1_最大子列和问题(在线处理算法)

题目

给你一个整数数组 nums ， 请你找出一个具有最大和的连续子数组（子数组最少包含一个元素），返回其最大和。

子数组是数组中的一个连续部分。

示例 1：

```
输入：nums = [-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 。
```

示例 2：

```
输入：nums = [1]
输出：1
```

示例 3：

```
输入：nums = [5,4,-1,7,8]
输出：23
```

提示：

```
1 <= nums.length <= 105
-104 <= nums[i] <= 104
```

进阶：如果你已经实现复杂度为 O(n) 的解法，尝试使用更为精妙的 分治法 求解。

算法

1

```
int MaxSubseqSum1(int num[], int n)
{
    int ThisSum, MaxSum = 0;
    int i, j, k;
    for (i = 0; i < n; i++) //i是子列左端点
    {
        for (j = 1; j < n; j++) //j是子列左端点
        {
            ThisSum = 0;
            for (k = i; k <= j; k++)
            {
                ThisSum += num[k];
            }
            if (ThisSum > MaxSum)
            {
```

```
                MaxSum = ThisSum;
            }
        }
    }
    return 0;
}
```

时间复杂度

$$T(N) = O(N^3)$$

2

对于相同的 `i`，不同的 `j`，只要在 `j-1` 次循环的基础上累加一项即可。

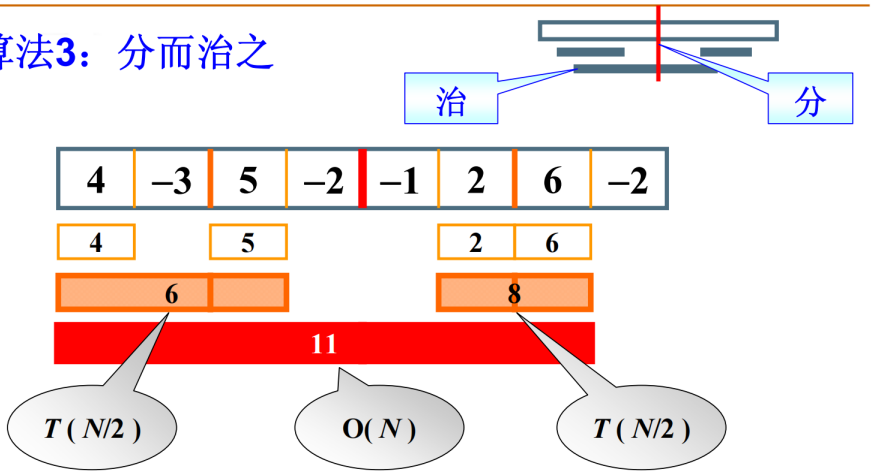
```
int MaxSubseqSum1(int num[], int n)
{
    int ThisSum, MaxSum = 0;
    int i, j, k;
    for (i = 0; i < n; i++) //i是子列左端点
    {
        ThisSum = 0;
        for (j = i; j < n; j++) //j是子列左端点
        {
            ThisSum += num[j];
            if(ThisSum > MaxSum)
            {
                MaxSum = ThisSum;
            }
        }

    }
    return MaxSum;
}
```

时间复杂度

$$T(N) = O(N^2)$$

3 递归、分而治之

算法3：分而治之



$$T(N) = 2\,T(N/2) + c\,N, \quad T(1) = O(1)$$
$$= 2\,[2\,T(N/2^2) + c\,N/2] + c\,N$$
$$= 2^k\,O(1) + c\,k\,N \quad \text{其中 } N/2^k = 1$$
$$= O(N \log N)$$

```
int Max3( int A, int B, int C )
{ /* 返回3个整数中的最大值 */
    return A > B ? (A > C ? A : C):(B > C ? B : C);
}

int DivideAndConquer( int List[], int left, int right )
{ /* 分治法求List[left]到List[right]的最大子列和 */
    int MaxLeftSum, MaxRightSum; /* 存放左右子问题的解 */
    int MaxLeftBorderSum, MaxRightBorderSum; /*存放跨分界线的结果*/

    int LeftBorderSum, RightBorderSum;
    int center, i;

    if( left == right )
    { /* 递归的终止条件，子列只有1个数字 */
        if( List[left] > 0 )
            return List[left];
        else
            return 0;
    }

    /* 下面是"分"的过程 */
    center = ( left + right ) / 2; /* 找到中分点 */
    /* 递归求得两边子列的最大和 */
    MaxLeftSum = DivideAndConquer( List, left, center );
    MaxRightSum = DivideAndConquer( List, center+1, right );

    /* 下面求跨分界线的最大子列和 */
    MaxLeftBorderSum = 0;
    LeftBorderSum = 0;
    for( i = center; i >= left; i-- )
    { /* 从中线向左扫描 */
        LeftBorderSum += List[i];
        if( LeftBorderSum > MaxLeftBorderSum )
            MaxLeftBorderSum = LeftBorderSum;
    } /* 左边扫描结束 */

    MaxRightBorderSum = 0; RightBorderSum = 0;
```

```
    for( i=center+1; i<=right; i++ )
    { /* 从中线向右扫描 */
        RightBorderSum += List[i];
        if( RightBorderSum > MaxRightBorderSum )
            MaxRightBorderSum = RightBorderSum;
    } /* 右边扫描结束 */

    /* 下面返回"治"的结果 */
    return Max3( MaxLeftSum, MaxRightSum, MaxLeftBorderSum + MaxRightBorderSum );
}

int MaxSubseqSum3( int List[], int N )
{ /* 保持与前2种算法相同的函数接口 */
    return DivideAndConquer( List, 0, N-1 );
}

```

4 在线处理

```
int MaxSubseSum4(int num[], int n)
{
    int ThisSum, MaxSum;
    int i;
    ThisSum = MaxSum = 0;
    for (i = 0; i < n; i++)
    {
        ThisSum += num[i];      //向右累加
        if (ThisSum > MaxSum)
        {
            MaxSum = ThisSum;   //发现更大的则更新当前结果
        }
        else if(ThisSum < 0)  //如果发现当前子列和为负
        {
            ThisSum = 0;
        }
    }
    return MaxSum;
}
```

$$T(N) = O(N)$$