**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра вычислительной техники.**

отчет

**по лабораторной работе №2**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: Алгоритмы «разделяй и властвуй».

|  |  |  |
| --- | --- | --- |
| Студент гр. 1302 |  | Фомин В. В. |
| Преподаватель |  |  |

Санкт-Петербург

2022

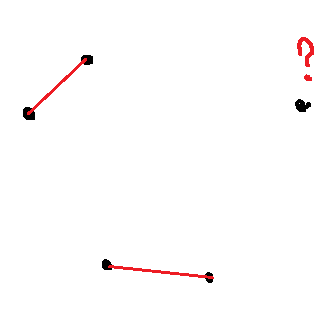
Постановка задачи.

На квадрате земли 1 км2 располагаются улитки-гермафродиты. В момент времени каждая из улиток с постоянной скоростью 1 cм/с ползет к улитке, являющейся ближайшей к ней в момент времени t = 0, выбрав её в качестве спутника жизни. Определить время, через которое первая пара улиток достигнет друг друга или наличие ситуации, приводящей улиток в замешательство, наиболее эффективным способом

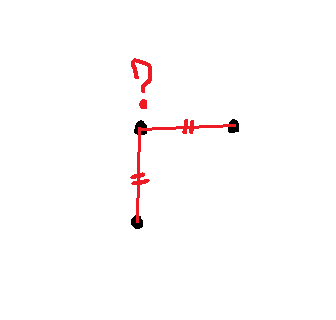


Ситуации замешательства:

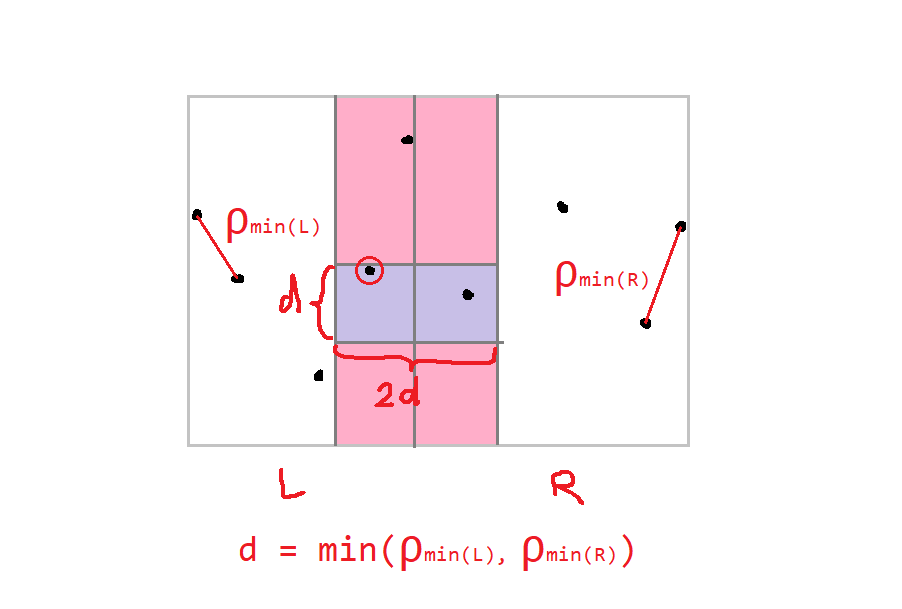
улиток нечетное число или 0.



минимальное расстояние встречается несколько раз.



В качестве основы для решения взят алгоритм Препарата-Шамос.



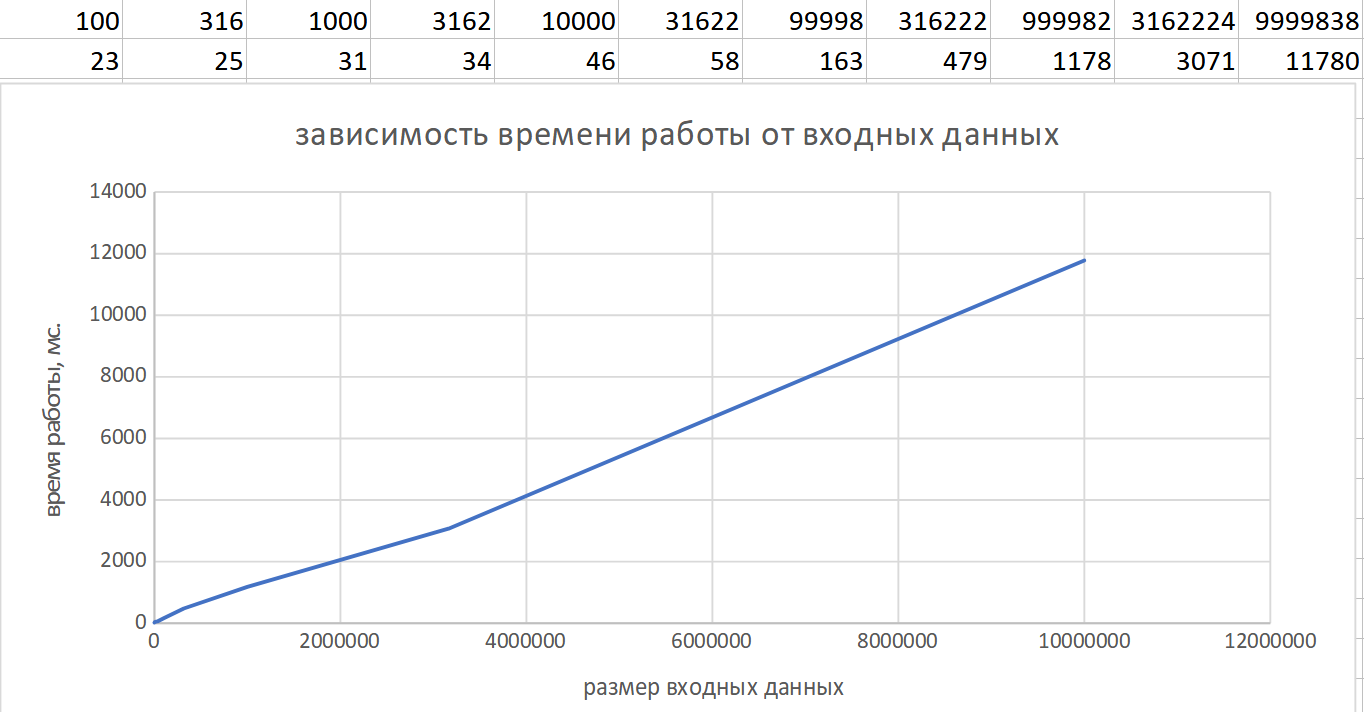
Описание классов и методов.

|  |  |  |
| --- | --- | --- |
| Метод | Временная сложность | Описание метода |
| Main.mergeSort | O(n\*log(n)) | сортировка массива слиянием |
| Main.binarySearch | O(log(n)) | быстрое нахождение индекса элемента в отсортированном массиве |
| Main.minDistance | O(n\*log(n)) | нахождение минимального расстояния среди набора точек и выявление факта замешательства |
| Point.distanceTo | O(1) | расстояние между двумя точками |

Теоретическая асимптотическая сложность.

T(n) = 2T(n/2) + O(n) = **O(n\*log(n))**

Эмпирическая оценка временной сложности



Эмпирически проверено, что программа работает за **T(n).**

Пример работы

**4**

**0 0**

**0 1**

**1 0**

**1 1**

**определить время невозможно.**

**1**

**1 1**

**определить время невозможно.**

**3**

**1 1**

**1 5**

**8 1**

**определить время невозможно.**

**6**

**0 0**

**-7 -3**

**3 4**

**2 10**

**6.34 2.91**

**15.5 -0.7**

**первая встреча будет через 1.7566801074754619 секунд между улитками из точек {x=3.0, y=4.0} и {x=6.34, y=2.91}.**

**4**

**0 0**

**2 1**

**5 0**

**10 0**

**первая встреча будет через 1.118033988749895 секунд между улитками из точек {x=0.0, y=0.0} и {x=2.0, y=1.0}.**

Листинг

Среда разработки: Intellij Idea. Язык: Java.

import java.util.Comparator;  
import java.util.Locale;  
import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*).useLocale(Locale.*US*);  
 Point[] points = new Point[sc.nextInt()];  
 Point[] temp = new Point[points.length];  
 for (int i = 0; i < points.length; i++) {  
 points[i] = new Point(sc.nextDouble(), sc.nextDouble());  
 }  
 if (points.length > 1 && points.length % 2 == 0) {  
 *mergeSort*(points, temp, 0, points.length - 1, new XComp());  
 *minDistance*(points, 0, points.length - 1);  
 } else {  
 *isExist* = false;  
 }  
 System.*out*.println(*isExist* ? "первая встреча будет через " + *min* / 2 + " секунд между улитками из точек "  
 + *p1*.toString() + " и " + *p2*.toString() + "." : "определить время невозможно.");  
 }  
  
 public static double *min* = Double.*MAX\_VALUE*;  
 public static boolean *isExist* = true;  
  
 public static Point *p1*; //first point in min pair  
 public static Point *p2*; //second point in min pair  
  
 public static void minDistance(Point[] points, int left, int right) {  
 if (right - left < 1) {  
 return;  
 } else if (right - left == 1) {  
 double x = points[left].distanceTo(points[right]);  
 if (Math.*abs*(x - *min*) < 0.0000001) {  
 *isExist* = false;  
 } else if (x < *min*) {  
 *min* = x;  
 *isExist* = true;  
 *p1* = points[left];  
 *p2* = points[right];  
 }  
 } else if (right - left == 2) {  
 double x1 = points[left].distanceTo(points[right]);  
 if (Math.*abs*(x1 - *min*) < 0.0000001) {  
 *isExist* = false;  
 } else if (x1 < *min*) {  
 *min* = x1;  
 *isExist* = true;  
 *p1* = points[left];  
 *p2* = points[right];  
 }  
 double x2 = points[left + 1].distanceTo(points[right]);  
 if (Math.*abs*(x2 - *min*) < 0.0000001) {  
 *isExist* = false;  
 } else if (x2 < *min*) {  
 *min* = x2;  
 *isExist* = true;  
 *p1* = points[left + 1];  
 *p2* = points[right];  
 }  
 double x3 = points[left].distanceTo(points[left + 1]);  
 if (Math.*abs*(x3 - *min*) < 0.0000001) {  
 *isExist* = false;  
 } else if (x3 < *min*) {  
 *min* = x3;  
 *isExist* = true;  
 *p1* = points[left];  
 *p2* = points[left + 1];  
 }  
 } else {  
 *minDistance*(points, left, (left + right) / 2);  
 *minDistance*(points, (left + right) / 2 + 1, right);  
 Point m = points[(left + right) / 2]; //middle point on segment  
 Point[] subArray = new Point[right - left + 1];  
 Point[] temp = new Point[subArray.length];  
 int realSize = 0;  
 for (int i = left; i <= right; i++) {  
 if (Math.*abs*(m.x - points[i].x) < *min*) {  
 subArray[realSize++] = points[i];  
 }  
 }  
 *mergeSort*(subArray, temp, 0, realSize - 1, new YComp());  
 for (int i = 0; i < realSize; i++) {  
 int start = *binarySearch*(subArray, 0, realSize, new Point(subArray[i].x, subArray[i].y - *min*), new YComp());  
 if (start < 0) {  
 start++;  
 start = -start;  
 }  
 for (int j = start; j < i; j++) {  
 double x = subArray[i].distanceTo(subArray[j]);  
 if (Math.*abs*(x - *min*) < 0.0000001 && !((*p1* == subArray[i] || *p2* == subArray[i]) && (*p1* == subArray[j] || *p2* == subArray[j]))) {  
 *isExist* = false;  
 } else if (x < *min*) {  
 *min* = x;  
 *isExist* = true;  
 *p1* = subArray[i];  
 *p2* = subArray[j];  
 }  
 }  
 }  
 }  
 }  
  
 public static <T> void mergeSort(T[] array, T[] temp, int left, int right, Comparator<T> cmp) {  
 if (right - left == 0) {  
 return;  
 } else {  
 *mergeSort*(array, temp, left, (left + right) / 2, cmp);  
 *mergeSort*(array, temp, (left + right) / 2 + 1, right, cmp);  
 int i = left, j = (left + right) / 2 + 1;  
 for (int k = 0; k < right - left + 1; k++) {  
 if (i == (left + right) / 2 + 1) {  
 temp[k] = array[j++];  
 } else if (j == right + 1) {  
 temp[k] = array[i++];  
 } else if (cmp.compare(array[i], array[j]) < 1) {  
 temp[k] = array[i++];  
 } else {  
 temp[k] = array[j++];  
 }  
 }  
 for (int k = 0; k < right - left + 1; k++) {  
 array[left + k] = temp[k];  
 }  
 }  
 }  
  
 public static <T> int binarySearch(T[] a, int left, int right,  
 T key, Comparator<T> cmp) {  
 if (cmp == null) {  
 throw new NullPointerException();  
 }  
  
 int low = left;  
 int high = right - 1;  
  
 while (low <= high) {  
 int mid = (low + high) >>> 1;  
 T midVal = a[mid];  
 int c = cmp.compare(midVal, key);  
 if (c < 0)  
 low = mid + 1;  
 else if (c > 0)  
 high = mid - 1;  
 else  
 return mid; // key found  
 }  
 return -(low + 1); // key not found.  
 }  
}

public class Point {  
 final double x;  
 final double y;  
  
 public Point(double x, double y) {  
 this.x = x;  
 this.y = y;  
 }  
  
 public double distanceTo(Point p) {  
 return Math.*sqrt*(Math.*abs*(x - p.x) \* Math.*abs*(x - p.x) + Math.*abs*(y - p.y) \* Math.*abs*(y - p.y));  
 }  
  
 @Override  
 public String toString() {  
 return "{" +  
 "x=" + x +  
 ", y=" + y +  
 '}';  
 }  
}  
  
class XComp implements Comparator<Point> {  
  
 @Override  
 public int compare(Point o1, Point o2) {  
 return Double.*compare*(o1.x, o2.x);  
 }  
}  
  
class YComp implements Comparator<Point> {  
  
 @Override  
 public int compare(Point o1, Point o2) {  
 return Double.*compare*(o1.y, o2.y);  
 }  
}