连续数列

一、题目

给定一个整数数组（有正数有负数），找出总和最大的连续数列，并返回总和。

示例：

输入： [-2,1,-3,4,-1,2,1,-5,4]
输出： 6
解释： 连续子数组 [4,-1,2,1] 的和最大，为 6。

二、解决方法

1、自制垃圾算法

```
class Solution {
   public static int maxSubArray(int[] nums) {
        int len = nums.length;
        int[] sums=new int[len];
        int max=Integer.MIN_VALUE;
        int i,j;
        if(len==1) return nums[0];
        for( i=0;i<len;i++){
            int temp=Integer.MIN_VALUE;
            int Besum=0;
            for( j=i;j<len;j++){
                Besum=Besum+nums[j];
                temp=Math.max(temp,Besum);
            }
            sums[i]=temp;
        }

        for(int k=0;k<len;k++){
            if(max<=sums[k])
                max=sums[k];
        }
        return max;
    }

}
```

2、动态规划

```
class Solution {
    public int maxSubArray(int[] nums) {
        int len = nums.length;
        if (len == 0)
            return 0;
        if(len == 1)
            return nums[0];
        int[] dp = new int[len];
        //初始状态
        //dp[i]表示从左到右，包含元素nums[i]的最大和
        dp[0] = nums[0];
        int max = dp[0];
        for (int i = 1; i < len; i++) {
            dp[i] = Math.max(nums[i] + dp[i-1],nums[i]);
        }
        //寻找最大值
        for (int i = 1; i < len; i++) {
            if (dp[i] > max)
                max = dp[i];
        }
        return max;
    }
}
```

3、分治算法

```
class Solution {
    int [] nums;

    public int maxSubArray(int[] nums) {
        this.nums=nums;
        return MaxSum(0,nums.length-1);
        }

 //   此函数计算nums[left]到nums[right]之间的最大连续总和最大连续总和只可能出现在数组的左边，或者右边，或者中间
    private int MaxSum(int left,int right){
        if(left==right)
            return nums[left];

        int mid=(left+right)/2;
        int leftMaxSum=MaxSum(left,mid); //左边部分最大连续总和
        int rightMaxSum=MaxSum(mid+1,right); ////右边部分最大连续总和

        //下面计算中间部分最大连续总和
```

```
        int sum=0;
        int leftBorderMax=Integer.MIN_VALUE;
        for(int i=mid;i>=left;i--){ //从中间往左边延伸,找到左边边界最大和
            sum+=nums[i];
            leftBorderMax=Math.max(leftBorderMax,sum);
        }
        sum=0;
        int rightBorderMax=Integer.MIN_VALUE;
        for(int i=mid+1;i<=right;i++){ //从中间往右边延伸,找到右边边界最大和
            sum+=nums[i];
            rightBorderMax=Math.max(rightBorderMax,sum);
        }
        //返回左边，右边，中间之中的最大值
        return Math.max(leftMaxSum,Math.max(rightMaxSum,
leftBorderMax+rightBorderMax));
    }
}
```