**Process Followed...**

**1. Dealing with large file size**

* I stripped the file DnaseIClustered.fa, using stripfile.py, in order to remove all chromosome lines, and to replace all instances of multiple “N”s with a single “N”
  + Problems encountered:
    - Initially I didn't realise that the chromosomes lines were separate, so I just let them run into each other.
    - Later, I tried to do it so that if a line was just a single “N”, it would run into the next line. Unfortunately, By accident this also made it so that ANY line ending in an N would run on into the next line. It seemed too much effort to correct this problem.
  + I eventually reached a compression rate of 2/3, from 420MB to 280MB
* Using splitfile.py, I Split the file into 4 separate files, each with the same number of lines.  
  This was done in order to facilitate later multi-processing.

**2. Creating Kmer Lists**

* Originally, using createkmerlists.py, I created a file with all 8mers, starting with an “A” or a “C”.
  + Naively, I expected that this would also include all 8mers starting with a “T” or a “G”  
    But, of course, this is not the case, as we take **reverse** complements. See eg. (“TTTTCTA”)
  + Later on I add all Kmers Starting with a G and ending in an A or C, as well as all Kmers starting with a T and ending in an A.
* For multiprocessing purposes, I copied the file four times, one for each concurrent process

**3. Counting the kmers...**

* I used the countkmersx.py script to count the number of lines in which each kmer or its reverse complement occurs.
  + There were a few false starts, but this was due to problems with the way I had split the file
  + It seems that the preferred answer would be the number of kmers, not the number of lines in which they appear. This is something I can revise.
* Due to the multi-processing, 4 output files were produced. These had to be combined, into a final list showing “8mer \t RC \t Count \n” using combinefiles.py
* I later had to repeat the process for the [“G”,”T”] kmers above. To save time I did this separately and used bash's cat to combine it with the [“A”,”C”] kmer combined file.
* The original [“A”,”C”] process counted 32768 kmers in roughly 2.5 hours.
* The later [“G”,”T”] process counted 12288 kmers in exactly 3533.2 seconds, so 7 seconds short of an hour.

**5. Creating the main file**

* Firstly, I used getline from linecache, and used simple mathematics to discover the exact line for a given kmer (or it's revcomp if this was less) This would then print the Kmer, its RC, and the count, and return the count...
* This obviously had to be revised later to include the [“G”,”T”] set...
* I developed a recursive algorithm for counting dotted Kmers.
* The main file is KmerCount.py, and the function to call in it is also KmerCount, taking a single 8mer with as many gaps as you like as an argument.

**6.Problems**

* The only final problem is that in the final list of Kmer, Rc and Count, there are still some Kmers which have repeated lines....
  + For example “AAAGTATT AATACAAA 356789”  
    and “ AATACAAA AAAGTATT 356789”

will both be in the file...

* + This doesn't cause a bug in the program, but removing them might make the list file size probably about 1/8th or so smaller, although it is only 1MB, and changing things would complicate my line counting algorithms.

**7. Going Forward:**

* My plans are to:
  + Implement a GUI for KmerCount, just to Learn how to...
  + Change the counting process in countkmersx.py so that it counts number of instances of the kmer rather than number of lines...
  + Compare my process with other algorithms for efficiency etc.
  + Consider automating the entire file creation process so that I can easily make files for 9mers, 10mers and so on. Depends on the efficiency of my algorithm
    - KmerCount is fairly open ended already... if it has a list file for Kmers, it should be able to work on it – all that may need to be changed is the line counting algorith, but not much.