算法设计与分析第二章上机作业

班级：2018211313 班　　学号：2018211366　　姓名：蒋潇逸

版本：1.0

更新：*November 6, 2020*

本文档是算法设计与分析第二章上机作业，具体代码详见附录。

目录

1 排序算法（作业 1、2）

1.1 题目描述

分析不同的问题规模、输入对归并排序和快速排序算法运行时间的影响，实验中考察以下两点。(n 为序列长度)

- 不同问题规模 n 下，算法/程序运行时间
- 相同问题规模 n 下，算法/程序针对不同的输入 I——具有不同的 $DD(n)$ 的运行时间

1.2 实验步骤以及要求

- 取 N=30,000，生成长度为 n、最大值 $x_{n-1} \le N$ 的 L 组序列 $SEQ(n,N)$，每组序列 $SEQ(n,N)$ 中有 M 个长度均为 n、组成序列的数字相同，但数字间顺序不同的序列。要求：
  - L=6 种不同长度，n=2,000，5,000，10,000，15,000，20,000，30,000
  - M=5，即每组有 5 个长度相同的序列 $SEQ(n,N)$
  - 对同组内长度相同的序列，ADD、DD 值不相同
- 对全部 $L*M=30$ 个序列 $SEQ(n,N)$，采用递归合并排序，非递归合并排序, 快速排序算法 1，快速排序算法 2。对序列 $SEQ(n,N)$ 从小到大进行排序，要求:
  - 用表格记录每个序列的长度、ADD、DD、排序时间
  - 对递归算法，观察统计递归层次
- 针对 $L=6$ 种不同序列长度，从表中选 6 行，考察对同一输入 $SEQ(n,N)$，考察问题规模 n、DD 相同时，四种排序算法运行时间 $T(n,I)$ 差异
- 观察在长度 n 相同的同一组内，同一算法的运行时间随 ADD、DD 增长的变化情况
- 考察问题规模 n 对算法运行时间的影响针对 $L=6$ 种不同序列长度 n=2000, 5000, 10000, 15000, 20000, 30000
  - 统计 6 组相同长度的输入序列的平均 $avgDD$、$avgADD$
  - 计算四种算法对每组中的 M 个序列进行排序的平均时间 $avgT(n)$
  - 观察同一算法的 $avgT(n)$ 随 n 的变化情况
- 考察对排序难度相近、问题规模 n 不同的序列，n 对算法运行时间的影响
  - 从 $L=6$ 个组中，每组分别挑选 1 个序列，共挑选出 6 个长度 n 不同的序列 $SEQ(n,N)$，要求：这 6 个序列 $SEQ(n,N)$ 的 ADD 尽可能相同或接近
  - 观察对同一算法，算法运行时间 $T(n,I)$ 随 n 的变化情况

1.3 代码实现

该程序利用 C++ 实现，包含三个文件，其中含有一个 Sort 类和 main.cpp 文件。在 Sort 类中实现了随机生成序列并调用 C++ 的算法库中的乱序函数打乱序列的功能，并完成四种排序的代码功能，Sort 的 process 方法可以将序列的信息以及四种排序所花费的时间，递归深度输出。各种排序的实现方法以及代码细节在此处不在赘述。详情见附录。

1.4 算法改进

- 参照 PPT 讲义内容，当输入数组 $a[p:r]$ 已经按照非递减序排列好时，直接返回 $a[p:r]$，作为排序结果；当 $a[p:r]$ 已经按照非递增序排列好时，返回 $a[p:r]$ 中元素的逆序，作为排序结果，实现代码如下图所示

```
if (p < r)
{
    int flag_inc = 0, flag_dec = 0;
    for (int i = p; i < r && (flag_inc == 0 || flag_dec == 0); i++)
    {
        if (a[i] < a[i + 1])
        {
            flag_dec = 1;
        }
        else if (a[i] > a[i + 1])
        {
            flag_inc = 1;
        }
    }
    if (flag_dec == 0)
    {
        //reverse(a[p], a[r]);
        for (int i = p; i < (r - p + 1) / 2; i++)
        {
            swap(a[i], a[r + p - i]);
        }
        return;
    }
    if (flag_inc == 0)
    {
        return;
    }
    int q = RandomizedPartition(a, p, r);
    Quick_Sort_2(a, p, q - 1, level + 1);
```

图 1: 算法改进 1

- 在排序主程序中设置全局变量，记录排序过程的递归层次，实现方法为添加传递一个函数的参数为递归层次，在函数的开始比较全局变量的递归层次与目前函数的递归层次，若目前的递归层次更深则替换，实现代码如下图所示

```
void Sort::Quick_Sort_1(int a[], int p, int r, int level)
{
    if (level_max < level)
    {
        level_max = level;
    }
```

图 2: 算法改进 2

- 划分基准元素 x 的选取采用 2 种方式, 第一种固定选取左端 $a[p]$，Partition——快排算法 1；第二种随机 $a[p:r]$ 中元素，RandomizedPartition—快排算法 2。比较 2 种划分方式下递归层次的差异，通过程序的运行结果发现利用随机选择的递归层次在平均水平上会少于选择第一个元素的递归层次。

1.5 运行结果

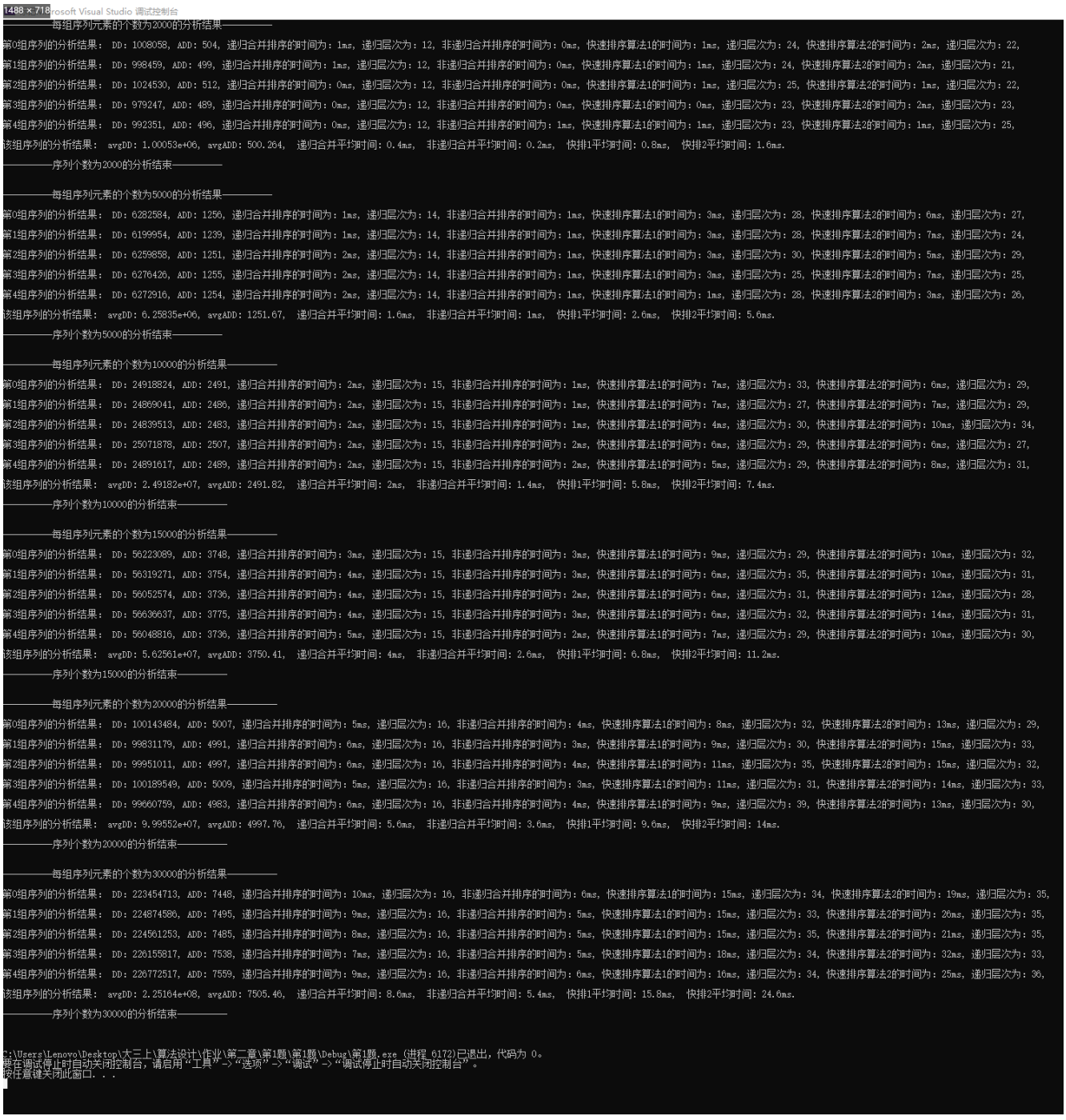

图 3: 运行结果

1.6 结果分析

表 1: 结果分析

序列 SEQ 编号	长度 n	组号	DD	ADD	递归合并时间/层次	非递归合并时间	快排 1 时间/层次	快排 2 时间/层次
1	2000	1	979247	489	0ms/12	0ms	0ms/23	2ms/23
2	2000	1	992351	496	0ms/12	1ms	1ms/23	1ms/25
3	2000	1	998459	499	1ms/12	0ms	1ms/24	2ms/21
4	2000	1	1008058	504	1ms/12	0ms	1ms/24	2ms/22
5	2000	1	1024530	512	0ms/12	0ms	1ms/25	1ms/22
6	5000	2	6199954	1239	1ms/14	1ms	3ms/28	7ms/24
7	5000	2	6259858	1251	2ms/14	1ms	3ms/30	5ms/29
8	5000	2	6272916	1254	2ms/14	1ms	1ms/28	3ms/26
9	5000	2	6276426	1255	2ms/14	1ms	3ms/25	7ms/25
10	5000	2	6282584	1256	1ms/14	1ms	3ms/28	6ms/27
11	10000	3	24839513	2483	2ms/15	1ms	4ms/30	10ms/34
12	10000	3	24869041	2486	2ms/15	1ms	7ms/27	7ms/29
13	10000	3	24891617	2489	2ms/15	2ms	5ms/29	8ms/31
14	10000	3	24918824	2491	2ms/15	1ms	7ms/33	6ms/29
15	10000	3	25071878	2507	2ms/15	2ms	6ms/29	6ms/27
16	15000	4	56048816	3736	5ms/15	2ms	7ms/29	10ms/30
17	15000	4	56052574	3736	4ms/15	2ms	6ms/31	12ms/28
18	15000	4	56223089	3748	3ms/15	3ms	9ms/29	10ms/32
19	15000	4	56319271	3754	4ms/15	3ms	6ms/35	10ms/31
20	15000	4	56636637	3775	4ms/15	3ms	6ms/32	14ms/31
21	20000	5	99660759	4983	6ms/16	4ms	9ms/39	13ms/30
22	20000	5	99831179	4991	6ms/16	3ms	9ms/30	15ms/33
23	20000	5	99951011	4997	6ms/16	4ms	11ms/35	15ms/32
24	20000	5	100143484	5007	5ms/16	4ms	8ms/32	13ms/29
25	20000	5	100189549	5009	5ms/16	3ms	11ms/31	14ms/33
26	30000	6	223454713	7448	10ms/16	6ms	15ms/34	19ms/35
27	30000	6	224561253	7485	8ms/16	5ms	15ms/35	21ms/35
28	30000	6	224874586	7495	9ms/16	5ms	15ms/33	26ms/35
29	30000	6	226155817	7538	7ms/16	5ms	18ms/34	32ms/33
30	30000	6	226772517	7559	9ms/16	6ms	16ms/34	25ms/36

表 2: 结果分析

长度 n	组号	avgDD	avgADD	递归合并 平均时间	非递归合并 平均时间	快排 1 平均时间	快排 2 平均时间
2000	1	1000530	500.264	0.4ms	0.2ms	0.8ms	1.6ms
5000	2	6258350	1251.67	1.6ms	1ms	2.6ms	5.6ms
10000	3	24918200	2491.82	2ms	1.4ms	5.8ms	7.4ms
15000	4	56256100	3750.41	4ms	2.6ms	6.8ms	11.2ms
20000	5	99552000	4997.76	5.6ms	3.6ms	9.6ms	14ms
30000	6	225164000	7505.46	8.6ms	5.4ms	15.8ms	24.6ms

表 3: 序列长度与运行时间

长度 n	组号	递归合并 平均时间	非递归合并 平均时间	快排 1 平均时间	快排 2 平均时间
2000	1	0.6ms	0.2ms	0.6ms	0.6ms
3000	2	0.8ms	0.6ms	1.2ms	1.6ms
4000	3	1.2ms	0.8ms	1.4ms	3.4ms
5000	4	1.4ms	1ms	1.6ms	4.4ms
7000	5	1.4ms	1ms	2.4ms	4.2ms
9000	6	1.6ms	1ms	2.8ms	4.2ms
10000	7	1.8ms	1.2ms	3ms	4.4ms
12000	8	2.8ms	1.8ms	4.4ms	6.8ms
15000	9	4ms	2ms	5.6ms	8.6ms
18000	10	4ms	2.2ms	6.6ms	9.6ms
20000	11	4.6ms	2.2ms	7.2ms	9.8ms
25000	12	5.6ms	3.2ms	9.4ms	14.4ms
30000	13	7ms	4.2ms	11.6ms	16.8ms
40000	14	10.2ms	8.2ms	20.6ms	24.6ms
50000	15	12.8ms	8.4ms	25.4ms	31.6ms
60000	16	15.6ms	9.2ms	26.4ms	37.4ms
70000	17	20.2ms	12.8ms	34.6ms	48.6ms
80000	18	23.2ms	14.4ms	44ms	59.4ms
90000	19	25.4ms	16ms	46.4ms	62.8ms
100000	20	29ms	17.8ms	51ms	67.2ms

由表1可得，在序列长度相同的情况下，若该序列的 DD 越大，运行时间越长，说明了在一定程度上，可以用 DD 来表示一个序列的乱序程度。比较表中两种快排算法的递归深度发现，利用随机选择的递归层次在平均水平上会少于选择第一个元素的递归层次。对于相同 n 和 DD 的序列排序中，非递归合并排序所执行的时间最短，之后分别是递归合并排序，快排 1 和快排 2。通过表2可以看出，四种算法的运行时间大体上是随着序列长度的增加而线性增加。对于不同长度的序列，其序列的 ADD 明显呈现递增趋势，故无法挑选出 ADD 相近的 6 个长度不同的序列，也由此可以看出序列的 ADD 和序列的长度 N 有关。为了更好的拟合序列长度和程序运行时间的关系，本人利用该程序又测试 20 组数据，结果如表3所示。根据上表数据，利用 python 的 curve_fit 函数通过非线性最小二乘法拟合，得到的结果如下图4所示。

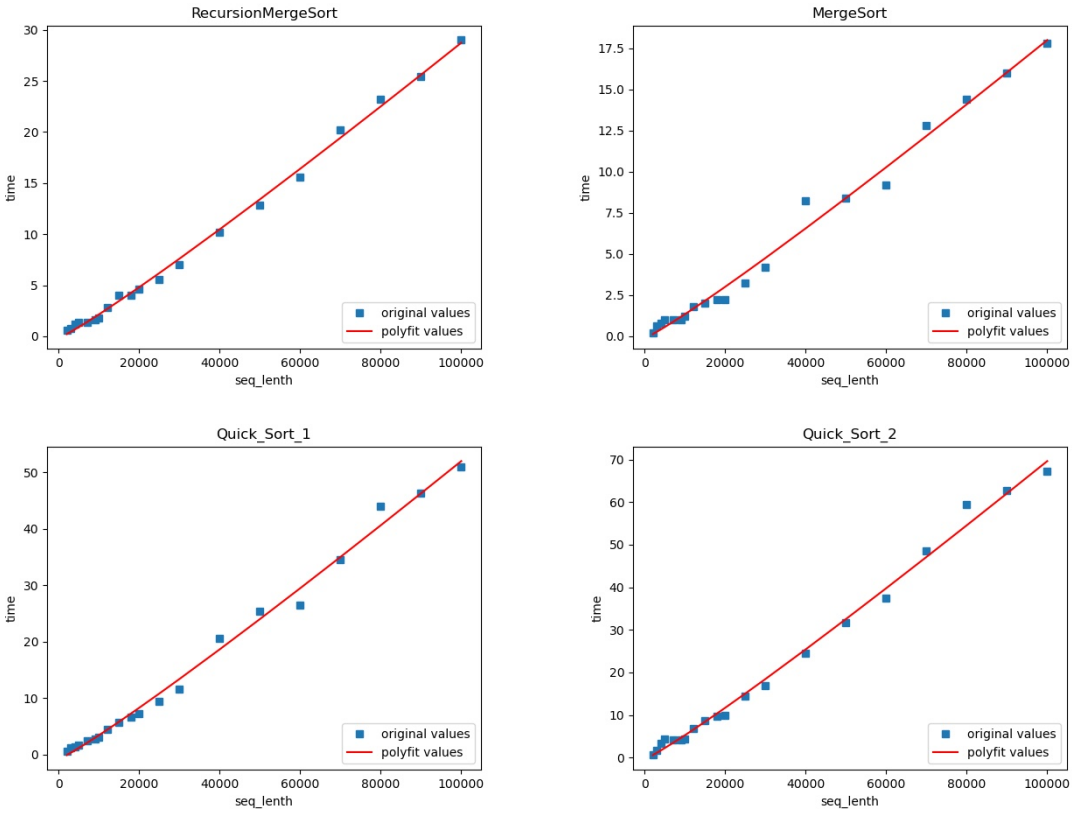

图 4: 四种排序算法序列长度 n 与执行时间的关系

从图中可以看出，拟合效果良好，可以验证这四种排序的时间复杂度为 $O(nlogn)$，下表给出了四种排序算法的拟合方程表达式。

表 4: 拟合函数

排序方式	拟合函数
递归归并排序	$y = 0.0000521x \log x - 0.000312x + 0.49632$
非递归归并排序	$y = 0.0000256x \log x - 0.000114x + 0.124$
快速排序 1	$y = 0.0000774x \log x - 0.000365x - 0.05677$
快速排序 2	$y = 0.0001132x \log x - 0.000606x + 0.92725$

2 线性时间选择 (作业 3)

2.1 题目描述

采用线性时间选择算法，根据基站 k-dist 距离，挑选出 k-dist 值最小的基站，k-dist 第 5 小的基站，k-dist 值第 50 小的基站，k-dist 值最大的基站。

2.2 实验步骤以及要求

- 在排序主程序中设置全局变量，记录选择划分过程的递归层次
- 参照讲义 PPT，将教科书上的“一分为二”的子问题划分方法，改进为“一分为三”，比较这 2 种划分方式下，选择过程递归层次的差异

2.3 代码实现

首先将 excle 表存为.csv 文件的格式方便代码读取，利用文件流操作读出.csv 文件的一行，再将整行字符串 line 读入到字符串流 istringstream 中，然后把将字符串流 sin 中的字符读入到 field 字符串中，以逗号为分隔符，最后清除掉向量 fields 中的无效字符，并赋值给变量 k_dist。一分为二和一分为三的区别就在于最后判断，一分为三中若 j=k 则直接返回 a[j], 这对降低递归层数有一定的作用。线性时间选择的实现方法以及代码细节在此处不在赘述。详情见附录。

```cpp
ifstream fin("C:\\Users\\Lenovo\\Desktop\\data_1.csv"); //打开文件流操作
string line;
double* k_dist = new double[num_of_k_dist];
int num = 0;
getline(fin, line);
while (getline(fin, line))   //整行读取，换行符“\n”区分，遇到文件尾标志eof终止读取
{
    //cout << "原始字符串: " << line << endl; //整行输出
    istringstream sin(line); //将整行字符串line读入到字符串流istringstream中
    vector<string> fields; //声明一个字符串向量
    string field;
    while (getline(sin, field, ',')) //将字符串流sin中的字符读入到field字符串中，以逗号为分隔符
    {
        fields.push_back(field); //将刚刚读取的字符串添加到向量fields中
    }
    k_dist[num] = atof(Trim(fields[3]).c_str()); //清除掉向量fields中第一个元素的无效字符，并赋
    num++;
}
```

图 5: 读入.csv 文件

```cpp
/*
//一分为二
if (k <= j)
    return Select(a, p, i, k, level + 1);
else
    return Select(a, i + 1, r, k - j, level + 1);
    */

//一分为三
if (j == k)
    return a[j];
else
{
    if (k < j)
        return Select(a, p, i - 1, k, level + 1);
    else
        return Select(a, i + 1, r, k - j, level + 1);
}
```

图 6: 一分为二和一分为三的代码区别

2.4 运行结果

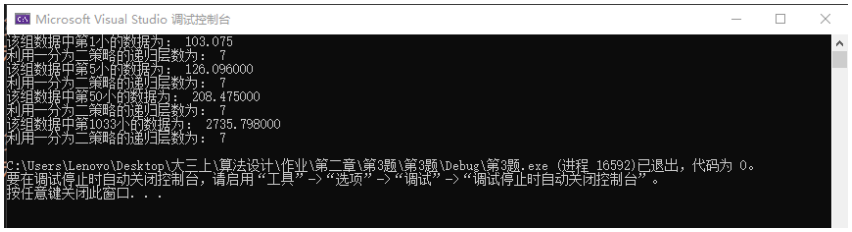

图 7: 一分为二运行结果

图 8: 一分为三运行结果

2.5 结果分析

与 excel 表中数据对比，结果符合预期。一分为三策略相比与一分为二在寻找最小 k-dist 值中，递归层数少了 1，说明一分为三策略对于降低函数的递归层数有一定的作用和效果。

3 最近平面点对 (作业 4)

3.1 题目描述

采用平面最近点对算法，根据基站经纬度，挑选出距离最近的 2 个基站和距离次最近的 2 个基站。

3.2 实验步骤以及要求

- 最近/次最近的 2 个基站间距离
- 最近/次最近的 2 个基站点对（用基站 ENodeBID 表示）

3.3 代码实现

读入文件的操作和第三题一致，此处不再赘述。求最近点对直接利用最近平面点对算法即可。求次最近点对，该程序的思路是先求出最近点对保存这两个点的 ENodeBID，对基站进行遍历，把这两个点交换到最后两个位置，利用最近平面点对算法将前 n-1 个基站的最近平面点对求出，然后交换最后两个基站的位置，再次利用最近平面点对算法将前 n-1 个基站的最近平面点对求出，从这两中选出最近的点对即为次最近点对。

```
int flag = 0;
for (int i = 0; i < num - 2; i++)
{
    if (point[i].ENODEBID == p1.ENODEBID || point[i].ENODEBID == p2.ENODEBID)
    {
        flag++;
        swap(point[i], point[num - flag]);
        i--;
    }
}
Point p1_temp_1, p2_temp_1;
double d_temp_1 = cloest_pair(0, num - 2, p1_temp_1, p2_temp_1);
swap(point[num - 2], point[num - 1]);
Point p1_temp_2, p2_temp_2;
double d_temp_2 = cloest_pair(0, num - 2, p1_temp_2, p2_temp_2);
if (d_temp_1 > d_temp_2)
{
    p1 = p1_temp_2;
    p2 = p2_temp_2;
    d = d_temp_2;
}
else
{
    p1 = p1_temp_1;
    p2 = p2_temp_1;
    d = d_temp_1;
}
```

图 9: 一分为三运行结果

3.4 运行结果

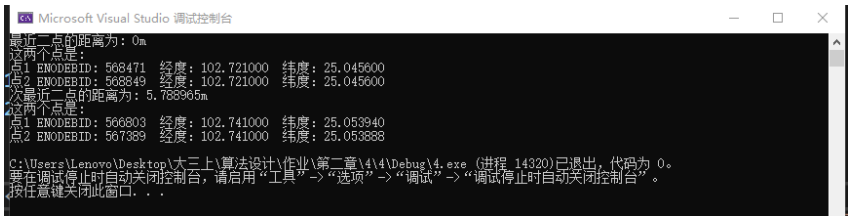

图 10: 最近平面点对运行结果

3.5 结果分析

经过验证，结果符合预期。

4 实验总结

通过本章的学习和上机作业的巩固，我对于排序算法，线性时间选择算法，最近平面点对算法有了更深入的理解和认识。这次上机作业让我理解到动手实践的重要性，在上课听讲的过程中，我一度认为排序算法比较好实现，直到亲自动手写代码时才发现有很多需要注意的细节。在实现递归的归并排序时，由于我将判断条件小于等于写成了等于导致程序错误，这个 bug 足足改了我两个小时。在最近平面点对实验第一次输出最短距离是 0 时，我一度怀疑是自己程序错误，在 excel 表中比对发现有两个基站的经度和纬度一模一样，当时的心情无比喜悦。总之，这次实验虽然较难，但是对于我来说收获是非常大的。

A 排序算法代码

```cpp
//Sort.cpp文件
#include "Sort.h"
#include <stdlib.h>
#include <time.h>
#include <iostream>
#include <algorithm>
#include <vector>
#include <random>
#include <time.h>

using namespace std;
Sort::Sort(int num) :n(num)
{
    seq_dd = new int[m];
    //初始化序列的乱序数组
    arr = new int* [m];
    //初始化二维数组
    b = new int[n];
    sort_time = new int* [m];
    for (int i = 0; i < m; i++)
    {
        sort_time[i] = new int[4];
    }
    for (int i = 0; i < m; i++)
    {
        arr[i] = new int[n];
    }
    srand((unsigned)time(NULL));
    for (int i = 0; i < n; i++)
    {
        //随机产生长度为n的数组
        arr[0][i] = rand() % N;
        //arr[0][i] = i + 1;
        for (int j = 1; j < m; j++)
        {
            arr[j][i] = arr[0][i];
        }
```

```
    }
    seq_dd[0] = count_seq_dd(0);
    for (int j = 1; j < m; j++)
    {
        //乱序
        do
        {
            random_shuffle(arr[j], arr[j] + n);
            seq_dd[j] = count_seq_dd(j);
        } while (compare_seq_dd(j));
    }
    /*for (int i = 0; i < n; i++)
    {
        cout << arr[0][i] << " ";
    }
    cout << endl;
    for (int i = 0; i < n; i++)
    {
        cout << arr[1][i] << " ";
    }
    cout << endl;
    for (int i = 0; i < n; i++)
    {
        cout << arr[2][i] << " ";
    }
    cout << endl;
    for (int i = 0; i < n; i++)
    {
        cout << arr[3][i] << " ";
    }
    cout << endl;
    for (int i = 0; i < n; i++)
    {
        cout << arr[4][i] << " ";
    }
    cout << endl;
    for (int i = 0; i < m; i++)
    {
        cout << seq_dd[i] << "  ";
```

```
77      }
78      cout << endl;
79      MergeSort(arr[0]);
80      RecursionMergeSort(arr[1], 0, n - 1);
81      Quick_Sort_1(arr[2], 0, n - 1);
82      Quick_Sort_2(arr[3], 0, n - 1);*/
83  }
84  int Sort::count_seq_dd(int arr_seq)//计算乱序程度，传入计算第几个序列
        的参数
85  {
86      int ans = 0;
87      for (int i = 0; i < n; i++)
88      {
89          for (int j = i + 1; j < n; j++)
90          {
91              if (arr[arr_seq][i] > arr[arr_seq][j])
92              {
93                  ans++;
94              }
95          }
96      }
97      /*cout << ans << endl;*/
98      return ans;
99  }
100 bool Sort::compare_seq_dd(int arr_seq)//比较同组内长度相同的序列，若
        DD相同则返回true否则返回false
101 {
102     bool flag = false;
103     for (int i = 0; i < arr_seq; i++)
104     {
105         if (seq_dd[i] == seq_dd[arr_seq])
106         {
107             flag = true;
108         }
109     }
110     return flag;
111 }
112 void Sort::MergeSort(int a[], int n)
113 {
```

```
114     int s = 1;
115     while (s < n)
116     {
117         MergePass(a, b, s, n);
118         s += s;
119         MergePass(b, a, s, n);
120         s += s;
121     }
122 }
123 void Sort::MergePass(int x[], int y[], int s, int n)
124 {
125     int i = 0;
126     while (i <= n - 2 * s)
127     {
128         Merge(x, y, i, i + s - 1, i + 2 * s - 1);
129         i = i + 2 * s;
130     }
131     if (i + s <= n)
132     {
133         Merge(x, y, i, i + s - 1, n - 1);
134     }
135     else
136     {
137         for (int j = i; j < n; j++)
138         {
139             y[j] = x[j];
140         }
141     }
142 }
143 void Sort::Merge(int c[], int d[], int l, int m, int r)
144 {
145     int i = l, j = m + 1, k = l;
146     while ((i <= m) && (j <= r))
147     {
148         if (c[i] <= c[j])
149         {
150             d[k++] = c[i++];
151         }
152         else
```

```
153         {
154             d[k++] = c[j++];
155         }
156     }
157     if (i > m)
158     {
159         for (int q = j; q <= r; q++)
160         {
161             d[k++] = c[q];
162         }
163     }
164     else
165     {
166         for (int q = i; q <= m; q++)
167         {
168             d[k++] = c[q];
169         }
170     }
171 }
172 void Sort::RecursionMergeSort(int a[], int left, int right, int level
    )
173 {
174     if (level_max < level)
175     {
176         level_max = level;
177     }
178     if (left < right)
179     {
180         int i = (left + right) / 2;
181         RecursionMergeSort(a, left, i, level + 1);
182         RecursionMergeSort(a, i + 1, right, level + 1);
183         Merge(a, b, left, i, right);
184         for (int i = left; i <= right; i++)
185         {
186             a[i] = b[i];
187         }
188     }
189 }
190 void Sort::Quick_Sort_1(int a[], int p, int r, int level)
```

```
191 {
192     if (level_max < level)
193     {
194         level_max = level;
195     }
196     if (p < r)
197     {
198         int flag_inc = 0, flag_dec = 0;
199         for (int i = p; i < r && (flag_inc == 0 || flag_dec == 0); i
               ++)
200         {
201             if (a[i] < a[i + 1])
202             {
203                 flag_dec = 1;
204             }
205             else if (a[i] > a[i + 1])
206             {
207                 flag_inc = 1;
208             }
209         }
210         if (flag_dec == 0)
211         {
212             //reverse(a[p], a[r]);
213             for (int i = p; i < (r - p + 1) / 2; i++)
214             {
215                 swap(a[i], a[r + p - i]);
216             }
217             return;
218         }
219         if (flag_inc == 0)
220         {
221             return;
222         }
223         int q = Partition(a, p, r);
224         Quick_Sort_1(a, p, q - 1, level + 1);
225         Quick_Sort_1(a, q + 1, r, level + 1);
226     }
227 }
228 int Sort::Partition(int a[], int p, int r)
```

```
229 {
230     int i = p, j = r + 1;
231     int x = a[p];
232     while (true)
233     {
234         while (a[++i] < x && i < r);
235         while (a[--j] > x);
236         if (i >= j)
237         {
238             break;
239         }
240         swap(a[i], a[j]);
241     }
242     a[p] = a[j];
243     a[j] = x;
244     return j;
245 }
246 void Sort::Quick_Sort_2(int a[], int p, int r, int level)
247 {
248     if (level_max < level)
249     {
250         level_max = level;
251     }
252     if (p < r)
253     {
254         int flag_inc = 0, flag_dec = 0;
255         for (int i = p; i < r && (flag_inc == 0 || flag_dec == 0); i
               ++)
256         {
257             if (a[i] < a[i + 1])
258             {
259                 flag_dec = 1;
260             }
261             else if (a[i] > a[i + 1])
262             {
263                 flag_inc = 1;
264             }
265         }
266         if (flag_dec == 0)
```

```
267         {
268             //reverse(a[p], a[r]);
269             for (int i = p; i < (r - p + 1) / 2; i++)
270             {
271                 swap(a[i], a[r + p - i]);
272             }
273             return;
274         }
275         if (flag_inc == 0)
276         {
277             return;
278         }
279         int q = RandomizedPartition(a, p, r);
280         Quick_Sort_2(a, p, q - 1, level + 1);
281         Quick_Sort_2(a, q + 1, r, level + 1);
282     }
283 }
284 int Sort::RandomizedPartition(int a[], int p, int r)
285 {
286     srand((unsigned)time(NULL));
287     int i = p + rand() % (r - p + 1); //随机选1个a[i]
288     swap(a[i], a[p]);        //将a[i]交换到a[]的第1个位置a[p]
289     return Partition(a, p, r);
290 }
291 void Sort::running()
292 {
293     cout << "----------每组序列元素的个数为" << n << "的分析结果
        ----------" << endl;
294     cout << endl;
295     clock_t start, finish;
296     clock_t duration;
297     int* temp = new int[n];
298     for (int k = 0; k < m; k++)
299     {
300         cout << "第" << k << "组序列的分析结果:  ";
301         memcpy(temp, arr[k], n * sizeof(int));
302         int ans = 0;
303         for (int i = 0; i < n; i++)
304         {
```

```
            for (int j = i + 1; j < n; j++)
            {
                if (temp[i] > temp[j])
                {
                    ans++;
                }
            }
        }
        cout << "DD: " << ans << ", ADD: " << ans / n << ", ";
        level_max = 1;
        start = clock();
        RecursionMergeSort(temp, 0, n - 1, level_max);
        finish = clock();
        duration = (finish - start);
        sort_time[k][0] = duration;
        cout << "递归合并排序的时间为: " << duration << "ms, 递归层次
            为: " << level_max << ", ";

        memcpy(temp, arr[k], n * sizeof(int));
        start = clock();
        MergeSort(temp, n);
        finish = clock();
        duration = (finish - start);
        sort_time[k][1] = duration;
        cout << "非递归合并排序的时间为: " << duration << "ms, ";

        memcpy(temp, arr[k], n * sizeof(int));
        level_max = 1;
        start = clock();
        Quick_Sort_1(temp, 0, n - 1, level_max);
        finish = clock();
        duration = (finish - start);
        sort_time[k][2] = duration;
        cout << "快速排序算法1的时间为: " << duration << "ms, 递归层
            次为: " << level_max << ", ";

        memcpy(temp, arr[k], n * sizeof(int));
        level_max = 1;
        start = clock();
```

```
342         Quick_Sort_2(temp, 0, n - 1, level_max);
343         finish = clock();
344         duration = (finish - start);
345         sort_time[k][3] = duration;
346         cout << "快速排序算法2的时间为: " << duration << "ms, 递归层
               次为: " << level_max << ", " << endl;
347         cout << endl;
348     }
349     float DD = 0, avg_time_merge = 0, avg_time_reg = 0,
           avg_time_quick1 = 0, avg_time_quick2 = 0;
350     for (int i = 0; i < m; i++)
351     {
352         DD += seq_dd[i];
353     }
354     for (int i = 0; i < m; i++)
355     {
356         avg_time_reg += sort_time[i][0];
357         avg_time_merge += sort_time[i][1];
358         avg_time_quick1 += sort_time[i][2];
359         avg_time_quick2 += sort_time[i][3];
360     }
361     avg_time_reg /= m;
362     avg_time_merge /= m;
363     avg_time_quick1 /= m;
364     avg_time_quick2 /= m;
365     cout << "该组序列的分析结果: avgDD: " << DD / m << ", avgADD: "
           << DD / (m * n) <<
366         ", 递归合并平均时间: " << avg_time_reg << "ms, 非递归合并平
               均时间: " << avg_time_merge <<
367         "ms, 快排1平均时间: " << avg_time_quick1 << "ms, 快排2平均
               时间: " << avg_time_quick2 << "ms." << endl;
368     cout << endl;
369     cout << "----------序列个数为" << n << "的分析结束----------" <<
           endl;
370     cout << endl;
371     cout << endl;
372 }
373 Sort::~Sort()
374 {
```

```
375     delete[] seq_dd;
376     seq_dd = NULL;
377     delete[] b;
378     b = NULL;
379     for (int i = 0; i < m; i++)
380     {
381         delete[] arr[i];
382         arr[i] = NULL;
383     }
384     delete[] arr;
385     arr = NULL;
386     for (int i = 0; i < 4; i++)
387     {
388         delete[] sort_time[i];
389         sort_time[i] = NULL;
390     }
391     delete[] sort_time;
392     sort_time = NULL;
393 }
394
395
396 //Sort.h文件中
397 #pragma once
398
399 class Sort
400 {
401 public:
402     Sort(int num);
403     ~Sort();
404     void running();//执行
405 private:
406     int N = 30000, n, m = 5, ** arr;
407     int* seq_dd, * b, level_max, ** sort_time;
408     bool compare_seq_dd(int arr_seq);//比较同组内长度相同的序列，若DD
        相同则返回true否则返回false
409     int count_seq_dd(int arr_seq);//计算出数组的乱序程度
410     void MergeSort(int a[], int n);//非递归的合并排序
411     void MergePass(int x[], int y[], int s, int n);
412     void Merge(int c[], int d[], int l, int m, int r);
```

```
413     void RecursionMergeSort(int a[], int left, int right, int level);
            //递归的合并排序
414     void Quick_Sort_1(int a[], int p, int r, int level);//快排算法1
415     int Partition(int a[], int p, int r);
416     void Quick_Sort_2(int a[], int p, int r, int level);//快排算法2
417     int RandomizedPartition(int a[], int p, int r);
418 };
419
420 //main.cpp文件中
421 #include <iostream>
422 #include "Sort.h"
423 using namespace std;
424
425 int main()
426 {
427     Sort sort1(2000);
428     sort1.running();
429     Sort sort2(5000);
430     sort2.running();
431     Sort sort3(10000);
432     sort3.running();
433     Sort sort4(15000);
434     sort4.running();
435     Sort sort5(20000);
436     sort5.running();
437     Sort sort6(30000);
438     sort6.running();
439     return 0;
440 }
```

B 拟合函数

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy.optimize import curve_fit
4
5
```

```
# 自定义函数 e指数形式
def func(x, a, b, c):
    return a * x * np.log(x) + b*x+c


# 定义x、y散点坐标
x = [2000, 3000, 4000, 5000, 7000, 9000, 10000, 12000,
     15000, 18000, 20000, 25000, 30000, 40000, 50000, 60000, 70000,
     80000, 90000, 100000]
x = np.array(x)
# num = [0.6, 0.8, 1.2, 1.4, 1.4, 1.6, 1.8, 2.8, 4, 4, 4.6, 5.6, 7,
   10.2, 12.8, 15.6, 20.2, 23.2, 25.4, 29]
num = [0.2, 0.6, 0.8, 1, 1, 1, 1.2, 1.8, 2, 2.2, 2.2, 3.2, 4.2, 8.2,
   8.4, 9.2, 12.8, 14.4, 16, 17.8]
# num = [0.6, 1.2, 1.4, 1.6, 2.4, 2.8, 3, 4.4, 5.6, 6.6, 7.2, 9.4,
   11.6, 20.6, 25.4, 26.4, 34.6, 44, 46.4, 51]
# num = [0.6, 1.6, 3.4, 4.4, 4.2, 4.2, 4.4, 6.8, 8.6, 9.6, 9.8, 14.4,
    16.8, 24.6, 31.6, 37.4, 48.6, 59.4, 62.8, 67.2]
y = np.array(num)

# 非线性最小二乘法拟合
popt, pcov = curve_fit(func, x, y)
# 获取popt里面是拟合系数
print(popt)
a = popt[0]
b = popt[1]
c = popt[2]
yvals = func(x, a, b,c)  # 拟合y值
print('popt:', popt)
print('系数a:', a)
print('系数b:', b)
print('系数c:', c)
print('系数pcov:', pcov)
print('系数yvals:', yvals)
# 绘图
plot1 = plt.plot(x, y, 's', label='original values')
plot2 = plt.plot(x, yvals, 'r', label='polyfit values')
plt.xlabel('seq_lenth')
plt.ylabel('time')
```

```
plt.legend(loc=4)  # 指定legend的位置右下角
plt.title('MergeSort')
plt.savefig("C:\\Users\\Lenovo\\Desktop\\mer.jpg")
plt.show()
```

C 线性时间选择代码

```
#include <iostream>
#include <vector>
#include <fstream>
#include <sstream>

constexpr auto num_of_k_dist = 2000;
using namespace std;
string Trim(string& str);
void SelectSort(double a[], int p, int r);
void Swap(double& a, double& b);
int Partition(double a[], int p, int r, double x);
int SearchMid(double a[], int p, int r);
double Select(double a[], int p, int r, int k, int level);
int deep = 0;

int main()
{
    ifstream fin("C:\\Users\\Lenovo\\Desktop\\data_1.csv"); //打开文件流操作
    string line;
    double* k_dist = new double[num_of_k_dist];
    int num = 0;
    getline(fin, line);
    while (getline(fin, line))   //整行读取，换行符“\n”区分，遇到文件尾标志eof终止读取
    {
        //cout << "原始字符串: " << line << endl; //整行输出
        istringstream sin(line); //将整行字符串line读入到字符串流istringstream中
        vector<string> fields; //声明一个字符串向量
```

```
28          string field;
29          while (getline(sin, field, ',')) //将字符串流sin中的字符读入
                到field字符串中，以逗号为分隔符
30          {
31              fields.push_back(field); //将刚刚读取的字符串添加到向量
                    fields中
32          }
33          k_dist[num] = atof(Trim(fields[3]).c_str()); //清除掉向量
                fields中第一个元素的无效字符，并赋值给变量name
34          num++;
35      }
36      /*for (int i = 1; i < 50; i++)
37      {
38          cout << (double)Select(k_dist, 0, num - 1, i, 1) << endl;
39      }*/
40      cout << "该组数据中第1小的数据为:  " << Select(k_dist, 0, num -
            1, 1, 1) << endl;
41      cout << "利用一分为二策略的递归层数为:  " << deep << endl;
42      //cout << "利用一分为三策略的递归层数为:  " << deep << endl;
43      deep = 1;
44      cout << "该组数据中第5小的数据为:  " << Select(k_dist, 0, num -
            1, 5, 1) << endl;
45      cout << "利用一分为二策略的递归层数为:  " << deep << endl;
46      //cout << "利用一分为三策略的递归层数为:  " << deep << endl;
47      deep = 1;
48      cout << "该组数据中第50小的数据为:  " << Select(k_dist, 0, num -
            1, 50, 1) << endl;
49      cout << "利用一分为二策略的递归层数为:  " << deep << endl;
50      //cout << "利用一分为三策略的递归层数为:  " << deep << endl;
51      deep = 1;
52      cout << "该组数据中第1033小的数据为:  " << Select(k_dist, 0, num
            - 1, 1033, 1) << endl;
53      cout << "利用一分为二策略的递归层数为:  " << deep << endl;
54      //cout << "利用一分为三策略的递归层数为:  " << deep << endl;
55      return 0;
56  }
57  string Trim(string& str)
58  {
59      //str.find_first_not_of(" \t\r\n"),在字符串str中从索引0开始，返回
```

```
        首次不匹配"\t\r\n"的位置
    str.erase(0, str.find_first_not_of(" \t\r\n"));
    str.erase(str.find_last_not_of(" \t\r\n") + 1);
    return str;
}

void Swap(double& a, double& b)
{
    double c = a;
    a = b;
    b = c;
}
//选择排序

void SelectSort(double a[], int p, int r)
{
    for (int i = p; i < r; ++i)
    {
        int index = i;
        for (int j = i + 1; j <= r; ++j)
        {
            if (a[j] < a[index])
                index = j;
        }
        Swap(a[i], a[index]);
    }
}
//按x划分，返回划分基准下标

int Partition(double a[], int p, int r, double x)
{
    int i = p - 1, j = r + 1;
    while (true)
    {
        while (a[++i] < x && i < r);
        while (a[--j] > x && j > p);
        if (i >= j)break;
        Swap(a[i], a[j]);
    }
```

```
98      return j;
99  }
100 //找到中位数（用于找每组的5个数的中位数）
101
102 int SearchMid(double a[], int p, int r)
103 {
104     double* b = new double[r - p + 1];
105     for (int i = p; i <= r; ++i)
106     {
107         b[i - p] = a[i];
108     }
109     SelectSort(b, 0, r - p);
110     for (int i = p; i <= r; ++i)
111     {
112         if (a[i] == b[(r - p + 1) / 2])
113             return i;
114     }
115     return 0;
116 }
117 //p第一个数下标，r最后一个数下标，k要找的第k个数
118
119 double Select(double a[], int p, int r, int k, int level)
120 {
121     if (deep < level)
122         deep = level;
123     //规模小于20时直接排序查找
124     if (r - p < 20)
125     {
126         SelectSort(a, p, r);
127         return a[p + k - 1];
128     }
129     //分成n/5组，每组5个；找到每组中位数，放置队首
130     for (int i = 0; i <= (r - p - 4) / 5; ++i)
131     {
132         int mid = SearchMid(a, p + 5 * i, p + 5 * i + 4);
133         Swap(a[mid], a[p + i]);
134     }
135     //找到所有中位数的中位数
136     double x = Select(a, p, p + (r - p - 4) / 5, (r - p - 4) / 10 +
```

```
        1, level + 1);
    //按中位数划分
    int i = Partition(a, p, r, x);
    int j = i - p + 1;
    if (k <= j)
        return Select(a, p, i, k, level + 1);
    else
        return Select(a, i + 1, r, k - j, level + 1);
    /*
        一分为三
        if (j == k)
            return a[j];
        else
        {
            if (k < j)
                return Select(a, p, i - 1, k, level + 1);
            else
                return Select(a, i + 1, r, k - j, level + 1);
        }

    */
}
```

D 最近平面点对代码

```
#include <cstdio>
#include <cstdlib>
#define _USE_MATH_DEFINES
#include <math.h>
#include <iostream>
#include <vector>
#include <fstream>
#include <sstream>

constexpr auto INF = 100000000;
constexpr auto N = 100000;
const double EARTH_RADIUS = 6378.137;//赤道半径km
```

```
using namespace std;
struct Point
{
    int ENODEBID;
    double LONGITUDE;
    double LATITUDE;
}point[N];

string Trim(string& str)
{
    //str.find_first_not_of(" \t\r\n"),在字符串str中从索引0开始，返回首次不匹配"\t\r\n"的位置
    str.erase(0, str.find_first_not_of(" \t\r\n"));
    str.erase(str.find_last_not_of(" \t\r\n") + 1);
    return str;
}

int cmpy(const void* y1, const void* y2) //根据y坐标进行排序
{
    if (*(double*)y1 < *(double*)y2)
        return 1;
    return 0;
}

int cmpxy(const void* index1, const void* index2)
{
    if (fabs((*(struct Point*) index1).LONGITUDE - (*(struct Point*)index2).LONGITUDE) > 1e-3)  //如果x坐标不同
        if ((*(struct Point*) index1).LONGITUDE < (*(struct Point*)index2).LONGITUDE)         //根据x坐标排序
            return 1;
        else   //x坐标相同的情况下，根据y坐标排序
        {
            if ((*(struct Point*) index1).LATITUDE < (*(struct Point*) index1).LATITUDE)
                return 1;
        }
    return 0;
}
```

```
48
49  double rad(double LatOrLon)
50  {
51      return LatOrLon * M_PI / 180.0;
52  }
53
54
55  double GetDistance2(double lng1, double lat1, double lng2, double
        lat2)
56  {
57      //将对应的经纬度转化为弧度
58      double radLat1 = rad(lat1);
59      double radLat2 = rad(lat2);
60      double radlng1 = rad(lng1);
61      double radlng2 = rad(lng2);
62      //利用正弦余弦公式求距离
63      double s = acos(cos(radLat1) * cos(radLat2) * cos(radlng1 -
            radlng2) + sin(radLat1) * sin(radLat2));
64      s = s * EARTH_RADIUS;
65      s = s * 1000;
66      return s;
67  }
68
69  int temp[N];
70  int ids[N];
71
72  // 返回点区间上最近两点间的距离,
73  // left为区间最左边的点, right为最右边的点
74  // p1, p2为最近点对
75  double cloest_pair(int left, int right, Point& p1, Point& p2)
76  {
77      double d = INF;
78      if (right == left)                  //如果只有一个点
79      {
80          p1 = p2 = point[left];
81          return d;
82      }
83      else if (right == left + 1)         //如果只有两个点
84      {
```

```
85          p1 = point[left];
86          p2 = point[right];
87          return GetDistance2(point[left].LONGITUDE, point[left].
               LATITUDE, point[right].LONGITUDE, point[right].LATITUDE);
                //如果有三个及以上的点
88      }
89      else
90      {
91          int mid = (left + right) >> 1;  //取区间中点
92          Point p3, p4, p5, p6;
93          double d1 = cloest_pair(left, mid, p3, p4);      //d1=左半区间
               最近点距离
94          double d2 = cloest_pair(mid + 1, right, p5, p6);  //d2=右半区
               间最近点距离
95          if (d1 < d2)
96          {
97              p1 = p3;
98              p2 = p4;
99              d = d1;
100         }
101         else
102         {
103             p1 = p5;
104             p2 = p6;
105             d = d2;
106         }
107
108         int i, j, k = 0;
109         //以左半区间和右半区间的垂直分割线为中心，分离出宽度为2d的区
               间
110         for (i = left; i <= right; i++)
111         {
112             if (fabs(point[i].LONGITUDE - point[mid].LONGITUDE) <= d)
113                 temp[k++] = i;
114         }
115
116         //将距离2d之内的点按y坐标排序
117         qsort(temp, k, sizeof(int), cmpy);
118
```

```
119         //对y坐标线性扫描
120         for (i = 0; i <= k; i++)
121         {
122             //对于x坐标在宽度为d的区间内的每个点，y坐标在宽度为2d之内
123             for (j = i + 1; j < k && fabs(point[temp[j]].LATITUDE -
                    point[temp[i]].LATITUDE) < d; j++)
124             {
125                 double d3 = GetDistance2(point[temp[i]].LONGITUDE,
                        point[temp[i]].LATITUDE, point[temp[j]].LONGITUDE,
                         point[temp[j]].LATITUDE);
126                 if (d3 < d)
127                 {
128                     p1 = point[temp[i]];
129                     p2 = point[temp[j]];
130                     d = d3;  //更新最短距离
131                 }
132             }
133         }
134     }
135     return d;
136 }
137
138 int main()
139 {
140     ifstream fin("C:\\Users\\Lenovo\\Desktop\\data_1.csv"); //打开文
        件流操作
141     string line;
142     int num = 0;
143     getline(fin, line);
144     while (getline(fin, line))   //整行读取，换行符“\n”区分，遇到文
        件尾标志eof终止读取
145     {
146         //cout << "原始字符串: " << line << endl; //整行输出
147         istringstream sin(line); //将整行字符串line读入到字符串流
            istringstream中
148         vector<string> fields; //声明一个字符串向量
149         string field;
150         while (getline(sin, field, ',')) //将字符串流sin中的字符读入
            到field字符串中，以逗号为分隔符
```

```
151         {
152             fields.push_back(field); //将刚刚读取的字符串添加到向量
                fields中
153         }
154         point[num].ENODEBID = atoi(Trim(fields[0]).c_str()); //清除掉
            向量fields中第一个元素的无效字符，并赋值给变量name
155         point[num].LONGITUDE = atof(Trim(fields[1]).c_str());
156         point[num].LATITUDE = atof(Trim(fields[2]).c_str());
157         num++;
158     }
159     qsort(point, num, sizeof(point[0]), cmpxy);
160     Point p1, p2;
161     double d = cloest_pair(0, num - 1, p1, p2);
162     cout << "最近二点的距离为：" << d << "m" << endl;
163     cout << fixed << "这两个点是：\n" << "点1 ENODEBID: " << p1.
        ENODEBID << "  经度：" << p1.LONGITUDE << "  纬度：" << p1.
        LATITUDE << endl;
164     cout << fixed << "点2 ENODEBID: " << p2.ENODEBID << "  经度：" <<
        p2.LONGITUDE << "  纬度：" << p2.LATITUDE << endl;
165     int flag = 0;
166     for (int i = 0; i < num - 2; i++)
167     {
168         if (point[i].ENODEBID == p1.ENODEBID || point[i].ENODEBID ==
            p2.ENODEBID)
169         {
170             flag++;
171             swap(point[i], point[num - flag]);
172             i--;
173         }
174     }
175     Point p1_temp_1, p2_temp_1;
176     double d_temp_1 = cloest_pair(0, num - 2, p1_temp_1, p2_temp_1);
177     swap(point[num - 2], point[num - 1]);
178     Point p1_temp_2, p2_temp_2;
179     double d_temp_2 = cloest_pair(0, num - 2, p1_temp_2, p2_temp_2);
180     if (d_temp_1 > d_temp_2)
181     {
182         p1 = p1_temp_2;
183         p2 = p2_temp_2;
```

```
184             d = d_temp_2;
185         }
186         else
187         {
188             p1 = p1_temp_1;
189             p2 = p2_temp_1;
190             d = d_temp_1;
191         }
192         cout << "次最近二点的距离为: " << d << "m" << endl;
193         cout << fixed << "这两个点是: \n" << "点1 ENODEBID: " << p1.
                ENODEBID << "  经度: " << p1.LONGITUDE << "  纬度: " << p1.
                LATITUDE << endl;
194         cout << fixed << "点2 ENODEBID: " << p2.ENODEBID << "  经度: " <<
                p2.LONGITUDE << "  纬度: " << p2.LATITUDE << endl;
195         //cout << GetDistance2(102.741, 25.05394, 102.741, 25.053888);
196         //cout << GetDistance2(102.791, 25.03979, 102.791, 25.039722);
197         return 0;
198 }
```