Part 1:

For naïve algorithm:

```
public static void findClosestPair(XYPoint points[], boolean print)
    {
          int nPoints = points.length;
          double minNaive=INF;
          double d;
          XYPoint po1 = null,po2=null;
          for(int i=0;i<nPoints-1;i++)
            for(int j=i+1;j<nPoints;j++){
                  d=points[i].dist(points[j]);
                  if(d<minNaive){
                        minNaive=d;
                        po1=points[i];
                        po2=points[j];
                  }
            }

          if (print)
          System.out.println("NAIVE " +
po1.toString()+po2.toString()+(double)Math.round(100000*minNaive)/1000
00);
    }
```

Explain:

Points[] is an array of XYPoint class, it’s like[((x,y),item),……]

I use 2 for loop to compare every two points’ distance with the class function:x[0].disc(x[1]):

When i=0,we calculate x[0].disc(x[1]),x[0].disc(x[2])………..

When i=1,we calculate x[1].disc(x[2]),x[1].disc(x[3])………..

Every time I calculate a new distance(d) and it’s smaller than the current minNaive, I update minNaive’s value to be d, and update po1 and po2 to record the two points’ (x,y) value

For divide-and-conquer algorithm:

```
public class ClosestPairDC {

    public static XYPoint x1,x2;
    public final static double INF =
java.lang.Double.POSITIVE_INFINITY;
    public static double temp = INF,minDis=INF;

    public static double findClosestPair(XYPoint pointsByX[],
                                XYPoint pointsByY[],
                                boolean print)
        {
         int n = pointsByX.length,mid;
         double disLR,disL,disR;
         if(n==1)
              temp=INF;
         else
              if(n==2){
                   temp=pointsByX[0].dist(pointsByX[1]);
              if  (temp<minDis){
                        minDis=temp;
                        x1=pointsByX[0];
                        x2=pointsByX[1];
                      }}
                                                   //base is 2 or 1
              if(n>2){
                   mid=n/2;
                   XYPoint XL [] = Arrays.copyOfRange(pointsByX, 0,mid);
                   XYPoint XR [] = Arrays.copyOfRange(pointsByX, mid,n);
                    XYPoint YL [] = Arrays.copyOfRange(pointsByY, 0,mid);
                    XYPoint YR [] = Arrays.copyOfRange(pointsByY, mid,n);

                    disL=findClosestPair(XL,YL,false);
                    disR=findClosestPair(XR,YR,false);
                    disLR=(disL<disR)?disL:disR;
                    minDis=disLR; //find the closest pair in the left or
                    XYPoint midPoint=pointsByX[mid];
                   int i=mid,j=mid,k=mid;
                   double crossTemp=INF;
                   for(i=mid;i>=0&&midPoint.x-pointsByX[i].x<=disLR;i--);
                        j=i+1;
                   for(i=mid;i<n&&pointsByX[i].x-midPoint.x<=disLR;i++);
                        k=i;
                   XYPoint ystrip[]=Arrays.copyOfRange(pointsByX, j, k);
                   for(j=0;j<=ystrip.length-2;j++)
                        for(k=j+1;k<=ystrip.length-1;k++)
                             if(ystrip[k].y-ystrip[j].y<=disLR){
                             crossTemp=ystrip[j].dist(ystrip[k]);
                             if(crossTemp<minDis){
                             minDis=crossTemp;
                             x1=ystrip[j];
                             x2=ystrip[k];
                              }   }     }
                   if (print)
                           System.out.println("DC " +
x1.toString()+x2.toString()+(double)Math.round(100000*minDis)/100000);

         return minDis;
        }
        };
```

Explain:

pointsByX and pointsByY are the sorted version of Points[], by X increasing order and by Y increasing order.

I use array.copyofrange() to split the two arrays in the middle into 4 arrays:xl,xr,yl,yr

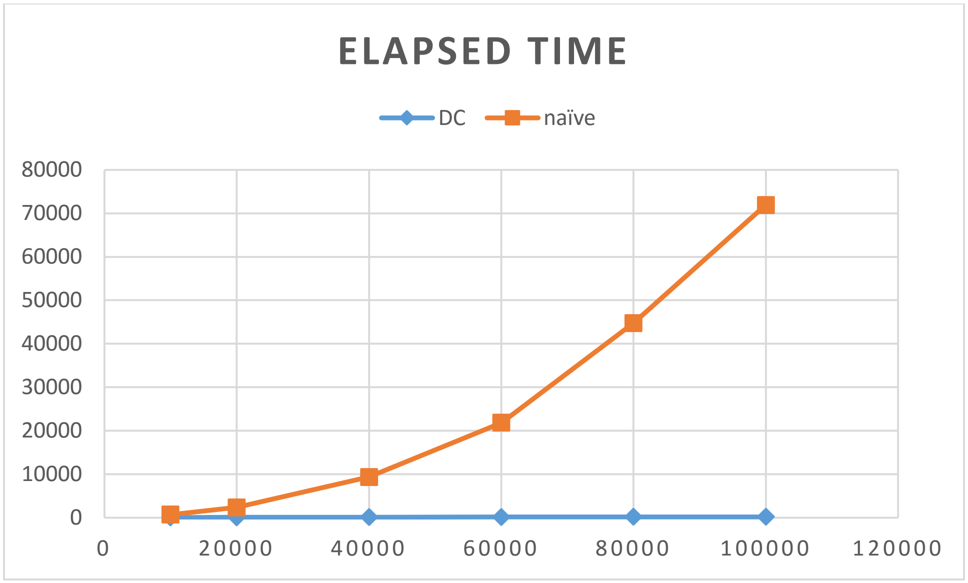
To every array,the size has be cut down to half of the original. And we use getClosetPair() …, this function will cut down n again and again until n is 1 or 2, when it is 1,we return inf so that we can add it into the ystrip later if its distance from the middle line is less than delta. If n=2, we calculate the distance of the two points. If this new distance is smaller than the current one, we set the mindis to be the new value, and we also update x1 and x2' value to keep track of the current (x,y)

For every disl and disr, we choose the min of them,set it dislr. And we use that to build a middle strip called ystrip, we put all the points that their x –middlepoint.x is smaller than dislr. At first, I don't know we can use arraylist in java to creat dynamic array, so I calculate the start and the end of the array we need to copy, and use array.copyofrange again to creat the exact ystrip. In ystrip, I use the same method in naïve algorithm to get the closet pair, and I use the same method to update the current mindis and current x1,x2.

Part 2:

The running time of the 2 algorithms are as below:

n	DC	naïve
10000	33	680
20000	53	2328
40000	87	9342
60000	136	21811
80000	160	44754
100000	169	71909



I add a for loop to run the DC algorithm 100 times:

When a new random xypoint array is created every loop:

The maximum is 176 milliseconds, the minimum is 17, average is 39.3

When the same xypoint array is used every loop:

The maximum is 145 milliseconds, the minimum is 17, average is 24.5

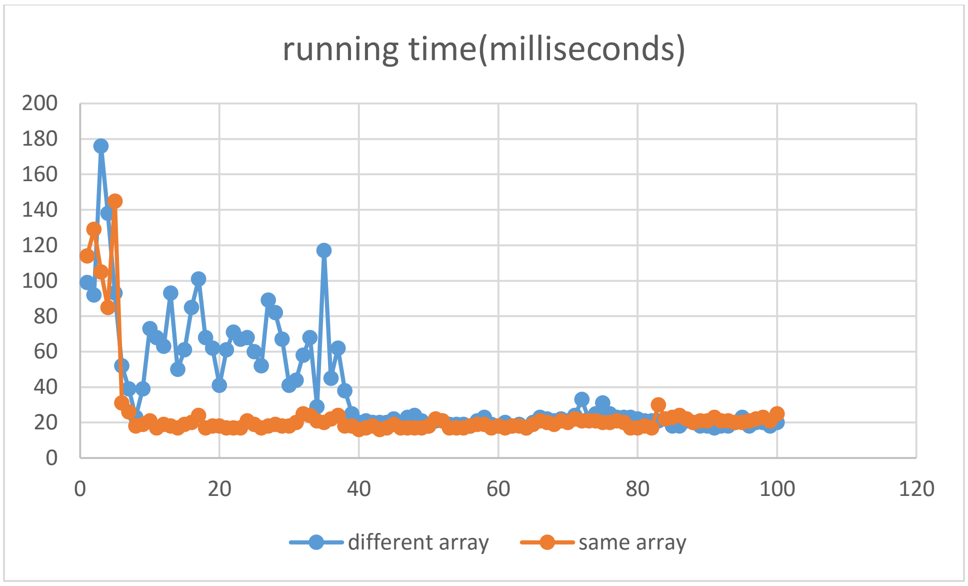
I found:

The 1st way (different array) costs more time than the 2nd way (same array).

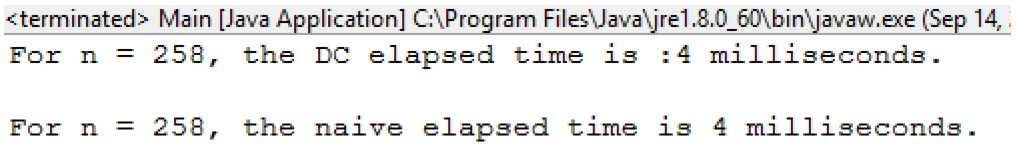
The first few times of loops cost more time than the later times, no matter which way.

I have a doubt: I thought the 2nd way (same array) should costs the same time or almost the same every loop. But the time won't look stable until about the 10th loop. At first, it varies a lot and becomes a straight line.

And I thought the 1st way(different array) should varies a lot, but after about 40 times of loop, the cost of every loop decreases to 20 milliseconds.

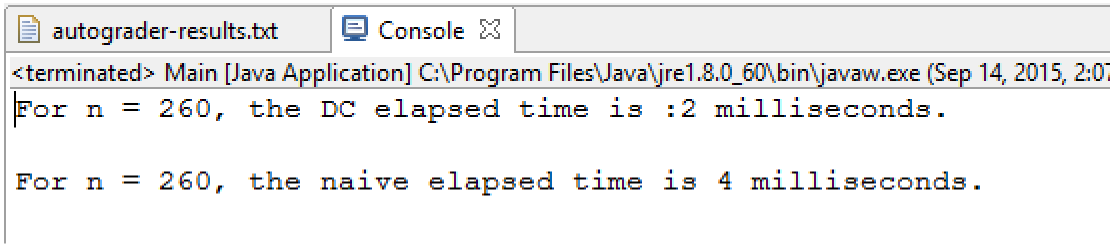


Part 3:



```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (Sep 14,
For n = 258, the DC elapsed time is :4 milliseconds.

For n = 258, the naive elapsed time is 4 milliseconds.
```



```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (Sep 14, 2015, 2:07
For n = 260, the DC elapsed time is :2 milliseconds.

For n = 260, the naive elapsed time is 4 milliseconds.
```

N=260, dc starts to win.