Computer Science and Engineering Discipline
Khulna University, Khulna - 9208
Roll#:190208
Name:Tanvir Hassan
email:tanvir1908@cseku.ac.bd
Mobile:01989567104

PROBLEM : The problem is to find the closest pair of points in a set of points in x-y plane. The problem can be solved in O(n^2) time by calculating distances of every pair of points and comparing the distances to find the minimum. The Divide and Conquer algorithm solves the problem in O(nLogn) time.
solve the above problem using divide and conquer method.

1) Describe the method of solution using an example.
2) Write pseudocode for the method/algorithm.
3) Analysis the method and show its complexity in term of O (big-oh).
4) Write a program and execute it. Show the input and out of the program.

Solution :

CLOSEST PAIR OF POINTS

If given n points that are in 2D plane . We have to find the closest path of the array.
We know the distance between two points in the Euclidean distance given by the formula,

$$\|pq\| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

For points p(x_x,y_y) and q(x_x,y_y)

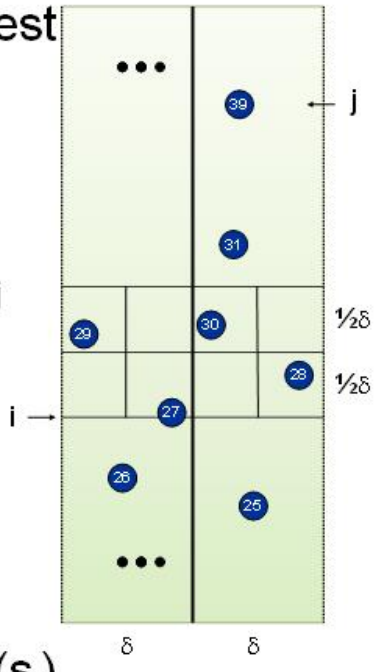
Def. Let s_i have the i^{th} smallest y-coordinate among points in the 2δ-width-strip.

Claim. If $|i - j| > 8$, then the distance between s_i and s_j is $> \delta$.

Pf: No two points lie in the same $\frac{1}{2}\delta$-by-$\frac{1}{2}\delta$ box:

$$\sqrt{\left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^2} = \sqrt{\frac{1}{2}} = \frac{\sqrt{2}}{2} \approx 0.7 < 1$$

so ≤ 8 boxes within $+\delta$ of $y(s_i)$.



PSUDOCODE:

```
Closest-Pair(p1, …, pn) {
if(n <= ??) return ??
```

Compute separation line L such that half the points are on one side and half on the other side.

```
$1 = Closest-Pair(left half)
$2 = Closest-Pair(right half)
$ = min($1, $2)
```

Delete all points further than $ from separation line L

Sort remaining points p[1]…p[m] by y-coordinate.

```
for i = 1..m
k = 1
```

```
 while i+k <= m && p[i+k].y < p[i].y + $
     $ = min($,distance between p[i] and p[i+k]);
     k++;
return $
}
```

ANALYSIS : Let D(n) be the number of pairwise distance calculations in the Closest-Pair Algorithm when run on n >= 1 points

$$D(n) \le \begin{cases} 0 & n = 1 \\ 2D(n/2) + 7n & n > 1 \end{cases} \Rightarrow D(n) = O(n \log n)$$

BUT – that's only the number of distance calculations

What if we counted comparisons?

Let C(n) be the number of comparisons between coordinates/distances in the Closest-Pair Algorithm when run on n >= 1 points

$$C(n) \le \begin{cases} 0 & n = 1 \\ 2C(n/2) + O(n \log n) & n > 1 \end{cases} \Rightarrow C(n) = O(n \log^2 n)$$

Now it the question that how can we achieve O(nlogn)

Sort by x at top level only.

Each recursive call returns $ and list of all points sorted by y

Sort by merging two pre-sorted lists.

$$T(n) \leq 2T(n/2) + O(n) \quad \Rightarrow \quad T(n) = O(n \log n)$$

PROGRAM :

```
#include<stdio.h>
#include<math.h>
float min;
int i,j,n,k=0;

struct point{
        int x;
        int y;
};

float minimum(float [], int , int );

int main()
{
   printf("Enter number of points : \n");
   scanf("%d",&n);
   struct point points[n];
        printf("Enter points : \n");
        for(i=0;i<=n-1;i++)
   {
      scanf("%d",&points[i].x);
      scanf("%d",&points[i].y);
   }
   ;
   float D[n],Dx,Dy;
   for(i=0;i<n-1;i++)
   {
      for(j=i+1;j<n;j++)
      {
         Dx = points[i].x - points[j].x ;
         Dy = points[i].y - points[j].y ;
         D[k] = sqrt(pow(Dx,2) + pow(Dy,2));
```

```
            k++;
        }
    }
        float min = minimum(D, 0, n - 1);
        printf("Minimum distance :  %f",min);
        return 0;
}

float minimum(float A[], int L, int H)
{
        float  min, minL, minR;
        int mid;

        if (L == H)
        {
                min = A[L];
                return min;
        }

        if (H == L + 1)
        {
                if (A[L] > A[H])
                {
                        min = A[H];
                }
                else
                {
                        min = A[L];
                }
                return min;
        }

        mid = (L + H) / 2;
        minL = minimum(A, L, mid);
        minR = minimum(A, mid + 1, H);

        if (minL < minR)
                min = minL;
        else
                min = minR;
        return min;
}
```

Input Output :

Input =

```
Enter number of points :
6
Enter points :
2
3
12
30
40
50
5
1
12
10
3
4
```

Output =

```
Minimum distance :  1.414214
```