### 实验一：最大子段和问题不同算法的时间消耗测试

**题目描述：**

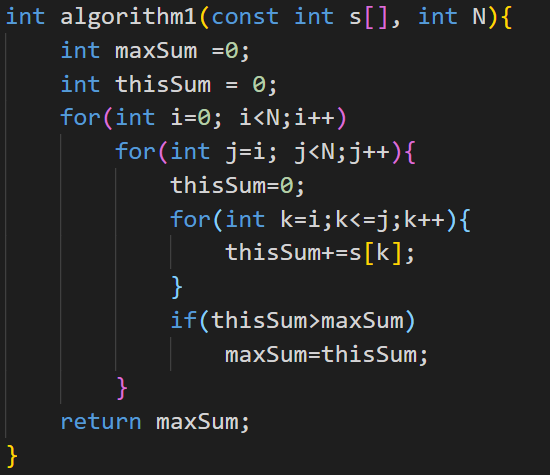
给定由n个整数组成的序列，求该序列子段和的最大值。当所有整数均为负整数时定义其最大子段和为0

**算法一：穷举法——时间复杂度（）**

**算法思想：**

穷举出[,]区间上所有取值，取最大值后与0进行比较

**代码：**



**算法二：优化的穷举法——时间复杂度（）**

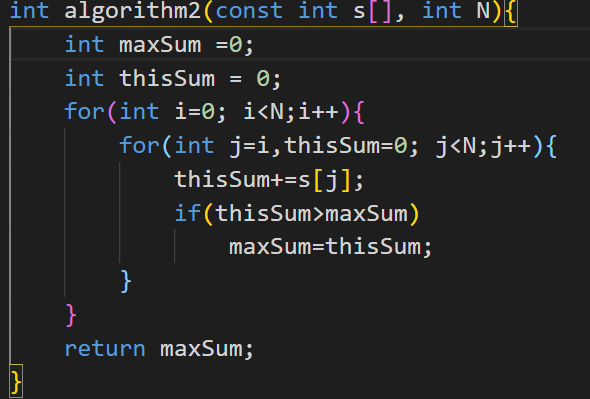
**算法思想：**

穷举出[,]区间上所有取值，取最大值后与0进行比较

**优化部分：**

在选取区间的时候，直接累加

**代码：**



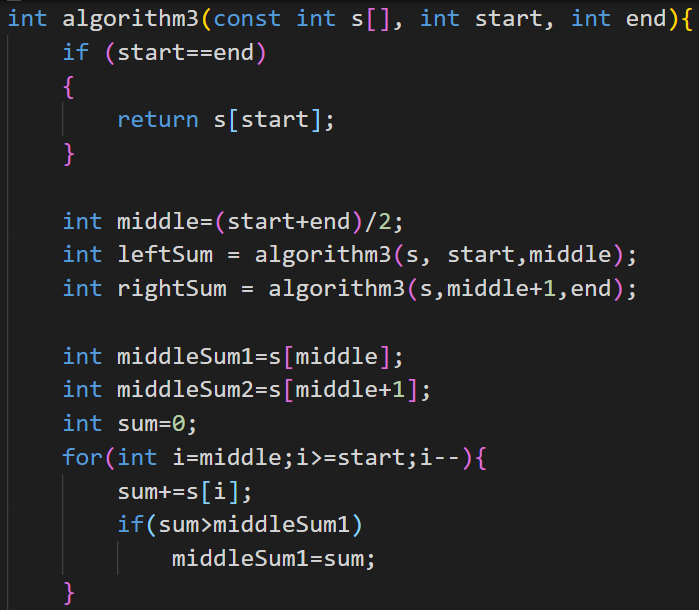
**算法三：分治算法——时间复杂度（）**

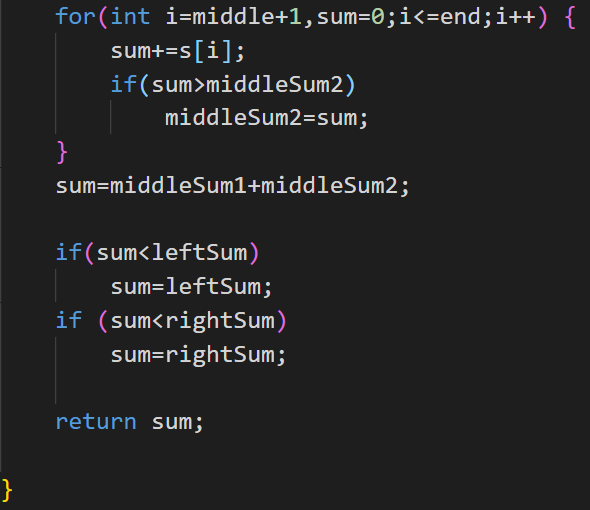
**算法思想：**

最大子段和分三种情况：

1. 在左子段中取得
2. 在右子段中取得
3. 跨越左右子段

**代码：**





**算法四：优化的动态规划算法——时间复杂度（）**

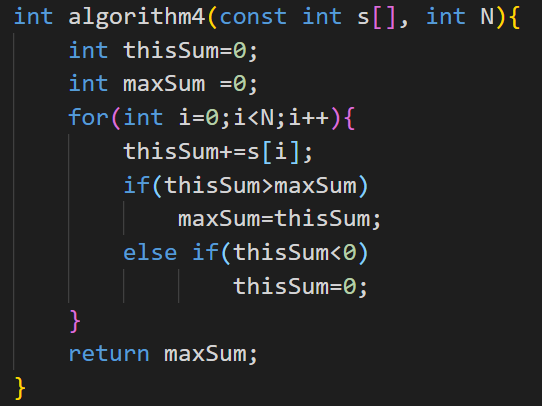
**算法思想：**



**优化部分：**

不进行存储

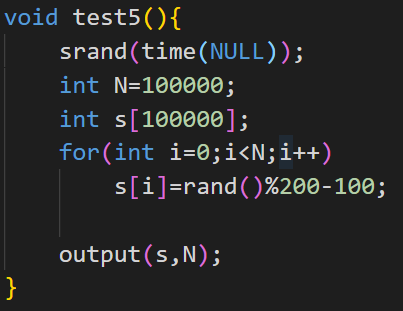
**代码：**



**测试数据生成：**

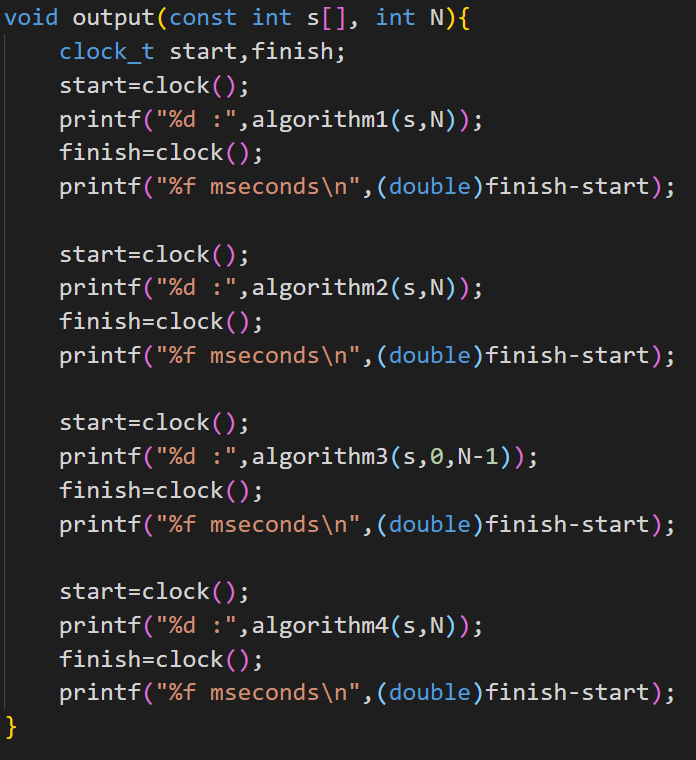
使用函数随机生成数据

例：生成100,000组数据



**输出函数：**

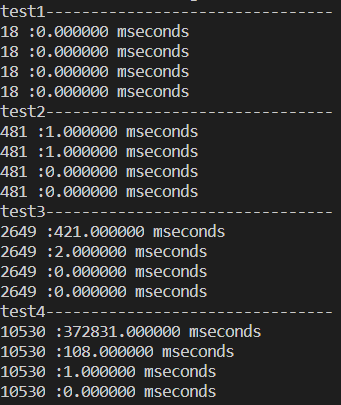
输出每个算法所消耗的时间



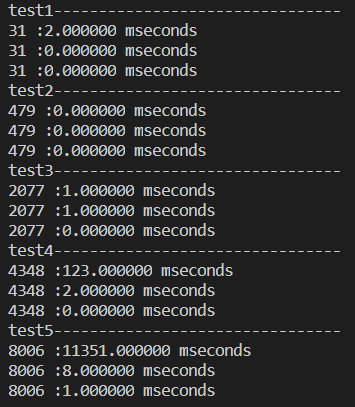
**输出结果：**

因为算法一的时间复杂度太高，所以

1. 在包含算法一的情况下给出每个算法10，100，1000，10000个整数的消耗时间：



1. 在不包含算法一的情况下给出每个算法10，100，1000，10000，100000个整数的消耗时间：



**结果分析：**

由以上输出可直观地看出，随着数据规模不断增大，时间复杂度高的算法消耗的时间越长，算法一在10000个整数时需要近6分钟时间，而算法四100000个整数时消耗的时间仍然很短

**实验总结和体会：**

经过这次实验，我感受到了算法时间复杂度的重要性，一个低时间复杂度的算法在数据规模很大的情况下效率是很高的，对于大型企业来说，一个算法的改进会带来效率很大的提升。

**附录：**

源代码：

#include <cstdio>

#include <stdlib.h>

#include <time.h>

using namespace std;

int algorithm1(const int s[], int N){

int maxSum =0;

int thisSum = 0;

for(int i=0; i<N;i++)

for(int j=i; j<N;j++){

thisSum=0;

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

thisSum+=s[k];

}

if(thisSum>maxSum)

maxSum=thisSum;

}

return maxSum;

}

int algorithm2(const int s[], int N){

int maxSum =0;

int thisSum = 0;

for(int i=0; i<N;i++){

for(int j=i,thisSum=0; j<N;j++){

thisSum+=s[j];

if(thisSum>maxSum)

maxSum=thisSum;

}

}

return maxSum;

}

int algorithm3(const int s[], int start, int end){

if (start==end)

{

return s[start];

}

int middle=(start+end)/2;

int leftSum = algorithm3(s, start,middle);

int rightSum = algorithm3(s,middle+1,end);

int middleSum1=s[middle];

int middleSum2=s[middle+1];

int sum=0;

for(int i=middle;i>=start;i--){

sum+=s[i];

if(sum>middleSum1)

middleSum1=sum;

}

for(int i=middle+1,sum=0;i<=end;i++) {

sum+=s[i];

if(sum>middleSum2)

middleSum2=sum;

}

sum=middleSum1+middleSum2;

if(sum<leftSum)

sum=leftSum;

if (sum<rightSum)

sum=rightSum;

return sum;

}

int algorithm4(const int s[], int N){

int thisSum=0;

int maxSum =0;

for(int i=0;i<N;i++){

thisSum+=s[i];

if(thisSum>maxSum)

maxSum=thisSum;

else if(thisSum<0)

thisSum=0;

}

return maxSum;

}

void output(const int s[], int N){

clock\_t start,finish;

start=clock();

printf("%d :",algorithm1(s,N));

finish=clock();

printf("%f mseconds\n",(double)finish-start);

start=clock();

printf("%d :",algorithm2(s,N));

finish=clock();

printf("%f mseconds\n",(double)finish-start);

start=clock();

printf("%d :",algorithm3(s,0,N-1));

finish=clock();

printf("%f mseconds\n",(double)finish-start);

start=clock();

printf("%d :",algorithm4(s,N));

finish=clock();

printf("%f mseconds\n",(double)finish-start);

}

void test1(){

srand(time(NULL));

int N=10;

int s[10];

for(int i=0;i<N;i++)

s[i]=rand()%20-10;

output(s,N);

}

void test2(){

srand(time(NULL));

int N=100;

int s[100];

for(int i=0;i<N;i++)

s[i]=rand()%200-100;

output(s,N);

}

void test3(){

srand(time(NULL));

int N=1000;

int s[1000];

for(int i=0;i<N;i++)

s[i]=rand()%200-100;

output(s,N);

}

void test4(){

srand(time(NULL));

int N=10000;

int s[10000];

for(int i=0;i<N;i++)

s[i]=rand()%200-100;

output(s,N);

}

void test5(){

srand(time(NULL));

int N=100000;

int s[100000];

for(int i=0;i<N;i++)

s[i]=rand()%200-100;

output(s,N);

}

int main(){

printf("test1--------------------------------\n");

test1();

printf("test2--------------------------------\n");

test2();

printf("test3--------------------------------\n");

test3();

printf("test4--------------------------------\n");

test4();

printf("test5--------------------------------\n");

test5();

}