Pseudo-code:

```
int main(int argc,char **argv){


    read points from given points
    merge sort points by x direction

    float dim=closet_pair(p,size);

    print result to file
  return 0;
}
```

```
float cloest_cross_pair(point* left,int n1,point* right,int n2,float dmin){
    float result=dmin;
    for(int i=0;i<n1;i++){
        for(int j=0;j<n2;j++){
            float d=get_distance(right[i],left[j]);
            if(d<=result){
                update shortest distance and add points into pair_arr.
            }
        }
    }
    return result;
}
```

```
float closet_pair(point *p,int size){
    if(size<=3){
        float d1,d2,d3,dmin1;
        d1=get_distance(p[0],p[1]);
        if(size==2){
            find the distance of two points, then return their distance
        }
        else{
            d1=get_distance(p[0],p[1]);
            d2=get_distance(p[1],p[2]);
            d3=get_distance(p[0],p[2]);
            dmin1=min(d1,d2);
            dmin1=min(dmin1,d3);
            if(dmin1==d1){
                store first and second point
            }
            if(dmin1==d2){
                store the second and third point
            }
            if(dmin1==d3){
                store the first and third point
            }
            return dmin1;
        }
    }
    else
    {
        middle=(n/2)
        n1=medin,n2=size-medin;

        float dL=closet_pair(left,n1);
        float dR=closet_pair(right,n2);
        float dmin2=min(dL,dR);
        dmin2=cloest_cross_pair(left,n1,right,n2,dmin2);
        return dmin2;
    }
}
```

Analysis:

closet_pair

    If(size<=3)

        Find shortest distance of base case points (2 or 3 points).

    Else

        m = (n/2)

        n1=m, n2=size-m

        dl = closet_pair (left, n1)

        dr = closet_pair (right, n2)

        dmin2 = min (dl, dr)

        dmin2 = cloest_cross_pair(left, n1, right, n2, dmin2)

return dmin2

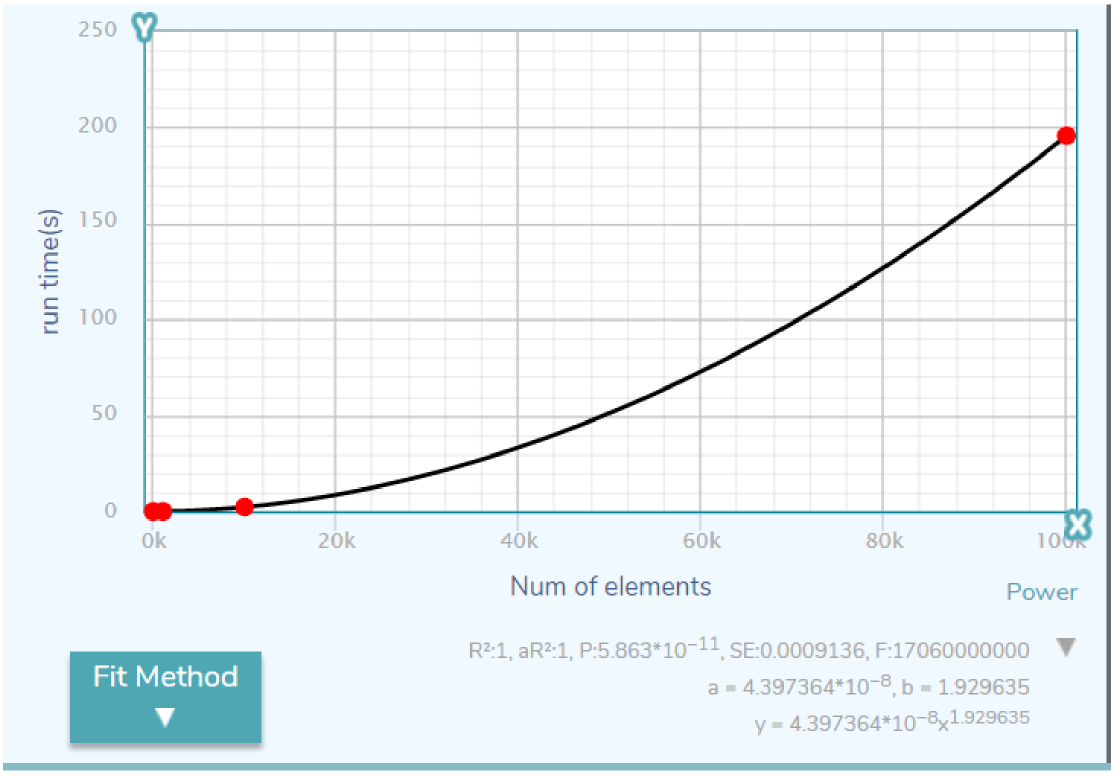
```
cloest_cross_pair
    for i in n1
        for j in n2
            get distance of right[ i ] and left[ j ]
            compare to the old distance
                if smaller
                    upgrade pair
                end if
        end for

    end for
```

The divide and conquer algorithm was deployed at function closet_pair, whose base case only have sub array contained 2 or 3 points to compare their distance. After this in sub array comparation the left array and right array is going to get a cross compare which implemented at function cloest_cross_pair. It has a for loop nested in another for loop, so the time complicity would be O(n^2).

For each base case it has inner sub-array comparation which has O(1) time complexity and cross pair comparation with O(n^2), totally it has O(n^2) for each base case. For the whole recursive it divide array by two parts and call recursive twice, so T(n) = 2 T(n/2) + cn^2. Applying the master theory we get a = 2, b = 2, d = 2. a/(b^d) = 0.5 > 1, so T(n) = Θ (n^d) = Θ (n^2). So the overall time complexity would be T(n) = Θ (n^2).

	10^2	10^3	10^4	10^5
1	363 (Micro)	29926(Micro)	3 seconds	194seconds
2	305	23228	2	195
3	331	37339	2	196
4	444	36992	2	199
5	209	37666	2	196
6	210	40010	3	201
7	267	19360	3	197
8	222	19555	2	193

9	282	19678	2	193
10	406	19619	2	192
AVG	0.0003039s	0.0283373s	2.3s	195.6s



The curve fit on the plot which made by average points shown that time complexity is O(n^2)