CSE 551 - Foundations of Algorithms

Assignment 4

Submitted by: Sannidhya Pathania (spathan3)

Solution 1:

1.1

Let us assume that the given numbers are n-bit. Now we need to reduce the complexity of multiplying such large numbers.

In the question, it's given we have split the numbers into the format, a+b*i, and c+d*i, and our multiplication is s = (a+b*i)*(c+d*i), where i is some random variable (i can be $2^{n/2}$ or i can also be taken as √-1 i.e complex number presentation).

The above multiplication can be transformed as:

$s = (a+b*i)*(c+d*i) = ac + ad * i + bc * i + bd * i^2$

$= ac + ((a+b) * (c+d) - ac - bd) + bd * i^2$

Here we can see we have 3 multiplications of n/2 bits instead of n bits. Thus, transformed the basic expression with 4 n/2 bits multiplication to 3 n/2 bits multiplications.

- ac
- bd
- (a+b) (c+d)

1.2

The final expression is ***ac + ((a+b) * (c+d) - ac - bd) + bd * i^2***

Solution 2:

2.1

Given, If |i – j| >= x, where x = 12, then the distance between si and sj is at least **δ**.

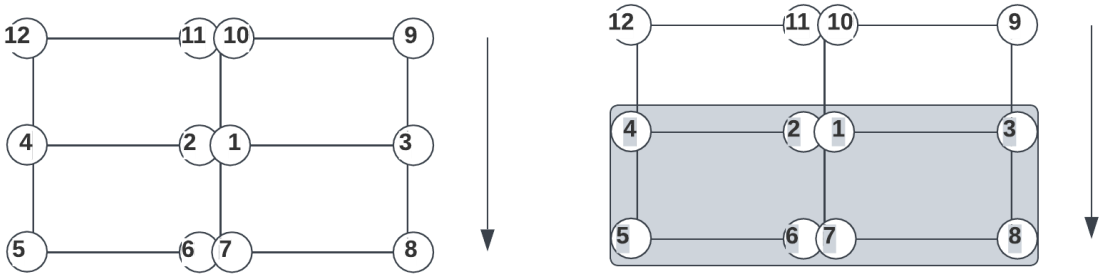
Here, **x can be reduced to 7** i.e. we can say we need to check **at most 7** neighboring points to find a distance smaller than **δ**.

This can be reduced to 7 neighbors by considering the 2**δ** * **δ** box as shown in the figure below as grey area.

Let us consider 1 as the current point.

Each point must lie either on the left square or right square of point 1. Both squares have a dimension of **δ** * **δ**. We just need to check the points that are ahead of point 1, because the points behind are already checked in previous iterations.

So, we can skip checking the 4 points that are behind point 1. And we are left with 8 points only. Out of which one is the point itself. Thus, I need to check 7 neighbors.



2.2

IDE used = Visual Studio Code

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Scanner;

public class ClosestPair {

   public static void main(String args[]){
       System.out.println("Closest Pair of Points Algorithm");
       System.out.println("--------------------------------");
       Scanner sc = new Scanner(System.in);

       System.out.println("Enter the number of points");
       int n = sc.nextInt();
       int[][] points = new int[n][2];

       System.out.println("Enter x and y coordinate points");
       for(int i=0; i<n; i++){
           points[i][0] = sc.nextInt();
           points[i][1] = sc.nextInt();
       }
       Arrays.sort(points, (a,b)->a[0]-b[0]);
```

```
        //calling the algorithm
        double dist = closestPair(points, 0, n-1);
        sc.close();
        System.out.println("Distance between closest pair of points is " +
String.format("%.2f", dist));
    }

    public static double closestPair(int[][] points, int si, int ei){
        if(ei - si <= 0)
            return Double.MAX_VALUE;

        if(ei-si == 1)
            return calEucDist(points[si], points[ei]);

        int mid = si + (ei - si)/2;

        double diff1 = closestPair(points, si, mid);
        double diff2 = closestPair(points, mid, ei);

        double diff = Math.min(diff1, diff2);

        //making a set of points with distance diff from separation line
        ArrayList<int[]> setOfY = new ArrayList<>();
        for(int i=si; i<=ei; i++){
            if(Math.abs(points[i][0] - points[mid][0]) <= diff)
                setOfY.add(points[i]);
        }

        //Sorting setOfY
        Collections.sort(setOfY, (a,b)->(a[1]-b[1]));

        //Scanning points comparing the distance between each point and next x-1
neighbors
        for(int i=0; i<setOfY.size(); i++){
            for(int j = 1; j<=7 && i+j < setOfY.size() ; j++){
                double dist = calEucDist(setOfY.get(i), setOfY.get(i+j));
                diff = Math.min(diff, dist);
            }
        }
        return diff;
    }

    public static double calEucDist(int[] p1, int[] p2){
```

```
        int distX = Math.abs(p1[0]-p2[0]);
        int distY = Math.abs(p1[1]-p2[1]);

        double dist = Math.sqrt(distX * distX + distY * distY);
        return dist;
    }
}
```

2.3

Input for Code:

Let the 16 points:

1. (1, 13)
2. (3, 14)
3. (4, 11)
4. (5, 5)
5. (2, 6)
6. (7, 3)
7. (8, 8)
8. (9, 2)
9. (15, 9)
10. (11, 12)
11. (10, 15)
12. (14, 16)
13. (13, 1)
14. (6, 4)
15. (16, 7)
16. (12, 10)

Output from Code:

The distance between the closest pair of points is **1.41**

Screenshot of output:

```
spats@Sannidhyas-MacBook-Air Code %  /usr/bin/env /Library/Jav
review -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/spat
824c68264e09/redhat.java/jdt_ws/Code_3d556bb7/bin ClosestPair
Closest Pair of Points Algorithm
--------------------------------
Enter the number of points
16
Enter x and y coordinate points
1 13
3 14
4 11
5 5
2 6
7 3
8 8
9 2
15 9
11 12
10 15
14 16
13 1
6 4
16 7
12 10
Distance between closest pair of points is 1.41
spats@Sannidhyas-MacBook-Air Code %
```