《算法设计与分析》第一次作业

一、题目

分析PPT中求解最大字段和的四种算法效率。

二、模块设计

2.1 数据生成模块

输入	输出	功能
需要生成随机数列的数量N	生成的数列指针	分配内存空间，并生成随机数

2.2 算法模块

输入	输出	功能
待求数列	完成计算后的时间	使用相应算法完成最大字段和的计算

2.3 时间模块

输入	输出	功能
无	当前时间（us）	求出当前时间（精确到微秒）

2.4 主模块

输入	输出	功能
无	四种算法在不同N下对应的计算时间	求出不同数据规模下四种算法的效率

三、软件设计

3.1 数据生成模块

DataCreate.h

```
class DataCreate
{
public:
    DataCreate(int);
    ~DataCreate();
    int* GetData();
private:
    int N;  //需要随机生成的数据量
    int* data;  //随机生成的数据数组
};
```

DataCreate.cpp

```cpp
#include "DataCreate.h"
#include <cstdlib>
#include <ctime>

DataCreate::DataCreate(int n)
{
    N = n;
    data = new int[N];
}

int* DataCreate::GetData()
{
    srand((int)time(0));  // 产生随机种子
    for (int i = 0; i < N; i++)
    {
        data[i] = (rand() % 200) - 100; //产生 -100~99 的随机数
    }
    return data;
}

DataCreate::~DataCreate()
{
    delete[] data;
}
```

3.2 算法模块

Algorithm1.h

```cpp
#pragma once
#include "ustime.h"
class Algorithm1
{
public:
    Algorithm1(int, int*);
    ~Algorithm1();
    long long Run();
private:
    int N;
    int* Data;
    long long RunTime;
};
```

Algorithm1.cpp

```cpp
#include "Algorithm1.h"
#include <cstdlib>
#include <ctime>

Algorithm1::Algorithm1(int n, int* data)
{
    N = n;
    Data = data;
}
```

```
long long Algorithm1::Run()
{
    long long StartTime, EndTime;
    StartTime = GetSysTimeMicros();
    long long ThisSum, MaxSum;
    int i, j, k;
    MaxSum = 0;
    for (i = 0; i < N; i++)
    for (j = i; j < N; j++) {
        ThisSum = 0;
        for (k = i; k <= j; k++)
            ThisSum += Data[k];
            if (ThisSum > MaxSum)
                MaxSum = ThisSum;
        }
    EndTime = GetSysTimeMicros();
    RunTime = EndTime - StartTime;
    return RunTime;
}

Algorithm1::~Algorithm1()
{

}
```

Algorithm2.h

```
#pragma once
#include "ustime.h"
class Algorithm2
{
public:
    Algorithm2(int, int*);
    ~Algorithm2();
    long long Run();
private:
    int N;
    int* Data;
    long long RunTime;
};
```

Algorithm2.cpp

```
#include "Algorithm2.h"
#include <cstdlib>
#include <ctime>

Algorithm2::Algorithm2(int n, int* data)
{
    N = n;
    Data = data;
}

long long Algorithm2::Run()
{
```

```
    long long StartTime, EndTime;
    StartTime = GetSysTimeMicros();
    long long ThisSum, MaxSum;
    int i, j, k;
    MaxSum = 0;
    for (i = 0; i < N; i++) {
        ThisSum = 0;
        for (j = i; j < N; j++) {
            ThisSum += Data[j];
            if (ThisSum > MaxSum)
            MaxSum = ThisSum;
        }
    }
    EndTime = GetSysTimeMicros();
    RunTime = EndTime - StartTime;
    return RunTime;
}

Algorithm2::~Algorithm2()
{

}
```

Algorithm3.h

```
#include "ustime.h"
class Algorithm3
{
public:
    Algorithm3(int, int*);
    ~Algorithm3();
    long long Run();
    long long Work(int, int);
private:
    int N;
    int* Data;
    long long RunTime;
};
```

Algorithm3.cpp

```
#include "Algorithm3.h"
#include <cstdlib>
#define gmax(_a, _b) ((_a) > (_b) ? (_a) : (_b))

Algorithm3::Algorithm3(int n, int* data)
{
    N = n;
    Data = data;
}

long long Algorithm3::Run()
{
    long long StartTime, EndTime;
```

```
    StartTime = GetSysTimeMicros();
    Work(1, N);
    EndTime = GetSysTimeMicros();
    RunTime = EndTime - StartTime;
    return RunTime;
}

long long Algorithm3::Work(int left, int right)
{
    if (right == left)
        return gmax(Data[left], 0);

    int center = (left + right) / 2;
    long long LeftMax = Work(left, center);
    long long RightMax = Work(center + 1, right);
    long long sum = 0;
    long long left_max = 0;
    for (int i = center; i >= left; i--)
    {
        sum += Data[i];
        left_max = gmax(left_max, sum);
    }
    sum = 0;
    long long right_max = 0;
    for (int i = center + 1; i <= right; i++)
    {
        sum += Data[i];
        right_max = gmax(right_max, sum);
    }
    return gmax(gmax(LeftMax, RightMax), left_max + right_max);
}
```

Algorithm4.h

```
#pragma once
#include "ustime.h"
class Algorithm4
{
public:
    Algorithm4(int, int*);
    ~Algorithm4();
    long long Run();
private:
    int N;
    int* Data;
    long long RunTime;
};
```

Algorithm4.cpp

```
#include "Algorithm4.h"

#include <cstdlib>
#include <ctime>
```

```
Algorithm4::Algorithm4(int n, int* data)
{
    N = n;
    Data = data;
}

long long Algorithm4::Run()
{
    long long StartTime, EndTime;
    StartTime = GetSysTimeMicros();
    long long ThisSum, MaxSum;
    int i, j;
    ThisSum = MaxSum = 0;
    for (j = 0; j < N; j++) {
        ThisSum += Data[j];
        if (ThisSum > MaxSum)
            MaxSum = ThisSum;
        else if (ThisSum < 0)
            ThisSum = 0;
    }
    EndTime = GetSysTimeMicros();
    RunTime = EndTime - StartTime;
    return RunTime;
}

Algorithm4::~Algorithm4()
{

}
```

3.3 时间模块

ustime.h

```
#ifdef _WIN32
#include <windows.h>
#else
#include <time.h>
#endif  // _WIND32

// 定义64位整形
#if defined(_WIN32) && !defined(CYGWIN)
typedef __int64 int64_t;
#else
typedef long long int64t;
#endif  // _WIN32

int64_t GetSysTimeMicros();
```

ustime.cpp

```
#include "ustime.h"
```

```
// 获取系统的当前时间，单位微秒(us)
int64_t GetSysTimeMicros()
{
#ifdef _WIN32
    // 从1601年1月1日0:0:0:000到1970年1月1日0:0:0:000的时间(单位100ns)
#define EPOCHFILETIME   (116444736000000000UL)
    FILETIME ft;
    LARGE_INTEGER li;
    int64_t tt = 0;
    GetSystemTimeAsFileTime(&ft);
    li.LowPart = ft.dwLowDateTime;
    li.HighPart = ft.dwHighDateTime;
    // 从1970年1月1日0:0:0:000到现在的微秒数(UTC时间)
    tt = (li.QuadPart - EPOCHFILETIME) / 10;
    return tt;
#else
    timeval tv;
    gettimeofday(&tv, 0);
    return (int64_t)tv.tv_sec * 1000000 + (int64_t)tv.tv_usec;
#endif // _WIN32
    return 0;
}
```

3.4 主模块

Homework1.cpp

```
#include "DataCreate.h";
#include "Algorithm1.h"
#include "Algorithm2.h"
#include "Algorithm3.h"
#include "Algorithm4.h"
#include <iostream>

using namespace std;
const int test[4][7] = { {10, 100, 1000, 10000, 0, 0, 0},
                         {10, 100, 1000, 10000, 100000, 1000000, 0},
                         {10, 100, 1000, 10000, 100000, 1000000, 10000000},
                         {10, 100, 1000, 10000, 100000, 1000000, 10000000} };

int main()
{
    freopen("out.txt", "w", stdout);
    cout << "Algo1" << endl;
    for (int n = 1000; n <= 10000000; n+= 1000)
    {
        DataCreate RandData(n);
        Algorithm1 Algo(n, RandData.GetData());
        int RunTime = Algo.Run();
        if (RunTime > 500000000) break;
        cout << n << "," << RunTime / 1000000.0 << endl;
    }
    cout << "Algo2" << endl;
    for (int n = 1000; n <= 10000000; n += 1000)
    {
        DataCreate RandData(n);
```

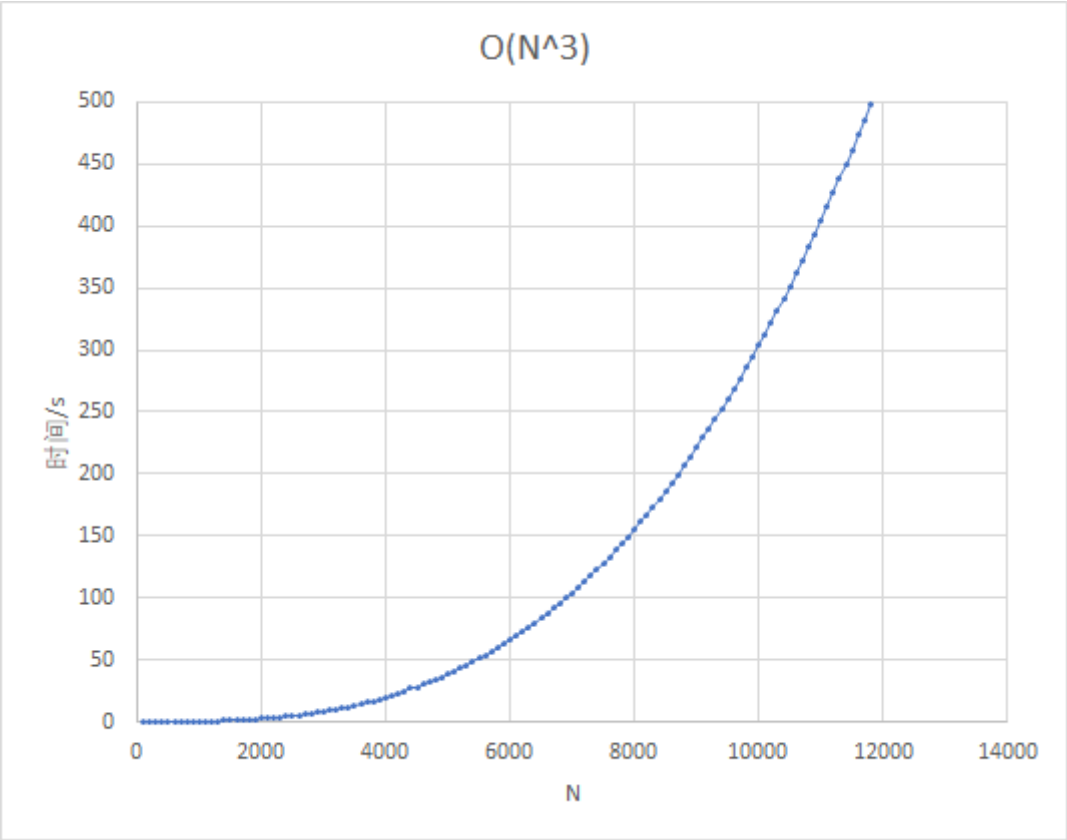
```
        Algorithm2 Algo(n, RandData.GetData());
        int RunTime = Algo.Run();
        if (RunTime > 500000000) break;
        cout << n << "," << RunTime / 1000000.0 << endl;
    }
    cout << "Algo3" << endl;
    for (int n = 1000; n <= 10000000; n += 1000)
    {
        DataCreate RandData(n);
        Algorithm3 Algo(n, RandData.GetData());
        long long RunTime = Algo.Run();
        if (RunTime > 500000000) break;
        cout << n << "," << RunTime / 1000000.0 << endl;
    }
    cout << "Algo4" << endl;
    for (int n = 1000; n <= 10000000; n += 1000)
    {
        DataCreate RandData(n);
        Algorithm4 Algo(n, RandData.GetData());
        long long RunTime = Algo.Run();
        if (RunTime > 500000000) break;
        cout << n << "," << RunTime / 1000000.0 << endl;
    }
}
```

四、运行结果

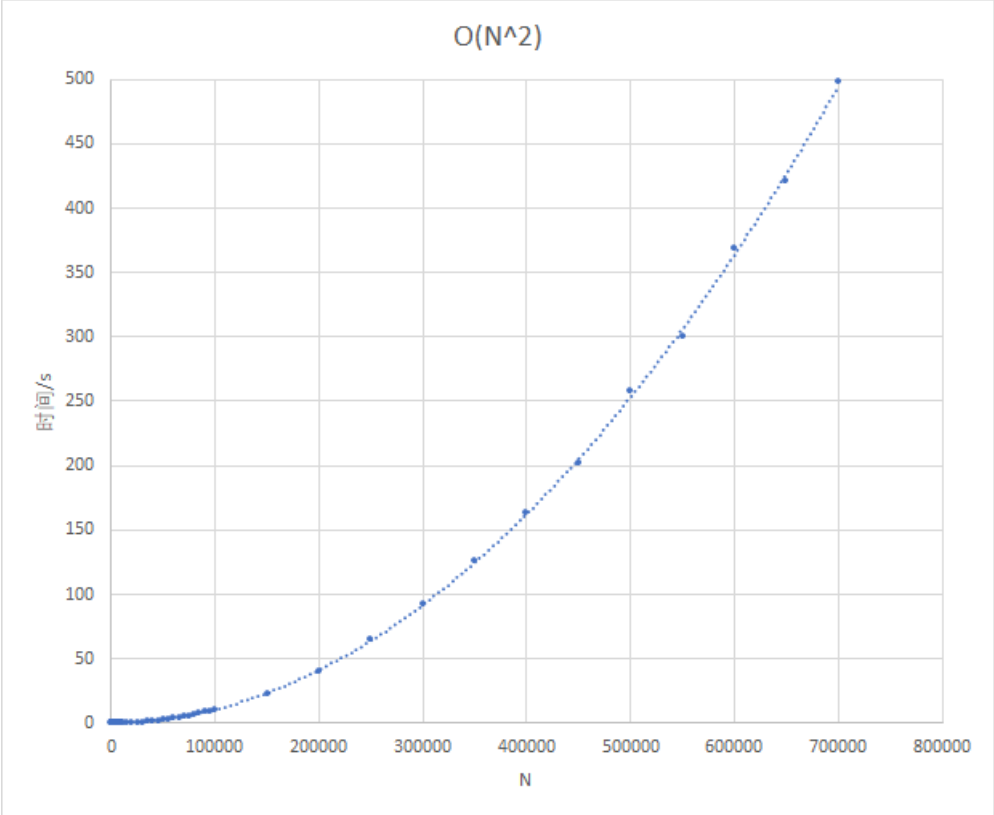
4.1 概括

	$O(N^3)$	$O(N^2)$	$O(NlogN)$	$O(N)$
N=10	0.000001	0.000001	0.000001	0.000001
N=100	0.000001	0.000001	0.000001	0.000001
N=1000	0.320679	0.000998	0.000001	0.000001
N=10000	304.746	0.109557	0.000997	0.000001
N=100000	NA	10.4	0.010998	0.000855
N=1000000	NA	1020.52	0.119684	0.002968
N=10000000	NA	NA	1.28261	0.033904

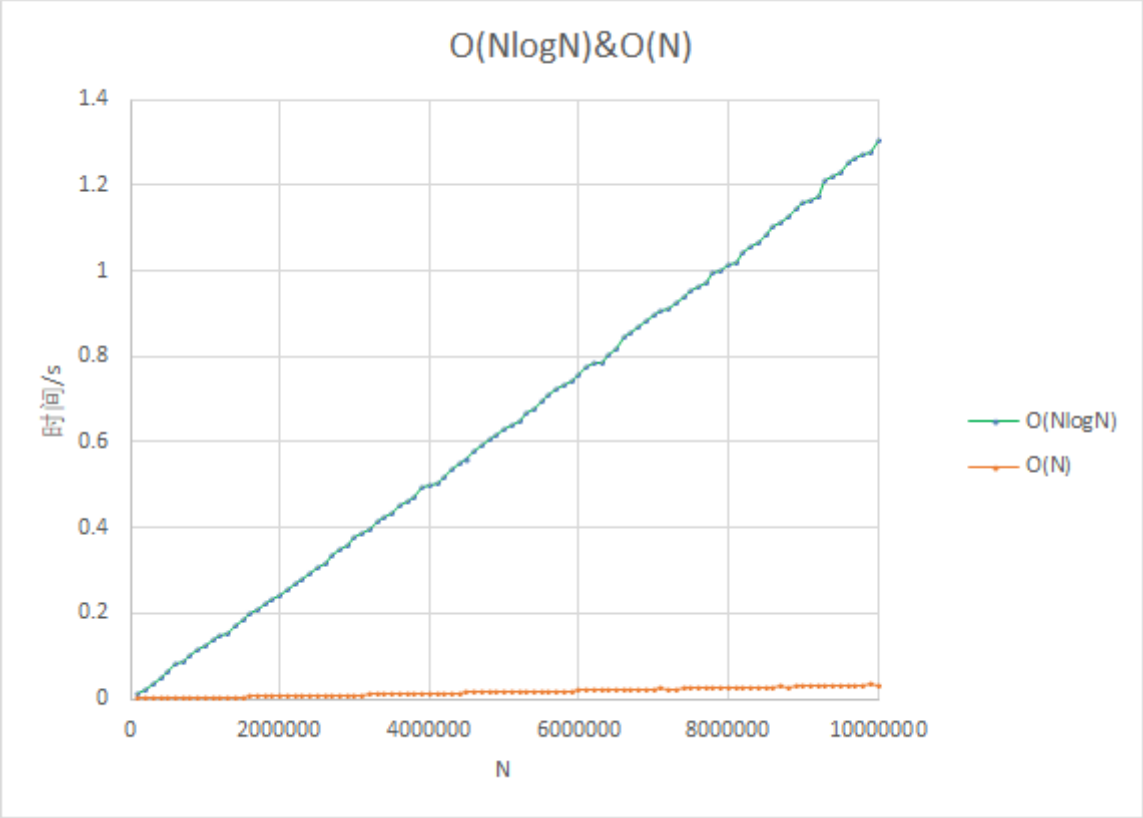
4.2 $O(N^3)$



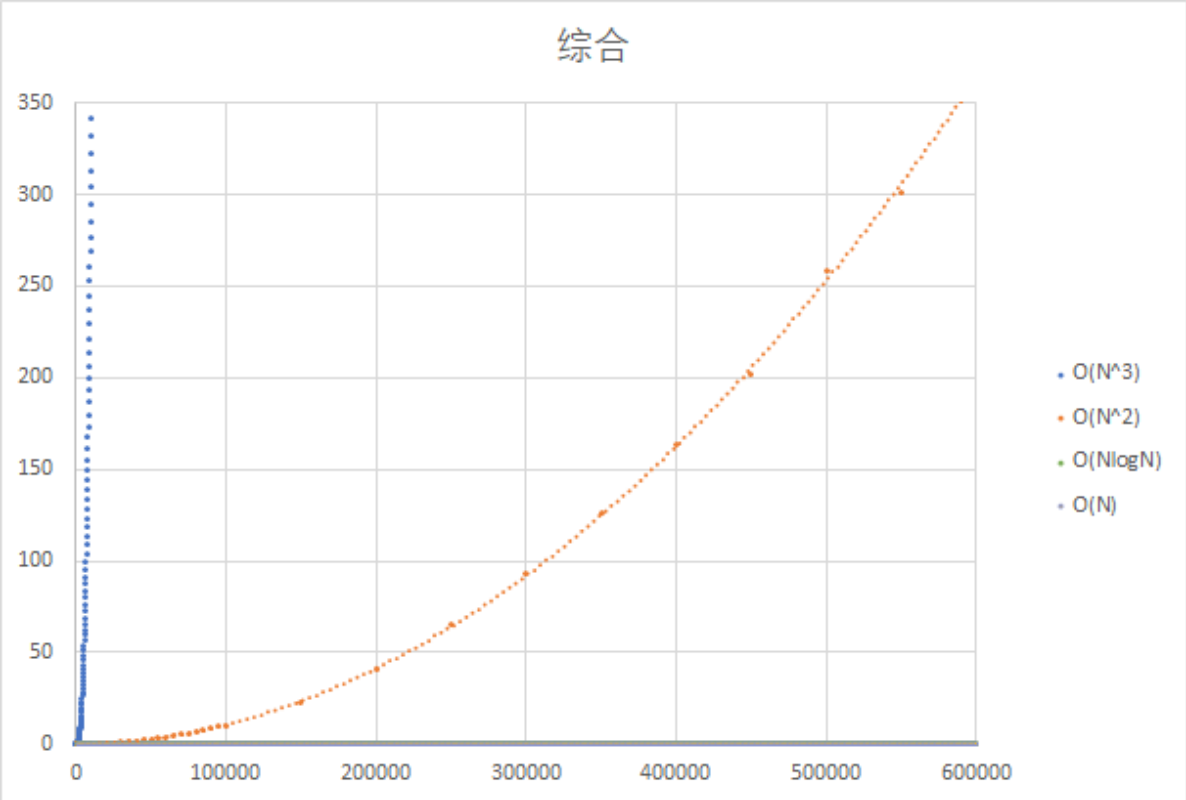
4.3 $O(N^2)$



4.4 $O(NlogN)\&O(N)$



4.5 综合



五、结果分析

1. $O(N^3)$算法的时间开销图基本符合$f(x)=cx^3$的图像，$O(N^2)$算法的时间开销图也基本符合$f(x)=cx^2$的图像，且$O(N^3)$的增长速度快于$O(N^2)$，都符合预期。
2. 与$O(N^2),O(N^3)$相比，$O(NlogN)$与$O(N)$算法所花费时间几乎可以忽略不计。在较大数据规模范围下$O(NlogN)$与$O(N)$算法的性能远高于前两者。

3. 在$O(NlogN)$&$O(N)$时间开销图中，可以看到两条函数线都大致呈线性状态，事实上这并不符合我们对应$O(NlogN)$的预期。推测由于我取的数据范围与数据间隔都过大，在较大数据规模下$O(NlogN)$呈现出线性。因此，需要在较小数据范围与数据间隔下，重新计算并绘图。与此同时，为了避免数据规模较小时出现的统计的误差，需要采用精度更高的计时方式，并多次运算取平均值。

六、心得体会

在本次最大字段和算法效率的探究实验中，我通过编写数据生成程序、调用程序、计时程序、算法程序，将他们综合起来使用，对探究算法效率有了一个初步的体验。同时，通过探索$O(N^3), O(N^2), O(NlogN), O(N)$四种不同复杂度算法的运行时间，我对于算法复杂度对于运行效率的影响有了一个更为清晰的认知。