最大的子序列和问题：给定的整数，求的最大值。

例：max\_subsequence\_sum\_test

// 穷举法求最大子序列和

int MaxSubsequenceSum1(const vector<int>& array)

{

int max\_sub\_sum = INT\_MIN;

for (int i = 0; i != array.size(); ++i)

{

for (int j = i; j != array.size(); ++j)

{

int sum = 0;

for (int k = i; k <= j; ++k)

{

sum += array[k];

}

if (sum > max\_sub\_sum)

{

max\_sub\_sum = sum;

}

}

}

return max\_sub\_sum;

}

其算法的时间复杂度为：

在算法中，有3个for循环，其中第一个for循环为，第二个for循环为，最坏情况下为，第3个for循环为，也可能为。

// 改进后的穷举法求最大子序列和

int MaxSubsequenceSum2(const vector<int>& array)

{

int max\_sub\_sum = INT\_MIN;

for (int i = 0; i != array.size(); ++i)

{

int sum = 0;

for (int j = i; j != array.size(); ++j)

{

sum += array[j];

if (sum > max\_sub\_sum)

{

max\_sub\_sum = sum;

}

}

}

return max\_sub\_sum;

}

改进后的算法的时间复杂度为，其有两个for循环，第一个为，第二个最坏也为。

用递归的方式求解

// 递归求解

// 求个整数的最大值

int Max3(int a, int b, int c)

{

if (a < b)

{

a = b;

}

if (a < c)

{

a = c;

}

return a;

}

int MaxSumRec(const vector<int>& array, int left, int right)

{

if (left == right) // 基准情形

{

return array[left];

}

int center = static\_cast<int>((left + right) / 2);

// 左半部分最大子序列和

int max\_left\_sum = MaxSumRec(array, left, center);

// 右半部分最大子序列和

int max\_right\_sum = MaxSumRec(array, center + 1, right);

// 跨数据中部从而占据左右两半部分的最大子序列和

// 这个子序列包含左半部分的最后一个元素以及

// 右半部分的第一个元素

int max\_left\_border\_sum = INT\_MIN;

int left\_border\_sum = 0;

for (int i = center; i >= left; --i)

{

left\_border\_sum += array[i];

if (left\_border\_sum > max\_left\_border\_sum)

{

max\_left\_border\_sum = left\_border\_sum;

}

}

int max\_right\_border\_sum = INT\_MIN;

int right\_border\_sum = 0;

for (int j = center + 1; j <= right; ++j)

{

right\_border\_sum += array[j];

if (right\_border\_sum > max\_right\_border\_sum)

{

max\_right\_border\_sum = right\_border\_sum;

}

}

return Max3(max\_left\_sum, max\_right\_sum, max\_right\_border\_sum + max\_left\_border\_sum);

}

int MaxSubsequenceSum3(const vector<int>& array)

{

return MaxSumRec(array, 0, array.size() - 1);

}

递归的思想是：要么最大子序列出现在数组的左半部分，要么出现在右半部分，要么出现在跨数据中部包含左右两半部分。

出现在左半部分和右半部分可以用递归计算，而跨中间部分的最大子序列需要包含左半部分的最后一个元素和右半部分的第一个元素。

递归算法的时间复杂度为：（以2为底）

推导过程如下：

令是求解大小为的最大子序列和的时间



当时，会触发递归调用，以及之后的两个for循环。

两个递归花费的时间为，而递归中的for循环花费的时间为，即刚进入递归时，left=0，right=。



根据：



若，则