TABLE OF CONTENTS

SYNOPSIS

Clustering is a division of data into groups of similar objects. One of clustering technique is K-Mean. K-means is one of the simplest unsupervised learned algorithm that solves the clustering problem. The main idea with the K-means is to define k centroids, one for each cluster. It is mainly used to partition n observations into (k) clusters in which each observation belong to the cluster with the nearest mean and forming a cluster with the nearest data. Hence, over large data set, traditional k-means produce high complexity. Hence the proposed scheme will reduce the drawbacks of the traditional k-mean.

In proposed scheme, instead of choosing the value from random data set, we first sort the data set using merge sort, based on difference between corresponding dataset and find the corresponding centroid of each dataset. The results obtained by the proposed k-means clustering algorithm over a dataset and the results obtained by applying enhanced k-means algorithm are compared.

Hence, when sorted, the number of iterations obtained between with sort and without sort are compared and the graph representation is presented for the understanding purpose. The experiment results in reducing the complexity and number of iterations compared to traditional K-Means algorithm over a large data. Futuristic improvement like implementing greedy algorithm along with enhanced k-mean clustering technique will be likely possible to improve more efficiency.

HARDWARE REQUIREMENTS:

System	:	Intel core i-5,64-bit OS
Hard Disk	:	1TB.
Monitor	:	15.6 VGA Colour.
Ram	:	8GB

SOFTWARE REQUIREMENTS:

Operating system	:	Windows 7.
Coding Language	:	Java, Data Sources(ODBC)
IDE	:	

PROJECT DESCRIPTION

The K-Means algorithm is an algorithm to cluster objects based on the attributes into k partitions where k<n. K-Means crack to find centres of clusters in data. The procedure of K-Means follows by classifying a given data set into number of clusters (approximate value of clusters) that are fixed prior. The main objective is to define centres to each cluster. Centres are chosen randomly in this algorithm. Later, the difference is calculated between the points and the centres. That difference will give the nearest distance which routes to the nearest centre. The one with the nearest distance will eventually be in the same cluster. The first step will be completed only when there are no points pending and then grouping is done. After the first step, the second one is calculated with the mean and with the centroids. The re-calculations are done until the data in the clusters are same as the previous step. We notice that, after few steps, the centres change their location step by step until no more changes occur. At last, the main objective is to achieve minimum intra cluster variance or to minimize the squared error function.

$$J(V)= \sum \sum (|x_i - v_j|)^2$$

Where, $(|x_i - v_j|)$ is the Euclidean distance between x_i and v_j .

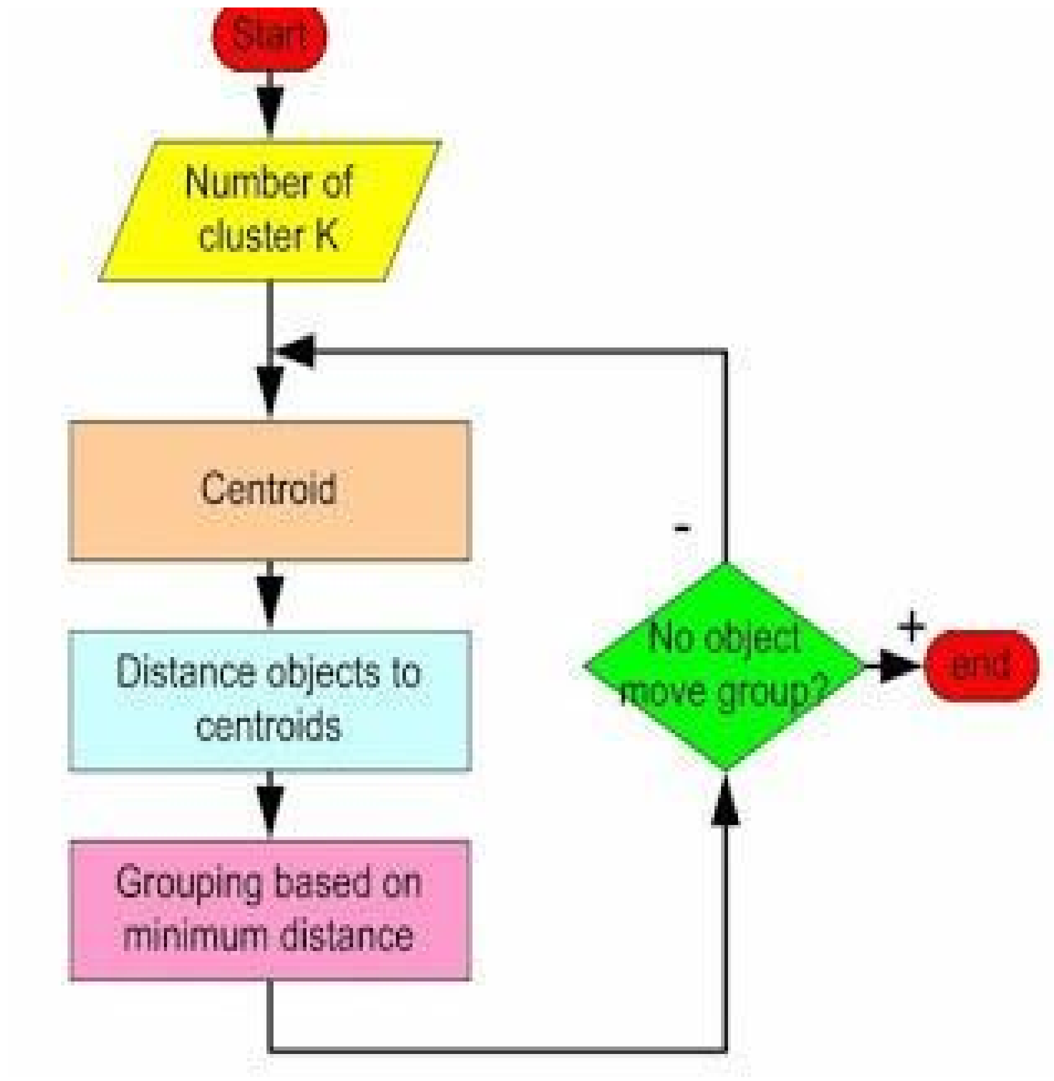
K-Means algorithm is mainly suitable for the small amount of dataset. When the large dataset is given, the traditional k-means will take high time to produce the output and hence the complexity will be high. Hence, different approaches are made in order to reduce the complexity. In terms of performance, the algorithm is not certain to return a global optimum. The drawback of the k-means algorithm is that the number of clusters k is an input parameter. An inappropriate choice of k may lead to the poor results.

K-Means Algotithm:

Algorithm: k-means .The k-means algorithm for partitioning, where each cluster's centre is represented by the mean value of the objects in the cluster.

Input:
K: the number of clusters.
D: a data set containing n objects.

Output:
A set of k clusters.


Start
Number of cluster K
Centroid
Distance objects to centroids
Grouping based on minimum distance
-
No object move group?
+
end

Method:

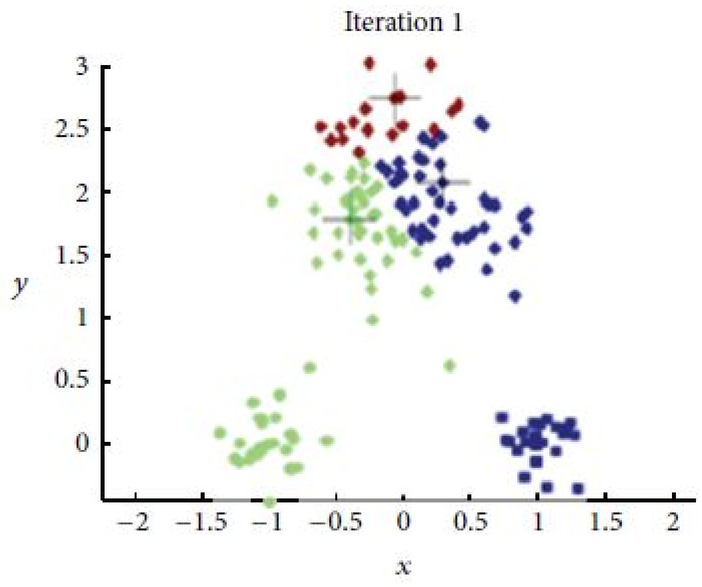
(1) randomly choose k objects from D as the initial cluster centres;

(2) **Repeat**

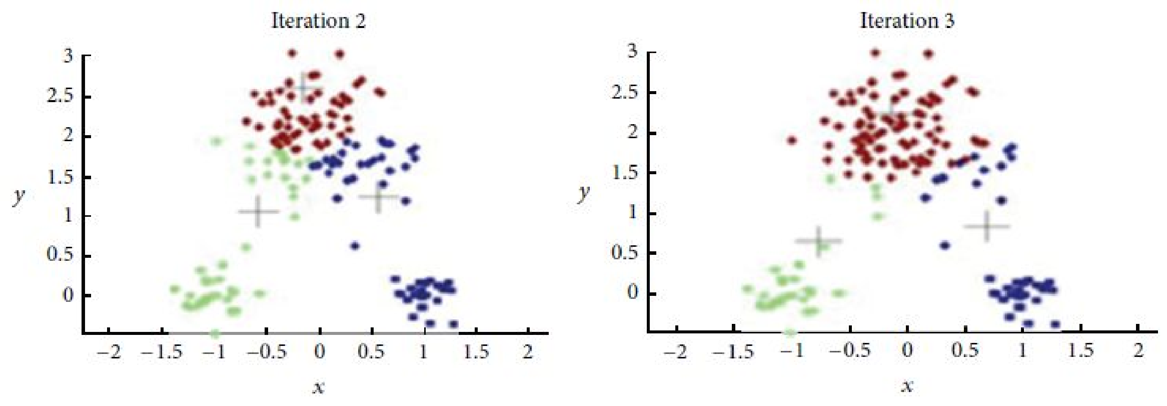
(3) (re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster;

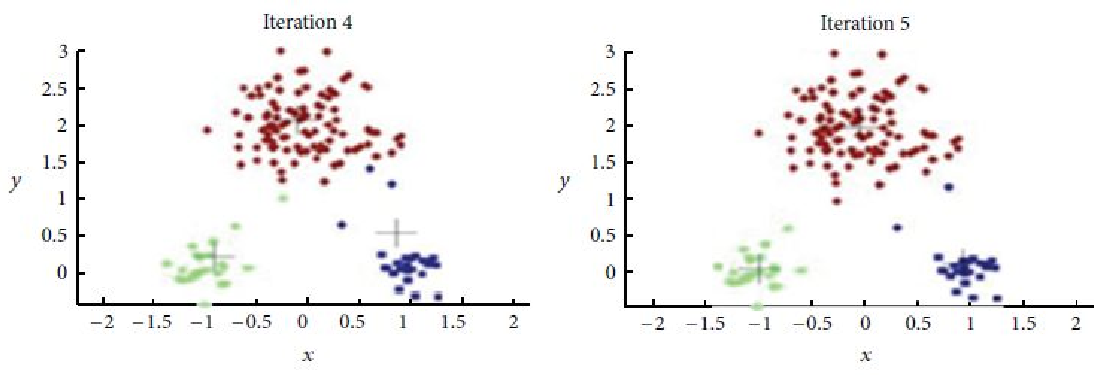
(4) update the cluster means , i.e., calculate the mean value of the objects for each clusters;

(5) **until** no change;

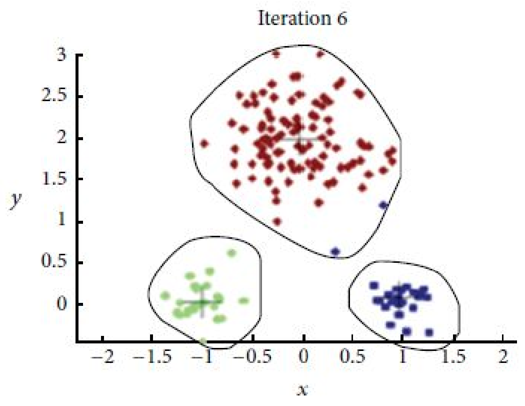


(a)





(b)



(c)

FIGURE 1: (a) Initial centroids depiction. (b) Repositioning of centroids. (c)k-means converges.

K-means algorithm is composed of four steps:

(1) Randomly place k elements in a space representing the item coordinates that are being clustered.
(2) Allocate each item in the space to a group that is most similar to it.
(3) After the assignment of all items in the space, recomputed the k centroid elements and change their positions, respectively.
(4) Repeat steps (2) and (3) until the centroids reach a position where they no longer change with respect to the distances between all the elements of their group.

However, traditional K-means mostly doesn’t give any optimum solutions. The algorithm is made run different times to avoid the difficulties or situations of not being optimum in giving solutions and hence it will choose different set of centroids each time, making it very hard to compare it with the initial conditions.

In figure 1(a) it shows how the initial centroid selections are made with the colouring factor. Plus symbol depicts that particular elements are centroids.

In figure 1(b) number of iterations are concluded. Here, the assignment of each element to its closest centroid is done by the distance calculation between the initial cluster centroids and the each point in the space. Re-calculations are done at each iteration over this phase.

In figure 1(c) is where the algorithm reaches to the final position. This happens when the algorithm compares the centroids from the last step to its current step and notices that there are no changes in the centroids; thus the full clusters have been reached.

To avoid such minimum global optimum, enhancement of k-mean clustering is used. The data here is normally given in unsorted order. Hence, before the initial centroids are chosen, the given data is sorted by using one of the sorting techniques with the best time complexity. It is because, the sorting algorithm puts the elements in correct order. Sorting algorithms are generally meant for the best optimization solutions. Hence, Merge sort is the best sorting technique that is opted. It is because, it has the best time complexity of O(nlogn).

Merge sort is a sorting technique based on divide and conquer method. It is a recursive type algorithm in which it divides or splits the list continuously in to half. If there is only one item in the list, it is sorted by definition. If the list has more than one item, it splits recursively invoking merge sort on both halves. Once these two halves are sorted, merge function is applied. Merging is the process of taking smaller lists and making it to one, sorted, new list.

Merge sort Algorithm:

To sort the entire sequence A[1...n],make the initial call to the procedure MERGE-SORT(A,1,n)

MERGE-SORT (A,p,r)

```
1. IF p<r                              //Check for base case
2. THEN q=FLOOR[(p+r)]/2               //Divide step
3. MERGE(A,p,q)                        //conquer step
4. MERGE(A,q+1,r)                      //conquer step
```

```
5. MERGE(A,p,q,r)                          //conquer step
```

Pseudo code for the Merge is as follows:

MERGE (A,p,q,r)

```
1.  $n_1$ □ q-p+1
2.  $n_2$ □ r-q
3.  Create arrays L[1....$n_1$+1] and R[1...$n_2$+1]
4.  **FOR** i□1 TO $n_1$
5.  **DO** L[i] □A[p+i-1]
6.  **FOR** j□1 TO $n_2$
7.  **DO** R[$j$] ← A[$q + j$ ]
8.      L[$n_1$ + 1] ← ∞
9.    R[$n_2$ + 1] ← ∞
10.   $i \leftarrow 1$
11.   $j \leftarrow 1$
12.   **FOR** $k \leftarrow p$ **TO** $r$
13.       **DO IF** L[$i$ ] ≤ R[ $j$]
14.   **THEN** A[$k$] ← L[$i$]
15.             $i \leftarrow i + 1$
16.     **ELSE** A[k] ← R[j]
17.             $j \leftarrow j + 1$
```

Running Time:

The first two **for** loops (that is, the loop in line 4 and the loop in line 6) take $\Theta(n_1 + n_2) = \Theta(n)$ time. The last **for** loop (that is, the loop in line 12) makes n iterations, each taking constant time, for $\Theta(n)$ time. Therefore, the total running time is $\Theta(n)$.

Analyzing Merge sort:

For simplicity, assume that n is a power of 2 ,so that each divide step yields two sub problems, both of size of n/2.

Base case occurs when n=1.

When n>=2, time for merge sort steps:

Divide : Just compute q as the average of p and r, which takes constant time i.e. $\Theta(1)$.

Conquer: Recursively solve 2 sub problems, each of size $n/2$, which is $2T(n/2)$.

Combine: MERGE on an *n*-element sub array takes Θ (*n*) time.

Summed together they give a function that is linear in *n*, which is Θ(*n*). Therefore, the recurrence for merge sort running time is

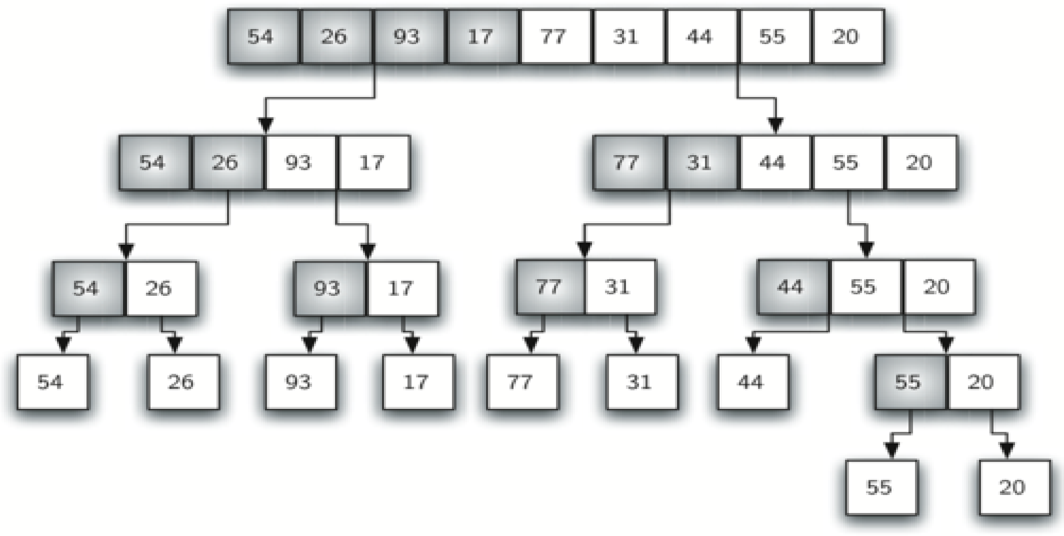
$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 , \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 . \end{cases}$$

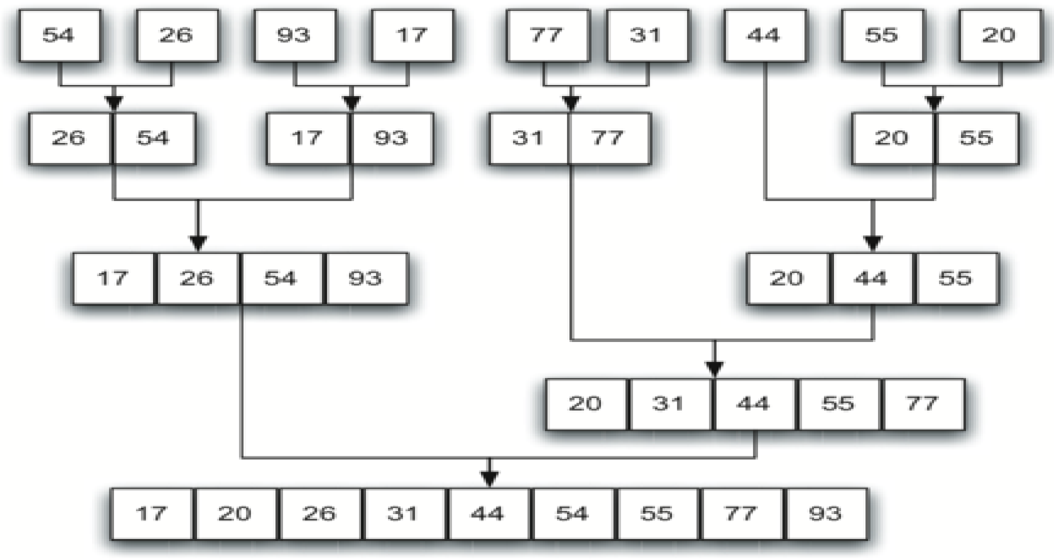
The merge sort recurrence has the solution: T(n) = Θ(*n log n)*

MERGE Sort diagram:

a) Splitting the list in merge sort order.

b) Lists as they are Merged together.





Code:

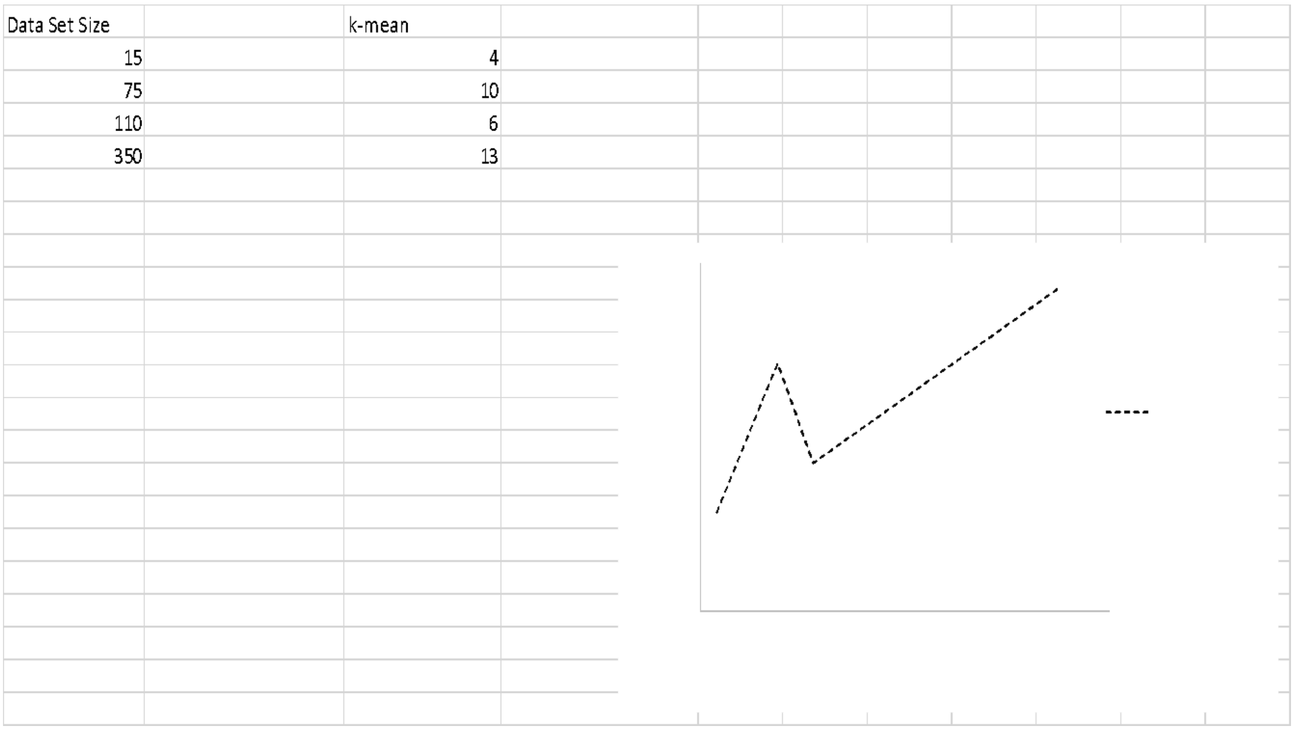
K-Means without sorting:

```
import java.util.*;
import java.sql.*;

class KMEAN2withoutsort
{
static int count1,count2,count3;
static int d[];
static int k[][];
static int tempk[][];
static double m[];
static double diff[];
static int n,p;
static int st;
static int count;
static int cal_diff(int a) // This method will determine the
cluster in which an element go at a particular step.
{
int temp1=0;
for(int i=0;i<p;++i)
{
if(a>m[i])
diff[i]=a-m[i];
else
diff[i]=m[i]-a;
}
int val=0;
```

```
double temp=diff[0];
for(int i=0;i<p;++i)
{
if(diff[i]<temp)
{
temp=diff[i];
val=i;
}
}//end of for loop
return val;
}

static void cal_mean() // This method will determine
intermediate mean values
{
for(int i=0;i<p;++i)
m[i]=0; // initializing means to 0
int cnt=0;
for(int i=0;i<p;++i)
{
cnt=0;
for(int j=0;j<n-1;++j)
{
if(k[i][j]!=-1)
{
m[i]+=k[i][j];
++cnt;
}}
m[i]=m[i]/cnt;
}
}

static int check1() // This checks if previous k ie. tempk and
current k are same.Used as terminating case.
{
for(int i=0;i<p;++i)
for(int j=0;j<n;++j)
if(tempk[i][j]!=k[i][j])
{
return 0;
}
return 1;
}
public static void mergeSort(int [ ] a)
          {
          int[] tmp = new int[a.length];
          System.out.println("Calling From Merge  Sort");


          for(int km=0;km<a.length;km++)
          System.out.println(a[km]);

          mergeSort(a, tmp,  0,  a.length - 1);
          }
private static void mergeSort(int [ ] a, int [ ] tmp, int
left, int right)
          {
               if( left < right )
               {
                    int center = (left + right) / 2;
                    mergeSort(a, tmp, left, center);

                    mergeSort(a, tmp, center + 1, right);
                    merge(a, tmp, left, center + 1, right);

               }
          }
private static void merge(int[ ] a, int[ ] tmp, int left, int
right, int rightEnd )
           {
                     int leftEnd = right - 1;
                    int k = left;
                    int num = rightEnd - left + 1;
                           while(left <= leftEnd && right <=
rightEnd)
                           if( (a[left]) - (a[right]) <= 0)
                                  tmp[k++] = a[left++];
                                  else
                                   tmp[k++] = a[right++];

                         while(left <= leftEnd)    // Copy
rest of first half
                       tmp[k++] = a[left++];

                    while(right <= rightEnd)  // Copy rest of
right half
                             tmp[k++] = a[right++];

        // Copy tmp back
                           for(int i = 0; i < num; i++,
rightEnd--)
                       a[rightEnd] = tmp[rightEnd];
          }
```

```
public static void main(String args[])
{

try{

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection c =
DriverManager.getConnection("jdbc:odbc:final_book");
Statement st = c.createStatement();
ResultSet rs = st.executeQuery("select * from [Sheet1$]");

int no_of_iter=0;
Scanner scr=new Scanner(System.in);
/* Accepting number of elements */
//System.out.println("Enter the number of elements ");
//n=scr.nextInt();
int dup[]=new int[200];

/* Accepting elements */
//System.out.println("Enter "+n+" elements: ");
//for(int i=0;i<n;++i)
//d[i]=scr.nextInt();


int i1=0;
while(rs.next()){

dup[i1++]=rs.getInt("marks");
System.out.println(dup[i1-1]);


}

int d[]=new int[i1];

for(int km=0;km<i1;km++)
     d[km]=dup[km];

n=i1;

System.out.println("\nBefore merge sort:\n");
for(int j=0;j<n;j++)
     System.out.println(d[j]+"   ");

//mergeSort(d);
```

```
System.out.println("|nafter merge sort:\n");
for(int j=0;j<n;j++)
     System.out.println(d[j]+"   ");
/* Accepting num of clusters */
System.out.println("Enter the number of clusters: ");
p=scr.nextInt();
/* Initialising arrays */
k=new int[p][n];
tempk=new int[p][n];
m=new double[p];
diff=new double[p];
/* Initializing m */
for(int i=0;i<p;++i)
m[i]=d[i];

int temp=0;
int flag=0;
do
{
for(int i=0;i<p;++i)
for(int j=0;j<n;++j)
{
k[i][j]=-1;
}
for(int i=0;i<n;++i) // for loop will cal cal_diff(int) for
every element.
{
temp=cal_diff(d[i]);
if(temp==0)
k[temp][count1++]=d[i];
else
if(temp==1)
k[temp][count2++]=d[i];
else
if(temp==2)
k[temp][count3++]=d[i];
}
cal_mean(); // call to method which will calculate mean at
this step.
no_of_iter++;
flag=check1(); // check if terminating condition is satisfied.
if(flag!=1)
/*Take backup of k in tempk so that you can check for
equivalence in next step*/
for(int i=0;i<p;++i)
for(int j=0;j<n;++j)
tempk[i][j]=k[i][j];
System.out.println("\n\nAt this step");
System.out.println("\nValue of clusters");
for(int i=0;i<p;++i)
{
System.out.print("K"+(i+1)+"{ ");
for(int j=0;k[i][j]!=-1 && j<n-1;++j)
System.out.print(k[i][j]+" ");
System.out.println("}");
}//end of for loop
System.out.println("\nValue of m ");
for(int i=0;i<p;++i)
System.out.print("m"+(i+1)+"="+m[i]+"  ");

count1=0;count2=0;count3=0;
}
while(flag==0);

System.out.println("\n\n\nThe Final Clusters By Kmeans are as
follows: ");
System.out.println("\n\n\nTotal no. of iterations:
"+no_of_iter);
for(int i=0;i<p;++i)
{
System.out.print("K"+(i+1)+"{ ");
for(int j=0;k[i][j]!=-1 && j<n-1;++j)
System.out.print(k[i][j]+" ");
System.out.println("}");
}

}
catch(Exception e){

System.out.println(e);

}

}
}
```



Data Set Size		k-mean
15		4
75		10
110		6
350		13

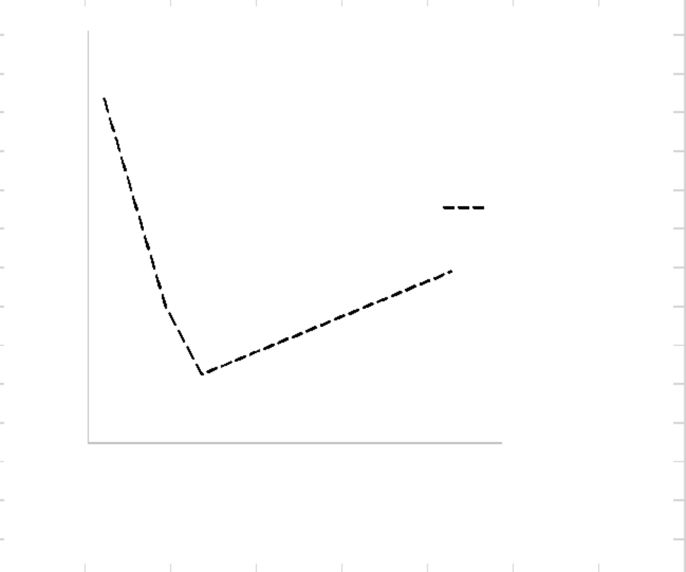
K-Means with sorting(Merge Sort):

```
import java.util.*;
import java.sql.*;

class KMEAN2withsort
{
static int count1,count2,count3;
static int d[];
static int k[][];
static int tempk[][];
static double m[];
static double diff[];
static int n,p;
static int st;
static int count;
static int cal_diff(int a) // This method will determine the
cluster in which an element go at a particular step.
{
int temp1=0;
for(int i=0;i<p;++i)
{
if(a>m[i])
diff[i]=a-m[i];
else
diff[i]=m[i]-a;
}
int val=0;
double temp=diff[0];
for(int i=0;i<p;++i)
{
if(diff[i]<temp)
{
temp=diff[i];
val=i;
}
}//end of for loop
return val;
}

static void cal_mean() // This method will determine
intermediate mean values
{
for(int i=0;i<p;++i)
m[i]=0; // initializing means to 0
int cnt=0;
for(int i=0;i<p;++i)
{
cnt=0;
for(int j=0;j<n-1;++j)
{
if(k[i][j]!=-1)
{
m[i]+=k[i][j];
++cnt;
}}
m[i]=m[i]/cnt;
}
}

static int check1() // This checks if previous k ie. tempk and
current k are same.Used as terminating case.
{
for(int i=0;i<p;++i)
for(int j=0;j<n;++j)
if(tempk[i][j]!=k[i][j])
{
return 0;
}
return 1;
}
public static void mergeSort(int [ ] a)
          {
          int[] tmp = new int[a.length];

          System.out.println("Calling From Merge  Sort");


          for(int km=0;km<a.length;km++)
          System.out.println(a[km]);

          mergeSort(a, tmp,  0,  a.length - 1);
          }
private static void mergeSort(int [ ] a, int [ ] tmp, int
left, int right)
          {
               if( left < right )
               {
                    int center = (left + right) / 2;
                    mergeSort(a, tmp, left, center);

                    mergeSort(a, tmp, center + 1, right);
                    merge(a, tmp, left, center + 1, right);

               }
          }
private static void merge(int[ ] a, int[ ] tmp, int left, int
right, int rightEnd )
           {
                     int leftEnd = right - 1;
                    int k = left;
                    int num = rightEnd - left + 1;
                           while(left <= leftEnd && right <=
rightEnd)
                           if( (a[left]) - (a[right]) <= 0)
                                   tmp[k++] = a[left++];
                                   else
                                    tmp[k++] = a[right++];

                         while(left <= leftEnd)    // Copy
rest of first half
                       tmp[k++] = a[left++];
                    while(right <= rightEnd)  // Copy rest of
right half
                              tmp[k++] = a[right++];

        // Copy tmp back
                           for(int i = 0; i < num; i++,
rightEnd--)
                       a[rightEnd] = tmp[rightEnd];
          }



public static void main(String args[])
{

try{

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection c =
DriverManager.getConnection("jdbc:odbc:final_book");
Statement st = c.createStatement();
ResultSet rs = st.executeQuery("select * from [Sheet1$]");

int no_of_iter=0;
Scanner scr=new Scanner(System.in);
/* Accepting number of elements */
//System.out.println("Enter the number of elements ");
//n=scr.nextInt();
int dup[]=new int[1000];

/* Accepting elements */
//System.out.println("Enter "+n+" elements: ");
//for(int i=0;i<n;++i)
//d[i]=scr.nextInt();


int i1=0;
while(rs.next()){

dup[i1++]=rs.getInt("marks");
System.out.println(dup[i1-1]);


}

int d[]=new int[i1];
```

```
for(int km=0;km<i1;km++)
     d[km]=dup[km];

n=i1;

System.out.println("\nBefore merge sort:\n");
for(int j=0;j<n;j++)
     System.out.println(d[j]+"   ");

mergeSort(d);

System.out.println("|nafter merge sort:\n");
for(int j=0;j<n;j++)
     System.out.println(d[j]+"   ");
/* Accepting num of clusters */
System.out.println("Enter the number of clusters: ");
p=scr.nextInt();
/* Initialising arrays */
k=new int[p][n];
tempk=new int[p][n];
m=new double[p];
diff=new double[p];
/* Initializing m */
for(int i=0;i<p;++i)
m[i]=d[i];

int temp=0;
int flag=0;
do
{
for(int i=0;i<p;++i)
for(int j=0;j<n;++j)
{
k[i][j]=-1;
}
for(int i=0;i<n;++i) // for loop will cal cal_diff(int) for
every element.
{
temp=cal_diff(d[i]);
if(temp==0)
k[temp][count1++]=d[i];
else
if(temp==1)
k[temp][count2++]=d[i];
else
if(temp==2)
k[temp][count3++]=d[i];
}
cal_mean(); // call to method which will calculate mean at
this step.
no_of_iter++;
flag=check1(); // check if terminating condition is satisfied.
if(flag!=1)
/*Take backup of k in tempk so that you can check for
equivalence in next step*/
for(int i=0;i<p;++i)
for(int j=0;j<n;++j)
tempk[i][j]=k[i][j];

System.out.println("\n\nAt this step");
System.out.println("\nValue of clusters");
for(int i=0;i<p;++i)
{
System.out.print("K"+(i+1)+"{ ");
for(int j=0;k[i][j]!=-1 && j<n-1;++j)
System.out.print(k[i][j]+" ");
System.out.println("}");
}//end of for loop
System.out.println("\nValue of m ");
for(int i=0;i<p;++i)
System.out.print("m"+(i+1)+"="+m[i]+"  ");

count1=0;count2=0;count3=0;
}
while(flag==0);

System.out.println("\n\n\nThe Final Clusters By Kmeans are as
follows: ");
System.out.println("\n\n\nTotal no. of iterations:
"+no_of_iter);
for(int i=0;i<p;++i)
{
System.out.print("K"+(i+1)+"{ ");
for(int j=0;k[i][j]!=-1 && j<n-1;++j)
System.out.print(k[i][j]+" ");
System.out.println("}");
}

}
catch(Exception e){

System.out.println(e);

}

}
```

```
}
```

Data Set Size	Enhanced k-mean
15	10
75	4
110	2
350	5

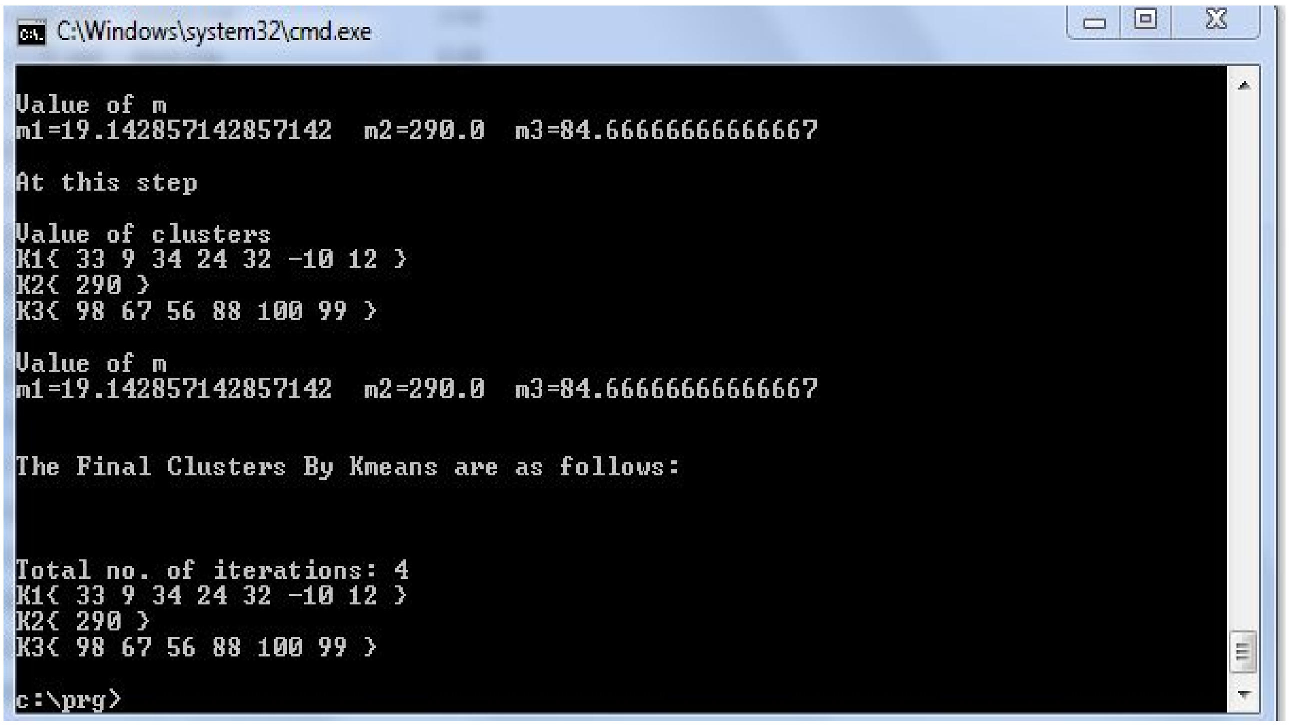


Test output:

K-Means without sort:

Clusters used: 3

Dataset: 15 elements

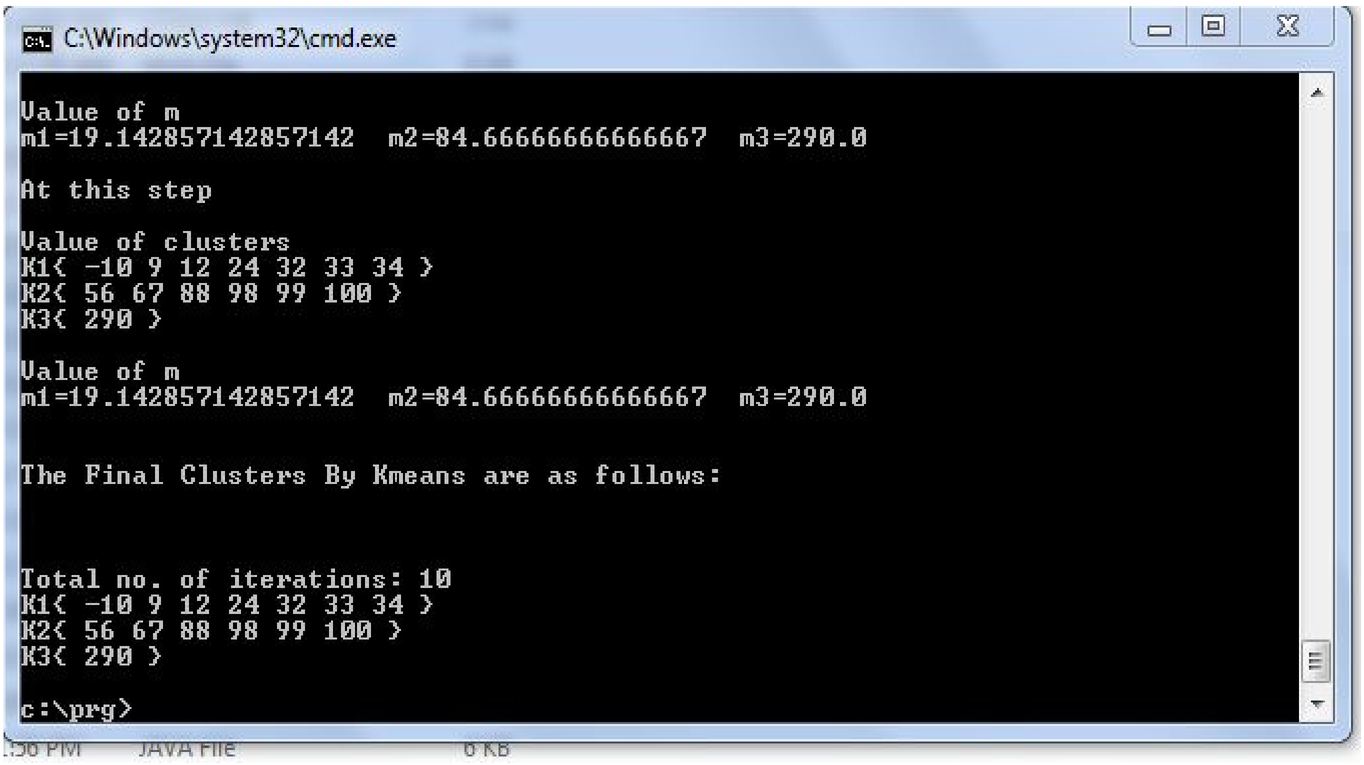

C:\Windows\system32\cmd.exe

```
Value of m
m1=19.142857142857142  m2=290.0  m3=84.66666666666667

At this step

Value of clusters
K1{ 33 9 34 24 32 -10 12 }
K2{ 290 }
K3{ 98 67 56 88 100 99 }

Value of m
m1=19.142857142857142  m2=290.0  m3=84.66666666666667


The Final Clusters By Kmeans are as follows:



Total no. of iterations: 4
K1{ 33 9 34 24 32 -10 12 }
K2{ 290 }
K3{ 98 67 56 88 100 99 }

c:\prg>
```

K-Means with sort:

Clusters used: 3

Data set: 15 elements:



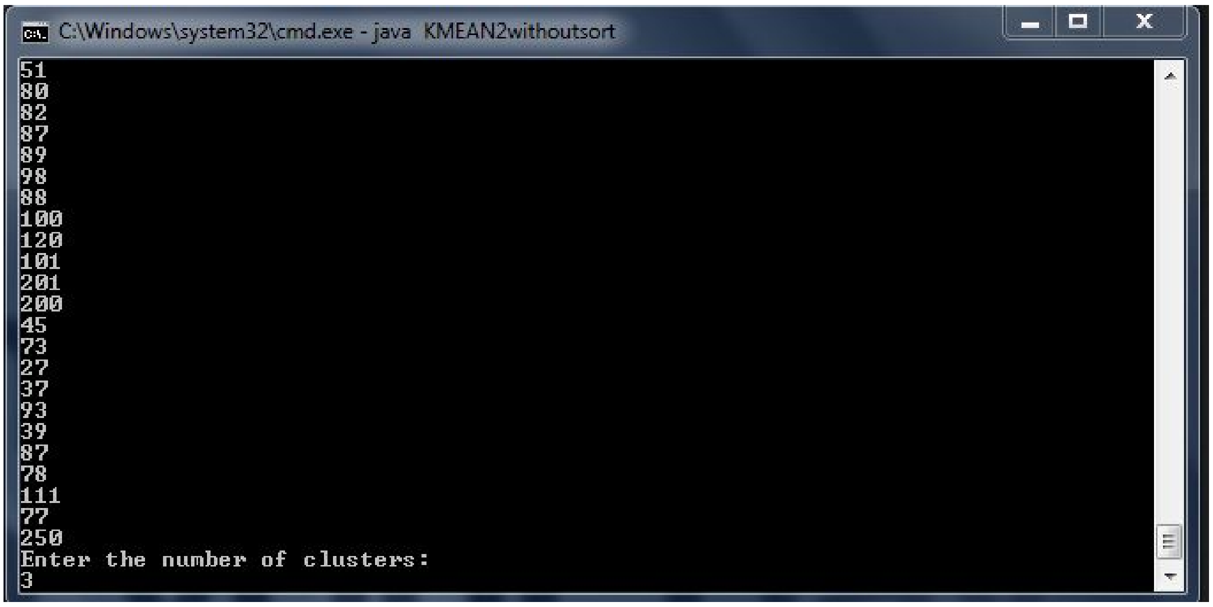
```
Value of m
m1=19.142857142857142  m2=84.66666666666667  m3=290.0

At this step

Value of clusters
K1{ -10 9 12 24 32 33 34 }
K2{ 56 67 88 98 99 100 }
K3{ 290 }

Value of m
m1=19.142857142857142  m2=84.66666666666667  m3=290.0


The Final Clusters By Kmeans are as follows:



Total no. of iterations: 10
K1{ -10 9 12 24 32 33 34 }
K2{ 56 67 88 98 99 100 }
K3{ 290 }

c:\prg>
```

Test output 2:

K-Means without sort

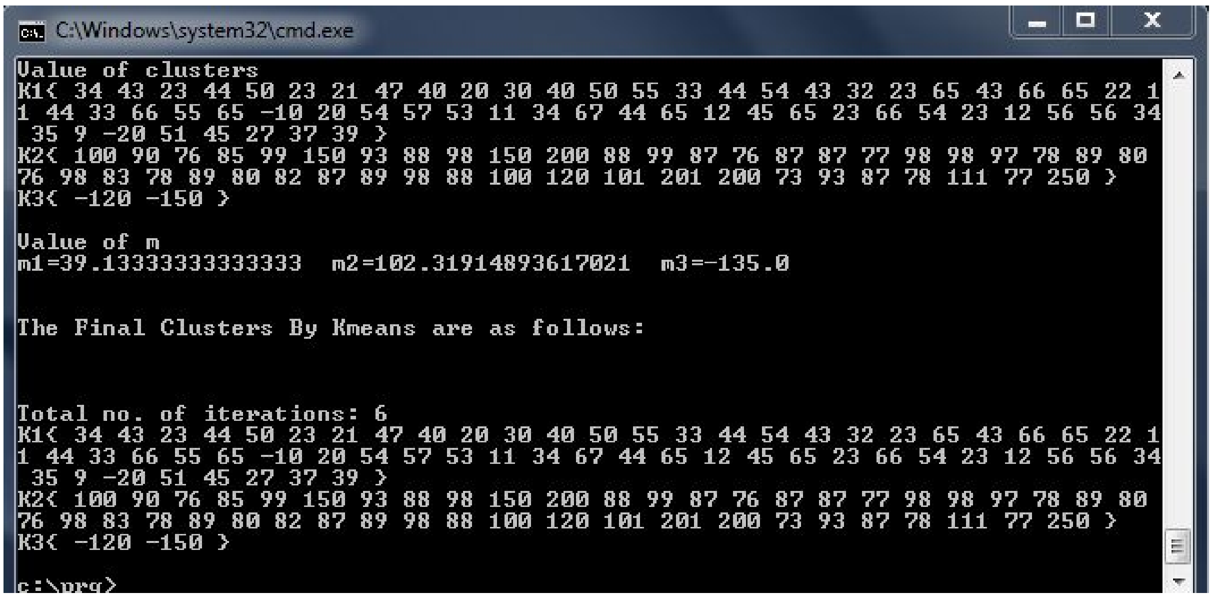
Clusters: 3

Dataset : 110 elements



C:\Windows\system32\cmd.exe - java KMEAN2withoutsort

```
51
80
82
87
89
98
88
100
120
101
201
200
45
73
27
37
93
39
87
78
111
77
250
Enter the number of clusters:
3
```



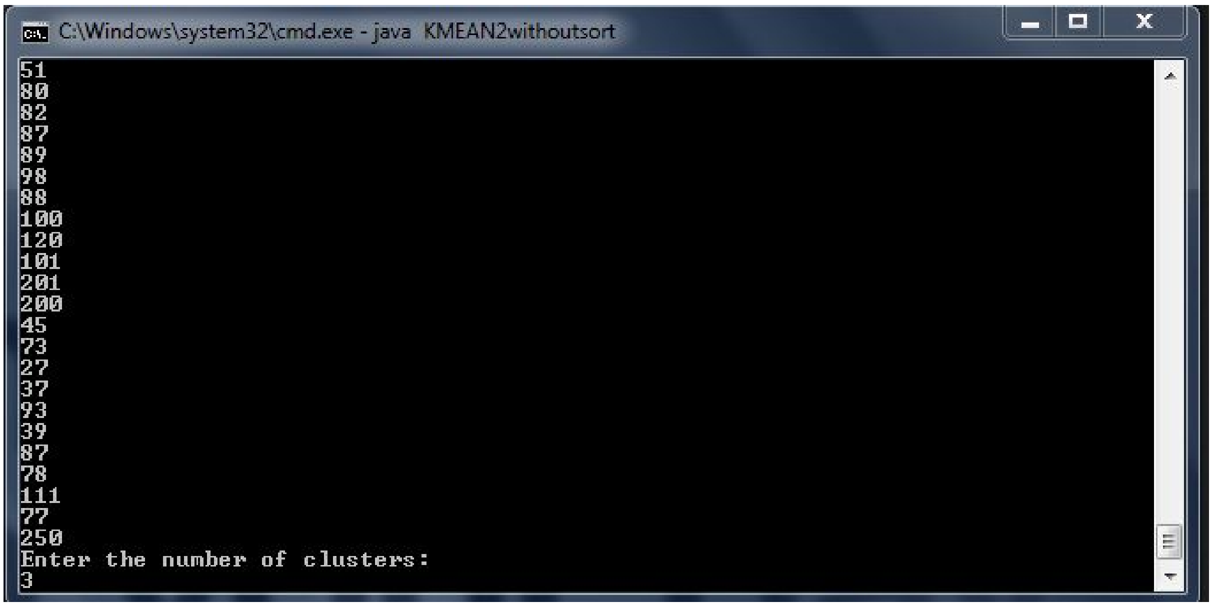
C:\Windows\system32\cmd.exe

```
Value of clusters
K1{ 34 43 23 44 50 23 21 47 40 20 30 40 50 55 33 44 54 43 32 23 65 43 66 65 22 1
1 44 33 66 55 65 -10 20 54 57 53 11 34 67 44 65 12 45 65 23 66 54 23 12 56 56 34
 35 9 -20 51 45 27 37 39 }
K2{ 100 90 76 85 99 150 93 88 98 150 200 88 99 87 76 87 87 77 98 98 97 78 89 80
76 98 83 78 89 80 82 87 89 98 88 100 120 101 201 200 73 93 87 78 111 77 250 }
K3{ -120 -150 }

Value of m
m1=39.13333333333333  m2=102.31914893617021  m3=-135.0


The Final Clusters By Kmeans are as follows:



Total no. of iterations: 6
K1{ 34 43 23 44 50 23 21 47 40 20 30 40 50 55 33 44 54 43 32 23 65 43 66 65 22 1
1 44 33 66 55 65 -10 20 54 57 53 11 34 67 44 65 12 45 65 23 66 54 23 12 56 56 34
 35 9 -20 51 45 27 37 39 }
K2{ 100 90 76 85 99 150 93 88 98 150 200 88 99 87 76 87 87 77 98 98 97 78 89 80
76 98 83 78 89 80 82 87 89 98 88 100 120 101 201 200 73 93 87 78 111 77 250 }
K3{ -120 -150 }

c:\prg>
```

K-Means with sort

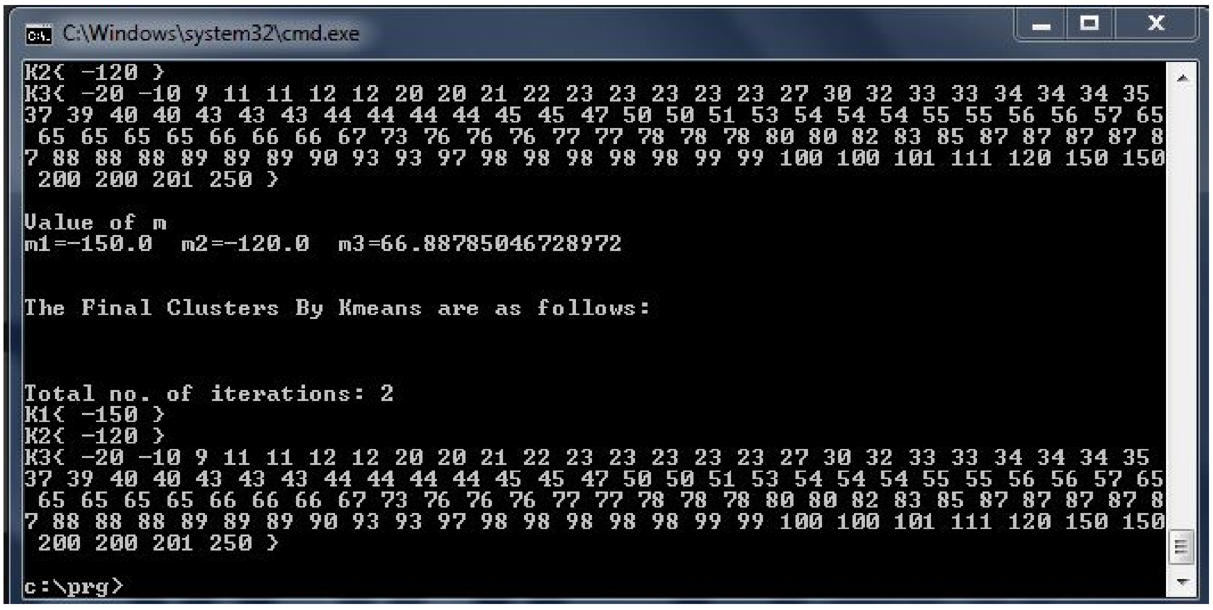
Clusters: 3

Dataset: 110 elements



C:\Windows\system32\cmd.exe - java KMEAN2withoutsort

```
51
80
82
87
89
98
88
100
120
101
201
200
45
73
27
37
93
39
87
78
111
77
250
Enter the number of clusters:
3
```



C:\Windows\system32\cmd.exe

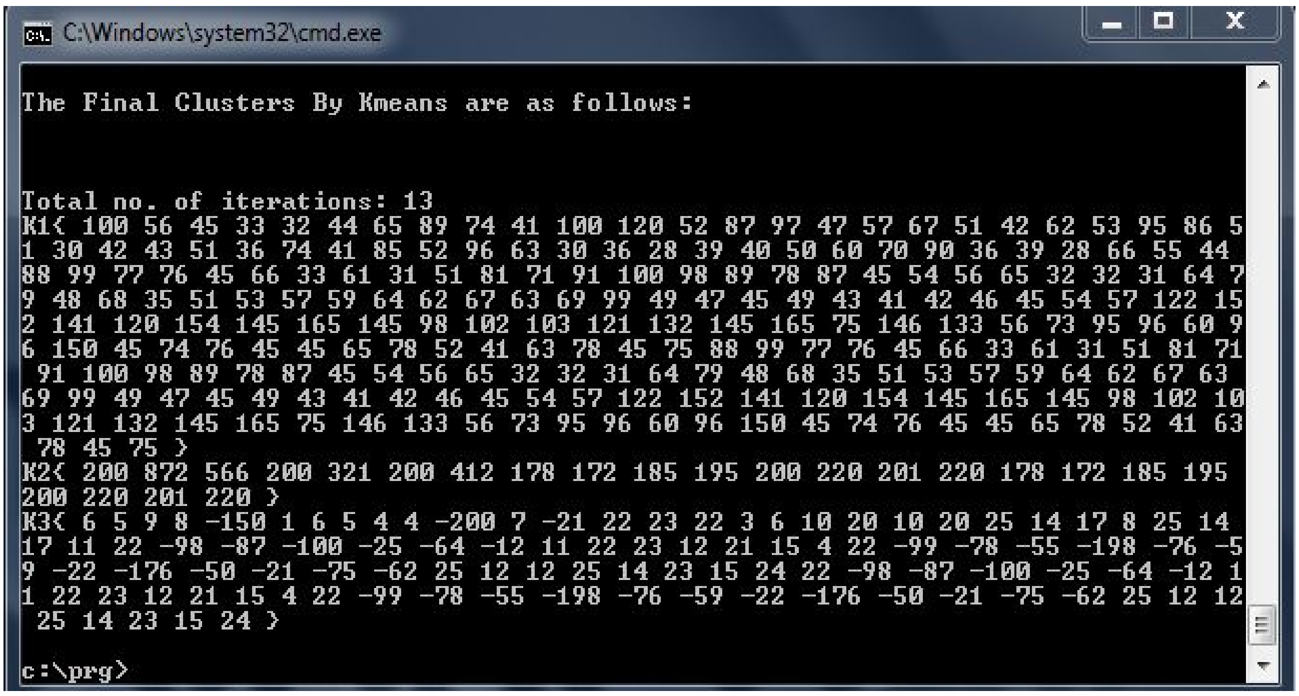
```
K2{ -120 }
K3{ -20 -10 9 11 11 12 12 20 20 21 22 23 23 23 23 23 27 30 32 33 33 34 34 34 35
37 39 40 40 43 43 43 44 44 44 44 45 45 47 50 50 51 53 54 54 54 55 55 56 56 57 65
 65 65 65 65 66 66 66 67 73 76 76 76 77 77 78 78 78 80 80 82 83 85 87 87 87 87 8
7 88 88 88 89 89 89 90 93 93 97 98 98 98 98 98 99 99 100 100 101 111 120 150 150
 200 200 201 250 }

Value of m
m1=-150.0  m2=-120.0  m3=66.88785046728972


The Final Clusters By Kmeans are as follows:



Total no. of iterations: 2
K1{ -150 }
K2{ -120 }
K3{ -20 -10 9 11 11 12 12 20 20 21 22 23 23 23 23 23 27 30 32 33 33 34 34 34 35
37 39 40 40 43 43 43 44 44 44 44 45 45 47 50 50 51 53 54 54 54 55 55 56 56 57 65
 65 65 65 65 66 66 66 67 73 76 76 76 77 77 78 78 78 80 80 82 83 85 87 87 87 87 8
7 88 88 88 89 89 89 90 93 93 97 98 98 98 98 98 99 99 100 100 101 111 120 150 150
 200 200 201 250 }

c:\prg>
```

Test case 3:

K-Means without sort

Clusters: 3

Dataset : 350 elements



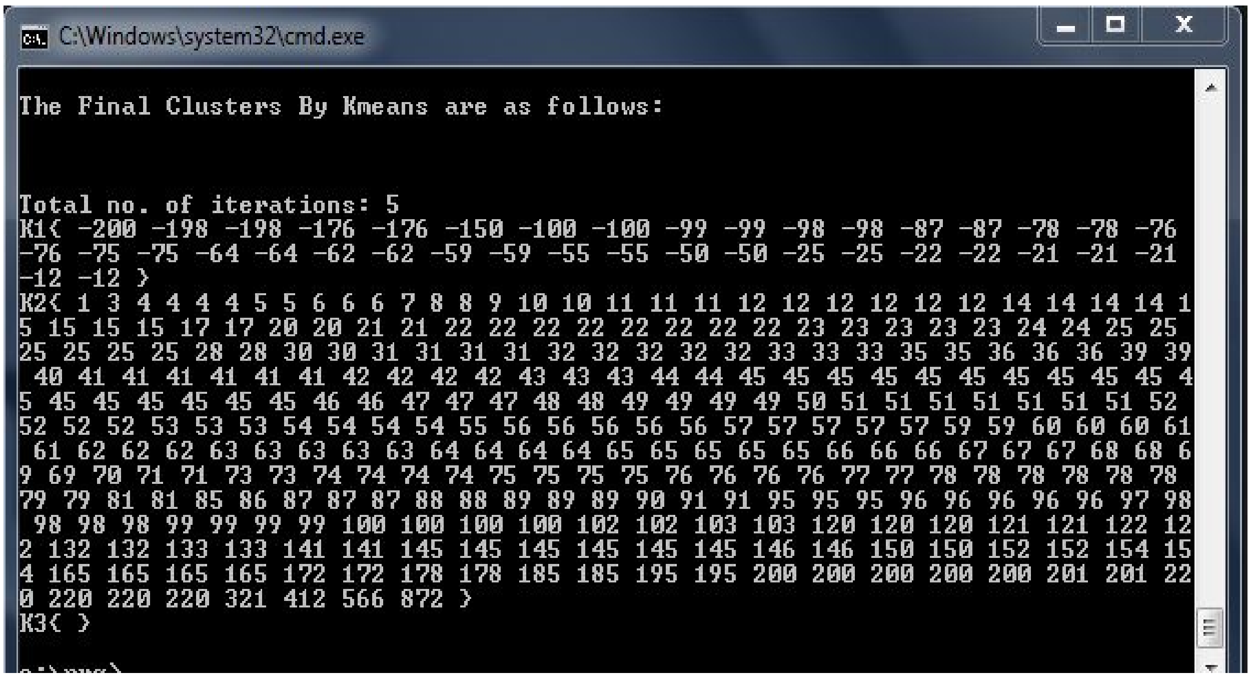
C:\Windows\system32\cmd.exe

```
The Final Clusters By Kmeans are as follows:



Total no. of iterations: 13
K1< 100 56 45 33 32 44 65 89 74 41 100 120 52 87 97 47 57 67 51 42 62 53 95 86 5
1 30 42 43 51 36 74 41 85 52 96 63 30 36 28 39 40 50 60 70 90 36 39 28 66 55 44
88 99 77 76 45 66 33 61 31 51 81 71 91 100 98 89 78 87 45 54 56 65 32 32 31 64 7
9 48 68 35 51 53 57 59 64 62 67 63 69 99 49 47 45 49 43 41 42 46 45 54 57 122 15
2 141 120 154 145 165 145 98 102 103 121 132 145 165 75 146 133 56 73 95 96 60 9
6 150 45 74 76 45 45 65 78 52 41 63 78 45 75 88 99 77 76 45 66 33 61 31 51 81 71
 91 100 98 89 78 87 45 54 56 65 32 32 31 64 79 48 68 35 51 53 57 59 64 62 67 63
69 99 49 47 45 49 43 41 42 46 45 54 57 122 152 141 120 154 145 165 145 98 102 10
3 121 132 145 165 75 146 133 56 73 95 96 60 96 150 45 74 76 45 45 65 78 52 41 63
 78 45 75 >
K2< 200 872 566 200 321 200 412 178 172 185 195 200 220 201 220 178 172 185 195
200 220 201 220 >
K3< 6 5 9 8 -150 1 6 5 4 4 -200 7 -21 22 23 22 3 6 10 20 10 20 25 14 17 8 25 14
17 11 22 -98 -87 -100 -25 -64 -12 11 22 23 12 21 15 4 22 -99 -78 -55 -198 -76 -5
9 -22 -176 -50 -21 -75 -62 25 12 12 25 14 23 15 24 22 -98 -87 -100 -25 -64 -12 1
1 22 23 12 21 15 4 22 -99 -78 -55 -198 -76 -59 -22 -176 -50 -21 -75 -62 25 12 12
 25 14 23 15 24 >

c:\prg>
```

K-Means with sort:

Clusters: 3

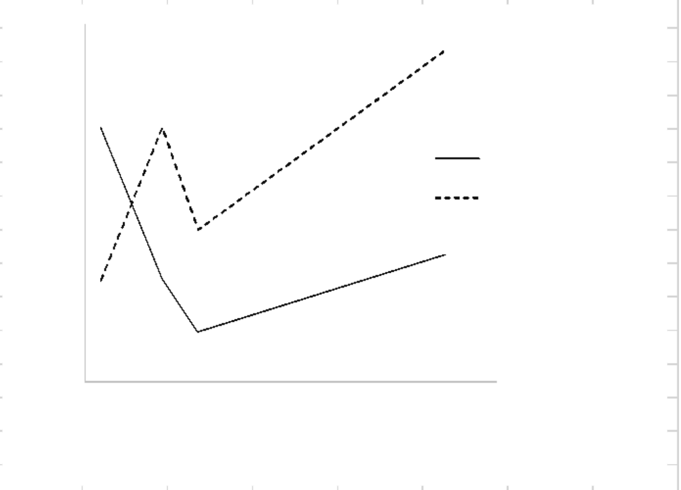
Dataset : 350 elements



```
The Final Clusters By Kmeans are as follows:



Total no. of iterations: 5
K1< -200 -198 -198 -176 -176 -150 -100 -100 -99 -99 -98 -98 -87 -87 -78 -78 -76
-76 -75 -75 -64 -64 -62 -62 -59 -59 -55 -55 -50 -50 -25 -25 -22 -22 -21 -21 -21
-12 -12 >
K2< 1 3 4 4 4 4 5 5 6 6 6 7 8 8 9 10 10 11 11 11 12 12 12 12 12 12 14 14 14 14 1
5 15 15 15 17 17 20 20 21 21 22 22 22 22 22 22 22 22 23 23 23 23 23 24 24 25 25
25 25 25 25 28 28 30 30 31 31 31 31 32 32 32 32 32 33 33 33 35 35 36 36 36 39 39
 40 41 41 41 41 41 41 42 42 42 42 43 43 43 44 44 45 45 45 45 45 45 45 45 45 45 4
5 45 45 45 45 45 45 46 46 47 47 47 48 48 49 49 49 49 50 51 51 51 51 51 51 51 52
52 52 52 53 53 53 54 54 54 54 55 56 56 56 56 56 57 57 57 57 57 59 59 60 60 60 61
 61 62 62 62 63 63 63 63 63 64 64 64 64 65 65 65 65 65 66 66 66 67 67 67 68 68 6
9 69 70 71 71 73 73 74 74 74 74 75 75 75 75 76 76 76 76 77 77 78 78 78 78 78 78
79 79 81 81 85 86 87 87 87 88 88 89 89 89 90 91 91 95 95 95 96 96 96 96 96 97 98
 98 98 98 99 99 99 99 100 100 100 100 102 102 103 103 120 120 120 121 121 122 12
2 132 132 133 133 141 141 145 145 145 145 145 145 146 146 150 150 152 152 154 15
4 165 165 165 165 172 172 178 178 185 185 195 195 200 200 200 200 200 201 201 22
0 220 220 220 321 412 566 872 >
K3< >
```

Difference between K-Means with sort(Enhanced K-mean) and K-Mean without sort:

Data Set Size	Enhanced k-mean	k-mean
15	10	4
75	4	10
110	2	6
350	5	13



Hence, for the less number of data set elements given to the unsorted K-Means, the iterations are very less. For example, for the given 15 elements, the iterations are 4. The same amount of data is given to the Sorted K-means (Merge sort) where the iterations are noted as 10. In the data mining process, the values are always taken in the upper bound. Therefore, with this, the traditional k-means is proved as best for the lower bound values.

After using the sorting technique with the K-means, which is called as Enhanced K-means. with the high amount of dataset given to the algorithm, like 110 elements, the iterations that are obtained to the traditional k-means is 6 and the same dataset when given to the sorting K-means , the iterations obtained are 2. Hence with this, the algorithm proved that the upper bounds are chosen as best in the clustering process. Also, the enhanced k-mean technique worked which gave the higher efficiency because of the usage of sorting algorithm which once again proved that such algorithms gives or improves optimization

CONCLUSION:

Further improvement can be done with the Greedy approach. A greedy algorithm is an algorithm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum. Henceforth, combining the greedy algorithm and K-means algorithm which can be obtained as G-means will give the best results for the large amount data such as for high health care datasets. Already, G-means approach and F-scores approach together have proved that these techniques are better than the existing k-means. Futuristic improvement can be made even higher with the best time complexity when the sorting technique is used before G-means technique.