一些说明：

1、由于数据量较小，因此很难看出递归合并和非递归合并的差别（递归是 0.001 秒，非递归偶尔是 0.000 秒）

2、很明显，一分为三的快速排序比一分为二的递归层次要小很多

注：部分代码可能需要 C++11 版本

注：为了方便读取，将 xls 的文件改成了 csv

注：所有题目均在一个 cpp 文件下，主函数中依次进行各个算法

代码：

```
// Algorithm1.cpp : Designed by Xiao Yunming.
//

#include "stdafx.h"  // Vs Projects head file

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <time.h>
#include <cmath>

using namespace std;

#define NUM_STATION 1033
#define EARTH_RADIUS 6378.137
#define M_PI 3.14159265358979323846
#define INF 100000.0

// ------------------------------ Over All ---------------------------------------

class BaseStation
{
public:
    int enodebid;
    double longitude, latitude, k_dist;

    BaseStation();
    BaseStation(string line);
    ~BaseStation() {};
    BaseStation(const BaseStation &b);

    bool operator<(BaseStation &b) { return this->k_dist < b.k_dist; };
    bool operator<=(BaseStation &b) { return this->k_dist <= b.k_dist; };
    bool operator>(BaseStation &b) { return this->k_dist > b.k_dist; };
    bool operator>=(BaseStation &b) { return this->k_dist >= b.k_dist; };
    bool operator==(BaseStation &b) { return this->k_dist == b.k_dist; };
};

// this class is for the convinience of sorting by longitude in the closest pair algorithm
```

```
class BaseStationLongitude :public BaseStation
{
public:
    BaseStationLongitude() :BaseStation() {};
    BaseStationLongitude(const BaseStationLongitude &b);
    BaseStationLongitude operator=(BaseStation &b);

    bool operator<(BaseStationLongitude &b) { return this->longitude < b.longitude; };
    bool operator<=(BaseStationLongitude &b) { return this->longitude <= b.longitude; };
    bool operator==(BaseStationLongitude &b) { return this->longitude == b.longitude; };
    bool operator>=(BaseStationLongitude &b) { return this->longitude >= b.longitude; };
    bool operator>(BaseStationLongitude &b) { return this->longitude > b.longitude; };
};

// this class is for the convinience of sorting by latitude in the closest pair algorithm
class BaseStationLatitude :public BaseStation
{
public:
    int p;

    BaseStationLatitude(BaseStationLongitude &b, int p);
    BaseStationLatitude() :BaseStation() { p = 0; };
    BaseStationLatitude(const BaseStationLatitude &b);
    BaseStationLatitude operator=(BaseStation &b);

    bool operator<(BaseStationLatitude &b) { return this->latitude < b.latitude; };
    bool operator<=(BaseStationLatitude &b) { return this->latitude <= b.latitude; };
    bool operator==(BaseStationLatitude &b) { return this->latitude == b.latitude; };
    bool operator>=(BaseStationLatitude &b) { return this->latitude >= b.latitude; };
    bool operator>(BaseStationLatitude &b) { return this->latitude > b.latitude; };
};

BaseStation::BaseStation()
{
    this->enodebid = 0;
    this->k_dist = 0;
    this->latitude = 0;
    this->longitude = 0;
}

BaseStation::BaseStation(string line)
{
    size_t pos = 0;
    size_t len = line.length();
    size_t delim_len = 1;
    // to devide the line by ','
    vector<string> s;
    while (pos < len) {
        int find_pos = line.find(',', pos);
```

```
        if (find_pos < 0) {
            s.push_back(line.substr(pos, len - pos));
            break;
        }
        s.push_back(line.substr(pos, find_pos - pos));
        pos = find_pos + delim_len;
    }

    this->enodebid = stoi(s[0]);
    this->longitude = stod(s[1]);
    this->latitude = stod(s[2]);
    this->k_dist = stod(s[3]);
}

BaseStation::BaseStation(const BaseStation &b)
{
    this->enodebid = b.enodebid;
    this->longitude = b.longitude;
    this->latitude = b.latitude;
    this->k_dist = b.k_dist;
}

BaseStationLongitude::BaseStationLongitude(const BaseStationLongitude &b)
{
    this->enodebid = b.enodebid;
    this->longitude = b.longitude;
    this->latitude = b.latitude;
    this->k_dist = b.k_dist;
}

BaseStationLatitude::BaseStationLatitude(const BaseStationLatitude &b)
{
    this->enodebid = b.enodebid;
    this->longitude = b.longitude;
    this->latitude = b.latitude;
    this->k_dist = b.k_dist;
    this->p = b.p;
}

BaseStationLatitude BaseStationLatitude::operator=(BaseStation &b)
{
    this->enodebid = b.enodebid;
    this->k_dist = b.k_dist;
    this->latitude = b.latitude;
    this->longitude = b.longitude;
    return *this;
}

BaseStationLongitude BaseStationLongitude::operator=(BaseStation &b)
```

```
{
    this->enodebid = b.enodebid;
    this->k_dist = b.k_dist;
    this->latitude = b.latitude;
    this->longitude = b.longitude;
    return *this;
}

BaseStationLatitude::BaseStationLatitude(BaseStationLongitude &b, int p)
{
    this->enodebid = b.enodebid;
    this->k_dist = b.k_dist;
    this->latitude = b.latitude;
    this->longitude = b.longitude;
    this->p = p;
}

static vector<BaseStation> base; // file reading

void Init()
{
    fstream fp("1033base.csv");
    string s;
    getline(fp, s); // the first line

    while (getline(fp, s) && fp.good()) {
        BaseStation b(s);
        base.push_back(b);
    }
}

static int depthest; // wildly used to get the depth of many algorithms

double GetDistance(BaseStation A, BaseStation B)
{
    auto rad = [](const double& f) {return f * M_PI / 180.0; };
    double radLatA = rad(A.latitude), radLatB = rad(B.latitude);
    double radLonA = rad(A.longitude), radLonB = rad(B.longitude);
    double s = 1000 * EARTH_RADIUS * acos(cos(radLatA) * cos(radLatB) * cos(radLonA - radLonB) +
sin(radLatA) * sin(radLatB));
    //s = round(s * 1000);// round(s*1000,6)????
    return s;
}

// ------------------------ MergeSort With Recurrence -------------------------------

template <class Type>
void Merge(Type c[], Type d[], int left, int middle, int right)
{
```

```
    int i = left, j = middle + 1, k = left;
    while ((i <= middle) && (j <= right)) {
        if (c[i] < c[j])
            d[k++] = c[i++];
        else
            d[k++] = c[j++];
    }

    if (i > middle)
        while (j <= right)
            d[k++] = c[j++];
    else
        while (i <= middle)
            d[k++] = c[i++];
}

template <class Type>
void MergeSort(Type a[], Type b[], int left, int right, int depth = 0)
{
    depth += 1;
    if (depth > depthest)
        depthest = depth;

    if (left < right) {
        int i = (left + right) / 2;
        MergeSort(a, b, left, i, depth);
        MergeSort(a, b, i + 1, right, depth);
        Merge(a, b, left, i, right);
        for (int i = left; i <= right; i++)
            a[i] = b[i];
    }
}

// ---------------------- MergeSort Without Recurrence -------------------------------

template <class Type>
void MergePass(Type x[], Type y[], int s, int n)
{
    int i = 0;
    while (i <= n - 2 * s) {
        // Share the same funcion Merge with the above merge sort with recurrence
        Merge(x, y, i, i + s - 1, i + 2 * s - 1);
        i = i + 2 * s;
    }
    if (i + s <= n)
        Merge(x, y, i, i + s - 1, n - 1);
    else
        for (int j = i; j < n; j++) // Attention! The ending condition is wrong in PPT, should be j < n
            y[j] = x[j];
```

```
}

template <class Type>
void MergeSortNoRe(Type a[], int n)
{
    Type b[NUM_STATION];
    int s = 1;
    while (s < n) {
        MergePass(a, b, s, n);
        s += s;
        MergePass(b, a, s, n);
        s += s;
    }
}

// ------------------------------- Quick Sort ---------------------------------------

template<class Type>
int Partition(Type a[], int p, int r)
{
    int i = p, j = r + 1;
    Type x = a[p];
    while (true) {
        while (a[++i] < x && i < r);
        while (a[--j] > x);

        if (i >= j)
            break;

        // swap a[i] and a[j]
        Type t = a[i];
        a[i] = a[j];
        a[j] = t;
    }
    a[p] = a[j];
    a[j] = x;
    return j;
}

template<class Type>
void QuickSort(Type a[], int p, int r, int depth = 0)
{
    depth += 1;
    if (depth > depthest)
        depthest = depth;

    // to check if a[p:r] fits the condition that it doesn't decrease as the index increases
    int j = p;
    while (j < r && a[j] <= a[j + 1]) ++j;
```

```
    if (j >= r)
        return;

    if (p < r) {
        int q = Partition(a, p, r);
        QuickSort(a, p, q - 1, depth);
        QuickSort(a, q + 1, r, depth);
    }
}

// --------------------------- Randomized Quick Sort ---------------------------------

template<class Type>
int RandomizedPartition(Type a[], int p, int r)
{
    // get a random value from p to r
    srand((unsigned int)clock());
    int i = p + ((int)rand()) % (r - p);

    // swap a[i] and a[p]
    Type t = a[p];
    a[p] = a[i];
    a[i] = t;

    return Partition(a, p, r);
}

template<class Type>
void RandomizedQuickSort(Type a[], int p, int r, int depth = 0)
{
    depth += 1;
    if (depth > depthest)
        depthest = depth;

    // to check if a[p:r] fits the condition that it doesn't decrease as the index increases
    int j = p;
    while (j < r && a[j] <= a[j + 1]) ++j;
    if (j >= r)
        return;

    if (p < r) {
        int q = Partition(a, p, r);
        RandomizedQuickSort(a, p, q - 1,depth);
        RandomizedQuickSort(a, q + 1, r, depth);
    }
}

// ---------------------------- Randomized Select ---------------------------------
```

```
// 2 parts
template<class Type>
Type RandomizedSelect2(Type a[], int p, int r, int k, int depth = 0)
{
    depth += 1;
    if (depth > depthest)
        depthest = depth;

    if (p == r)
        return a[p];
    int i = RandomizedPartition(a, p, r);
    int L = i - p + 1;
    if (k <= L)
        return RandomizedSelect2(a, p, i, k, depth);
    else
        return RandomizedSelect2(a, i + 1, r, k - L, depth);
}

// 3 parts
template<class Type>
Type RandomizedSelect3(Type a[], int p, int r, int k, int depth = 0)
{
    depth += 1;
    if (depth > depthest)
        depthest = depth;

    if (p == r)
        return a[p];
    int i = RandomizedPartition(a, p, r);
    int L = i - p + 1;
    if (k == L)
        return a[i];
    else if (k < L)
        return RandomizedSelect3(a, p, i - 1, k, depth);
    else
        return RandomizedSelect3(a, i + 1, r, k - L, depth);
}

// ------------------------------ Closest Pair ------------------------------------

class Pair
{
public:
    BaseStation a, b;
    double dist;

    Pair() { dist = INF; };
    Pair(BaseStation a, BaseStation b, double d);
    Pair(const Pair &p) { this->a = p.a; this->b = p.b; this->dist = p.dist; };
```

```
    bool operator==(Pair &p) { return (this->a == p.a && this->b == p.b); };
    Pair operator=(Pair &p) { this->a = p.a; this->b = p.b; this->dist = p.dist; return *this; };
};

Pair::Pair(BaseStation a, BaseStation b, double d)
{
    this->a = a;
    this->b = b;
    this->dist = d;
}

Pair ClosestPair(BaseStationLongitude x[], BaseStationLatitude y[], BaseStationLatitude z[], int left,
int right, int depth = 0)
{
    depth += 1;
    if (depth > depthest)
        depthest = depth;

    if (right - left <= 0)
        return Pair();
    else if (right - left == 1)
        return Pair(x[left], x[right], GetDistance(x[left], x[right]));
    else if (right - left == 2) {
        double d1 = GetDistance(x[left], x[left + 1]);
        double d2 = GetDistance(x[left + 1], x[right]);
        double d3 = GetDistance(x[left], x[right]);
        if (d1 <= d2 && d1 <= d3)
            return Pair(x[left], x[left + 1], d1);
        else if (d2 <= d3)
            return Pair(x[left + 1], x[right], d2);
        else
            return Pair(x[left], x[right], d3);
    }


    int mid = (left + right) / 2;
    int f = left;
    for (int i = left; i <= right; i++) {
        if (y[i].longitude <= x[mid].longitude)
            z[f++] = y[i];
    }
    for (int i = left; i <= right; i++) {
        if (y[i].longitude > x[mid].longitude)
            z[f++] = y[i];
    }

    Pair dmin = ClosestPair(x, z, y, left, mid, depth);
    Pair dminr = ClosestPair(x, z, y, mid + 1, right, depth);
```

```
    dmin = dmin.dist < dminr.dist ? dmin : dminr;

    Merge(z, y, left, mid, right);

    int k = left;
    auto angle = [](const double& f) {return f * 180.0 / M_PI / EARTH_RADIUS / 1000; };
    double d = angle(dmin.dist);
    for (int i = left; i <= right; i++)
        if (abs(x[mid].longitude - y[i].longitude) <= d)
            z[k++] = y[i];

    for (int i = left; i < k; i++) {
        for (int j = i + 1; j < k && abs(z[j].latitude - z[i].latitude) <= d; j++) {
            double dp = GetDistance(z[i], z[j]);
            if (dp < dmin.dist)
                dmin = Pair(z[i], z[j], dp);
        }
    }

    return dmin;
}

// return the pair with smallest distance
Pair cpair2(BaseStation x[])
{
    // For the convinience of the calculation of the closest pair,
    // here we sort the bases by longitude and latitude respectively
    BaseStationLongitude LO[NUM_STATION];
    BaseStationLatitude LA1[NUM_STATION], LA2[NUM_STATION];
    for (int i = 0; i < NUM_STATION; i++)
        LO[i] = x[i];
    MergeSortNoRe(LO, NUM_STATION); // sort LO by longitude

    for (int i = 0; i < NUM_STATION; i++)
        LA1[i] = BaseStationLatitude(LO[i], i);
    MergeSortNoRe(LA1, NUM_STATION); // sort LA1 by latitude

    return ClosestPair(LO, LA1, LA2, 0, NUM_STATION - 1);
}

// ------------------------------ Closest Pair 2 -----------------------------------

void Compare(Pair &min1, Pair &min2, Pair &tmin1, Pair &tmin2)
{
    // To check and change if the min1 and min2 are
    // not the smallest and 2nd smallest number
    // in above 4 parameters
    if (tmin1.dist < min1.dist) {
```

```
        min2 = min1;
        min1 = tmin1;
        if (tmin2.dist < min2.dist)
            min2 = tmin2;
    }
    else if (tmin1.dist > min1.dist && tmin1.dist < min2.dist)
        min2 = tmin1;
    else if (tmin1.dist == min1.dist && tmin2.dist < min2.dist)
        min2 = tmin2;
}

void ClosestPair2(BaseStationLongitude x[], BaseStationLatitude y[], BaseStationLatitude z[], Pair &min1,
Pair &min2, int left, int right, int depth = 0)
{
    depth += 1;
    if (depth > depthest)
        depthest = depth;

    Pair tmin1, tmin2; //temperarily variable

    if (right - left <= 0)
        return;
    else if (right - left == 1) {
        tmin1 = Pair(x[left], x[right], GetDistance(x[left], x[right]));
        Compare(min1, min2, tmin1, tmin2);
        return;
    }
    else if (right - left == 2) {
        double d1 = GetDistance(x[left], x[left + 1]);
        double d2 = GetDistance(x[left + 1], x[right]);
        double d3 = GetDistance(x[left], x[right]);
        if (d1 <= d2 && d1 <= d3) {
            tmin1 = Pair(x[left], x[left + 1], d1);
            tmin2 = d2 <= d3 ? Pair(x[left + 1], x[right], d2) : Pair(x[left], x[right], d3);
        }
        else if (d2 <= d3) {
            tmin1 = Pair(x[left + 1], x[right], d2);
            tmin2 = d1 <= d3 ? Pair(x[left], x[left + 1], d1) : Pair(x[left], x[right], d3);
        }
        else {
            tmin1 = Pair(x[left], x[right], d3);
            tmin2 = d1 <= d2 ? Pair(x[left], x[left + 1], d1) : Pair(x[left + 1], x[right], d2);
        }
        Compare(min1, min2, tmin1, tmin2);
        return;
    }


    int mid = (left + right) / 2;
```

```
    int f = left;
    for (int i = left; i <= right; i++) {
        if (y[i].longitude <= x[mid].longitude)
            z[f++] = y[i];
    }
    for (int i = left; i <= right; i++) {
        if (y[i].longitude > x[mid].longitude)
            z[f++] = y[i];
    }

    ClosestPair2(x, z, y, tmin1, tmin2, left, mid, depth);
    Compare(min1, min2, tmin1, tmin2);
    ClosestPair2(x, z, y, tmin1, tmin2, mid + 1, right, depth);
    Compare(min1, min2, tmin1, tmin2);

    Merge(z, y, left, mid, right);

    int k = left;
    auto angle = [](const double& f) {return f * 180.0 / M_PI / EARTH_RADIUS / 1000 ; };
    double d = angle(min2.dist);
    for (int i = left; i <= right; i++)
        if (abs(x[mid].longitude - y[i].longitude) <= d)
            z[k++] = y[i];

    for (int i = left; i < k; i++) {
        for (int j = i + 1; j < k && abs(z[j].latitude - z[i].latitude) <= d; j++) {
            double dp = GetDistance(z[i], z[j]);
            if (dp < min1.dist) {
                min2 = min1;
                min1 = Pair(z[i], z[j], dp);
            }
            else if (dp > min1.dist && dp < min2.dist) {
                min2 = Pair(z[i], z[j], dp);
            }
        }
    }
}

// return both the pairs with smallest and 2nd smallest distances by min1 & min2 respectively
void cpair22(BaseStation x[], Pair &min1, Pair &min2)
{
    BaseStationLongitude LO[NUM_STATION];
    BaseStationLatitude LA1[NUM_STATION], LA2[NUM_STATION];
    for (int i = 0; i < NUM_STATION; i++)
        LO[i] = x[i];
    MergeSortNoRe(LO, NUM_STATION); // sort LO by longitude

    for (int i = 0; i < NUM_STATION; i++)
        LA1[i] = BaseStationLatitude(LO[i], i);
```

```
    MergeSortNoRe(LA1, NUM_STATION); // sort LA1 by latitude

    ClosestPair2(LO, LA1, LA2, min1, min2, 0, NUM_STATION - 1);
}

// ---------------------------------- Main ----------------------------------------

int main()
{
    // Read file and initialization
    Init();
    wfstream fp("Result.txt");
    BaseStation a[NUM_STATION], b[NUM_STATION];
    for (int i = 0; i < NUM_STATION; i++)
        a[i] = base[i];
    double duration;
    clock_t start, end;

    // Merge Sort With Recurrence
    depthest = 0;
    start = clock();
    MergeSort(a, b, 0, NUM_STATION - 1);
    end = clock();
    duration = (double)(end - start);
    fp << "The Merge Sort With Recurrence costs " << duration / CLOCKS_PER_SEC << " secondes, ";
    fp << "The depth of the recurrence is " << depthest << endl;
    fp << "Here's the sorting result:" << endl << "The format is: ENODEBID/LONGITUDE/LATITUDE/K_DIST " <<
endl;
    for (int i = 0; i < NUM_STATION; i++)
        fp << a[i].enodebid << ", " << a[i].longitude << ", " << a[i].latitude << ", " << a[i].k_dist <<
endl;
    fp << endl << endl;

    // Merge Sort Without Recurrence
    for (int i = 0; i < NUM_STATION; i++)
        a[i] = base[i];
    start = clock();
    MergeSortNoRe(a, NUM_STATION);
    end = clock();
    duration = (double)(end - start);
    fp << "The Merge Sort Without Recurrence costs " << duration / CLOCKS_PER_SEC << " secondes" << endl;
    fp << "Here's the sorting result:" << endl << "The format is: ENODEBID/LONGITUDE/LATITUDE/K_DIST " <<
endl;
    for (int i = 0; i < NUM_STATION; i++)
        fp << a[i].enodebid << ", " << a[i].longitude << ", " << a[i].latitude << ", " << a[i].k_dist <<
endl;
    fp << endl << endl;

    // Quick Sort
```

```
    for (int i = 0; i < NUM_STATION; i++)
        a[i] = base[i];
    depthest = 0;
    start = clock();
    QuickSort(a, 0, NUM_STATION - 1);
    end = clock();
    duration = (double)(end - start);
    fp << "The Quick Sort costs " << duration / CLOCKS_PER_SEC << " secondes, ";
    fp << "The depth of the recurrence is " << depthest << endl;
    fp << "Here's the sorting result:" << endl << "The format is: ENODEBID/LONGITUDE/LATITUDE/K_DIST " <<
endl;
    for (int i = 0; i < NUM_STATION; i++)
        fp << a[i].enodebid << ", " << a[i].longitude << ", " << a[i].latitude << ", " << a[i].k_dist <<
endl;
    fp << endl << endl;

    // Randomized Quick Sort
    for (int i = 0; i < NUM_STATION; i++)
        a[i] = base[i];
    depthest = 0;
    start = clock();
    RandomizedQuickSort(a, 0, NUM_STATION - 1);
    end = clock();
    duration = (double)(end - start);
    fp << "The Randomized Quick Sort costs " << duration / CLOCKS_PER_SEC << " secondes, ";
    fp << "The depth of the recurrence is " << depthest << endl;
    fp << "Here's the sorting result:" << endl << "The format is: ENODEBID/LONGITUDE/LATITUDE/K_DIST " <<
endl;
    for (int i = 0; i < NUM_STATION; i++)
        fp << a[i].enodebid << ", " << a[i].longitude << ", " << a[i].latitude << ", " << a[i].k_dist <<
endl;
    fp << endl << endl;

    BaseStation temp;

    // Randomized Select (2 parts) for 1st smallest k_dist
    for (int i = 0; i < NUM_STATION; i++)
        a[i] = base[i];
    depthest = 0;
    start = clock();
    temp = RandomizedSelect2(a, 0, NUM_STATION - 1, 1);
    end = clock();
    duration = (double)(end - start);
    fp << "The Randomized Select (2 parts) costs " << duration / CLOCKS_PER_SEC << " secondes, ";
    fp << "The depth of the recurrence is " << depthest << endl;
    fp << "The station with 1st smallest k_dist is " << temp.enodebid << ", and its k_dist is " <<
temp.k_dist << endl << endl;

    // Randomized Select (3 parts) for 1st smallest k_dist
```

```
    for (int i = 0; i < NUM_STATION; i++)
        a[i] = base[i];
    depthest = 0;
    start = clock();
    temp = RandomizedSelect3(a, 0, NUM_STATION - 1, 1);
    end = clock();
    duration = (double)(end - start);
    fp << "The Randomized Select (3 parts) costs " << duration / CLOCKS_PER_SEC << " secondes, ";
    fp << "The depth of the recurrence is " << depthest << endl;
    fp << "The station with 1st smallest k_dist is " << temp.enodebid << ", and its k_dist is " <<
temp.k_dist << endl << endl;

    // Randomized Select (2 parts) for 5th smallest k_dist
    for (int i = 0; i < NUM_STATION; i++)
        a[i] = base[i];
    depthest = 0;
    start = clock();
    temp = RandomizedSelect2(a, 0, NUM_STATION - 1, 5);
    end = clock();
    duration = (double)(end - start);
    fp << "The Randomized Select (2 parts) costs " << duration / CLOCKS_PER_SEC << " secondes, ";
    fp << "The depth of the recurrence is " << depthest << endl;
    fp << "The station with 5th smallest k_dist is " << temp.enodebid << ", and its k_dist is " <<
temp.k_dist << endl << endl;

    // Randomized Select (3 parts) for 5th smallest k_dist
    for (int i = 0; i < NUM_STATION; i++)
        a[i] = base[i];
    depthest = 0;
    start = clock();
    temp = RandomizedSelect3(a, 0, NUM_STATION - 1, 5);
    end = clock();
    duration = (double)(end - start);
    fp << "The Randomized Select (3 parts) costs " << duration / CLOCKS_PER_SEC << " secondes, ";
    fp << "The depth of the recurrence is " << depthest << endl;
    fp << "The station with 5th smallest k_dist is " << temp.enodebid << ", and its k_dist is " <<
temp.k_dist << endl << endl;

    // Randomized Select (2 parts) for 50th smallest k_dist
    for (int i = 0; i < NUM_STATION; i++)
        a[i] = base[i];
    depthest = 0;
    start = clock();
    temp = RandomizedSelect2(a, 0, NUM_STATION - 1, 50);
    end = clock();
    duration = (double)(end - start);
    fp << "The Randomized Select (2 parts) costs " << duration / CLOCKS_PER_SEC << " secondes, ";
    fp << "The depth of the recurrence is " << depthest << endl;
    fp << "The station with 50th smallest k_dist is " << temp.enodebid << ", and its k_dist is " <<
```

```
temp.k_dist << endl << endl;

    // Randomized Select (3 parts) for 50th smallest k_dist
    for (int i = 0; i < NUM_STATION; i++)
        a[i] = base[i];
    depthest = 0;
    start = clock();
    temp = RandomizedSelect3(a, 0, NUM_STATION - 1, 50);
    end = clock();
    duration = (double)(end - start);
    fp << "The Randomized Select (3 parts) costs " << duration / CLOCKS_PER_SEC << " secondes, ";
    fp << "The depth of the recurrence is " << depthest << endl;
    fp << "The station with 50th smallest k_dist is " << temp.enodebid << ", and its k_dist is " <<
temp.k_dist << endl << endl;

    // Randomized Select (2 parts) for largest k_dist
    for (int i = 0; i < NUM_STATION; i++)
        a[i] = base[i];
    depthest = 0;
    start = clock();
    temp = RandomizedSelect2(a, 0, NUM_STATION - 1, NUM_STATION);
    end = clock();
    duration = (double)(end - start);
    fp << "The Randomized Select (2 parts) costs " << duration / CLOCKS_PER_SEC << " secondes, ";
    fp << "The depth of the recurrence is " << depthest << endl;
    fp << "The station with largest k_dist is " << temp.enodebid << ", and its k_dist is " << temp.k_dist
<< endl << endl;

    // Randomized Select (3 parts) for largest k_dist
    for (int i = 0; i < NUM_STATION; i++)
        a[i] = base[i];
    depthest = 0;
    start = clock();
    temp = RandomizedSelect3(a, 0, NUM_STATION - 1, NUM_STATION);
    end = clock();
    duration = (double)(end - start);
    fp << "The Randomized Select (3 parts) costs " << duration / CLOCKS_PER_SEC << " secondes, ";
    fp << "The depth of the recurrence is " << depthest << endl;
    fp << "The station with largest k_dist is " << temp.enodebid << ", and its k_dist is " << temp.k_dist
<< endl << endl;

    // Closest Pair
    for (int i = 0; i < NUM_STATION; i++)
        a[i] = base[i];
    depthest = 0;
    start = clock();
    Pair dmin = cpair2(a);
    end = clock();
    duration = (double)(end - start);
```

```
    fp << "The Closest Pair costs " << duration / CLOCKS_PER_SEC << " secondes, ";
    fp << "The depth of the recurrence is " << depthest << endl;
    fp << "The pair station with minimun distance are " << dmin.a.enodebid << " and " << dmin.b.enodebid;
    fp << ", and its distance is " << dmin.dist << endl << endl;

    // Closest Pair 2
    Pair min1, min2;
    for (int i = 0; i < NUM_STATION; i++)
        a[i] = base[i];
    depthest = 0;
    start = clock();
    cpair22(a, min1, min2);
    end = clock();
    duration = (double)(end - start);
    fp << "The Closest Pair 2 costs " << duration / CLOCKS_PER_SEC << " secondes, ";
    fp << "The depth of the recurrence is " << depthest << endl;
    fp << "The pair station with minimun distance are " << min1.a.enodebid << " and " <<min1.b.enodebid;
    fp << ", and its distance is " << min1.dist << endl;
    fp << "The pair station with 2nd minimun distance are " << min2.a.enodebid << " and " <<
min2.b.enodebid;
    fp << ", and its distance is " << min2.dist << endl<< endl;
    fp.close();
}
```

结果部分：

```
The Merge Sort With Recurrence costs 0.001 secondes, The depth of the recurrence is 12
Here's the sorting result:
The format is: ENODEBID/LONGITUDE/LATITUDE/K_DIST
568030, 102.676, 25.0101, 103.075
566670, 102.72, 25.0458, 103.783
567387, 102.71, 24.9989, 107.679
566310, 102.711, 25.0404, 121.091
567883, 102.741, 25.0522, 126.096
566449, 102.707, 25.0408, 128.825
33635, 102.701, 25.0452, 136.177
566714, 102.742, 25.0529, 148.4
567074, 102.722, 25.0964, 161.289
566492, 102.71, 25.0412, 161.578
33246, 102.728, 25.0639, 171.394
33069, 102.718, 25.0705, 173.016
567759, 102.741, 25.0532, 183.765
33158, 102.753, 25.0336, 188.45
565672, 102.712, 25.0406, 189.012
567605, 102.722, 25.0384, 190.333
565623, 102.731, 25.0101, 195.087
567760, 102.74, 25.0513, 195.452
567603, 102.724, 25.077, 196.968
```

569113, 102.721, 25.0385, 197.559
566805, 102.72, 25.0369, 197.559
567759, 102.74, 25.0524, 198.127
565820, 102.729, 25.0417, 198.899
568112, 102.748, 25.0872, 200.609
567440, 102.747, 25.0888, 200.609
566054, 102.71, 25.0349, 201.75
566958, 102.754, 25.035, 201.965
567610, 102.726, 25.0208, 202.631
33099, 102.735, 25.0499, 202.746
568046, 102.721, 25.0969, 202.774
568184, 102.677, 25.054, 203.213
568182, 102.679, 25.0538, 203.213
566657, 102.737, 25.0501, 203.67
566623, 102.739, 25.0498, 203.67
567129, 102.697, 25.0475, 203.713
568177, 102.758, 25.0007, 204.172
565695, 102.706, 25.0392, 204.684
566803, 102.741, 25.0539, 205.009
565957, 102.71, 25.0333, 206.113
567389, 102.741, 25.0539, 206.125
566739, 102.739, 25.0543, 206.125
565366, 102.696, 25.0491, 206.318
33113, 102.743, 25.0758, 206.464
568175, 102.721, 25.0243, 206.595
567442, 102.699, 25.0478, 206.923
566769, 102.728, 25.0699, 207.014
566724, 102.726, 25.0695, 207.014
565051, 102.713, 25.0392, 207.063
567609, 102.738, 25.041, 207.314
568074, 102.753, 25.0353, 208.475
566475, 102.708, 25.0392, 209.546
565601, 102.722, 25.0062, 209.559
566403, 102.678, 25.0555, 210.516
565450, 102.721, 25.0076, 211.509
568172, 102.726, 25.0259, 212.159
566788, 102.737, 25.0589, 212.472
565587, 102.7, 25.0438, 212.478
567059, 102.709, 25.0423, 212.48
565660, 102.724, 25.0265, 212.484
565632, 102.723, 25.0282, 212.484
565052, 102.698, 25.0495, 213.456
566320, 102.7, 25.0393, 213.46
568186, 102.728, 25.0481, 213.947
567890, 102.722, 25.0271, 214.319
566414, 102.694, 25.0538, 214.436
567091, 102.719, 25.0693, 215.521
566674, 102.717, 25.0699, 215.521
565367, 102.693, 25.0521, 217.036

566796, 102.701, 25.0409, 217.053
566447, 102.699, 25.0402, 217.053
566324, 102.699, 25.0421, 217.465
566328, 102.711, 25.043, 217.788
566722, 102.732, 25.0839, 218.327
566798, 102.759, 25.0439, 218.432
33153, 102.737, 25.0166, 218.773
566598, 102.741, 25.0461, 218.872
567884, 102.735, 25.059, 219.362
567133, 102.736, 25.0572, 219.362
565477, 102.725, 25.0414, 219.368
566592, 102.729, 25.0651, 219.57
32788, 102.715, 24.985, 220.379
566314, 102.702, 25.0446, 220.944
568120, 102.728, 25.0401, 221.406
565356, 102.7, 25.046, 222.139
566624, 102.733, 25.0649, 222.156
566474, 102.71, 25.0398, 222.286
566306, 102.714, 25.0486, 223.745
566658, 102.744, 25.0544, 225.316
566354, 102.695, 25.0514, 225.317
565365, 102.694, 25.0496, 225.317
567058, 102.712, 25.0495, 226.21
566368, 102.678, 25.059, 227.308
566343, 102.677, 25.0572, 227.308
568173, 102.723, 25.0239, 229.318
565395, 102.724, 25.0221, 229.318
565829, 102.704, 25.0293, 229.339
567538, 102.719, 25.0443, 229.904
567088, 102.721, 25.0433, 229.904
567306, 102.752, 25.0272, 230.116
566415, 102.708, 25.0463, 230.814
567589, 102.722, 25.0781, 230.878
565625, 102.73, 25.0076, 230.979
567760, 102.74, 25.0519, 233.506
568849, 102.721, 25.0456, 233.518
568471, 102.721, 25.0456, 233.518
566603, 102.722, 25.0437, 233.518
565970, 102.678, 25.0031, 233.744
565969, 102.68, 25.0041, 233.744
568064, 102.756, 25.0351, 233.755
567098, 102.725, 25.0228, 234.28
566423, 102.712, 25.0449, 234.322
33164, 102.724, 25.0209, 235.335
567515, 102.746, 25.0883, 236.543
33075, 102.76, 25.0757, 236.825
567457, 102.706, 25.0403, 236.96
33070, 102.731, 25.0572, 237.699
33186, 102.691, 25.0324, 238.147

565981, 102.68, 25.0115, 238.354
565857, 102.682, 25.0103, 238.354
565448, 102.723, 25.0082, 238.361
565638, 102.721, 25.02, 238.399
568179, 102.688, 25.0525, 238.891
566472, 102.686, 25.0514, 238.891
565832, 102.709, 25.0359, 239.259
566342, 102.697, 25.0414, 239.505
566483, 102.709, 25.0581, 239.554
567232, 102.711, 25.0581, 239.857
566451, 102.71, 25.06, 239.857
567138, 102.719, 25.0353, 239.995
566262, 102.711, 25.0527, 240.819
566631, 102.746, 25.0465, 241.075
566535, 102.734, 25.0404, 242.493
565368, 102.689, 25.0527, 243.603
567443, 102.746, 25.0533, 244.28
567260, 102.748, 25.0521, 244.28
566655, 102.727, 25.0679, 244.408
568174, 102.717, 25.0297, 244.437
33155, 102.738, 25.0182, 244.703
33116, 102.751, 25.0736, 245.008
33117, 102.734, 25.0486, 245.042
567081, 102.74, 25.0414, 246.549
566645, 102.739, 25.0434, 246.549
33100, 102.727, 25.0659, 246.947
566673, 102.732, 25.0635, 247.456
567003, 102.694, 25.0453, 248.898
565669, 102.696, 25.0466, 248.898
566298, 102.708, 25.0289, 249.574
32777, 102.706, 25.0302, 249.574
567002, 102.746, 25.0285, 250.102
33159, 102.748, 25.0272, 250.102
32841, 102.719, 25.0211, 250.176
33067, 102.707, 25.0186, 250.524
566648, 102.712, 25.0633, 251.245
567076, 102.723, 25.0398, 251.542
567744, 102.72, 25.0306, 252.05
565523, 102.719, 25.0285, 252.05
568114, 102.734, 25.0825, 252.151
567191, 102.727, 25.0772, 252.158
566717, 102.725, 25.0786, 252.158
567526, 102.713, 24.9983, 253.606
565964, 102.711, 24.9969, 253.606
565728, 102.732, 25.009, 254.268
567646, 102.752, 25.0336, 254.845
566309, 102.708, 25.0378, 254.975
567467, 102.721, 25.0953, 255.447
567422, 102.719, 25.0967, 255.447

567469, 102.735, 25.0819, 255.609
565755, 102.733, 25.022, 255.61
565749, 102.735, 25.0234, 255.61
567498, 102.714, 25.0619, 256.025
32789, 102.728, 25.0067, 256.042
566442, 102.714, 25.0466, 257.297
33094, 102.716, 25.0481, 257.297
567116, 102.706, 25.0207, 257.676
566082, 102.705, 25.0185, 257.676
33192, 102.709, 24.985, 257.722
565773, 102.733, 24.9961, 258.403
566716, 102.717, 25.0578, 258.689
566416, 102.685, 25.0522, 259.026
567875, 102.708, 25.0329, 259.722
567111, 102.709, 25.0308, 259.722
565745, 102.721, 25.0224, 259.764
567139, 102.744, 25.0292, 261.57
567072, 102.693, 25.0403, 261.647
566393, 102.691, 25.0388, 261.647
568061, 102.745, 25.0314, 263.438
566259, 102.701, 25.0482, 264.135
566768, 102.741, 25.0762, 266.7
567109, 102.702, 25.0277, 266.91
566678, 102.713, 25.0598, 266.912
567342, 102.751, 25.0844, 267.11
566289, 102.714, 25.0433, 267.256
567709, 102.733, 25.0249, 267.646
565440, 102.731, 25.0233, 267.646
565906, 102.688, 25.0343, 267.95
566895, 102.74, 25.0302, 267.952
567238, 102.748, 25.0467, 268.978
566993, 102.747, 25.0445, 268.978
565830, 102.712, 25.0349, 269.102
565562, 102.721, 25.0046, 269.139
567073, 102.689, 25.0586, 269.589
567063, 102.691, 25.057, 269.589
568109, 102.733, 25.0753, 270.001
32778, 102.709, 24.9985, 271.965
566725, 102.759, 25.0776, 272.018
566741, 102.736, 25.097, 272.742
565382, 102.691, 25.0548, 273.544
33132, 102.693, 25.0532, 273.544
565578, 102.769, 25.0104, 273.596
567193, 102.728, 25.0791, 274.136
565420, 102.706, 25.0415, 274.312
32776, 102.704, 25.0432, 274.312
32997, 102.736, 25.0064, 274.961
566794, 102.724, 25.0721, 275.174
566601, 102.724, 25.0524, 275.813

565370, 102.687, 25.0493, 275.818
566804, 102.742, 25.0431, 275.826
566801, 102.755, 25.0784, 275.857
566662, 102.728, 25.0749, 276.209
565407, 102.693, 25.0561, 276.215
566628, 102.723, 25.0757, 277.245
567474, 102.698, 25.0211, 277.985
33636, 102.7, 25.0228, 277.985
566711, 102.698, 25.0564, 278.022
568102, 102.726, 25.0475, 278.877
566712, 102.679, 25.0437, 278.881
566004, 102.677, 25.042, 278.881
567553, 102.735, 25.0985, 279.313
567267, 102.753, 25.085, 279.317
566459, 102.684, 25.0498, 279.329
32881, 102.685, 25.0475, 279.329
568850, 102.734, 25.0047, 279.342
33566, 102.733, 25.0024, 279.342
565457, 102.738, 25.0135, 279.687
568116, 102.725, 25.0839, 279.755
567328, 102.739, 25.0891, 279.94
565593, 102.734, 25.0046, 280.313
568118, 102.737, 25.09, 280.368
565505, 102.739, 25.0159, 280.376
566639, 102.718, 25.0557, 280.564
33108, 102.716, 25.0539, 280.564
567077, 102.726, 25.0703, 280.622
567591, 102.736, 25.0749, 281.295
566486, 102.711, 25.0374, 281.409
568099, 102.728, 25.0509, 281.577
566637, 102.73, 25.0492, 281.577
567618, 102.728, 25.0014, 281.869
565912, 102.687, 25.0365, 282.292
566095, 102.692, 25.0285, 282.771
566615, 102.72, 25.0645, 283.51
568113, 102.748, 25.0912, 283.89
567231, 102.719, 25.0651, 285.042
566666, 102.718, 25.0627, 285.042
566669, 102.732, 25.0474, 285.491
567245, 102.73, 25.0825, 285.557
566737, 102.731, 25.0848, 285.557
567264, 102.723, 25.0413, 285.882
566650, 102.716, 25.0466, 285.888
566293, 102.714, 25.0448, 285.888
566652, 102.738, 25.0477, 288.486
567212, 102.685, 25.0383, 288.592
565691, 102.732, 25.0066, 289.107
567125, 102.726, 25.073, 291.43
565721, 102.731, 25.016, 291.497

566363, 102.761, 25.066, 292.245
565757, 102.715, 25.03, 292.562
33133, 102.693, 25.0471, 292.662
565677, 102.784, 25.0009, 292.78
565445, 102.727, 25.0204, 292.879
566280, 102.709, 25.0515, 293.715
567291, 102.749, 25.0869, 293.834
33114, 102.742, 25.0737, 293.909
565828, 102.713, 25.0319, 294.709
566963, 102.754, 25.0317, 294.788
567470, 102.734, 25.0858, 294.846
566757, 102.735, 25.0882, 294.846
566944, 102.735, 25.0381, 295.513
565804, 102.737, 25.0401, 295.513
566065, 102.712, 25.0296, 297.061
565681, 102.758, 24.99, 297.071
567135, 102.727, 25.0464, 297.136
565559, 102.721, 24.9853, 297.202
565558, 102.723, 24.9873, 297.202
567315, 102.738, 25.0044, 297.836
565877, 102.703, 25.0203, 298.11
33184, 102.702, 25.0228, 298.11
567865, 102.678, 25.0051, 298.114
566865, 102.69, 25.0323, 299.611
567694, 102.69, 25.0278, 300.44
566005, 102.686, 25.0192, 301.275
567492, 102.698, 25.0364, 302.167
565436, 102.736, 25.0356, 302.3
566260, 102.698, 25.0514, 302.544
567041, 102.763, 25.0641, 302.585
565455, 102.76, 25.064, 302.585
566003, 102.681, 25.0405, 302.62
565868, 102.678, 25.0406, 302.62
567355, 102.743, 25.0111, 302.645
565608, 102.73, 25.0024, 302.682
565894, 102.708, 24.9829, 302.713
565865, 102.711, 24.9829, 302.713
566671, 102.743, 25.0415, 302.733
566742, 102.749, 25.0532, 302.823
567099, 102.74, 25.0194, 302.843
567409, 102.688, 25.0547, 302.869
565994, 102.687, 25.0324, 302.884
566965, 102.775, 25.0384, 302.911
566081, 102.69, 24.9962, 302.973
566399, 102.688, 25.0572, 303.065
566597, 102.724, 25.0458, 303.148
565720, 102.735, 24.9946, 303.18
565780, 102.74, 25.0117, 303.232
565774, 102.737, 25.0116, 303.232

565463, 102.742, 25.0157, 303.293
567204, 102.677, 25.0076, 303.357
567524, 102.68, 25.0024, 303.401
565482, 102.73, 25.0289, 303.416
565732, 102.761, 24.9898, 303.432
565607, 102.74, 25.0195, 303.439
565461, 102.743, 25.0193, 303.439
566668, 102.714, 25.0576, 303.608
33112, 102.734, 25.0846, 303.618
565729, 102.719, 25.0245, 303.76
568066, 102.744, 25.0125, 303.815
565393, 102.714, 24.9832, 304.091
566614, 102.723, 25.0698, 304.099
566605, 102.722, 25.0369, 304.134
33077, 102.719, 25.0372, 304.134
33078, 102.757, 25.076, 304.141
565965, 102.683, 25.0201, 304.267
567303, 102.769, 24.9857, 304.305
567510, 102.714, 24.9972, 304.513
567415, 102.745, 25.0762, 304.537
566682, 102.716, 25.0494, 304.637
565951, 102.693, 25.0313, 304.681
566764, 102.75, 25.0757, 305.024
565682, 102.76, 25.0005, 305.166
568864, 102.687, 25.0455, 305.403
565369, 102.689, 25.0476, 305.403
565360, 102.748, 25.0074, 305.446
565850, 102.678, 25.0128, 305.895
565526, 102.736, 25.0253, 306.03
565976, 102.732, 25.0586, 306.116
566028, 102.685, 25.0221, 306.294
565861, 102.688, 25.0298, 306.502
567286, 102.68, 25.0198, 306.576
567693, 102.751, 25.0256, 307.03
565489, 102.752, 25.023, 307.03
566058, 102.696, 25.0316, 307.082
565903, 102.699, 25.0311, 307.082
567010, 102.756, 25.0286, 307.089
567205, 102.68, 25.0081, 307.139
33162, 102.757, 24.9715, 307.228
565634, 102.73, 25.0209, 307.299
568176, 102.74, 25.0111, 307.341
565905, 102.69, 25.036, 307.457
566610, 102.721, 25.0671, 307.544
565595, 102.763, 25.0064, 307.79
567481, 102.694, 24.9972, 307.802
567199, 102.684, 25.0247, 307.928
566772, 102.75, 25.0729, 307.979
566735, 102.753, 25.0724, 307.979

566684, 102.732, 25.0439, 308.048
566651, 102.729, 25.0444, 308.048
566799, 102.742, 25.0784, 308.172
567025, 102.716, 25.0324, 308.438
565971, 102.679, 25.0167, 308.983
565622, 102.735, 25.0095, 309.226
565430, 102.782, 25.0018, 309.245
565636, 102.719, 25.0319, 309.332
567889, 102.754, 25.0807, 309.519
567347, 102.757, 25.0801, 309.519
566006, 102.678, 25.0446, 309.655
566370, 102.697, 25.0444, 309.917
566986, 102.759, 25.0281, 310.47
566981, 102.757, 25.026, 310.47
566681, 102.719, 25.0774, 310.755
33110, 102.731, 25.0864, 310.811
565438, 102.774, 24.9818, 311.241
565496, 102.73, 24.9931, 311.352
566627, 102.725, 25.0646, 311.463
567413, 102.693, 24.9986, 311.886
566068, 102.696, 24.9993, 311.886
567295, 102.687, 25.0169, 312.008
565914, 102.685, 25.0147, 312.008
566316, 102.722, 24.9836, 312.331
565627, 102.725, 24.9829, 312.331
565550, 102.701, 25.0325, 312.817
565403, 102.723, 24.9809, 312.936
566686, 102.718, 25.0532, 312.99
567882, 102.731, 25.0271, 313.017
566608, 102.73, 25.0689, 314.108
566751, 102.76, 25.0469, 314.159
566740, 102.729, 25.0628, 314.423
565563, 102.75, 25.0089, 314.551
33157, 102.753, 25.0097, 314.551
33235, 102.745, 24.9613, 314.696
565042, 102.735, 25.0422, 314.878
565926, 102.703, 25.0255, 314.927
566749, 102.734, 25.0775, 315.004
566718, 102.731, 25.0767, 315.004
566394, 102.696, 25.0384, 315.096
566392, 102.693, 25.0376, 315.096
567110, 102.691, 25.0226, 315.132
567257, 102.738, 25.0453, 315.548
567134, 102.734, 25.0452, 315.557
566083, 102.691, 25.0108, 315.595
567505, 102.762, 25.0768, 315.632
33134, 102.71, 25.0473, 315.726
567310, 102.739, 25.0084, 315.88
565974, 102.71, 25.0131, 316.105

565801, 102.723, 25.0005, 316.134
565630, 102.726, 24.9997, 316.134
566719, 102.737, 25.0767, 316.276
565594, 102.761, 25.0083, 316.764
565980, 102.688, 25.0212, 317.064
567202, 102.733, 25.0905, 317.227
568181, 102.683, 25.0537, 317.263
567459, 102.724, 25.0822, 317.324
566758, 102.727, 25.0814, 317.324
566641, 102.75, 25.0697, 317.553
566607, 102.729, 25.0594, 317.648
568100, 102.745, 25.0633, 317.766
567658, 102.744, 25.066, 317.766
567622, 102.734, 25.0798, 317.941
567190, 102.731, 25.0789, 317.941
565631, 102.72, 24.9999, 318.182
567885, 102.733, 25.0681, 318.311
566629, 102.736, 25.069, 318.311
567087, 102.732, 25.0521, 318.349
566112, 102.729, 25.053, 318.349
565810, 102.777, 24.9921, 318.946
33629, 102.692, 24.995, 319.219
567082, 102.719, 25.059, 319.39
566422, 102.713, 25.0549, 319.471
566079, 102.696, 25.0216, 319.62
565973, 102.693, 25.0207, 319.62
565402, 102.747, 24.99, 319.828
565822, 102.682, 25.0301, 320.178
565652, 102.746, 25.01, 320.208
565459, 102.747, 25.0127, 320.208
567097, 102.775, 24.9956, 320.366
568069, 102.755, 24.9836, 320.689
567095, 102.739, 25.0562, 320.713
565616, 102.721, 25.0341, 320.736
566611, 102.721, 25.0731, 320.813
566635, 102.736, 25.0552, 320.854
565859, 102.714, 24.9858, 321.013
565610, 102.717, 24.9868, 321.013
567269, 102.764, 24.9974, 321.268
565532, 102.763, 25.0001, 321.268
567137, 102.743, 25.0562, 321.579
565740, 102.772, 24.9946, 321.743
565565, 102.775, 24.9936, 321.743
33547, 102.743, 24.9785, 322.31
566093, 102.702, 24.995, 322.502
566226, 102.733, 25.0542, 322.751
566057, 102.694, 25.059, 322.859
565598, 102.697, 25.058, 322.859
566616, 102.726, 25.054, 323.143

567149, 102.681, 24.9995, 323.269
566496, 102.707, 25.0496, 323.31
566263, 102.71, 25.0486, 323.31
565479, 102.738, 25.0371, 323.573
568119, 102.751, 25.0603, 323.638
568183, 102.704, 25.0466, 323.949
566327, 102.707, 25.0456, 323.949
33074, 102.761, 25.0799, 324.86
33120, 102.721, 25.0797, 325.035
567323, 102.763, 24.9699, 325.331
565736, 102.76, 24.971, 325.331
568178, 102.68, 25.0583, 325.54
566067, 102.702, 25.0352, 325.591
566225, 102.757, 25.0634, 325.939
566224, 102.754, 25.0645, 325.939
567542, 102.78, 24.9782, 326.01
565768, 102.766, 25.0059, 326.073
565486, 102.769, 25.0048, 326.073
566638, 102.735, 25.0424, 326.44
567888, 102.753, 25.0693, 326.795
566654, 102.751, 25.067, 326.795
566765, 102.748, 25.0781, 326.994
565618, 102.728, 25.0259, 327.282
567224, 102.687, 25.013, 327.312
566080, 102.682, 25.0113, 327.613
565986, 102.683, 25.0085, 327.613
565371, 102.694, 25.0425, 327.673
566793, 102.716, 25.065, 328.554
566344, 102.685, 25.0559, 328.852
566636, 102.736, 25.0531, 329.823
566761, 102.746, 25.055, 330.264
566766, 102.747, 25.0741, 330.671
566604, 102.74, 25.0616, 330.877
568106, 102.742, 25.062, 331.2
566643, 102.744, 25.0597, 331.2
566023, 102.688, 25.0028, 331.61
566059, 102.684, 25.0312, 331.671
33161, 102.739, 25.0011, 331.855
33156, 102.74, 25.0039, 331.855
567270, 102.767, 24.9983, 331.973
567602, 102.753, 25.076, 332.335
567130, 102.691, 25.041, 332.569
565904, 102.688, 25.0398, 332.569
33181, 102.675, 25.01, 333.02
565473, 102.748, 24.9612, 333.209
567067, 102.696, 25.0358, 333.511
566404, 102.693, 25.0345, 333.511
565380, 102.741, 25.028, 333.526
567381, 102.769, 24.9927, 333.608

566134, 102.715, 25.0419, 333.638
567207, 102.726, 25.0431, 333.962
566040, 102.684, 24.9985, 333.978
565979, 102.688, 25.0234, 334.481
565648, 102.728, 24.9953, 334.806
565654, 102.724, 25.031, 335.031
567745, 102.73, 25.0737, 335.036
566679, 102.742, 25.0447, 335.34
565571, 102.719, 25.0125, 335.46
565735, 102.762, 24.9876, 335.703
565733, 102.764, 24.99, 335.703
566664, 102.735, 25.0657, 335.893
33195, 102.702, 25.0284, 336.306
566653, 102.716, 25.0511, 336.341
566634, 102.719, 25.0498, 336.341
33109, 102.761, 25.0431, 336.404
568104, 102.738, 25.0831, 336.514
565743, 102.77, 24.988, 336.627
566602, 102.746, 25.0575, 337.305
566391, 102.68, 25.0523, 337.314
566278, 102.704, 25.0482, 337.817
565869, 102.689, 25.0086, 337.904
565414, 102.766, 24.9709, 338.218
565490, 102.744, 25.0082, 338.328
565962, 102.704, 24.9928, 338.375
565883, 102.702, 24.9904, 338.375
565516, 102.719, 24.9891, 338.444
566806, 102.743, 25.0897, 339.076
566626, 102.733, 25.0615, 339.272
566646, 102.748, 25.0656, 339.277
566871, 102.736, 25.1009, 339.708
565754, 102.733, 25.0174, 340.346
565840, 102.779, 25.0042, 340.633
565744, 102.776, 25.0028, 340.633
565431, 102.74, 25.0069, 341.048
566091, 102.675, 25.0132, 341.41
565851, 102.676, 25.0161, 341.41
566002, 102.686, 25.0099, 341.441
566277, 102.713, 25.0525, 341.603
567491, 102.707, 24.9864, 342.526
565844, 102.71, 24.9878, 342.526
565620, 102.74, 25.0344, 342.947
565379, 102.743, 25.0329, 342.947
566744, 102.752, 25.0573, 343.208
567071, 102.69, 25.0443, 343.451
568171, 102.721, 25.0157, 343.51
33079, 102.764, 25.0793, 343.676
565577, 102.719, 25.0199, 343.731
565896, 102.704, 24.9824, 343.767

566297, 102.729, 25.0561, 343.95
567293, 102.745, 25.0926, 344.139
566315, 102.705, 25.0362, 344.596
565442, 102.697, 25.0286, 344.598
566957, 102.772, 25.0215, 344.634
566748, 102.721, 25.0807, 345.069
565383, 102.684, 25.0478, 346.109
565408, 102.734, 25.0145, 347.267
566738, 102.751, 25.0792, 347.347
567485, 102.754, 25.0873, 347.459
565564, 102.775, 24.974, 348.297
567376, 102.747, 25.0681, 348.525
565902, 102.683, 25.0364, 348.784
33182, 102.681, 25.0339, 348.784
565723, 102.758, 24.9827, 349.196
565605, 102.756, 24.9852, 349.196
566755, 102.729, 25.0907, 350.005
33118, 102.731, 25.0881, 350.005
566593, 102.726, 25.0382, 350.541
567483, 102.683, 25.0025, 350.616
565998, 102.682, 25.0141, 350.988
567285, 102.749, 25.0173, 350.993
565456, 102.75, 25.0143, 350.993
567500, 102.717, 24.9947, 350.998
565898, 102.718, 24.9977, 350.998
568755, 102.693, 24.9831, 351.001
33589, 102.692, 24.9861, 351.001
567617, 102.706, 25.0263, 351.886
567263, 102.719, 25.0421, 352.116
33122, 102.676, 25.0522, 352.209
567417, 102.726, 25.0093, 352.452
566976, 102.769, 25.0224, 352.802
565968, 102.707, 25.0033, 352.887
565900, 102.71, 25.0017, 352.887
565621, 102.73, 24.9977, 352.899
567461, 102.701, 24.9813, 352.987
33051, 102.74, 25.025, 353.013
566090, 102.687, 25.0428, 353.12
565506, 102.779, 25.0012, 353.232
565497, 102.745, 24.9797, 353.756
565481, 102.743, 24.9823, 353.756
567354, 102.726, 25.0322, 354.19
565501, 102.727, 25.0291, 354.19
567208, 102.75, 25.0822, 354.566
566777, 102.747, 25.0806, 354.566
567494, 102.727, 25.0839, 354.795
565813, 102.78, 24.9905, 355.221
567411, 102.881, 25.0511, 356.326
565521, 102.795, 24.9908, 357.16

567747, 102.722, 25.0113, 357.944
566048, 102.689, 24.9933, 358.159
566773, 102.741, 25.0817, 358.575
566159, 102.706, 25.0598, 358.621
566956, 102.754, 25.06, 359.19
565950, 102.697, 25.0245, 359.293
568059, 102.748, 24.9836, 359.314
567298, 102.682, 25.017, 359.532
567531, 102.711, 24.9909, 360.073
567290, 102.77, 25.1004, 360.075
565560, 102.733, 25.0297, 360.14
33172, 102.724, 25.014, 360.395
567497, 102.727, 25.0046, 361.031
567598, 102.74, 25.0868, 361.219
566362, 102.755, 25.0666, 361.95
565396, 102.732, 25.0377, 362.901
566941, 102.742, 24.9605, 363.681
566754, 102.732, 25.0925, 364.032
566756, 102.741, 25.0919, 364.707
565551, 102.722, 24.9904, 364.884
565658, 102.76, 24.9632, 364.917
566864, 102.711, 25.0271, 366.031
566432, 102.715, 25.0388, 366.383
568020, 102.696, 25.0183, 366.536
566647, 102.726, 25.0576, 366.596
565955, 102.706, 25.0236, 366.665
566660, 102.718, 25.0731, 367.006
565975, 102.704, 25.0154, 367.018
565817, 102.735, 25.0325, 367.548
568054, 102.759, 24.9926, 367.584
565999, 102.694, 25.0267, 367.921
567313, 102.738, 25.08, 369.084
565775, 102.739, 25.0334, 369.156
565641, 102.761, 25.0036, 369.162
565576, 102.769, 24.979, 369.292
565378, 102.682, 25.045, 369.398
33163, 102.738, 25.0302, 370.227
565642, 102.751, 25.0291, 370.985
566677, 102.722, 25.0522, 371.261
566625, 102.724, 25.0494, 371.261
565824, 102.687, 24.9961, 371.311
33063, 102.728, 25.0179, 371.782
567528, 102.764, 25.0967, 371.883
33119, 102.724, 25.0503, 372.291
565913, 102.683, 25.0418, 372.367
567261, 102.694, 24.9872, 373.937
566092, 102.696, 24.9901, 373.937
566599, 102.739, 25.0693, 374.052
33115, 102.741, 25.0721, 374.052

568185, 102.704, 25.0476, 374.323
567891, 102.725, 24.9858, 374.97
565633, 102.727, 24.9887, 374.97
565639, 102.777, 25.0071, 375.047
565753, 102.723, 25.003, 375.591
565760, 102.751, 24.9854, 375.721
565494, 102.749, 24.9882, 375.721
566600, 102.742, 25.0478, 375.968
567294, 102.745, 24.9625, 376.472
566980, 102.762, 25.0293, 376.659
566069, 102.694, 25.0124, 376.662
566007, 102.693, 25.0091, 376.662
566591, 102.748, 25.0613, 377.134
33111, 102.727, 25.0918, 377.416
33106, 102.725, 25.0889, 377.416
565624, 102.732, 25.0116, 377.86
565758, 102.766, 24.9771, 379.778
565808, 102.772, 24.973, 380.455
565674, 102.769, 24.9751, 380.455
567488, 102.745, 24.9584, 380.483
33562, 102.727, 25.0868, 380.625
565592, 102.746, 25.0053, 380.693
566644, 102.737, 25.0628, 380.927
568122, 102.81, 25.0387, 381.609
565640, 102.758, 25.0057, 381.748
566031, 102.69, 25.0141, 382.027
565705, 102.783, 24.9985, 382.03
566612, 102.73, 25.041, 382.552
566732, 102.757, 25.0579, 383.003
565756, 102.746, 24.986, 386.372
565508, 102.772, 24.9915, 386.569
565719, 102.724, 25.0143, 386.748
33194, 102.713, 25.0137, 387.316
565572, 102.718, 24.9831, 389.371
565805, 102.774, 25.0058, 389.894
565354, 102.695, 25.061, 390.133
565841, 102.704, 24.9886, 390.769
567007, 102.77, 25.0272, 390.836
566974, 102.767, 25.025, 390.836
566734, 102.749, 25.0918, 391.06
567188, 102.687, 24.9833, 391.49
565878, 102.677, 25.0374, 392.06
565983, 102.708, 24.9909, 392.1
567895, 102.722, 25.0581, 393.841
566971, 102.77, 25.019, 396.002
565579, 102.772, 25.0089, 397.539
567439, 102.757, 25.0477, 397.695
566802, 102.755, 25.0508, 397.695
566656, 102.723, 25.0615, 398.836

567256, 102.754, 25.0556, 398.858
566775, 102.777, 25.0404, 399.013
566789, 102.737, 25.0725, 399.222
565647, 102.76, 24.9978, 399.329
565982, 102.683, 24.9957, 400.06
565881, 102.68, 24.998, 400.06
567897, 102.704, 25.0567, 400.474
567236, 102.746, 25.0261, 400.595
567209, 102.778, 25.0244, 401.032
566970, 102.781, 25.0221, 401.032
565480, 102.782, 24.9806, 402.472
566596, 102.733, 25.0724, 403.364
566467, 102.68, 25.06, 403.386
565491, 102.756, 25.0004, 403.571
565435, 102.755, 24.9942, 403.579
565513, 102.747, 25.0192, 403.681
566978, 102.778, 25.0163, 403.806
565845, 102.713, 25.0027, 403.809
565997, 102.711, 25.0188, 403.973
567964, 102.697, 25.0169, 404.081
566044, 102.701, 25.0167, 404.081
567640, 102.773, 25.0402, 404.31
566778, 102.787, 25.04, 404.449
565766, 102.766, 25.0018, 404.511
565566, 102.77, 25.0021, 404.511
565961, 102.705, 24.9963, 404.626
565458, 102.754, 25.0146, 404.629
566945, 102.871, 25.0472, 404.693
565680, 102.759, 24.9942, 404.868
565646, 102.763, 24.9939, 404.868
565535, 102.757, 25.0202, 405.263
565488, 102.753, 25.0205, 405.263
567322, 102.761, 25.0496, 405.608
566011, 102.711, 25.0166, 405.7
565827, 102.715, 25.036, 406.468
565666, 102.729, 24.9825, 406.577
565888, 102.716, 25.0112, 406.654
565582, 102.767, 25.012, 406.756
565671, 102.773, 24.9765, 406.786
565696, 102.747, 25.0413, 406.947
566261, 102.709, 25.0544, 407.126
565750, 102.759, 25.0602, 407.415
565612, 102.745, 24.9667, 407.798
567585, 102.739, 25.0995, 407.987
565035, 102.868, 25.0264, 408.191
565027, 102.872, 25.0269, 408.191
565659, 102.758, 24.965, 408.392
565613, 102.754, 24.9645, 408.392
33180, 102.698, 24.9878, 408.697

565416, 102.767, 24.9949, 408.922
566000, 102.714, 25.008, 409.393
566055, 102.688, 24.9867, 409.462
567203, 102.747, 25.0391, 410.107
565352, 102.699, 25.0616, 410.235
568056, 102.729, 24.9852, 410.46
565355, 102.705, 25.0537, 410.945
565643, 102.756, 24.997, 411.45
565806, 102.764, 24.9741, 411.459
565419, 102.724, 25.0172, 411.603
567320, 102.753, 25.0181, 411.713
565524, 102.729, 25.0141, 411.899
567547, 102.757, 25.0439, 411.942
567287, 102.696, 25.0067, 412.066
567175, 102.697, 25.0031, 412.066
565925, 102.702, 24.9849, 412.287
565710, 102.732, 24.9673, 412.367
565600, 102.736, 24.9681, 412.367
567604, 102.727, 25.0856, 412.378
565958, 102.709, 25.0232, 412.638
567009, 102.769, 25.0331, 412.804
566988, 102.765, 25.0338, 412.804
565819, 102.783, 24.984, 413.246
567557, 102.732, 25.096, 414.197
566877, 102.728, 25.0951, 414.197
566728, 102.776, 25.0439, 414.411
33140, 102.705, 25.0524, 415.165
567896, 102.7, 25.0547, 415.573
567701, 102.743, 25.085, 416.133
565759, 102.774, 25.0004, 416.215
33238, 102.77, 24.9995, 416.215
567023, 102.777, 25.0187, 416.45
566972, 102.774, 25.0161, 416.45
565492, 102.726, 24.9914, 416.655
567159, 102.686, 25.0051, 416.781
565415, 102.731, 25.033, 417.148
565531, 102.733, 25.001, 417.574
566785, 102.79, 25.0544, 418.008
566590, 102.741, 25.066, 418.094
565525, 102.771, 24.9824, 418.16
565454, 102.758, 25.0877, 418.393
33577, 102.755, 25.0903, 418.393
565510, 102.773, 24.9867, 418.972
566468, 102.71, 25.063, 419.824
566979, 102.763, 25.0261, 420.065
565604, 102.763, 25.0116, 420.091
565684, 102.776, 24.9894, 420.115
567163, 102.744, 25.0153, 420.454
566012, 102.717, 25.0057, 420.754

567167, 102.774, 24.9984, 421.569
566969, 102.758, 25.0362, 423.266
565884, 102.675, 25.0344, 423.44
565864, 102.679, 25.0332, 423.44
565739, 102.779, 24.9848, 423.729
565673, 102.777, 24.9815, 423.729
565427, 102.692, 24.9839, 424.217
565464, 102.757, 24.9613, 425.147
567061, 102.7, 25.053, 425.241
565978, 102.713, 24.9766, 425.445
33048, 102.717, 24.9778, 425.445
565889, 102.709, 25.0054, 426.221
565410, 102.715, 25.0187, 426.453
566074, 102.715, 24.9904, 429.13
565606, 102.743, 25.0234, 429.421
567078, 102.714, 25.0737, 429.838
33095, 102.716, 25.0771, 429.838
567522, 102.738, 24.9944, 430.183
565689, 102.735, 24.9972, 430.183
565678, 102.783, 24.9888, 430.509
565498, 102.74, 24.9795, 432.111
568097, 102.781, 25.0536, 433.102
565637, 102.767, 25.0097, 433.795
565468, 102.752, 24.9627, 434.346
565737, 102.76, 24.9768, 435.13
566975, 102.776, 25.023, 435.826
566783, 102.746, 25.0502, 436.755
565675, 102.72, 24.9939, 437.786
565609, 102.778, 24.9965, 439.131
565661, 102.756, 24.9753, 440.3
566010, 102.717, 25.0028, 440.898
566949, 102.874, 25.0478, 440.97
566948, 102.878, 25.0494, 440.97
565734, 102.766, 24.986, 441.156
566412, 102.705, 25.063, 441.38
565334, 102.701, 25.0614, 441.38
566750, 102.751, 25.0492, 441.824
565724, 102.715, 25.0216, 442.871
567094, 102.722, 25.0559, 443.223
568098, 102.754, 25.0538, 443.676
568027, 102.746, 24.9756, 446.31
565668, 102.732, 24.9822, 446.626
32775, 102.684, 25.06, 446.98
566967, 102.751, 25.0408, 448.388
567326, 102.677, 25.0203, 449.365
567089, 102.743, 25.0384, 449.817
567568, 102.742, 24.9878, 450.815
565800, 102.741, 24.9838, 450.815
33575, 102.748, 25.0633, 451.678

567527, 102.788, 25.0558, 453.568
567244, 102.784, 25.054, 453.568
566763, 102.767, 25.0929, 454.328
565885, 102.707, 25.0123, 456.488
565667, 102.732, 24.9866, 457.652
565527, 102.731, 24.9906, 457.652
566073, 102.698, 24.9829, 458.811
33160, 102.76, 25.0322, 461.012
566984, 102.767, 25.0293, 461.131
565645, 102.704, 25.0577, 463.071
565602, 102.758, 24.9672, 463.88
566753, 102.738, 25.0954, 463.903
565529, 102.754, 24.9671, 464.81
565472, 102.739, 24.9692, 465.08
565471, 102.743, 24.9671, 465.08
565657, 102.778, 24.9774, 465.258
567266, 102.689, 25.0007, 466.714
567241, 102.752, 24.9992, 467.504
567892, 102.747, 24.9699, 467.693
566985, 102.749, 24.9661, 467.693
567319, 102.761, 24.9794, 468.322
567265, 102.76, 25.0833, 469.267
565635, 102.748, 25.0358, 469.607
566680, 102.713, 25.0682, 469.887
565412, 102.755, 25.0045, 469.928
565048, 102.769, 25.0894, 471.047
566874, 102.73, 25.0989, 471.623
565767, 102.764, 25.0153, 472.531
568110, 102.773, 25.0471, 472.652
565514, 102.782, 24.9943, 474.278
567441, 102.763, 25.0567, 475.755
567615, 102.756, 24.9892, 475.995
567767, 102.699, 24.996, 476.844
567183, 102.739, 24.9648, 478.021
565512, 102.727, 24.9786, 478.58
567157, 102.808, 25.0416, 479.011
565350, 102.764, 25.0599, 479.381
566999, 102.756, 25.0401, 479.729
567292, 102.746, 25.0952, 482.743
565722, 102.715, 25.0235, 483.859
567353, 102.681, 25.0252, 486.818
566321, 102.679, 25.0292, 486.818
565381, 102.678, 25.0475, 488.2
566720, 102.764, 25.0445, 488.83
567465, 102.754, 25.0117, 493.36
566982, 102.761, 25.022, 496.973
565835, 102.685, 24.9916, 496.993
565478, 102.787, 24.9867, 497.307
566987, 102.763, 25.0357, 497.635

567027, 102.787, 25.0217, 497.99
567173, 102.696, 25.0022, 498.304
567220, 102.793, 25.0012, 500.165
565397, 102.796, 24.9976, 500.165
565487, 102.758, 25.0091, 500.513
567546, 102.778, 25.0471, 504.256
566787, 102.785, 25.0903, 504.318
566943, 102.782, 25.019, 505.143
565509, 102.753, 24.9812, 505.151
568101, 102.767, 25.0947, 505.495
566779, 102.768, 25.0398, 505.964
567299, 102.758, 25.0699, 508.568
566947, 102.884, 25.0508, 510.055
565726, 102.754, 24.9711, 510.265
565424, 102.766, 24.9803, 512.331
566745, 102.769, 25.0436, 513.484
566013, 102.704, 24.9741, 514.622
565765, 102.734, 24.9724, 514.667
567248, 102.719, 24.9781, 516.5
567596, 102.792, 24.9891, 517.855
33545, 102.787, 24.9902, 517.855
33073, 102.786, 25.0921, 518.471
565676, 102.74, 24.9572, 522.289
566996, 102.872, 25.0522, 522.512
565561, 102.74, 24.9988, 523.162
568065, 102.865, 25.0281, 528.19
565537, 102.768, 24.9684, 531.208
567008, 102.772, 25.0316, 531.575
567574, 102.772, 25.0944, 531.705
566072, 102.709, 24.9734, 533.609
565520, 102.782, 25.0055, 534.208
567600, 102.867, 25.0475, 534.254
565584, 102.798, 24.9904, 534.74
565522, 102.795, 24.9944, 534.74
566747, 102.765, 25.0492, 537.206
567626, 102.784, 25.059, 542.472
566759, 102.761, 25.0955, 543.092
566726, 102.763, 25.091, 543.092
567601, 102.704, 24.9781, 545.587
567357, 102.709, 24.9762, 545.587
565664, 102.729, 24.9671, 548.812
33234, 102.759, 25.0173, 551.784
567580, 102.675, 24.996, 553.677
567336, 102.87, 25.0367, 563.028
565030, 102.868, 25.0319, 563.028
566066, 102.695, 24.9785, 564.104
566026, 102.699, 24.975, 564.104
566781, 102.782, 25.0423, 567.074
567011, 102.785, 25.0257, 574.198

567184, 102.739, 24.9623, 575.534
565665, 102.752, 24.9588, 576.806
565031, 102.868, 25.0428, 577.898
565764, 102.736, 24.9774, 596.246
566557, 102.772, 25.0977, 602.687
565596, 102.728, 24.9719, 607.193
567715, 102.804, 25.0393, 608.7
567075, 102.75, 24.9759, 608.809
566977, 102.883, 25.0562, 609.526
566795, 102.778, 25.0508, 612.739
566743, 102.772, 25.05, 612.739
566731, 102.767, 25.0544, 612.994
567506, 102.704, 24.9718, 613.174
567537, 102.798, 25.0403, 615.236
568121, 102.792, 25.0511, 615.57
568115, 102.796, 25.0553, 615.57
567619, 102.736, 24.9753, 618.512
566946, 102.889, 25.0515, 621.359
565585, 102.801, 24.9956, 621.4
567021, 102.881, 25.0459, 625.702
566049, 102.702, 25.01, 628.806
565466, 102.746, 24.9555, 634.768
565046, 102.78, 25.0902, 640.069
565433, 102.749, 24.9922, 644.516
565045, 102.783, 25.0958, 647.035
566983, 102.777, 25.0353, 649.729
566767, 102.755, 25.0933, 652.989
567194, 102.874, 25.0286, 653.841
566770, 102.768, 25.1024, 664.088
568117, 102.802, 25.0506, 675.894
566730, 102.8, 25.0448, 675.894
567893, 102.873, 25.0561, 687.707
568075, 102.875, 25.0347, 687.71
567157, 102.812, 25.0366, 689.232
566997, 102.775, 24.9705, 696.678
566784, 102.791, 25.0398, 708.076
567222, 102.791, 25.0397, 708.696
566038, 102.686, 24.9803, 722.988
565547, 102.791, 25.0026, 730.589
566022, 102.676, 24.9925, 737.023
568209, 102.879, 25.0244, 748.165
565028, 102.872, 25.0222, 748.165
566989, 102.782, 25.0137, 753.612
565037, 102.882, 25.0372, 758.891
566992, 102.86, 25.0459, 801.405
566991, 102.854, 25.0411, 801.405
567530, 102.787, 25.0631, 810.249
567489, 102.886, 25.0401, 811.37
566968, 102.887, 25.0688, 811.387

568072, 102.851, 25.0364, 819.788
33248, 102.77, 25.0783, 823.261
565029, 102.874, 25.0622, 827.782
566966, 102.82, 25.0346, 835.284
567718, 102.891, 25.0747, 842.528
566821, 102.895, 25.0681, 842.528
566960, 102.89, 25.0591, 860.861
565034, 102.866, 25.0606, 861.984
566762, 102.746, 25.1005, 877.553
567016, 102.79, 25.0193, 878.178
566018, 102.772, 25.0862, 896.523
565036, 102.859, 25.034, 896.784
565043, 102.805, 25.0516, 914.185
33096, 102.755, 25.0994, 914.484
566786, 102.795, 25.0895, 950.703
567699, 102.797, 25.0146, 1011.9
567741, 102.823, 25.03, 1033.89
566776, 102.794, 25.0856, 1050.75
564595, 102.836, 25.0958, 1096.02
567258, 102.805, 25.0861, 1110.46
33237, 102.895, 25.0502, 1111.2
566942, 102.843, 25.0377, 1173.66
568298, 102.831, 25.0994, 1183.44
568297, 102.828, 25.0892, 1183.44
566990, 102.833, 25.032, 1188.65
565033, 102.88, 25.0789, 1322.61
565026, 102.872, 25.0147, 1366.48
568068, 102.804, 25.0211, 1426.7
565041, 102.782, 25.0767, 1463.87
565044, 102.803, 25.0675, 1531.77
565047, 102.812, 25.065, 1649.27
564499, 102.867, 25.0928, 2029.69
565025, 102.895, 25.095, 2346.04
568313, 102.863, 25.0983, 2735.8

The Merge Sort Without Recurrence costs 0 secondes
Share the same results as above

The Quick Sort costs 0 secondes, The depth of the recurrence is 21
Share the same results as above

The Randomized Quick Sort costs 0 secondes, The depth of the recurrence is 21
Share the same results as above

The Randomized Select (2 parts) costs 0 secondes, The depth of the recurrence is 11

```
The station with 1st smallest k_dist is 568030, and its k_dist is 103.075

The Randomized Select (3 parts) costs 0.001 secondes, The depth of the recurrence is 8
The station with 1st smallest k_dist is 568030, and its k_dist is 103.075

The Randomized Select (2 parts) costs 0 secondes, The depth of the recurrence is 6
The station with 5th smallest k_dist is 567883, and its k_dist is 126.096

The Randomized Select (3 parts) costs 0 secondes, The depth of the recurrence is 12
The station with 5th smallest k_dist is 567883, and its k_dist is 126.096

The Randomized Select (2 parts) costs 0 secondes, The depth of the recurrence is 14
The station with 50th smallest k_dist is 568074, and its k_dist is 208.475

The Randomized Select (3 parts) costs 0 secondes, The depth of the recurrence is 6
The station with 50th smallest k_dist is 568074, and its k_dist is 208.475

The Randomized Select (2 parts) costs 0 secondes, The depth of the recurrence is 9
The station with largest k_dist is 568313, and its k_dist is 2735.8

The Randomized Select (3 parts) costs 0.001 secondes, The depth of the recurrence is 5
The station with largest k_dist is 568313, and its k_dist is 2735.8

The Closest Pair costs 0.002 secondes, The depth of the recurrence is 10
The pair station with minimun distance are 568849 and 568471, and its distance is 0

The Closest Pair 2 costs 0.003 secondes, The depth of the recurrence is 10
The pair station with minimun distance are 568471 and 568849, and its distance is 0
The pair station with 2nd minimun distance are 567389 and 566803, and its distance is 5.78896
```