[1ai] Define your algorithm using pseudocode.

```
// I use brute force method to realize the function
// this is only show how to realize the brute force method
public RF(coordinate[], m){
      double min=Double.MaxValue
      int tempi=0; int    tempj=0;
      double temp;
      if (arr.size()==0||arr.size()==1|| result[].size==m) {
                return result[];
           } else {
                for (int i=0; i < arr.size(); i++) {
                     for (int j = i + 1; j < arr.size(); j++) {
                          temp = getDistance(coordinate[i], coordinate[j]);
                          if (temp < min) {
                               min = temp;
                               tempi=i;
                               tempj=j;
                          }
                     }

                }
                result[].add(min,coordinate[tempi], coordinate[tempj])
                coordinate[].remove(coordinate[tempi]&coordinate[tempj])
                RF(coordinate[],m);
           }
           return result[];
}
//there is also a algorithm to print the result
public printResult(result[])
StringBuffer buffer
           if (m>result[].size()){
                return error;
           }
           else {

           for(int i=0; i<m;i++){

                buffer.append(i+1    +    ".    "+"("+set.get(i).a.x+","+set.get(i).a.y+")"+"
"+"("+set.get(i).b.x+","+set.get(i).b.y+")"+'\n'+"distance=
"+set.get(i).distance+"\n"+"\n");

           }
           return buffer;
      }
```

}

1ai] Determine the worst-case running time (page 25) of your algorithm (call this the algorithm's worst-case running time).

The above algorithm will traverse all the input coordinates set, so the value range and sorting or not will not influence the running time.

But the coordinate sets varies through the process, and when the coordinate sets was traversed once, a result was output, m influence the running time, and when m=n/2, it is the worst case.

[2] Implement your algorithm. Your code must have a reasonable, consistent, style and documentation. It must have appropriate data structures, modularity, and error checking.

See HaoyuQi_PA01

[3] Perform and submit trace runs demonstrating the proper functioning of your code.

[4] Perform tests to measure the asymptotic behavior of your program (call this the code's worst case running time).

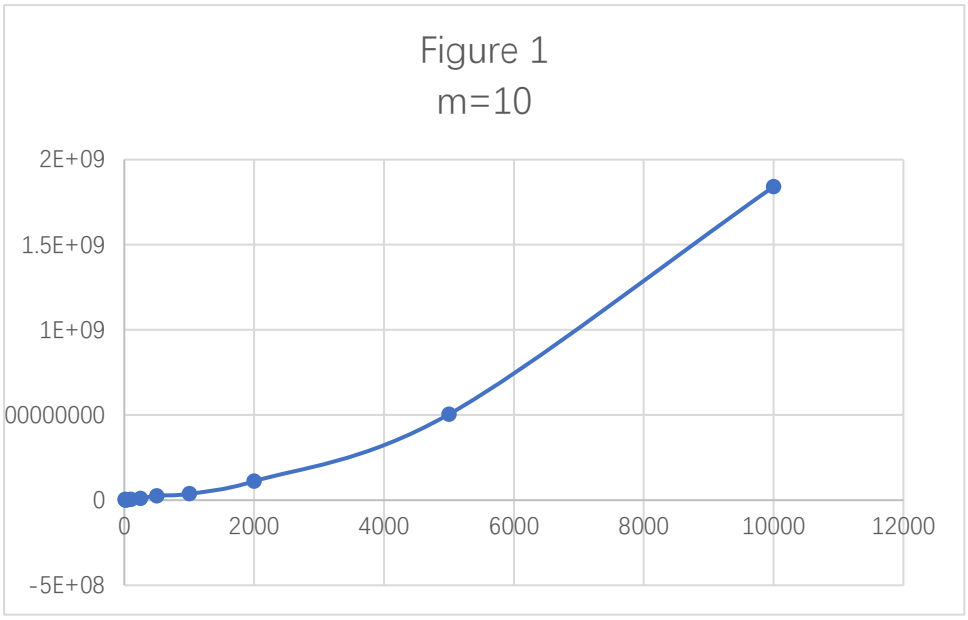
Use IDE or terminal command 'java test' trace the function, this program allow you to input the value of m or n as you wish, it also will display all the input coordinates in the set, and the m most closest coordinates pair with the distance and their coordinates. And because it is used to check the correctness and trace the function, so these output stream require a lot of space in the screen, here I use n=6, m=3 to demonstrate.

```
enter your size
6
enter your pairs m
3
your coordinate set is :
(3.652151483717136,1.2890139529789018)
(0.296009829056223,1.2205862528212656)
(4.094045114276158,1.8530556300682073)
(4.281414664707299,3.5749233819794872)
(1.4362844340722025,4.581234756308812)
(0.04138074212501841,2.2031124022721134)

VL The min distance and its correspond coordinates are
1. (3.652151483717136,1.2890139529789018)  (4.094045114276158,1.8530556300682073)
distance= 0.7165284156799316

2. (0.296009829056223,1.2205862528212656)  (0.04138074212501841,2.2031124022721134)
distance= 1.0149844884872437

3. (4.281414664707299,3.5749233819794872)  (1.4362844340722025,4.581234756308812)
distance= 3.0178515911102295


n=6   m=3
algorithm running time = 26140500 ns
```

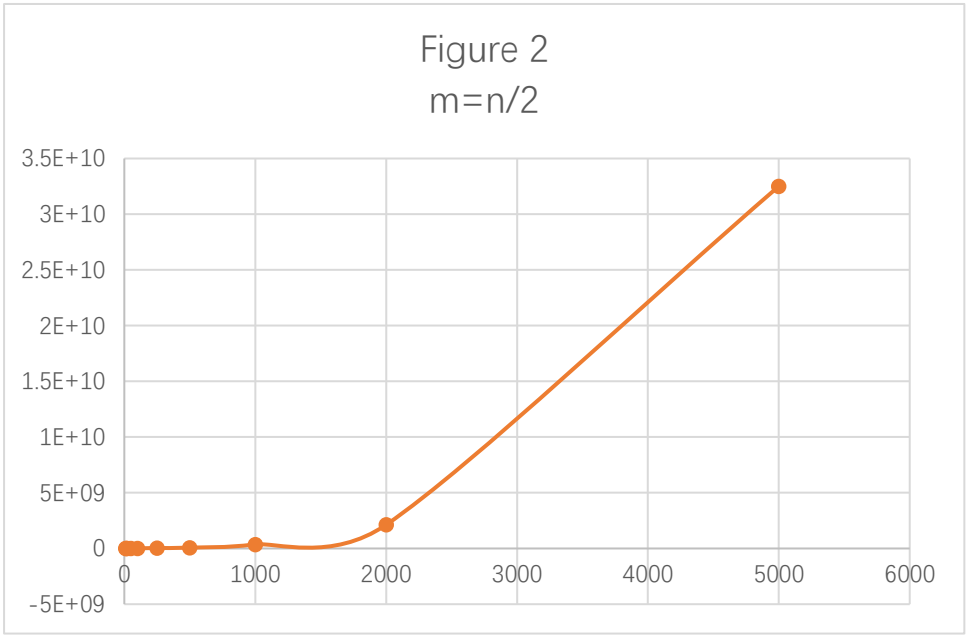
Also it has some error check function, if you enter a m > n/2, it will return the warning to remind you to modify the input m and n.

```
enter your size
6
enter your pairs m
6
your coordinate set is :
(3.652151483717136,1.2890139529789018)
(0.296009829056223,1.2205862528212656)
(4.094045114276158,1.8530556300682073)
(4.281414664707299,3.5749233819794872)
(1.4362844340722025,4.581234756308812)
(0.04138074212501841,2.2031124022721134)

ERROR, invalid m value, m should be at most n/2

VL The min distance and its correspond coordinates are
invalid m value, can not process the function

n=6   m=6
algorithm running time = 13269800 ns
```

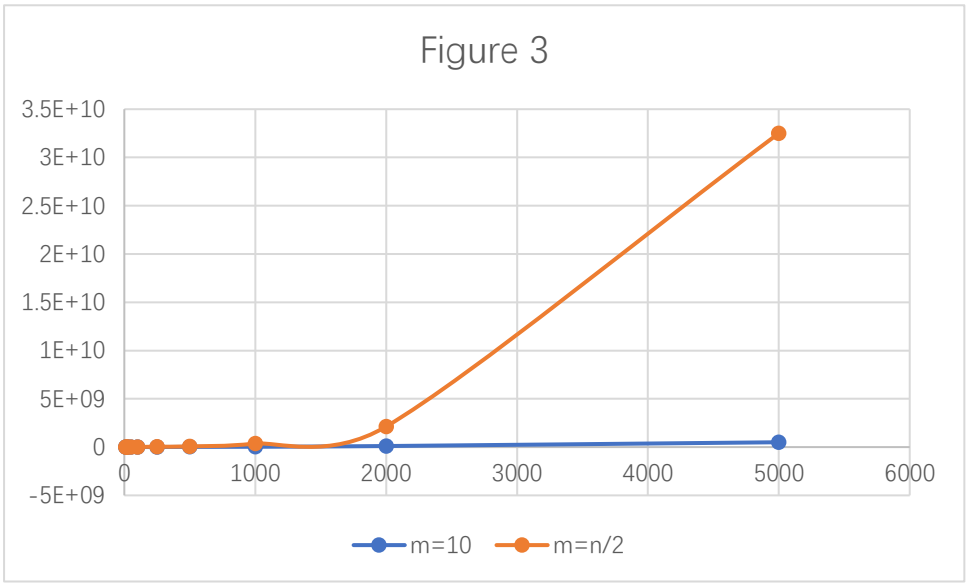
To perform the time complexity of the algorithm, first I perform the m=10, n={10,15,20,50,100,250,500,1000,2000,5000,10000}, and here is the n-time plot



You can see when m is constant, the n-time curve looks very similar nlgn curve. Then here is the worst time, which m=n/2

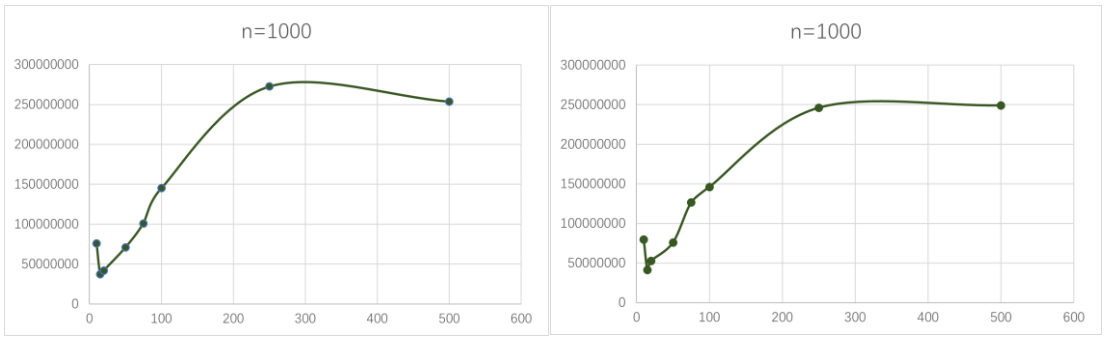


And let's put m=10 to the worst case for a comparison(figure 3), we can see the worst case has extremely more running time than m is a constant.

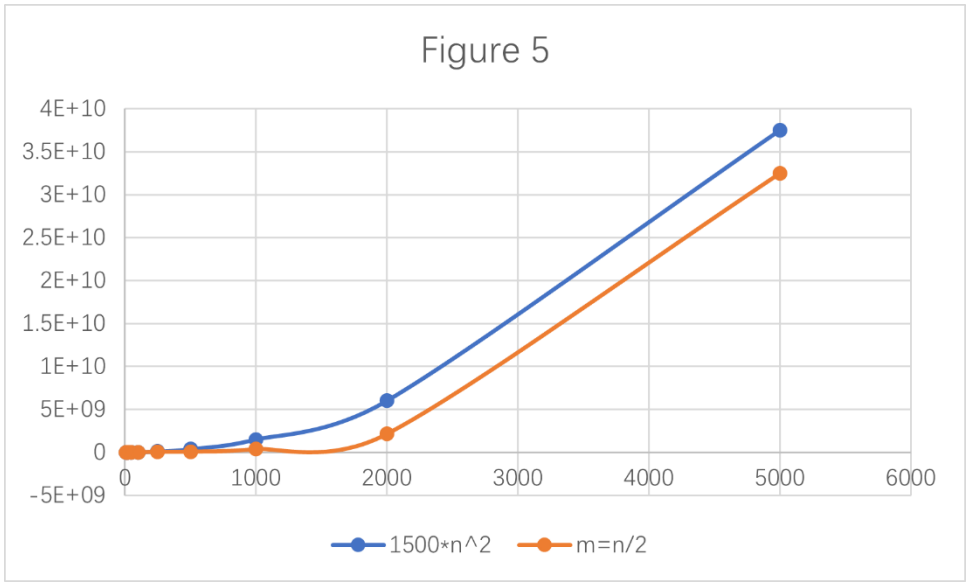


And here are 2 examples (figure 4)of n=1000, m= {10,15,20,50,75,100,250,500}.

Figure 4



To perform asymptotic behavior of my program, here is a comparison of the n-time curve of worst case and the curve of f(n)= 1500*x^2, the asymptotic behavior seems very significate.



This also perform the comparison between the code-worst case and the algorithm worst case.

3. Now that you have designed, implemented, and tested your algorithm, what aspects of your algorithm and/or code could change and reduce the worst-case running time of your algorithm? Be specific in your response to this question

Because I used the brute force method to deal with the problem, this method need to use 2 for circulation to traverse the set, so they need to multiple together makes the time complexcity to n^2, then the if front of them makes these code process m times. So the time complexity become the m n^2

```
if (arr.size()==0||arr.size()==1|| result[].size==m) {
        return result[];
    } else {
        for (int i=0; i < arr.size(); i++) {
            for (int j = i + 1; j < arr.size(); j++) {
                temp = getDistance(coordinate[i], coordinate[j]);
                if (temp < min) {
                    min = temp;
                    tempi=i;
                    tempj=j;
                }
            }
```

Take m time

Take n time

*

n time

One way to make progress of the algorithm is to adapt the divide-contour method which is introduced in CSRL P1039. But that solution can only out put the closet one pair. So for me, I plan to use the min heap to merge the sorted pairs of 2 sides, which is method of HW_03. 2, this will that O(n*lg2) time and it will get a heap that is easy to output m cloest pairs. This also do not need to be do m times of recursion.
Which means I do not need to do a recursion for the new method, it is no need to pop the cloucest pair of coordinates because we already have a sorted coordinate set and just traverse the set can realize the same function to the 2 for circulation in brute force method.
Using a liner sort can make the time complexity to O(n), heap needs O(lg n), print the result need O(m), so in total the time complexity could be O(n).