Problem:

KMEANS.

Problem Definition:

Let X = {x1,x2,x3,……..,xn} be the set of data points and V = {v1,v2,…….,vc} be the set of centers.

1) Randomly select ‘c’ cluster centers.

2) Calculate the distance between each data point and cluster centers.

3) Assign the data point to the cluster center whose distance from the cluster center is minimum of all the cluster centers.

4) Recalculate the new cluster center using Euclidean distance.

5) Recalculate the distance between each data point and new obtained cluster centers.

6) If no data point was reassigned then stop, otherwise repeat from step 3).Sample Solution:

Sample Solution:

```
public class Point {
/*theses will be the fields we use to decide distance from centroids
we could potentially have many more fields for classification but
for this example we will stick to three*/
private int x;
private int y;
private int z;
private int clusterId = 0;
```

```
public Point(int x, int y, int z){
        this.x = x;
        this.y = y;
        this.z = z;
}
public int getX(){
        return x;
}
public void setX(int x){
        this.x = x;
}
public int getY(){
        return y;
}
public void setY(int y){
        this.y = y;
}
public int getZ(){
        return z;
}
public void setZ(int z){
        this.z = z;
}
public int getClusterId(){
        return clusterId;
}
public void setClusterId(int clusterId){
        this.clusterId = clusterId;
```

```
	}
}
public class Cluster {

	ArrayList<Point> list;

	public Cluster(){
		list = new ArrayList<Point>();
	}
	public void addPoint(Point p){
		list.add(p);
	}
	public ArrayList<Point> getListOfPoints(){
		return list;
	}
}
public class Main {
	public static void main(String[] args) {
		int numOfCentroids = 3;
		int sizeOfDataset = 100;
		int numOfIterations = 10000;
		final ArrayList<Point> dataset = new ArrayList<Point>();;
		ArrayList<Point> oldCentroids = new ArrayList<Point>();
		ArrayList<Point> newCentroids = new ArrayList<Point>();

		initialize(numOfCentroids,sizeOfDataset,dataset,newCentroids);

	while(!testCentroids(oldCentroids,newCentroids,numOfCentroids)&&numOfIterations>0){
```

```
                ArrayList<Cluster> clusters = cluster(newCentroids,dataset);
                oldCentroids = newCentroids;
                newCentroids = getNewCentroids(clusters);
                outputIteration(dataset,oldCentroids);
                numOfIterations --;
        }
        System.out.println("K-Means ended --------------------------"+numOfIterations);
        //this will show us if the next iteration is the same as the previous
        ArrayList<Cluster> clusters = cluster(newCentroids,dataset);
        oldCentroids = newCentroids;
        newCentroids = getNewCentroids(clusters);
        outputIteration(dataset,oldCentroids);
        numOfIterations --;
}
private static void initialize(int numOfCentroids,int sizeOfDataset, ArrayList<Point> dataset,
                ArrayList<Point> newCentroids) {
        Random rand = new Random();

        //create the dataset
        for(int i=0;i<sizeOfDataset;i++){
                int x = rand.nextInt();
                int y = rand.nextInt();
                int z = rand.nextInt();
                Point p = new Point(x,y,z);
                dataset.add(p);
        }
        //create the new Centroids
                        for(int i=0;i<numOfCentroids;i++){
                                int x = rand.nextInt();
```

```
                int y = rand.nextInt();
                int z = rand.nextInt();
                Point p = new Point(x,y,z);
                newCentroids.add(p);
            }
        }
        // if centroids do not change then we are finished
        private static boolean testCentroids(ArrayList<Point> oldCentroids,ArrayList<Point>
newCentroids, int numOfCentroids) {
            for(int i=0;i<numOfCentroids;i++){
                if(oldCentroids.size()==0){
                    return false;
                }
                Point temp1 = oldCentroids.get(i);
                Point temp2 = newCentroids.get(i);

        if(temp1.getX()!=temp2.getX()&&temp1.getY()!=temp2.getY()&&temp1.getZ()!=temp2.getZ()){
                    return false;
                }
            }
            return true;

        }//test the points to decide which cluster it belongs to
        private static ArrayList<Cluster> cluster(ArrayList<Point> newCentroids,
                ArrayList<Point> dataset) {
            ArrayList<Cluster> result = new ArrayList<Cluster>();

            for(int i=0;i<newCentroids.size();i++){
```

```
                result.add(new Cluster());
            }
            for(Point p : dataset){
                int minDistance = Integer.MAX_VALUE;
                int index = 0;
                for(int i=0;i<newCentroids.size();i++){
                    // sqRt ((x2-x1)^2 +(y2-y1)^2+(z2-z1)^2)=distance
                    int distance = (int) Math.sqrt(Math.pow(p.getX()-
newCentroids.get(i).getX(), 2)+Math.pow(p.getY()-newCentroids.get(i).getY(), 2)+Math.pow(p.getZ()-
newCentroids.get(i).getZ(), 2));
                    if(distance<minDistance){
                        minDistance = distance;
                        index = i;
                    }
                }
                p.setClusterId(index+1);
                Cluster temp = result.get(index);
                temp.addPoint(p);
            }
            return result;
        }
        //test each cluster to find the centroid
        private static ArrayList<Point> getNewCentroids(ArrayList<Cluster> clusters) {
            ArrayList<Point> result = new ArrayList<Point>();
            for(Cluster c : clusters){
                int x = 0;
                int y = 0;
                int z = 0;
                int size = 0;
```

```
                for(Point p : c.getListOfPoints()){
                        x += p.getX();
                        y += p.getY();
                        z += p.getZ();
                        size++;
                }
                result.add(new Point(x/size,y/size,z/size));
        }
        return result;
    }
    //each iteration we will print the cetroids, all the points and which cluster they belonged to
    private static void outputIteration(ArrayList<Point> dataset,
                ArrayList<Point> oldCentroids) {
        System.out.println("Centroids: ");
        int count = 1;
        for(Point p : oldCentroids){
                System.out.println("cluster"+count+" - X="+p.getX()+" Y="+p.getY()+"
Z"+p.getZ());
                count++;
        }
        System.out.println("**********************************");
        for(Point p : dataset){
                System.out.println("X="+p.getX()+" Y="+p.getY()+" Z="+p.getZ()+"
Cluster="+p.getClusterId());
        }
        System.out.println("*****************End Of Iteration*****************");
    }
}
```

Conversation:

This is a quick and dirty solution to demonstrate Kmeans again there is room for improvement i.e. we could refactor the point class to hold an array of attributes then we would refactor are main class to test all attributes as opposed to only testing the static x, y, and z. Then we could read in large datasets with numerus attributes.