。（ A ）

A. 正确　　B. 错误

Tips

用 `static` 修饰的为类方法，类方法可由类名直接调用。

3.给定以下方法声明，调用执行 `mystery(1234)` 的输出结果是（ B ）。

```
1 //precondition: x >= 0
2 public void mystery (int x) {
3     System.out.print(x % 10);
4     if ((x / 10) != 0){
5         mystery(x / 10);
6     }
7     System.out.print(x % 10);
8 }
```

A. `1441`　　B. `43211234`　　C. `3443`　　D. `12344321`

解：

x%10 = 1234 % 10 = 4 ♢

Tips

该方法的第 3∼6 行的功能是实现逆序打印。

4.实现函数 `ToLowerCase()`，该函数接收一个字符串参数 `str`，并将该字符串中的大写字母转换成小写字母，之后返回新的字符串。OJ 链接-转换成小写字母

转换成小写字母

```
1 class Solution {
2     public String toLowerCase(String str) {
3         return str.toLowerCase();
4     }
5 }
```

Tips

转大写：`str.toUpperCase()`

转小写：`str.toLowerCase()`

5.给定一个数组，将数组中的元素向右移动 k 个位置，其中 k 是非负数。OJ 链接-旋转数组

旋转数组

```java
class Solution {
    public void rotate(int[] nums, int k) {
        k %= nums.length;
        reverse(nums,0,nums.length-1);
        reverse(nums,0,k-1);
        reverse(nums,k,nums.length-1);

    }
    public void reverse(int[] nums,int lo,int hi){
        while(lo < hi){
            int tmp = nums[lo];
            nums[lo] = nums[hi];
            nums[hi] = tmp;
            lo++;
            hi--;
        }
    }
}
```

Tips

在 C 语言做过这个题。

1. 要先把旋转的位置除数组的长度取余，因为向右移动数组长度个位置（或其倍数）等于没有移动。
2. 把整个数组旋转得到新的数组。
3. 依次旋转数组的前 k 个元素（ 0到k-1 ），和后边剩余的元素。

Day 2

1.如果类的方法没有返回值，该方法的返回类型是（ A ）。

A. `void` B. `null` C. `abstract` D. `default`

2. `Java Application(Java应用程序)` 源程序文件编译后的字节码文件的拓展名是（ B ）。

A. `java` B. `class` C. `exe` D. `jar`

Tips

Java 源文件的拓展名：`java`

Java 源文件编译后拓展名：`class`

3.关于继承的描述正确的是（ C ）。

A. 函数的出口应该尽可能少，最好只有一个出口

B. 为了防止程序中的内存泄漏，应该不使用动态内存分配

C. 在函数实现中应该少使用全局变量

D. 函数的功能应该单一

Tips

A. 函数应为每个逻辑分支提供出口。

B. 动态内存分配无法避免，应该及时回收空间和释放指针避免动态内存泄露。

C. 为保证封装性，尽量少用全局变量。

D. 函数可以根据不同的输入来实现相应的功能（重载）。

4.给定一个数组 `nums` 和一个值 `val`，你需要原地移除所有数值等于 `val` 的元素，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-移除元素

Tips

1. 边界情况：数组为空时，要返回 0。

分析：不考虑顺序，返回的为不含 `val` 的前边一段，思路：把 `val` 换到后边去。

2. 左指针 `lo` 指向数组的左边，找 `val`，如果没找到 `val`，就把这个指针向右移动；

3. 右指针 `hi` 指向数组的右边，`val` 就应该位于数组后边，所以如果值是 `val`，指针就左移。

4. 经过 2、3 步后，左指针指的是 `val`，右指针指的是非 `val`，将这俩指针对应元素交换。

5. 左右指针重合时，数组就遍历完了。

6. 左指针的左边一定是不含 `val` 的，其左边有 `lo` 个元素，如果此时的左指针对应元素的值时 `val`，那就返回 `lo`，如果左指针对应元素的值不是 `lo`，就返回 `lo + 1`。

移除元素

```
class Solution {
    public int removeElement(int[] nums, int val) {
        if(nums.length == 0){
            return 0;
        }
        int lo = 0;
        int hi = nums.length - 1;
        int len = 0;
        while(lo < hi){
            if(nums[lo] != val && lo < hi){
                lo++;
```

```
            }
            if(nums[hi] == val && lo < hi){
                hi--;
            }
            if(lo < hi){
                int tmp = nums[lo];
                nums[lo] = nums[hi];
                nums[hi] = tmp;
            }
        }
        if(nums[lo] == val){
            return lo;
        }
        return lo + 1;
    }
}
```

5.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，你可以假设数组中无重复元素。OJ 链接-搜索插入位置

Tips

有序数组，采用二分查找。

1. 找到元素：返回元素的下标。
2. 如果数组中没有该元素。退出循环的前一次左指针、右指针、中间指针指向同一元素，若该元素小于目标值，即 `nums[mid] < target`，左指针指向中间指针的下一个元素，此时左指针在右指针的右边，退出循环。同时左指针指向的位置也是目标元素要插入的位置。
3. 若该元素大于目标值，即 `target < nums[mid]`，右指针指向中间指针的前一个元素，右指针在左指针的左边，退出循环。左指针的左侧元素必定小于目标值，而左指针指向的元素大于目标值，目标值应该插入到左指针指向的位置处。
4. 综上，找到目标值就返回该目标值对应的下标，没找到该目标值就返回左指针指向的位置。

搜索插入位置

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

```
18            }
19            return lo;
20        }
21    }
```

Day 3

1.以下代码的循环次数是（ D ）。

```
public class Test {
    public static void main(String[] args) {
        int i = 7;
        do {
            System.out.println(--i);
            --i;
        } while (i != 0);
        System.out.println(i);
    }
}
```

A. 0　　B. 1　　C. 7　　D. 无限次

解：

i 的初始值为奇数，循环中每次减 2，退出条件为不等于 0，奇数减 2 永远不会为 0，即死循环。 ♢

2.下列选项中属于面向对象设计方法的主要特征是（ A ）。

A. 继承　　B. 自顶向下　　C. 模块化　　D. 逐步求精

Tips

面向对象最主要的几个特征：封装、继承、多态、抽象等。

3.下面的方法，当输入为 2 的时候返回值是多少（ D ）。

```
public static int getValue(int i) {
    int result = 0;
    switch (i) {
        case 1:
            result = result + i;
        case 2:
            result = result + i * 2;
        case 3:
            result = result + i * 3;
    }
    return result;
}
```

A. 0　　B. 2　　C. 4　　D. 10

解：

case 后没有 break，会继续执行后面的 case，所以结果为：$i \times 2 + i \times 3 = i \times 5 = 2 \times 5 = 10$ ♢

4.赎金信。OJ 链接-赎金信

给定一个赎金信（ransom）字符串和一个杂志（magazine）字符串，判断第一个字符串 ransom 能不能由第二个字符串 magazines 里面的字符构成。如果可以构成，返回 true；否则返回 false。

(题目说明：为了不暴露赎金信字迹，要从杂志上搜索各个需要的字母，组成单词来表达意思。杂志字符串中的每个字符只能在赎金信字符串中使用一次。)

假设两个字符串均只含有小写字母。

Tips

1. 杂志字符串长度小于赎金信字符串长度，一定不能构成赎金信。
2. 因为只有小写字母，所以分别统计各字符的个数，将两字符串各字符的个数放入数组中。
3. 分别比较两字符串对应字符的个数，赎金信相应字符的个数超过杂志该字符的个数，则一定不能表示，否则就能表示。

赎金信

```
class Solution {
    public boolean canConstruct(String ransomNote, String magazine) {
        if(ransomNote.length()>magazine.length()){
            return false;
        }
        char[] ch1 = ransomNote.toCharArray();
        char[] ch2 = magazine.toCharArray();
        int[] rans = new int[26];
        int[] maga = new int[26];
        for(char c1 : ch1){
            rans[c1 - 'a']++;
        }
        for(char c2 : ch2){
            maga[c2 - 'a']++;
        }
        for(int i = 0; i < 26; i++){
            if(rans[i] > maga[i]){
                return false;
            }
        }
        return true;
    }
}
```

5.判断一个整数是否是回文数。回文数是指正序（从左到右）和倒叙（从右向左）读都是一样的整数。OJ链接-回文数

方法 1：

Tips

1. 把数字转为字符串；
2. 把字符串转为字符串数组；
3. 使用左右指针指向数组的前边和后边，两个指针指向的字符相等指针就同时向中间移动，不相等就退出。
4. 左指针小于右指针，不是回文数，否则就是。

```
class Solution {
    public boolean isPalindrome(int x) {
      String str = "" + x;
      char[] ch1 = str.toCharArray();
      int lo = 0;
```

```
        int hi = ch1.length - 1;
        while(lo < hi){
            if(ch1[lo] != ch1 [hi]){
                break;
            }
            lo++;
            hi--;
        }
        if(lo < hi){
            return false;
        }
        return true;
    }
}
```

方法 2：

Tips

1. 根据题意，所有负数都不是回文数。
2. 小于 10 的自然数都是回文数。
3. 能被 10 整除的都不是回文数。（能被 10 整除，最后一位是 0，而第一位不会是 0，所以不是回文数）
4. 其他的数字 `x`，每次从后边取一位 `x%10`，放入新的数 `num` 中：`num = num * 10 + x%10`，
5. 每取完一位，就将最后一位拿掉：`x = x / 10`。

判断 `x` 是不是小于 `num`，如果小于，说明已经过半了，即：

6. 如果 `x` 是奇数位，例如 5 位，取了 2 位后，`x` 还有 3 位，此时 `num` 是 2 位，`num < x`。再取一位，`x` 变成 2 位，`num` 变成 3 位，两位数小于三位数，此时说明取的位数超过一半，就不用取了，如果 `x` 和 `num` 的前两位 `num` `10` 相等，说明是回文数，否则不是。
7. 如果 `x` 是偶数位，例如 4 位，取了 2 位后，`num` 是 2 位，`x` 剩余 2 位，如果 `num` 等于 `x`，说明是回文数。

```
class Solution {
    public boolean isPalindrome(int x) {
        if(x < 0){
            return false;
        }
        if(0 <= x && x < 10){
            return true;
        }
        if(x % 10 == 0){
            return false;
        }
        int num = 0;
        while(x > num){
            num = num * 10 + x % 10;
            x /= 10;
        }
        return x == num || x == num / 10;
    }
}
```

Day 4

1.下面有关 `java final` 的基本规则，描述错误的是（　B　）。

A. `final` 修饰的类不能被继承

B. `final` 修饰的成员变量只允许赋值一次，且只能在类方法赋值

C. `final` 修饰的局部变量即为常量，只能赋值一次

D. `final` 修饰的方法不允许被子类覆盖

Tips

`final` 修饰的类不能被继承，修饰的变量只允许被赋值一次且后边无法改变该变量的值，对于成员变量可以进行就地初始化，也可以进行代码块初始化，也可以使用构造方法初始化，但这三种方法只能使用一个，`final` 修饰的方法不支持重写，但是可以进行重载。

2.选项中哪一行代码可以替换 `//add code here` 而不产生编译错误（　　）。

A. `public abstract void method(int a);`

B. `consInt = constInt + 5;`

C. `public int method;`

D. `public abstract void anotherMethod(){}`

解：

没有代码，没法做。 ♢

3.第三行将输出什么（　B　）。

```
1  public class SwitchTest{
2      public static void main(String[] args) {
3          System.out.println("value=" + switchit(4));
4      }
5      public static int switchit(int x) {
6          int j = 1;
7          switch (x) {
8              case 1:j++;
9              case 2:j++;
10             case 3:j++;
11             case 4:j++;
12             case 5:j++;
13             default:j++;
14         }
15         return j+x;
16     }
17 }
```

A. `value=6;`　　B. `value=8;`　　C. `value=3;`　　D. `value=5;`

解：

没有遇到 `break` 会接着往后执行。 ♢

4.给定一个仅包含大小写字母和空格 `' '` 的字符串，返回其最后一个单词的长度。如果不存在最后一个单词，请返回 0。OJ 链接-最后一个单词的长度

Tips

1. 字符串长度为 0 或者字符串引用指向为空，那最后一个单词长度为 0.

2. 一般情况：

用右指针 hi 从字符串的最后一位开始遍历，如果是空格，指针就向前移动，如果遇到了字母就停下来，记录该位置。当 hi 小于 0 时，说明字符串遍历完毕也没遇到字母，即不存在最后一个单词，返回 0.

如果 hi 不小于 0，记录 hi 当前的位置，使用一个新的指针 lo 从该位置向前遍历，直到遇到空格（或遍历字符串结束即 lo 小于 0）停下来，lo 和 hi 之间的字符个数就是最后一个单词的长度。

最后一个单词的长度

```java
class Solution {
    public int lengthOfLastWord(String s) {
        //char[] ch = s.toCharArray();
        int hi = s.length() - 1;
        if(s.length() == 0 || s == null){
            return 0;
        }
        while(hi >= 0 && s.charAt(hi) == ' '){
            hi--;
        }
        if(hi < 0){
            return 0;
        }
        int lo = hi;
        while(lo >= 0 && s.charAt(lo) != ' '){
            lo--;
        }
        return hi - lo;
    }
}
```

5.给你两个有序整数数组 nums1 和 nums2，请你将 nums2 合并到 nums1 中，使 nums1 成为一个有序数组。

OJ 链接-合并两个有序数组

Tips

方法 1：将两个数组合并，然后重新排序。

方法 2：依据题目描述，此题为归并排序算法的归并部分。

因为最终的结果要存放在 nums1 中，为了避免 nums1 中的数据被覆盖，使用一个数组 arr 来存放 nums1 的数据。

步骤：

1. 使用两个指针分别指向数组 arr 和 nums2 的开头，比较其对应元素的大小，将较小的那个元素放入到 nums1 中，同时将 nums1 和拿出元素的那个数组的指针向后移动一位。

2. 其中一个数组遍历完后，将没有遍历完的那个数组的剩余部分放在 nums1 后边即可。

合并两个有序数组

```java
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        int[] arr = new int[m];
        for(int i = 0; i < m; i++){
            arr[i] = nums1[i];
        }
```

```
        int a1 = 0;
        int a2 = 0;
        int i = 0;
        while(a1 < m && a2 < n){
            if(arr[a1] <= nums2[a2]){
                nums1[i++] = arr[a1++];
            }else{
                nums1[i++] = nums2[a2++];
            }
        }
        for(;i < m + n; i++){
            if(a1 < m){
                nums1[i] = arr[a1++];
            }
            if(a2 < n){
               nums1[i] = nums2[a2++];
            }
        }
    }
}
```

Day 5

1.下列 Java 程序输出的结果为（ B ）。

```
public class Example{
    String str = new String("hello");
    char[] ch = {'a','b'};
    public static void main(String args[]){
        Example ex = new Example();
        ex.change(ex.str, ex.ch);
        System.out.print(ex.str + "and");
        System.out.print(ex.ch);
    }
    public void change(String str,char ch[]){
        str = "test ok";
        ch[0] = 'c';
    }
}
```

A. `hello and ab`　　B. `hello and cb`　　C. `hello and a`　　D. `test ok and ab`

解：

方法中的引用出了方法就会销毁,不会对外界造成影响,所以 `main()` 方法中的 `ex.str` 引用存放的依旧是 `"hello"` 的地址。

方法中改变数组，会对外界的数组造成影响。 ♢

2. `transient` 变量和下面哪一项有关（ B ）。

A. `Cloneable`　　B. `Serializable`　　C. `Runnable`　　D. `Comparable`

Tips

`transient` 关键字与序列化 `Serializable` 有关。　　`Cloneable`：克隆接口。

`Runnable`：多线程接口。　　`Comparable`：比较接口。

3.已知有下列 *Test* 类的说明，在该类的 *main* 方法内，则下列那个语句是正确的（ A ）。

```
public  class Test {
    private float f = 1.0f;
    int m = 12;
    static int n = 1;
    public static void main (String args[]){
        Test t = new Test();
    }
}
```

A. `t.f;`　　B. `this.n;`　　C. `Test.m;`　　D. `Test.f;`

Tips

由 `static` 修饰的为类变量或类方法，类变量可以通过类名访问也可以通过对象名访问，不能通过 `this` 关键字访问，成员变量只能通过对象访问。

4.给定一个整数数组，判断是否存在重复元素。如果存在一值在数组中出现至少两次，函数返回 `true` 。

如果数组中每个元素都不相同，则返回 `false` 。OJ 链接-存在重复元素

Tips

1. 对数组进行排序：（使用 `Arrays.sort(nums)` ）
2. 遍历数组，如果相邻元素有相等的，返回 `true` ，否则返回 `false` 。

```java
class Solution {
    public boolean containsDuplicate(int[] nums) {
        Arrays.sort(nums);
        for(int i = 0; i < nums.length - 1; i++) {
            if(nums[i] == nums[i + 1]){
                return true;
            }
        }
        return false;
    }
}
```

5.你的朋友正在使用键盘输入他的名字 `name` 。偶尔，在键入字符 `c` 时，按键可能会被长按，而字符可能被输入 1 次或多次。你将会检查键盘输入的字符 `typed` 。如果它对应的可能是你的朋友的名字（其中一些字符可能被长按），那么就返回 `True` 。OJ 链接-长按键入

Tips

`typed` 中的字符只有两种用途：

一是与 `name` 匹配，如果不满足一，属于长按输入，应与前一个字符相匹配。如果不满足这两个条件，应当直接返回 `false` 。

使用两个指针 `i` 和 `j` 分别追踪 `name` 和 `typed` 的位置，当 `type` 遍历完后，若 `i` 到达 `name` 的末尾，说明匹配上了，否则未匹配上。

```java
class Solution {
    public boolean isLongPressedName(String name, String typed) {
        int i = 0;//指向name
        int j = 0;//指向typed
        while(j < typed.length()){
            if(i < name.length() && name.charAt(i) == typed.charAt(j)){
                i++;
                j++;
            }else if(j > 0 && typed.charAt(j) == typed.charAt(j - 1)){
                j++;
            }else{
                return false;
            }
        }
        return i == name.length();
    }
}
```

Day 6

1.下面关于 Java 的垃圾回收机制正确的是（　B　）。

A. 当调用 `"System.gc()"` 来强制回收时，系统会立即回收垃圾

B. 垃圾回收不能确定具体的回收时间

C. 程序可明确地标识某个局部变量的引用不再被使用

D. 程序可以显式地立即释放对象占有的内存

Tips

调用 `"System.gc()"` 来建议执行垃圾回收器回收内存，但回收时间依旧不确定。

JVM 空闲时，自动回收每块可能被回收的内存，GC 是完全自动的，具体的回收时间，是不可知的。

局部变量存放在栈上，由 `finalize()` 来回收，JVM 回收的是堆上的内存。

2.以下会产生信息丢失的类型转换是（　B　）。

A. `float a = 10;`　B. `int a = (int)8846.0 ;`　C. `byte a = 10; int b = -a;`　D. `double d = 100`

解：

使用表示范围小的数据类型来存储表示范围大的数据，会产生信息丢失。`8846.0` 属于 `double` 类型。 ♢

Tips

`byte`：1 字节　`short`：2 字节　`int`：4 字节　`long`：8 字节

`float`：4 字节　`double`：8 字节　`char`：2 字节　`boolean`：不确定，不同的 JVM 实现方式不一样

3.面向对象方法的多态性指的是（　C　）。

A. 一个类可以派生出多个特殊类

B. 一个对象在不同的运行环境中可以有不同的变体

C. 针对一消息，不同的对象可以以适合自身的方式加以响应

D. 一个对象可以是由多个其他对象组合而成的

解：

多态即子类重写父类的方法。使用父类引用指向子类，调用方法时调用的是子类的方法。

方法又称为消息。 ♢

4.给定一个按非递减顺序排序的整数数组 `A`，返回每个数字的平方组成的新数组，要求新数组也按非递减顺序排序。（注意：非递减顺序即递增，要注意原数组里的负数）OJ 链接-有序数组的平方

Tips

1. 类似于归并排序，使用两个指针指向数组的左边和右边。

2. 判断两个指针指向的元素的平方的大小，将大的那个的平方填入新数组的后边，即如果左指针指向的元素的平方大，就把该元素平方填到新数组的后边，然后左指针向右移动一位，新数组中的指针向前移动一位，以此类推。

3. 退出条件：左指针越过右指针，或者新数组的指针小于 0. 两者是等价的

有序数组的平方

```
class Solution {
    public int[] sortedSquares(int[] nums) {
        int[] arr = new int[nums.length];
        int lo = 0;
        int hi = nums.length - 1;
        int i = nums.length - 1;
        while(lo <= hi){//或者写成while(i >= 0)
            if(nums[lo] * nums[lo] < nums[hi] * nums[hi]){
```

```
                arr[i--] = nums[hi] * nums[hi];
                hi--;
            }else{
                arr[i--] = nums[lo] * nums[lo];
                lo++;
            }
        }
        return arr;
    }
}
```

5.给定一个字符串 S，返回“反转后的”字符串，其中不是字母的字符都保留在原地，而所有字母的位置发生反转。OJ 链接-仅仅反转字母

Tips

1. 使用两个指针分别指向字符串的前后。
2. 判断左右指针指的元素是否为字母，如果左指针指的不是字母，左指针向右移动，右指针指的不是字母，右指针向左移动。
3. 经过第二步后，两个指针指的为字母，如果两个指针指的字母相等，不用交换，如果不相等，交换，交换完毕，左指针向右移动一下，右指针向左移动一下。第二步第三部。
4. 如果左指针没有越过右指针，或左指针和右指针重合之前，重复

仅仅反转字母

```
class Solution {
    private boolean isAlpha(char ch){
        return ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z'));
    }
    public String reverseOnlyLetters(String S) {
        int lo = 0;
        int hi = S.length() - 1;
        char[] ch = S.toCharArray();
        while(lo < hi){
            if(!isAlpha(S.charAt(lo))){
                lo++;
            }
            if(!isAlpha(S.charAt(hi))){
                hi--;
            }
            if(lo < hi && isAlpha(S.charAt(lo)) && isAlpha(S.charAt(hi))){
                if(S.charAt(lo) != S.charAt(hi)){
                    char tmp = ch[lo];
                    ch[lo] = ch[hi];
                    ch[hi] = tmp;
                }
                lo++;
                hi--;
            }
        }
```

```
26          return new String(ch);
27      }
28  }
```

Day 7

1.关于下面程序，哪些描述是正确的（ B、E ）。

```
1 public class While {
2     public void loop() {
3         int x = 10;
4         while (x) {
5             System.out.print("x minus one is " + (x - 1));
6             x -= 1;
7         }
8     }
9 }
```

A. 行 1 有语法错误　　B. 行 4 有语法错误　　C. 行 5 有语法错误

D. 行 6 有语法错误　　E. 行 2 有语法错误,loop 是关键字　　D. 程序能够正常编译和运行

解：

变量名、方法名、自定义类名不能与关键字重名，所以行 2 错误。

Java 中不能使用数字作为逻辑值，所以行 4 错误　　♢

2.在各自最优条件下，对 N 个数进行排序，算法复杂度最低的是（ A ）。

A. 插入排序　　B. 快速排序　　C. 堆排序　　D. 归并排序

Tips

排序方法	时间复杂度（平均）	时间复杂度（最坏）	时间复杂度（最好）	空间复杂度	稳定性
直接插入排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
希尔排序	$O(n \log n)$	$O(n^2)$	$O(n^{1.3})$	$O(1)$	不稳定
直接选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
堆排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	不稳定
冒泡排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
快速排序	$O(n \log n)$	$O(n^2)$	$O(n \log n)$	$O(n \log n)$	不稳定
归并排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	稳定
基数排序	$O(d(n+r))$	$O(d(n+r))$	$O(d(n+r))$	$O(n+r)$	稳定

注：基数排序的复杂度中，r 代表关键字的基数，d 代表长度，n 代表关键字的个数

3.在 java 中，下列对继承的说法，正确的是（ A ）。

A. 子类能继承父类所有的成员　　B. 子类继承父类的非私有方法和状态

C. 子类只能继承父类的 public 方法和状态　　D. 子类只能继承父类的方法

解：

子类继承父类的全部，但只能使用非私有成员变量和方法。　　♢

4.给定一个非负的整数数组 A，返回一个数组，在该数组中，A 的所有偶数元素之后跟着所有奇数元素。

OJ 链接-按奇偶排序数组

Tips

1. 使用左指针指向数组的左侧，右指针指向数组的右侧
2. 如果左指针指的是偶数，左指针向右移动，右指针指的是奇数，右指针向左移动
3. 经过上一步，左指针指向为奇数，右指针指向的为偶数，交换这两个指针指的元素的值。
4. 左指针在右指针的左侧时，重复第二步、第三部。

按奇偶排序数组

```java
class Solution {
    public int[] sortArrayByParity(int[] A) {
        int lo = 0;
        int hi = A.length - 1;
        while(lo < hi){
            if(A[lo] % 2 == 0){
                lo++;
            }
            if(A[hi] % 2 == 1){
                hi--;
            }
            if(lo < hi && A[lo] % 2 == 1 && A[hi] % 2 == 0){
                int tmp = A[lo];
                A[lo] = A[hi];
                A[hi] = tmp;
            }
        }
        return A;
    }
}
```

5.给定一个整数类型的数组 `nums`，请编写一个能够返回数组“中心索引”的方法。

我们是这样定义数组 中心索引 的：数组中心索引的左侧所有元素相加的和等于右侧所有元素相加的和。

如果数组不存在中心索引，那么我们应该返回 -1。如果数组有多个中心索引，那么我们应该返回最靠近左边的那一个。OJ 链接-寻找数组的中心索引

Tips

1. 计算出数组的和；
2. 从左至右遍历数组,计算出第 `i` 个元素左侧元素的和(包含该元素 `A[i]`)`sum1` 和右侧元素的和(包含该元素 `A[i]`)`sum2`,如果 `sum1` 和 `sum2` 相等，说明该位置为中心索引。
3. 遍历完依旧没有符合题意的返回 `-1`。

寻找数组的中心索引

```java
class Solution {
    public int pivotIndex(int[] nums) {
        int sum1 = 0;
        for(int x : nums){
            sum1 += x;
        }
        int sum2 = 0;
        for(int i = 0; i < nums.length; i++){
            sum2 += nums[i];
            if(sum1 == sum2){
                return i;
            }
```

```
13                 sum1 = sum1 - nums[i];
14             }
15             return -1;
16         }
17     }
```

Day 8

1.（**多选题**）已知 `boolean result = false`，则下面哪个选项是合法的（ B、D ）。

A. `result = 1`　　B. `result = true`　　C. `if (result != 0){ }`　　D. `if (result){ }`

解：

`boolean` 类型不能和其他类型进行相互赋值、比较。 ♢

2.（**多选题**）下面的 `switch` 语句中，`x` 可以是哪些类型的数据（ B、D ）。

```
switch(x){
    default:
    System.out.println("hello");
}
```

A. `long`　　B. `char`　　C. `float`　　D. `byte`　　E. `double`　　F. `Object`

Tips

`switch` 中支持的数据类型：`char`、`byte`、`short`、`int`、`Character`、`Byte`、`Short`、`Integer`、`String`、`char`、`enum`。

3.（**多选题**）关于运行时常量池，下列那个说法是正确的（ B、C、D ）。

A. 运行时常量池大小受栈区大小的影响

B. 运行时常量池大小受方法区大小的影响

C. 存放了编译时期生成的各种字面量

D. 存放编译时期生成的符号引用

Tips

运行时常量池是方法区的一部分。class 文件中除了有类的版本、字段、方法、接口等描述信息外，还有一项信息是常量池，用于存放编译期生成的各种字面量和符号引用，这部分内容将在类加载后进入方法区的运行时常量池中存放。

4.给定一个由**整数**组成的 **非空** 数组所表示的非负整数，在该数的基础上加一。最高位数字存放在数组的首位，数组中每个元素只存储单个数字。你可以假设除了整数 0 之外，这个整数不会以零开头。(第一个数字是 9 或者最后一个是 9 要考虑进位)OJ 链接-加一

Tips

1. 将数组的最后一个元素加一；
2. 加一后只有两种情况，等于 10 或小于 10.
3. 等于 10，将其除 10 取余即为该位的值，同时将指针前移一位，将前一位加一，然后重复 3、4 步；
4. 小于 10，不需要进位，直接返回。
5. 对于 9,99,999…这种加一后，长度增加的，直接创建一个比原数组长度长 1 的数组，将新数组首位初始化为 1，其余保持不变（默认初始化为 0）。

加一

```
1  class Solution {
2      public int[] plusOne(int[] digits) {
3          for(int i = digits.length - 1; i >= 0; i--){
4              digits[i]++;
5              digits[i] = digits[i] % 10;
6              if(digits[i] != 0){
7                  return digits;
8              }
9          }
```

```
        int[] arr = new int[digits.length + 1];
        arr[0] = 1;
        return arr;
    }
}
```

5.给你一个非空数组，返回此数组中 第三大的数。如果不存在，则返回数组中最大的数。OJ 链接-第三大的数

Tips

1. 数组长度为 1，直接返回该元素。
2. 数组长度为 2，返回两个中最大的那个。
3. 数组长度大于等于 3：

题目中数的范围为 int 的范围，所以数组中第三大的元素可能是 int 的最小值（ Integer.MIN_VALUE ）。

使用三个数 max1 、max2 、max3 分别表示数组中第一大,第二大,第三大的元素,将其初始化为 int 的最小值减一(即 Integer.MIN_VALUE - 1 ，把这个值记为 min)，因为超出了 int 的范围，所以：类型为 long 。

遍历数组，对于任意元素：

如果该元素大于 max1 ，就把 max2 的值赋给 max3 ，把 max1 的值赋给 max2 ，把该元素的值赋给 max1 ；

如果该元素等于 max1 ，检查下一个元素，即 continue ；

如果该元素小于 max1 ，则将该元素与 max2 比较：

如果该元素大于 max2 ，就把 max2 的值赋给 max3 ，把该元素的值赋给 max2 ；

如果该元素等于 max2 ，检查下一个元素，即 continue ；

如果该元素小于 max2 ，则将该元素与 max3 比较：

只有该元素大于 max3 时，才将该元素的值赋给 max3 ，否则不操作。

遍历完数组后，看 max3 的值是否为 min ，如果是，说明没有第 3 大的数，返回 max1 ，否则返回 max3 。因为其类型为 long ，还需要进行类型转换。

第三大的数

```
class Solution {
    public int thirdMax(int[] nums) {
        if(nums.length == 1){
            return nums[0];
        }
        if(nums.length == 2){
            return nums[0] > nums[1] ? nums[0] : nums[1];
        }
        long min = (long)Integer.MIN_VALUE - 1;
        long max1 = min;
        long max2 = max1;
        long max3 = max2;

        for(int i = 0; i < nums.length; i++){
            if(nums[i] > max1){
                max3 = max2;
                max2 = max1;
                max1 = nums[i];
            }else if(nums[i] == max1) {
                continue;
```

```
21              }else{
22                  if(nums[i] > max2){
23                      max3 = max2;
24                      max2 = nums[i];
25                  }else if(nums[i] == max2){
26                      continue;
27                  }else{
28                      if(nums[i] > max3){
29                          max3 = nums[i];
30                      }
31                  }
32              }
33          }
34          return max3 == min ? (int)max1 : (int)max3;
35      }
36  }
```

Day 9

1.下列类定义中哪些是合法的抽象类的定义（　C　）。

A. `abstract Animal{abstract void growl();}`

B. `class abstract Animal{abstract void growl();}`

C. `abstract class Animal{abstract void growl();}`

D. `abstract class Animal{abstract void growl(){System.out.println( “growl” );};}`

Tips

定义类使用 `class` 关键字，该关键字与类名紧邻，中间不能有其他关键字或符号。

抽象方法不能有方法体。

2.在 `java` 中，无论在何处调用，使用静态属性必须以类名做前缀。（　B　）

A. 正确　　B. 错误

解：

可以使用对象名来调用静态属性，但其本质上是通过对象找到其对应的类，然后调用该静态属性。 ♢

3.哪个关键字可以对对象加互斥锁（　A　）。

A. `synchronized` B. `volatile` C. `serialize` D. `static`

Tips

`synchronized`：用来给对象和方法或者代码块加锁. 当它锁定一个方法或者一个代码块的时候, 同一时刻最多只有一个线程执行这个段代码。

`volatile`：用来确保将变量的更新操作通知到其他线程, 当把变量声明为 `volatile` 类型后, 编译器与运行时都会注意到这个变量是共享的, 因此不会将该变量上的操作与其他内存操作一起重排序. 然而, 在访问 `volatile` 变量时, 不会执行加锁操作, 因此也就不会使执行线程阻塞, 因此, `volatile` 变量是一种比 `synchronized` 关键字更轻量级的同步机制。

`serialize`：Java 对象序列化为二进制文件。

`static`：修饰类变量、类方法，静态代码块。

4.给定一个整数数组 nums 和一个整数目标值 target，请你在该数组中找出 和为目标值 的那 两个整数，并返回它们的数组下标。你可以假设每种输入只会对应一个答案。但是，数组中同一个元素不能使用两遍。你可以按任意顺序返回答案。要求时间复杂度 $O(n)$, 当然也可以选择使用暴力的 $O(n^2)$ 的解法。OJ 链接-两数之和

Tips

1. 从左至右遍历数组。
2. 一个指针指向当前元素，另一个指针从当前元素的下一位至数组最后遍历，寻找与当前元素的和等于目标值的元素。找到后返回这两个指针对应的下标值。

两数之和

```java
class Solution {
    public int[] twoSum(int[] nums, int target) {
        for(int i = 0; i< nums.length;i++){
            for(int j = i + 1;j < nums.length;j++){
                if(nums[j] == target - nums[i]){
                    int[] ret = {i,j};
                    return ret;
                }
            }
        }
```

```
11          return new int[0];
12      }
13  }
```

5.给你两个二进制字符串，返回它们的和（用二进制表示）。输入为 **非空** 字符串且只包含数字 `1` 和 `0`。

OJ 链接-二进制求和

Tips

1. 首先将短的字符串用 0 将高位补齐。因为要对字符频繁的操作，所以使用可变字符序列对象 `StringBuilder`。可以先将题中的两个字符串翻转，在最短的字符串后补 0（使用 `append` 方法），最后计算结束再将字符串翻转。
2. 字符串补齐后，使用列竖式的方法计算每一位，用 `carry` 来表示进位，刚开始进位为 0,。
3. 获取每一位的字符，因为只有 0 和 1，所以减去字符 0 即为数字 0 和 1。每一位的和为 `carry + a对应位置的值 + b对应位置的值`，记为 `count`，因为是 2 进制，所以 `count` 大于等于 2 时，要进行进位，即 `carry` 置为 1，同时该位置的 `count%2`。
4. 计算结束后，要判断 `carry` 的值，如果这个值是 1，说明还需要进一位。
5. 别忘了翻转和返回值的类型。

二进制求和

```
class Solution {
    public String addBinary(String a, String b) {
        //求两个字符串的长度
        int a1 = a.length();
        int b1 = b.length();
        //求字符串的最大长度
        int max = a1 >= b1 ? a1 : b1;
        //将字符串存入可变字符序列并进行翻转
        StringBuilder stringBuilder1 = new StringBuilder(a).reverse();
        StringBuilder stringBuilder2 = new StringBuilder(b).reverse();
        //对短字符串进行补0
        while(stringBuilder1.length() < max){
            stringBuilder1.append("0");
        }
        while(stringBuilder2.length() < max){
            stringBuilder2.append("0");
        }
        //进位，初始为0
        int carry = 0;

        int x = 0;
        int y = 0;
        int count = 0;
        //存放相加的结果
        StringBuilder ret = new StringBuilder();
        for(int i = 0; i < max; i++){
            x = stringBuilder1.charAt(i) - '0';
            y = stringBuilder2.charAt(i) - '0';
            //每一位结果等于进位加各数
```

```
30              count = x + y + carry;
31              //二进制，结果超过2，就需要进位
32              if(count >= 2){
33                  carry = 1;
34                  ret.append(count % 2);
35              }else{
36                  carry = 0;
37                  ret.append(count);
38              }
39          }
40          //最高位的进位判断
41          if(carry == 1){
42              ret.append("1");
43          }
44          //别忘了翻转，注意方法的返回类型
45          return ret.reverse().toString();
46      }
47  }
```

Day 10

1.一个 Java 源程序文件中定义几个类和接口,则编译该文件后生成几个以.class 为后缀的字节码文件。(A)

A. 正确　　B. 错误

2.要使某个类能被同一个包中的其他类访问，但不能被这个包以外的类访问，可以（ A ）。

A. 让该类不使用任何关键字

B. 使用 `private` 关键字

C. 使用 `protected` 关键字

D. 使用 `void` 关键字

Tips

不使用关键字：包访问权限，即只能同包的类访问。

`private`：类访问权限，只能在类的内部访问，出了类则无法访问。

`protected`：同包的类可以访问，不通包的子类也可以访问。

`public`：任意包中的类都可以访问。

`void`：方法的返回类型，与访问权限无关。

3.判断对错。在 java 的多态调用中，new 的是哪一个类就是调用的哪个类的方法。(A)

A. 正确　　B. 错误

4.字符串转换函数OJ 链接-字符串转换整数 atoi。

请你来实现一个 `myAtoi(string s)` 函数，使其能将字符串转换成一个 32 位有符号整数（类似 `C/C++` 中的 `atoi` 函数）。

函数 `myAtoi(string s)` 的算法如下：

- 读入字符串并丢弃无用的前导空格。
- 检查第一个字符（假设还未到字符末尾）为正还是负号，读取该字符（如果有）。确定最终结果是负数还是正数。如果两者都不存在，则假定结果为正。
- 读入下一个字符，直到到达下一个非数字字符或到达输入的结尾。字符串的其余部分将被忽略。
- 将前面步骤读入的这些数字转换为整数（即，"123" → 123，"0032" →32）。如果没有读入数字，则整数为 0 。必要时更改符号（从步骤 2 开始）。
- 如果整数数超过 32 位有符号整数范围 $[-2^{31},\ 2^{31}\ -1]$ ，需要截断这个整数，使其保持在这个范围内。具体来说，小于 -2^{31} 的整数应该被固定为 -2^{31} ，大于 $2^{31}\ -1$ 的整数应该被固定为 $2^{31}\ -1$。
- 返回整数作为最终结果。

注意：

- 本题中的空白字符只包括空格字符 `' '` 。
- 除前导空格或数字后的其余字符串外，请勿忽略 任何其他字符。

Tips

1. 对于空字符串和 `null`，直接返回 0;
2. 一般字符串:

依题意，首先扫描空格，当指针没有到达末尾且指向的字符为空格时，指针向右移动。

空格扫描完后，判断指针是否到达末尾，如果到达末尾，返回 0.

没有到达末尾，扫描下一个字符是否是 `+` 或 `-`，使用 `flag` 作为正负的标记（默认为 1），若扫描到 `+`，`flag` 不变，指针向后移动一位。若扫描到 `1`，`flag` 置为 `-1`，指针向后移动一位。

再次判断是否到达末尾，如果到达末尾，返回 0.

没有到达末尾，继续扫描:（使用 `ret` 存储扫描的数字，因为数字可能超过 `int` 的最大范围，所以 `ret` 为 `long` 类型）

如果是数字,将其放到 `ret` 的后边,即 `ret = ret * 10` ,然后判断 `ret` 的大小,如果 `flag` 为 1,且 `ret` 超过整型的最大值,则返回 `Integer.MAX_VALUE` ,如果 `flag` 为-1，且 `-ret` 小于整型的最小值，则返回 `Integer.MIN_VALUE` 。

如果不是数字或者指针已经扫描到了末尾，退出循环。

如果 `ret` 没有超过整型的范围，返回 `flag * ret` 。要注意类型的转换。

字符串转换整数 atoi

```java
class Solution {
    public int myAtoi(String s) {
        if(s == null || s.length() == 0){
            return 0;
        }
        int i = 0;
        long ret = 0;
        int flag = 1;
        while(i < s.length() && s.charAt(i) == ' '){
            ++i;
        }
        if(i >= s.length()){
            return 0;
        }
        if(s.charAt(i) == '-'){
            flag = -1;
            i++;
        }else if(s.charAt(i) == '+'){
            i++;
        }

        if(i >= s.length()){
            return 0;
        }

        while(i < s.length() && (s.charAt(i) >= '0' && s.charAt(i) <= '9')){
            int x = s.charAt(i) - '0';
            ret = ret * 10 + x;
            if(flag > 0 && ret > Integer.MAX_VALUE){
                return Integer.MAX_VALUE;
            }
            if(flag < 0 && -ret < Integer.MIN_VALUE){
                return Integer.MIN_VALUE;
            }
            ++i;
        }
        return flag > 0 ? (int)ret : -(int)ret;
    }
}
```

5.给定一个按照升序排列的整数数组 `nums`，和一个目标值 `target`。找出给定目标值在数组中的开始位置和结束位置。如果数组中不存在目标值 `target`，返回 $[-1,-1]$。OJ 链接-在排序数组中查找元素的第一个和最后一个位置。

Tips

使用二分查找。

关键在于等于时的情况处理：

寻找下界:如果 `mid` 处的值等于目标值,判断 `mid` 的左侧值是否等于目标值,如果左侧没有值或者左侧不等于目标值,说明该 `mid` 为下界,否则需要从 `[lo,mid - 1]` 区间来查找下界，即 `hi = mid - 1`

寻找上界：与寻找下界逻辑相反，如果 `mid` 处的值等于目标值，判断 `mid` 的右侧值是否等于目标值，如果右侧没有值或者右侧不等于目标值，说明该 `mid` 为上界，否则需要从 `[mid + 1, hi]` 区间来查找上界，即 `lo = mid + 1`

在排序数组中查找元素的第一个和最后一个位置

```java
class Solution {
    private int findFirst(int[] nums, int target) {
        int lo = 0;
        int hi = nums.length - 1;
        int mid = 0;
        while(lo <= hi){
            mid = lo + ((hi - lo) >> 1);
            if(nums[mid] < target){
                lo = mid + 1;
            }else if(target < nums[mid]){
                hi = mid - 1;
            }else{
                if(mid == 0 || nums[mid - 1] != target){
                    return mid;
                }else{
                    hi = mid - 1;
                }
            }
        }
        return -1;
    }

    private int findLast(int[] nums, int target) {
        int lo = 0;
        int hi = nums.length - 1;
        int mid = 0;
        while(lo <= hi){
            mid = lo + ((hi - lo) >> 1);
            if(nums[mid] < target){
                lo = mid + 1;
            }else if(target < nums[mid]){
```

```
32                  hi = mid - 1;
33              }else{
34                  if(mid == nums.length - 1 || nums[mid + 1] != target){
35                      return mid;
36                  }else{
37                      lo = mid + 1;
38                  }
39              }
40          }
41          return -1;
42      }
43      public int[] searchRange(int[] nums, int target) {
44          return new int[]{findFirst(nums,target),findLast(nums,target)};
45
46      }
47  }
```

Day 11

1.执行下列代码的输出结果是（ C ）。

```java
public class Demo{
    public static void main(String args[]){
        int num = 10;
        System.out.println(test(num));
    }
    public static int test(int b){
        try {
            b += 10;
            return b;
        }catch(RuntimeException e){

        }catch(Exception e2){

        }finally{
            b += 10;
            return b;
        }
    }
}
```

A. 10 B. 20 C. 30 D. 40

Tips

如果有 finally 语句，在 try 和 catch 中遇到 return，先执行 finally 中的语句，如果在 finally 中遇到了 return，则直接返回，如果 finally 中没有 return 语句，则返回至 try 或 catch 中的 return 处。

2.下面关于 Java package 的描述，哪个是正确的（ B ）。

I. 包不提供将所有类名分区为更易管理的块的机制。

II. 包提供可见性控制机制。

III. 包的一个重要属性是包内定义的所有类都可以通过该包外的代码访问。

IV. 声明为包的一部分的类的.class 文件可以存储在多个目录中.

A. 只有 I B. 只有 II C. 只有 III D. 只有 IV

Tips

为了更好地组织类，Java 提供了包机制，用于区别类名的命名空间。

包的作用 1、把功能相似或相关的类或接口组织在同一个包中，方便类的查找和使用。

2、如同文件夹一样，包也采用了树形目录的存储方式。同一个包中的类名字是不同的，不同的包中的类的名字是可以相同的，当同时调用两个不同包中相同类名的类时，应该加上包名加以区别。因此，包可以避免名字冲突。

3、包也限定了访问权限，拥有包访问权限的类才能访问某个包中的类。

Java 使用包（package）这种机制是为了防止命名冲突，访问控制，提供搜索和定位类（class）、接口、枚举（enumerations）和注释（annotation）等。

3.Java 数据库连接库 JDBC 用到哪种设计模式（ B ）。

A. 生成器 B. 桥接模式 C. 抽象工厂 D. 单例模式

4.给定一个字符串，验证它是否是回文串，只考虑字母和数字字符，可以忽略字母的大小写。

说明：本题中，我们将空字符串定义为有效的回文串。OJ 链接-验证回文串

Tips

1. 处理空字符串：是回文串
2. 只关注字母和数字，所以写一个私有方法来判断当前字符是否为字母/数字。
3. 使用左右指针指向字符串的开头和结尾。
4. 如果不是字母或数字，左指针右移，右指针左移。
5. 经过上一后左右指针指向的字符则为字母或数字。题目中不区分大小写，所以如果字符是大写字母，将其转换为小写字母 (也可以把小写的字符转为大写的)。
6. 因为字符串是不可变对象，所以使用字符来接收左右指针指向的字符，大写字符转小写：`ch - 'A'+ 'a'`，字符相加减本质上是其对应的 ASCII 码相加减，因此还需将其强制转换为字符。
7. 大写转为小写后，比较两个字符是否相等，如果不相等，返回 `false`。如果相等，左指针左移，右指针右移。继续第 5 到第 7 步。
8. 若左指针和右指针重合，说明是回文串。

验证回文串

```
class Solution {
    private boolean isAlphaAndNum(char ch){
        if((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z') || (ch >= '0' &&
             ch <= '9')){
            return true;
        }
        return false;
    }
    public boolean isPalindrome(String s) {
        if(s == null || " ".equals(s)){
            return true;
        }
        int lo = 0;
        int hi = s.length() - 1;
        char a;
        char b;
        while(lo < hi){
            if(!isAlphaAndNum(s.charAt(lo))){
                lo++;
            }
            if(!isAlphaAndNum(s.charAt(hi))){
                hi--;
            }
            if(lo < hi && isAlphaAndNum(s.charAt(lo)) && isAlphaAndNum(s.charAt(
                hi))){
                a = s.charAt(lo);
                b = s.charAt(hi);
                if(a >= 'A' && a <= 'Z'){
                    a = (char)(a + 'a' - 'A');
                }
                if(b >= 'A' && b <= 'Z'){
                    b = (char)(b + 'a' - 'A');
                }
                if(a != b){
```

```
                    return false;
                }
                lo++;
                hi--;
            }
        }
        return true;
    }
}
```

5.给定一组字符，使用原地算法将其压缩。压缩后的长度必须始终小于或等于原数组长度。数组的每个元素应该是长度为 1 的字符（不是 int 整数类型）。在完成原地修改输入数组后，返回数组的新长度。OJ 链接-压缩字符串

Tips

使用三个指针。

1. 指针 i 指向开头。第三个指针 index 从头开始。

2. j 指向 i 的下一个位置。

3. 如果指针 j 指向的字符等于指针 i 指向的字符，指针 j 向后移动，直到与指针 i 指向的字符不相等（或者到字符串的末尾）停下来。

4. 两种情况：

a. 若 j 在 i 下一位停下来，表示 i 指向的字符只有一个，不需要压缩，此时 j - i 为 1。

b. 若 j 在 i 的后边位置停下来（超过 1 位），假设 i 为 0，j 为 4，说明位置 0、1、2、3 的字符都相等，即有连续 j - i 个字符相等。

5. 在 index 处记录 i 处的字符，然后向右移动一位，如果 j - i 为 1，不需要压缩，如果 j - i 大于 1，题目给的字符数组的长度最大为 1000，可能出现连续字符个数最大为 1000，即 j - i 超过一位数，把该数转换为字符数组，然后把数的每一位依次填入 index 处（填一位 index 向右移动一位）（代码的 12-15 行）。

6. 上一步执行完后，将指针 i 移到指针 j 处，执行 2-5 步，直到全部字符串扫描结束。

压缩字符串

```
class Solution {
    public int compress(char[] chars) {
        int index = 0;
        int i = 0;
        int j = 1;
        while(i < chars.length && index < chars.length){
            while(j < chars.length && chars[j] == chars[i]){
                j++;
            }
            chars[index++] = chars[i];
            int sum = j - i;
            if(sum > 1 && index < chars.length){
                for(char ch : ("" + sum).toCharArray()){
                    chars[index++] = ch;
                }
            }
            i = j;
            j = i + 1;
        }
```

```
20            return index;
21        }
22    }
```

Day 12

1.下列哪个选项是正确计算 42°（角度）的余弦值（　C　）。

A. `double d=Math.cos(42)`

B. `double d=Math.cosine(42)`

C. `double d=Math.cos(Math.toRadians(42))`

D `double d=Math.cos(Math.toDegrees(42))`

Tips

`Math.toDegrees()` 用于将 `()` 中的参数转化为角度。

`Math.toDegrees()` 用于将 `()` 中的参数转化为弧度。

`Math.cos()` 中的参数为弧度。

计算 42°（角度）的余弦值：把 42° 转化为弧度，然后求余弦。

2. `String s = new String("xyz")` 创建了几个 `String` 对象（　C　）。

A. 两个或一个都有可能　　B. 两个　　C. 一个　　D. 三个

3.下列代码输出结果为（　　）。

A.

动物可以移动

狗可以跑和走

狗可以吠叫

B. 动物可以移动 动物可以移动 狗可以吠叫

C. 运行错误

D. 编译错误

解：

没有代码。　　♢

4.给你一个整数数组 `nums` ，你需要找出一个**连续子数组**，如果对这个子数组进行升序排序，那么整个数组都会变为升序排序。请你找出符合题意的 最短 子数组，并输出它的长度。OJ 链接-最短无序连续子数组

Tips

1. 如果数组长度为 0 或长度为 1，则数组一定有序。
2. 对于长度大于 1 的数组：

符合题意的任意一个应该为：有序 1 + 无序 + 有序 2。

升序排序，所以无序中的最大值不会超过有序 2 中的最小值，无序中的最小值不会小于有序 1 中的最大值。

使用 `lo` 表示无序的左边界，使用 `ri` 表示无序的右边界。

从左至右找无序的右边界:将已经扫描过的最大元素记为 `max` ,显然有序 2 的所有元素都应该大于等于 `max` ,如果某个元素 `nums[i]<max` ,说明 `nums[i]` 不在有序 2 中，应该在无序中，把无序的右边界更新为 `i` 。

从右至左找左边界:与找右边界逻辑逻辑一致,将已经扫描过的最小元素记为 `min` ,有序 1 的所有元素都应该小于等于 `min` ,如果某个元素 `nums[j]>min` ,说明 `nums[j]` 不在有序 1 中，应该在无序中，把无序的左边界更新为 `j` 。

最短无序连续子数组

```
class Solution {
    public int findUnsortedSubarray(int[] nums) {
        if(nums.length == 0 || nums.length == 1 ){
            return 0;
        }
```

```
        int lo = nums.length - 1;
        int hi = 0;
        int max = nums[0];
        int min = nums[nums.length - 1];
        for(int i = 0; i < nums.length; i++){
            if(nums[i] < max){
                hi = i;
            }else{
                max = nums[i];
            }

            if(nums[nums.length - 1 - i] > min){
                lo = nums.length - 1 - i;
            }else{
                min = nums[nums.length - 1 - i];
            }
        }
        return hi - lo < 0 ? 0 : hi - lo + 1;
    }
}
```

5.根据**逆波兰表示法**，求表达式的值。有效的运算符包括 `+`, `-`, `*`, `/` 。每个运算对象可以是整数，也可以是另一个逆波兰表达式。OJ 链接-逆波兰表达式求值

说明：

- 整数除法只保留整数部分。
- 给定逆波兰表达式总是有效的。换句话说，表达式总会得出有效数值且不存在除数为 0 的情况。

Tips

使用题目给的提示：适合用栈操作运算：遇到数字则入栈；遇到算符则取出栈顶两个数字进行计算，并将结果压入栈中。

逆波兰表达式求值

```
class Solution {
    private boolean isNumber(String str){
        return !"+".equals(str) && !"-".equals(str) && !"*".equals(str) && !"/".
            equals(str);
    }

    public int evalRPN(String[] tokens) {
        Deque<Integer> stk = new ArrayDeque<>();
        for(String str : tokens){
            if(isNumber(str)){
                stk.push(Integer.valueOf(str));
                continue;
            }
            int n1 = stk.pop();
```

```
14              int n2 = stk.pop();
15              switch(str){
16                  case "+":
17                      stk.push(n2 + n1);
18                      break;
19                  case "-":
20                      stk.push(n2 - n1);
21                      break;
22                  case "*":
23                      stk.push(n2 * n1);
24                      break;
25                  case "/":
26                      stk.push(n2 / n1);
27                      break;
28              }
29
30          }
31          return stk.pop();
32      }
33  }
```

Day 13

1.（**多选题**）关于 Java 以下描述正确的有（ C、D ）。

A. Class 类是 Object 类的超类

B. Object 类是一个 final 类

C. String 类是一个 final 类

D. Class 类可以装载其它类

Tips

Object 是所有类的超类。

final 修饰的类不可以被继承。

2.（**多选题**）Java 中的集合类包括 ArrayList 、 LinkedList 、 HashMap 等类，下列关于集合类描述正确的是（ A、B、D ）。

A. ArrayList 和 LinkedList 均实现了 List 接口

B. ArrayList 的访问速度比 LinkedList 快

C. 添加和删除元素时， ArrayList 的表现更佳

D. HashMap 实现 Map 接口，它允许任何类型的键和值对象，并允许将 null 用作键或值

Tips

ArrayList 和 LinkedList 均实现了 List 接口。

ArrayList 在内存中基于数组实现的，为寻秩访问，访问速度快，但增加和删除元素慢；LinkedList 基于链表存放，为寻地址访问，访问速度慢，添加和删除元素略优。

HashMap 实现 Map 接口，允许任何类型的键和值对象，允许将 null 用作键或值

3.（**多选题**）关于 equals 和 hashCode 描述正确的是 （ A、B、C ）。

A. 两个 obj ，如果 equals() 相等， hashCode() 一定相等（符合代码规范的情况下）

B. 两个 obj ，如果 hashCode() 相等， equals() 不一定相等

C. 两个不同的 obj ， hashCode() 可能相等

D. 其他都不对

4.给你一个整数数组 nums ，数组中的元素互不相同。返回该数组所有可能的子集（幂集）。解集不能包含重复的子集。你可以按任意顺序返回解集。OJ 链接-子集

Tips

1. 对于互不相同元素组成的集合，其子集一共有 2^n 种。
2. 每种元素 a_i 只有两种状态，在子集中和不在子集中，用 1 表示在子集中，用 0 表示不在子集中。
3. 每个子集可以对应一个长度为 n 的 0/1 序列，第 i 位表示 a_i 是否在子集中。
4. 使用两层循环将每一个元素加入到对应的子集中。

子集

```
class Solution {
    List<Integer> t = new ArrayList();
    List<List<Integer>> ans = new ArrayList();
    public List<List<Integer>> subsets(int[] nums) {
        int n = nums.length;
        for(int mask = 0; mask < (1 << n);mask++){
            t.clear();
            for(int i = 0; i < n; i++){
                if((mask & (1 << i)) != 0 ){
```

```
                    t.add(nums[i]);
                }
            }
            ans.add(new ArrayList<Integer>(t));
        }
        return ans;
    }
}
```

5.给定一个整数矩阵，找出最长递增路径的长度。对于每个单元格，你可以往上，下，左，右四个方向移动。你不能在对角线方向上移动或移动到边界外（即不允许环绕）。OJ 链接-矩阵中最长递增路径

矩阵中最长递增路径

```
class Solution {
    public int longestIncreasingPath(int[][] matrix) {
        int[][] visited = new int[matrix.length][matrix[0].length];
        int max = 0;
        for(int i = 0;i < matrix.length;i++){
            for(int j = 0;j < matrix[0].length;j++){
                if(visited[i][j] == 0){
                    max = Math.max(max,dfs(i, j, matrix,visited));
                 }
            }
        }
        return max;
    }
    public int dfs(int i,int j, int[][] matrix,int[][] visited){
        if((i < 0 || i >= visited.length) && (j < 0 || j >= visited[0].length)){
            return 0;
        }
        if(visited[i][j] > 0){
            return visited[i][j];
        }
        int max = 0;
        if(i - 1 >= 0 && matrix[i - 1][j] < matrix[i][j]){
            max = Math.max(max,dfs(i - 1, j, matrix,visited));
        }
        if(i + 1 < visited.length && matrix[i + 1][j] < matrix[i][j]){
            max = Math.max(max,dfs(i + 1, j, matrix,visited));
        }
        if(j - 1 >= 0 && matrix[i][j - 1] < matrix[i][j]){
            max = Math.max(max,dfs(i, j - 1, matrix,visited));
        }
        if(j + 1 < visited[0].length && matrix[i][j + 1] < matrix[i][j]){
            max = Math.max(max,dfs(i, j + 1, matrix,visited));
        }
```

```
34
35          visited[i][j] = max + 1;
36          return  visited[i][j];
37      }
38  }
```

Day 14

1.（**多选题**）下面有关 `java` 的 `instanceof` 、 `?` 、 `&` 、 `&&` 说法正确的有（A、B、C、D）。

A. `instanceof` 可用来判断某个实例变量是否属于某种类的类型。

B. `"? :"` 三目运算符

C. `&` 在逻辑运算中是非短路逻辑与，在位运算中是按位与

D. `&&` 逻辑运算：逻辑与

2.（**多选题**）下面哪个语句是创建数组的正确语句？（A、B、D、E）。

A. `float f[][] = new float[6][6];`

B. `float []f[] = new float[6][6];`

C. `float f[][] = new float[][6];`

D. `float [][]f = new float[6][6];`

E. `float [][]f = new float[6][];`

Tips

多维数组行可以省略，列不能省略。等号右边的第一个 `[]` 是列，第二个 `[]` 是行。

3.下列类在多重 `catch` 中同时出现时，哪一个异常类应最后一个列出（ C ）。

A. `ArithmeticException` B. `NumberFormatException` C. `Exception` D. `ArrayIndexOutOfBoundException`

4.给定一棵二叉树，想象自己站在它的右侧，按照从顶部到底部的顺序，返回从右侧所能看到的节点值。

OJ 链接-二叉树的右视图

Tips

层次遍历

1. 空树直接返回。
2. 对于一般二叉树，使用一个队列。将根节点推入队列。
3. 当队列不为空时，首先记录当前队列的大小 `size`
4. 对队列（当前层）进行遍历：
   - 取出队首元素 `node` 。
   - 如果该元素 `node` 的左孩子不为空，左孩子入队。
   - 如果该元素 `node` 的左孩子不为空，右孩子入队。
5. 当前层遍历完后， `node` 为当前层的最右侧元素，将该元素的值入结果。
6. 继续对下一层执行 3-6 步。

二叉树的右视图

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
```

```
 *          this.left = left;
 *          this.right = right;
 *      }
 * }
 */
class Solution {
    public List<Integer> rightSideView(TreeNode root) {
        List<Integer> ans = new ArrayList<>();
        if(root == null){
            return ans;
        }
        TreeNode node = new TreeNode();
        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);
        while(!queue.isEmpty()){
            int size = queue.size();
            for (int i = 0; i < size; i++){
                node = queue.poll();
                if(node.left != null){
                    queue.offer(node.left);
                }
                if(node.right != null){
                    queue.offer(node.right);
                }
            }
            ans.add(node.val);
        }
        return ans;
    }
}
```

5.公交路线。OJ 链接-公交路线

我们有一系列公交路线。每一条路线 `routes[i]` 上都有一辆公交车在上面循环行驶。例如，有一条路线 `routes[0] = [1, 5, 7]`，表示第一辆 (下标为 0) 公交车会一直按照 $1 \to 5 \to 7 \to 1 \to 5 \to 7 \to 1 \to \ldots$ 的车站路线行驶。

假设我们从 S 车站开始（初始时不在公交车上），要去往 T 站。期间仅可乘坐公交车，求出最少乘坐的公交车数量。返回 -1 表示不可能到达终点车站。

公交路线

```
class Solution {
    public int numBusesToDestination(int[][] routes, int S, int T) {
        //不会做，解析也没看懂
    }
}
```

6.通配符匹配。OJ 链接-通配符匹配

给定一个字符串 (s) 和一个字符模式 (p)，实现一个支持 `'?'` 和 `'*'` 的通配符匹配。

- `'?'` 可以匹配任何单个字符。
- `'*'` 可以匹配任意字符串（包括空字符串）。

两个字符串完全匹配才算匹配成功。

说明:

- s 可能为空，且只包含从 `a-z` 的小写字母。
- s 可能为空，且只包含从 `a-z` 的小写字母，以及字符 `?` 和 `*`。

Tips

1. 使用动态规划算法
2. `match[i][j]` 表示当前 `p` 的前 `i` 个字符和 `s` 的前 `j` 个字符是否匹配
3. 状态转移：
   - 如果 `p[i - 1] == s[j - 1]` 或 `p[i - 1] == '?'`，表明当前字符匹配，`match[i][j]` 可以从 `match[i - 1][j - 1]` 转移而来。
   - 如果 `p[i - 1] == '*'`,这个位置可以匹配 0 到若干个字符，`match[i][j]` 从其前一个状态转移来。其前一个状态包括 `match[i - 1][j]`（表示模式串的前 i-1 个字符与字符串的前 j 个字符的匹配状态），`match[i][j - 1]`（表示模式串的前 i 个字符与字符串的前 j-1 个字符的匹配状态）
4. 初始条件：`match[0][0]` 表示两个空串匹配。

若模式串以若干个 * 开头，这些 * 可以匹配空串。

通配符匹配

```java
class Solution {
    public boolean isMatch(String s, String p) {
        boolean[][] match = new boolean[p.length() + 1][s.length() + 1];
        match[0][0] = true;
        for(int i = 1; i <= p.length();i++){
            if(p.charAt(i - 1) != '*'){
                break;
            }
            match[i][0] = true;
        }
        for (int i = 1; i <= p.length(); i++) {
            for (int j = 1; j <= s.length(); j++) {
                if (s.charAt(j - 1) == p.charAt(i - 1) || p.charAt(i - 1) == '?'
                    ){
                    match[i][j] = match[i - 1][j - 1];
                }else if (p.charAt(i - 1) == '*'){
                    match[i][j] = match[i-1][j] || match[i][j-1];
                }
            }
        }
        return match[p.length()][s.length()];
    }
}
```