# *COMPLEXITY ANALYSIS OF THE ALGORITHM ON FINDING SECOND CLOSEST POINT*

***Name: Sanjida Islam Era***

***Student ID: 1805116***

***Section : CSE-B2***

***Date: 18.06.2021***

## ***Divide and Conquer:***

In divide and conquer approach, the problem in hand, is divided into smaller sub-problems and then each problem is solved independently. When we keep on dividing the subproblems into even smaller sub-problems, we may eventually reach a stage where no more division is possible. Those "atomic" smallest possible sub-problem (fractions) are solved. The solution of all sub-problems is finally merged in order to obtain the solution of an original problem.

## ***Description Of the Problem:***

Given are some indices of houses, that can be considered as some points in a plane. There are n houses in Manhattan. Each house $h_i$ has a two- dimensional position ($x_i$ , $y_i$). Then we have to calculate the second closest distance from the given houses, showing the indices of the second nearest houses of Manhattan. Also time complexity should be maintained O(nlogn) here.

## ***Algorithm used in this problem:***

1. We sort all points according to x coordinates.

2. Then we divide all points in two halves.

3. Then we recursively find the smallest and second smallest distances in both subarrays. The both subarrays have two double elements.

4. We now have to take the minimum two of above smallest distances. Let the minimum be mins[2].

5. Now we have to create an array strip[] that stores all points which are at most second smallest distance away from the middle line dividing the two sets.

6. Then we have to find the smallest and second smallest distance in strip[].
7.Lastly we have to return the minimum of previous two smallest distances and the two smallest distances calculated in above step 6.

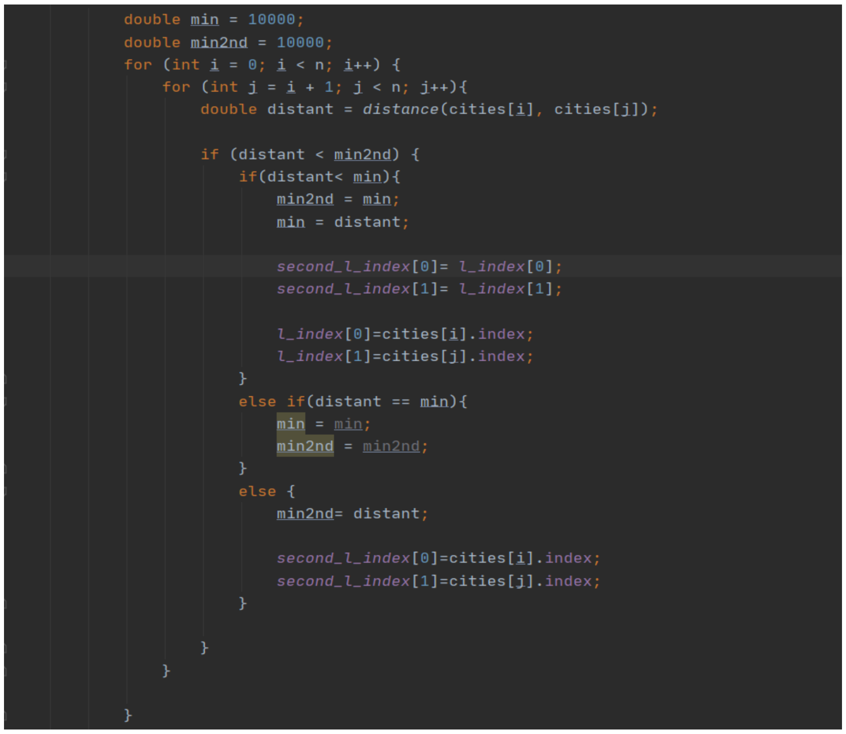
## ***Complexity Analysis:***

- Function used to perform the task is:

```
static double secondClosest(House H[], int n)
```

Here, in this function, the array of objects Hx was presorted using merge-sort according to an increasing order of x, which has a time complexity of O(nlogn). Another important thing to note here is that an array of objects Hy was also presorted in the same manner but according to the increasing order of y. Here, (x,y) represents the Cartesian coordinate of a house.

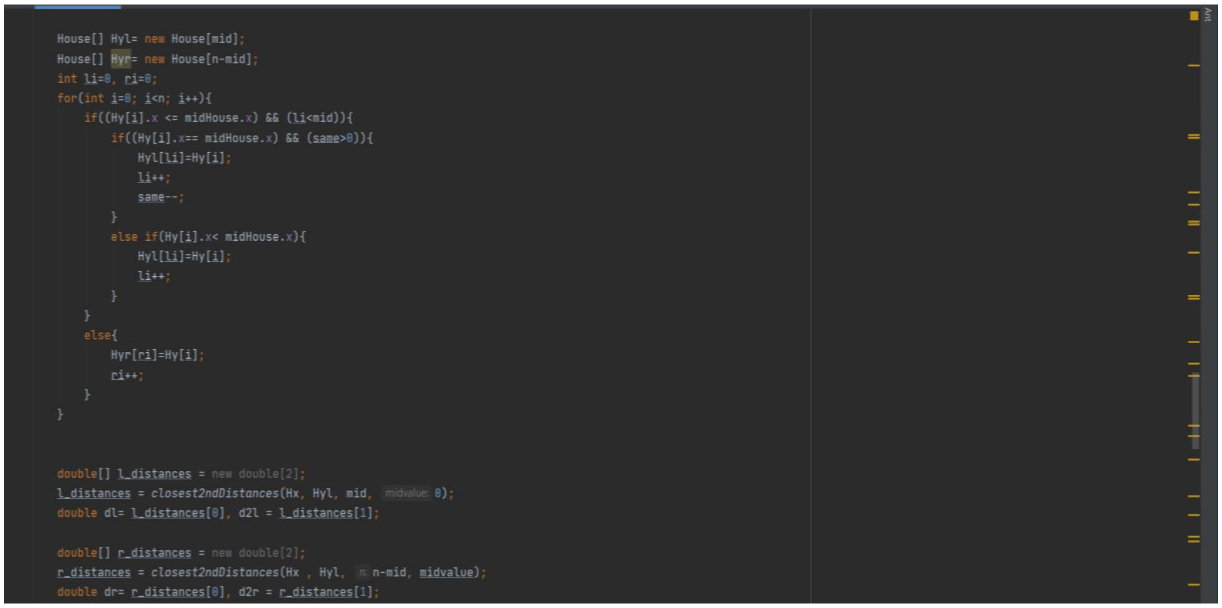
- As divide and conquer approach was used for this problem, recursion was used to solve it. The base case for this is:



```
double min = 10000;
double min2nd = 10000;
for (int i = 0; i < n; i++) {
    for (int j = i + 1; j < n; j++){
        double distant = distance(cities[i], cities[j]);

        if (distant < min2nd) {
            if(distant< min){
                min2nd = min;
                min = distant;

                second_l_index[0]= l_index[0];
                second_l_index[1]= l_index[1];

                l_index[0]=cities[i].index;
                l_index[1]=cities[j].index;
            }
            else if(distant == min){
                min = min;
                min2nd = min2nd;
            }
            else {
                min2nd= distant;

                second_l_index[0]=cities[i].index;
                second_l_index[1]=cities[j].index;
            }

        }
    }

}
```

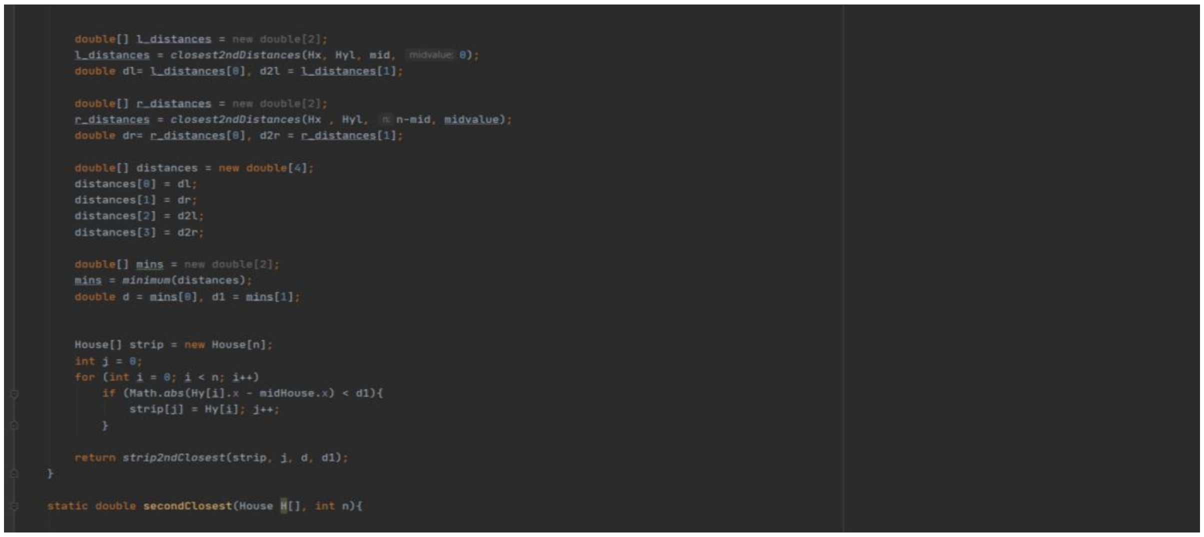
Here, for number of elements less than or equal to 3 will return an array of length 2 which will contain the closest and 2nd Closest distances 2 indexes, and the information about the closest the second closest elements are in l_index and second_l_index arrays.

- The divide & conquer section was approached in this manner:



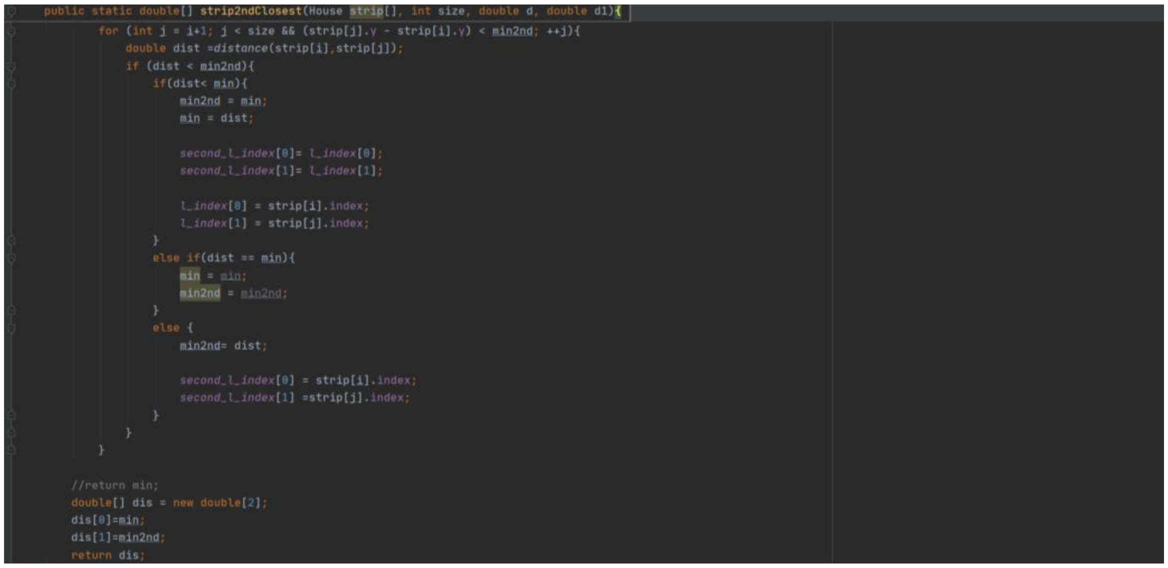
```java
House[] Hyl= new House[mid];
House[] Hyr= new House[n-mid];
int li=0, ri=0;
for(int i=0; i<n; i++){
    if((Hy[i].x <= midHouse.x) && (li<mid)){
        if((Hy[i].x== midHouse.x) && (same>0)){
            Hyl[li]=Hy[i];
            li++;
            same--;
        }
        else if(Hy[i].x< midHouse.x){
            Hyl[li]=Hy[i];
            li++;
        }
    }
    else{
        Hyr[ri]=Hy[i];
        ri++;
    }
}


double[] l_distances = new double[2];
l_distances = closest2ndDistances(Hx, Hyl, mid, midvalue: 0);
double dl= l_distances[0], d2l = l_distances[1];

double[] r_distances = new double[2];
r_distances = closest2ndDistances(Hx , Hyl, n: n-mid, midvalue);
double dr= r_distances[0], d2r = r_distances[1];
```

Here the Hy array was divided into two equal portions and then sent as argument in such a way that the sorted order was maintained. It takes O(n) time to divide the y sorted array around the mid vertical line. The smallest and second smallest distances is found recursively t in both subarrays. The both subarrays have two double elements.



```java
        double[] l_distances = new double[2];
        l_distances = closest2ndDistances(Hx, Hyl, mid, midvalue: 0);
        double dl= l_distances[0], d2l = l_distances[1];

        double[] r_distances = new double[2];
        r_distances = closest2ndDistances(Hx , Hyl, n: n-mid, midvalue);
        double dr= r_distances[0], d2r = r_distances[1];

        double[] distances = new double[4];
        distances[0] = dl;
        distances[1] = dr;
        distances[2] = d2l;
        distances[3] = d2r;

        double[] mins = new double[2];
        mins = minimum(distances);
        double d = mins[0], d1 = mins[1];


        House[] strip = new House[n];
        int j = 0;
        for (int i = 0; i < n; i++)
            if (Math.abs(Hy[i].x - midHouse.x) < d1){
                strip[j] = Hy[i]; j++;
            }

        return strip2ndClosest(strip, j, d, d1);
    }

    static double secondClosest(House H[], int n){
```

Here strip[] stores all points which are at most second smallest distance away from the middle line dividing the two sets.



```java
    public static double[] strip2ndClosest(House strip[], int size, double d, double d1){
            for (int j = i+1; j < size && (strip[j].y - strip[i].y) < min2nd; ++j){
                double dist =distance(strip[i],strip[j]);
                if (dist < min2nd){
                    if(dist< min){
                        min2nd = min;
                        min = dist;

                        second_l_index[0]= l_index[0];
                        second_l_index[1]= l_index[1];

                        l_index[0] = strip[i].index;
                        l_index[1] = strip[j].index;
                    }
                    else if(dist == min){
                        min = min;
                        min2nd = min2nd;
                    }
                    else {
                        min2nd= dist;

                        second_l_index[0] = strip[i].index;
                        second_l_index[1] =strip[j].index;
                    }
                }
            }

        //return min;
        double[] dis = new double[2];
        dis[0]=min;
        dis[1]=min2nd;
        return dis;
```

The smallest and second smallest distance in strip[] was found.

There are two for loops. The primary for loop takes at most n iterations, and thus works in O(n) time. The nested for loop is minimized to run at constant time which is less than n.Thus, the total nested for loop setup takes O(n) time to run.

So T(n) can be expressed as follows:

T’’(n) = 2T(n/2) + O(n) + O(n) +
O(n) T’’(n) = 2T(n/2) + O(n)

Now if we use Master’s theorem, we get a=2, b=2, f(n)= O(n)

$n \log_b a = n\log_2 2 = n1 = n$

T’’(n) = O(nLogn)

So total time complexity : T(n) = T’(n) + T’’(n) = O(nlogn) + O(nlogn) = O(nlogn).