For my approach to Color Quantization using clustering. I have chosen a method inspired a bit by k-means, median cut, kd trees, and octrees in a sense. I believe this is probably mean cut.

The method creates k boxes where k is the desired number of values. The first box is composed of the entire set of data.

For each box:

* Find the average value across the box
* Find the squared error for each dimension across that box

Next:

* Find the box with the most error to split into 2 separate boxes
  + The total error of a box is equal to the squared error for each dimension summed together.

Finally:

* Split the box into 2 boxes across the dimension with the most error

Repeat until you have k boxes, or you cannot split anymore.

After you have k boxes or cannot split anymore, find the average across each box to get k values.

This method is quite fast in comparison to Median Cut and produces results that either rival or look better than GIMPs implementation of octrees and is simple to implement.

The algorithm in this case take n unique values and performs the operations however, the algorithm does not depend on unique values. The use of unique values may contribute to the palette being more optimal with lower values of k than other approaches.

The running time of this algorithm is approximately O( N\*Log(K) ) where N is the input size, K is the desired output size, and Log is base 2.

More accurately, it is approximately equal to 3N\*Log(K) + (K(K+1))/2

This happens when the algorithm continuously picks the “largest” box. The size of each box is still being effectively cut in half so it still is relatively fast.

The running time of Median Cut is approximately O( N\*Log(N)\*Log(K) ) where N is the input size, K is the desired output size, and Log is base 2. This is due to the sorting that is necessary to do Median Cut but it could be reduced if the sorting was faster.

The speed of the algorithm can be improved by reusing values. The average of a box along with the errors do not need to be recalculated. Only new boxes must have the average and errors calculated. This in turn means that at the end of the algorithm, all the averages for each box has been calculated and can just be read off instead of resolving. These benefits speed quite a bit since for each step, you are at most calculating 2 averages and 2 error vectors.

Lastly, storing the boxes is done by storing in a ArrayList however, an array would do just fine if not better. Instead of removing a box from the list, the data in that box is replaced with one of the splits and the other split can be added as a new box at the end of the list. Those 2 boxes are set to be updated in the next pass. The order of the boxes does not matter in any way so we can avoid reordering the boxes.