MEDIAN IMPLEMENTATIONS

Approach 1: (in file *gossip_median.erl*)

Initialization phase:

1. Calculate median (Mi), sum (Si) and count (Ci) of numbers in fragment at each node.
2. Construct a new fragment locally at each node as follows:
    1. The fragment will contain Mi repetitively for (Ci/2 + 1) times, that is one more than half the number of values in the original fragment.
    2. The remaining (Ci/2 -1) numbers in the fragment are populated with the value

$$\text{Bal\_Sum} = \frac{Si - (Mi*(Ci/2 +1))}{Ci/2-1}$$

Thus the new fragment at node I whose fragment size is 7 will be

[Mi,Mi,Mi, Mi, Bal_Sum,Bal_Sum,Bal_Sum]

Step (a) ensures that Mi still remains the median of the newly constructed fragment. Step (b) ensures that the sum of all numbers in the fragment is preserved. The count of numbers in the fragment is evidently preserved too.

Gossip phase:

When two nodes converse they exchange their locally computed medians, sums and counts.

Upon receiving a neighbor's (node j's) fragment, node i will do :

1. Merge it's fragment with node j's fragment (which it can construct from Mj, Sj and Cj as per the steps in the initialization phase).
2. Compute

1. new median Mi from the merged fragment.
2. Compute new local sum Si= Si + Sj
3. Compute new count of elements in fragment Ci = Ci + Cj
4. Compute new fragment using steps (2.a) and (2.b) in the initialization step above.

Thus at any given instant a node contributes only the number of terms it has seen till that instant whose sum is the net sum that it has seen till then. Thus the sum as well as count of numbers it has seen till then is preserved. After log(n) rounds every node will attempt to converge on a median; also the sum of all fragments and count of numbers in all fragments is preserved.

Miscellaneous: (in file *median.erl*)

In this file, is a crude approach to test actual deviation from the median, where we do not conserve the sum, but conserve only the count and eliminate the variations in the numbers in a fragment.
In this approach we follow the same approach as above except while constructing the new fragment we replicate median Mi for Ci times instead of (Ci/2 +1) times.
It is obvious that this method will not converge towards the median value but it was done as a test to see the degree to which the median values deviate from the actual value when the size of fragments for each of the

nodes is constant versus when the size varies. From this we concluded that although the value Ci is important, more important is conserving the range of values in a given fragment as well atleast the true frequency of the local median.

Approach 2: (in file *medianminmax.erl*)

Based on the observations above we tried this approach which preserves the count of numbers (Ci) seen locally. In addition to computing the local median (Mi) we also compute four more values.

1. The local minimum value (Mini)
2. The local maximum value (Maxi)
3. the mid point on the number scale between Mini and Mi

   LMidi = (Mini + Mi)/2
4. the mid point on the number scale between Mi and Maxi

   RMidi = (Mi + Maxi)/2

The new fragment is constructed by repeating both Lmidi and Rmidi (Ci-3)/2 times and appending Mi, Mini and Maxi once.

Thus for a new fragment for Ci=7, the fragment will be

[Mini, LMidi, LMidi, Mi, RMidi, RMidi, Maxi]

This ensures that Mi is the median of the newly constructed fragment. We also conserve the minimum and maximum thus allowing the full range to be evaluated upon and restricting the movement of the calculated median values in further gossip rounds by adding the left and right mid points with maximum frequency to the newly constructed fragments.

Observations:

Intuitively the third approach works the best. But it is only the best of the worst. Both the methods achieve sufficient convergence for small to medium network sizes but not for large ones. They also show random behaviour when the fragments sizes at the nodes vary randomly in a large range, say from sizes 1 to 100. None of the approaches achieve sufficient accuracy consistently due to the loss of information about the frequencies of each individual numbers across the system. This information loss is aggravated in case of large variance in the fragment sizes. We tried to conserve the sum and count with a notion that in presence of these variances the median will be bounded by the average of numbers across the system. But this proves futile once the distribution of the numbers on the number scale have large void regions.
In the initialization code in all the files you can change the variance range for the fragment sizes across the network by altering the highlighted part in the code shown below to observe the behaviour as discussed above:

```
init_the_dhondus(N) ->
        %Values = lists:map(fun(_) -> random:uniform(100) end, lists:seq(0,N-1)),
        Fragments = lists:map(fun(_) -> lists:map(fun(_) -> random:uniform(1000)+0.5 end,
                        lists:seq(1, 50)) end, lists:seq(0,N-1)),
…....
```