Name: Mashiyat Mahjabin Prapty CSE 204

ID: 1805117 Offline-8

Second Closest Pair of Points using Divide and Conquer Method:

```
pair_points baseCase(Point p[], int n)
{
    float min_dist = std::numeric_limits<float>::max();
    pair_points points;
//This loop runs highest 9 times, so complexity is O(9)
    for(int i = 0; i < n; i++)
    {
        for(int j = i+1; j < n; j++)
        {
            if(distance(p[i], p[j]) < min_dist)
            {
                min_dist = distance(p[i], p[j]);
                points.p1 = p[i];
                points.p2 = p[j];
            }
        }
    }
    return points;
}

pair_points stripClosest(Point strip[], int n, pair_points d)
{
    float m = distance(d.p1, d.p2);
    pair_points q = d;

//This loop runs highest 7n times, so complexity of this nested loop is O(n)
    for(int i = 0; i < n; i++)
    {
        for(int j = i+1; j < n && (strip[j].getY()-strip[i].getY()) < m; j++)
        {
            if(distance(strip[i], strip[j]) < m)
            {
                m = distance(strip[i], strip[j]);
                q.p1 = strip[i];
                q.p2 = strip[j];
            }
        }
    }
    return q;
}
```

```
pair_points closest(Point px[], Point py[], int n)
{
    if(n <= 3)
    {
        return baseCase(px, n); //Complexity: O(9)
    }

    int mid = n/2;
    Point mid_point = px[mid];

    Point py_left[mid];
    Point py_right[n-mid];
    int l = 0, r = 0;

//Complexity: O(n)
    for(int i = 0; i < n; i++)
    {
        if(py[i].getX() <= mid_point.getX() && l < mid) py_left[l++] = py[i];
        else py_right[r++] = py[i];
    }

//Recursive calls are done by dividing the array into half each time, so
complexity of this method: O(nlgn)
    pair_points left = closest(px, py_left, mid);
    pair_points right = closest(px+mid, py_right, n-mid);

    pair_points d = minimum(left, right);

    Point strip[n];
    int j = 0;

//Complexity: O(n)
    for(int k = 0; k < n; k++)
    {
        if(abs(py[k].getX()-mid_point.getX()) < distance(d.p1, d.p2))
            strip[j++] = py[k];
    }

    pair_points strip_min = stripClosest(strip, j, d); //Complexity: O(n)

    return minimum(d, strip_min);
}

pair_points closestPoint(Point p[], int n)
```

```
{
    Point px[n], py[n];
//Complexity: O(n)
    for(int i = 0; i < n; i++)
    {
        px[i] = p[i];
        py[i] = p[i];
    }
    merge_sort(px, 0, n-1, 'X'); //Complexity: O(nlgn)
    merge_sort(py, 0, n-1, 'Y'); //Complexity: O(nlgn)

    return closest(px, py, n); //Complexity: O(nlgn)
}

int main()
{
    string myText;
    ifstream MyReadFile("input.txt");
    int n, tempX, tempY;

    MyReadFile >> n;
    Point array[n];
//Complexity: θ(n)
    for (int i = 0; i < n; i++)
    {
        MyReadFile >> tempX >> tempY;
        array[i].setVal(i, tempX, tempY);
    }
    MyReadFile.close();

    pair_points closest = closestPoint(array, n); //Complexity: O(nlgn)

//Complexity: O(n)
    for (int i = 0; i < n; i++)
    {
        Point temp;
        if (array[i].getX() == closest.p1.getX() && array[i].getY() == closest.p1
.getY())
        {
            temp = array[i];
            array[i] = array[n-1];
            array[n-1] = temp;
            break;
        }
    }
    pair_points aa = closestPoint(array, n-1); //Complexity: O(nlgn)

//Complexity: O(n)
    for (int i = 0; i < n; i++)
    {
        Point temp;
        if (array[i].getX() == closest.p2.getX() && array[i].getY() == closest.p2
.getY())
        {
            temp = array[i];
            array[i] = array[n-1];
            array[n-1] = temp;
            break;
        }
    }
    pair_points bb = closestPoint(array, n-1); //Complexity:O(nlgn)

    pair_points answer = minimum(aa, bb);
    cout << min(answer.p1.getIndex(), answer.p2.getIndex()) << " " << max(answer.
p1.getIndex(), answer.p2.getIndex()) << endl;
    cout << distance(answer.p1, answer.p2) << endl;
}
```

Total Complexity = $\theta(n)+2*O(n)+3*(3*O(n) + 3*O(nlgn)) = \theta(n) + 11*O(n) + 9*O(nlgn)$

$T(n) = O(nlgn)$

So, the complexity of the designed algorithm is O(nlgn).