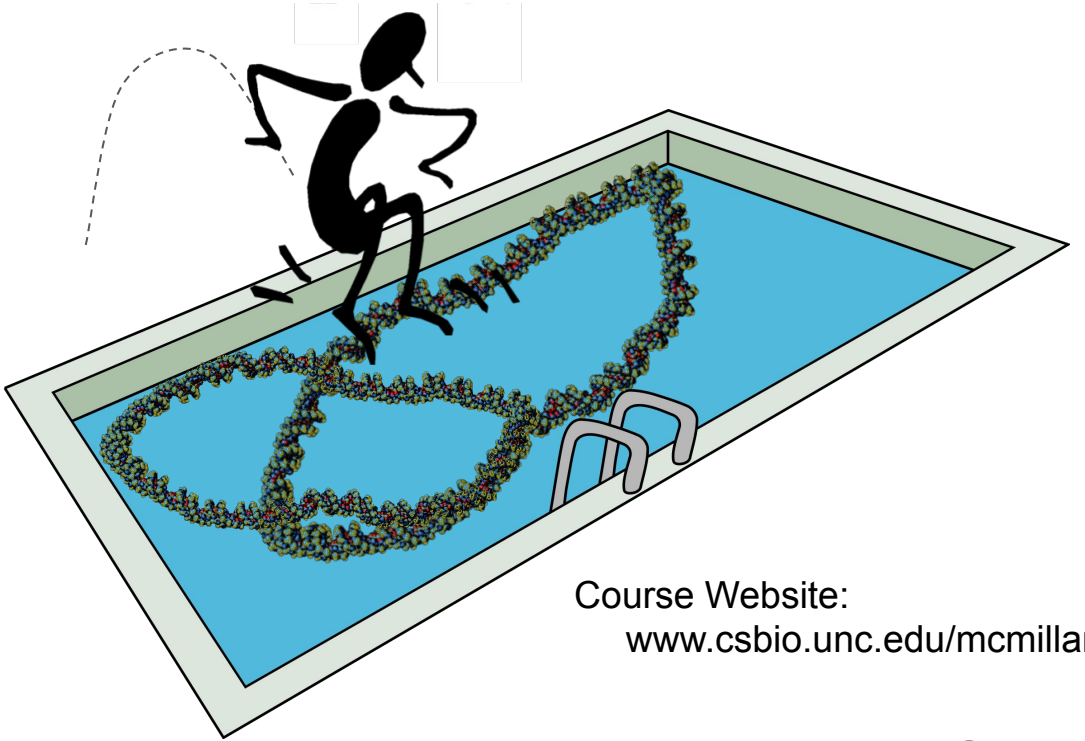
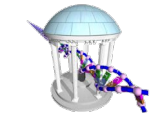
Comp 555 - BioAlgorithms - Spring 2020



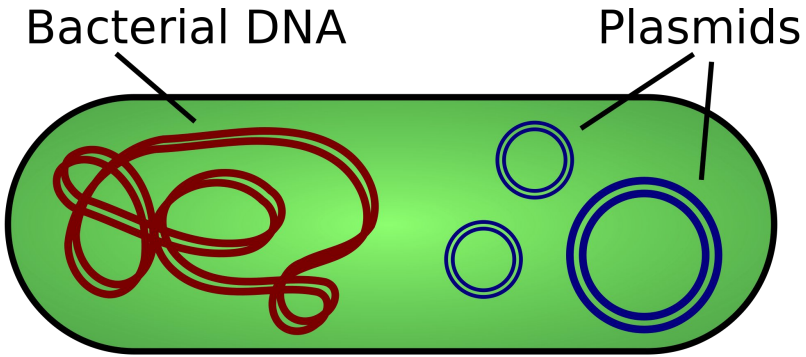
Course Website:
www.csbio.unc.edu/mcmillan/index.py?run=Courses.Comp555S20

Jumping into Genomes

A simple genome



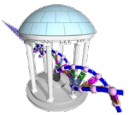
Let's first consider a Bacterial genome.



Characteristics of Bacterial DNA

- A "circular" primary chromosome (a few million bases) with essential genes
- Smaller chromosomes or circular plasmids (10-100K bases) with a few additional genes
- There can be multiple plasmid sequences with varible numbers of copies

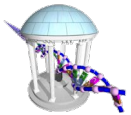
FASTA file format



FASTA is a common format for biological sequences

- **Each sequence is preceeded by a header line that starts with ‘>’**
- **Followed by multiple lines of sequence data from a standard alphabet**
  - For DNA, alphabet = “ACGT”
  - For Proteins, alphabet = “ACDEFGHIKLMNOPQRSTUVWY”
- **A sequence ends when either another header line is reached or the end-of-file**
- **Multiple sequences per file are allowed**
- **Sequences are 1-indexed rather than 0-indexed!**

An Example



```
In [1]: !head data/VibrioCholerae.fa
```

```
>gi|146313784|gb|CP000626.1| Vibrio cholerae O395 chromosome 1, complete genome
ACAATGAGGTCACTATGTTCGAGCTCTTCAAACCGGCTGCGCATACGCAGCGGCTGCCATCCGATAAGGT
GGACAGCGTCTATTCACGCCTTCGTTGGCAACTTTTCATCGGTATTTTTGTTGGCTATGCAGGCTACTAT
TTGGTTCGTAAGAACTTTAGCTTGGCAATGCCTTACCTGATTGAACAAGGCTTTAGTCGTGGCGATCTGG
GTGTGGCTCTCGGTGCGGTTTCAATCGCGTATGGTCTGTCTAAATTTTTGATGGGGAACGTCTCTGACCG
TTCTAACCCGCGCTACTTTCTGAGTGCAGGTCTACTCCTTTCGGCACTAGTGATGTTCTGCTTCGGCTTT
ATGCCATGGGCAACGGGCAGCATTACTGCGATGTTTATTCTGCTGTTCTTAAACGGCTGGTTCCAAGGCA
TGGGTTGGCCTGCTTGTGGCCGTACTATGGTGCACTGGTGGTCACGCAAAGAGCGTGGTGAGATTGTTTC
GGTCTGGAACGTCGCTCACAACGTCGGTGGTGGTTTGATTGGCCCCATTTTCCTGCTCGGCCTATGGATG
TTTAACGATGATTGGCGCACGGCCTTCTATGTCCCCGCTTTCTTTGCGGTGCTGGTTGCCGTATTTACTT
```

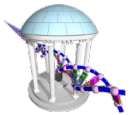

```
In [2]: !tail data/VibrioCholerae.fa
```

```
AAGTGGTGCCGGCTGCCGGAATCGAACTGGCGACCTACTGATTACAAGTCAGTTGCTCTACCTACTGAGC
TAAGCCGGCACACGTAACCTTTGCTGTTTGTGTCTTACACCAACAATCTAAAATTCGTGGTGCCCGGAGG
CGGAATCGAACCACCGACACGAGGATTTTCAATCCTCTGCTCTACCGACTGAGCTATCCGGGCAACGGAG
CGCTATTAAACGGATTTTCCCTTTCCCCGTCAACCTGTTTTTTGAAATATTTCGAAAAATCAGTTTGATT
GCCGTTATTTTCAGCAAACGGCGGGCTTTTTGTTATCCCGCGTTAAATTCCTTCTTAAATTTGGTCACTT
TTTCCAGATAACGACGCGCTTCCGCATTCGGATGTTTTTTGGTTAACGCCCAATACACTTGGTTAGGTTG
CAGGGCATTAAGGTCACGCATGGCGCGTTGACGATCACTGCTAAAGGTACTCAACACTCCGCCAGTGCCG
CCGTTATAGGCAGAAATCATGCTGTATTCGAGAGATGTGGGGTGGCGAACCTCTTTCAAATAGCGATTTT
TCAGGATGTAAAAATAGGCCGTACCCGTATCAATGTTGTTTTCTGGGTTAAACAGATACTCGGGGCTGG
```


```
In [3]: !wc data/VibrioCholerae.fa
```

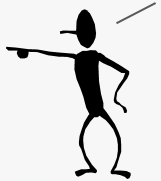
```
  59038   59050 4191517 data/VibrioCholerae.fa
```

A little code for reading FASTA



In [5]:
```
import gzip

def loadFasta(filename):
    """ Parses a classically formatted and possibly
        compressed FASTA file into two lists. One of
        headers and a second list of sequences.
        The ith index of each list correspond."""
    if (filename.endswith(".gz")):
        fp = gzip.open(filename, 'r')
    else:
        fp = open(filename, 'r')
    # split at headers
    data = fp.read().split('>')
    fp.close()
    # ignore whatever appears before the 1st header
    data.pop(0)
    headers = []
    sequences = []
    for sequence in data:
        lines = sequence.split('\n')
        headers.append(lines.pop(0))
        # add an extra "+" to make string "1-referenced"
        sequences.append('+' + ''.join(lines))
    return (headers, sequences)
```



"splits" the file at every header line. Then each of those sections is split at each return '\n'. "pop()" is used to remove the header line. The sequence is formed by joining together the remaining lines of sequences. A "+" is added to the front to give the string an offset of 1.

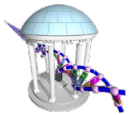
In [6]:
```
header, seq = loadFasta("data/VibrioCholerae.fa")

for i in range(len(header)):
    print(header[i])
    print(len(seq[i])-1, "bases", seq[i][:30], "...", seq[i][-30:])
    print()
```

```
gi|146313784|gb|CP000626.1| Vibrio cholerae O395 chromosome 1, complete genome
1108250 bases +ACAATGAGGTCACTATGTTCGAGCTCTTC ... CCGATAGTAGAGGTTTATACCATCGCAAAA

gi|147673035|ref|NC_009457.1| Vibrio cholerae O395 chromosome 2, complete genome
3024069 bases +GTTCGCCAGAGCGGTTTTTGACTAGCTTG ... TTTCTGGGTTAAACAGATACTCGGGGCTGG
```

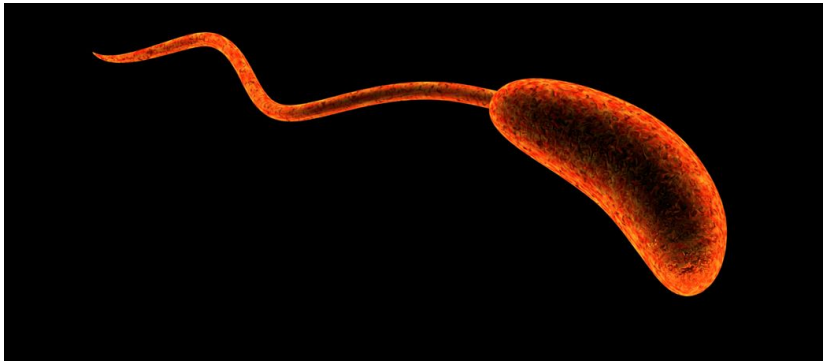
Vibrio Cholerae



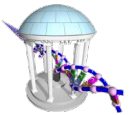
Aquatic microorganism that causes Cholera

An abundant marine and freshwater bacterium that causes Cholera. Vibrio can affect shellfish, finfish, and other marine animals and a number of species are pathogenic for humans. Vibrio cholerae colonizes the mucosal surface of the small intestines of humans where it causes, a severe and sudden onset diarrheal disease.

One famous outbreak was traced to a contaminated well in London in 1854 by John Snow. Epidemics, which can occur with extreme rapidity, are often associated with conditions of poor sanitation. The disease is highly lethal if untreated. Millions have died over the centuries incuding seven major pandemics between 1817 and today. Six were attributed to the classical biotype, while the 7th, which started in 1961, is associated with this El Tor biotype.



Let's take a minute to explore



Genome sequences are best understood by examining subsequences

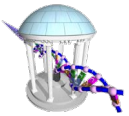
Often we examine subsequences of length *k*, called *k-mers*.

The statistics and patterns of k-mers can shed light on a genome's organization and local function.

Two simple rules to consider:

1) **There are 4^k possible DNA k-mers**
2) **A linear sequence of length N has *N - k + 1* k-mers**
   A circular sequence of length N has *N* k-mers

Genome “k-mer” statistics



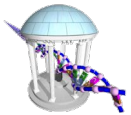
In [21]:

```
def kmerCounts(seq, k):
    kmerDict = {}
    for i in range(1,len(seq)-k+1):
        kmer = seq[i:i+k]
        kmerDict[kmer] = kmerDict.get(kmer,0) + 1
    return kmerDict

print('  k     k-mers              4^k      N-k+1          missing   repeated')
for k in range(3,25):
    kmers = kmerCounts(seq[0], k)
    print("%3d %10d %16d %10d %16d %10d" % (k, len(kmers), 4**k, (len(seq[0])-1)-k+1, 4**k-len(kmers), (len(seq[0
```

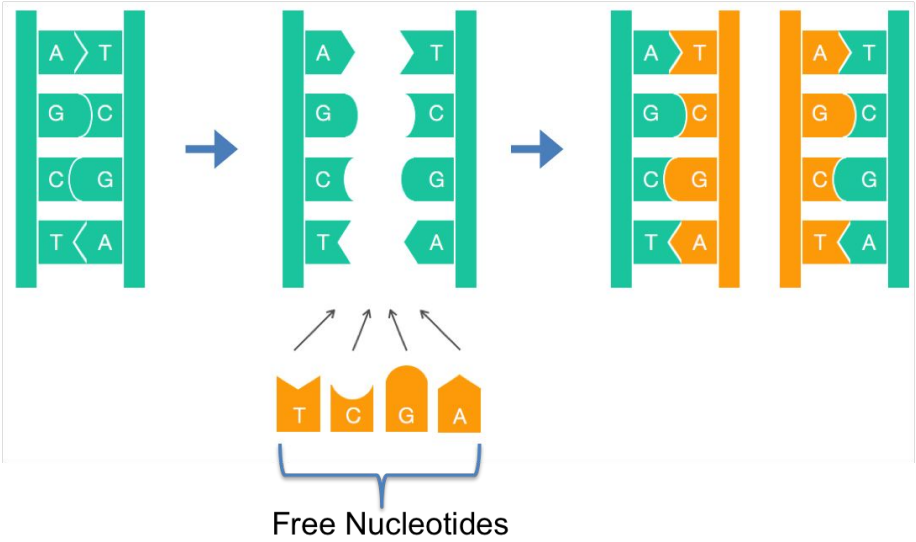
k	k-mers	4^k	N-k+1	missing	repeated
3	64	64	1108248	0	1108184
4	256	256	1108247	0	1107991
5	1024	1024	1108246	0	1107222
6	4096	4096	1108245	0	1104149
7	16382	16384	1108244	2	1091862
8	65099	65536	1108243	437	1043144
9	234316	262144	1108242	27828	873926
10	571913	1048576	1108241	476663	536328
11	870755	4194304	1108240	3323549	237485
12	1009883	16777216	1108239	15767333	98356
13	1056503	67108864	1108238	66052361	51735
14	1070862	268435456	1108237	267364594	37375
15	1075606	1073741824	1108236	1072666218	32630
16	1077604	4294967296	1108235	4293889692	30631
17	1078784	17179869184	1108234	17178790400	29450
18	1079674	68719476736	1108233	68718397062	28559
19	1080421	274877906944	1108232	274876826523	27811
20	1081116	1099511627776	1108231	1099510546660	27115
21	1081776	4398046511104	1108230	4398045429328	26454
22	1082397	17592186044416	1108229	17592184962019	25832
23	1082990	70368744177664	1108228	70368743094674	25238
24	1083559	281474976710656	1108227	281474975627097	24668

“Functional” Genome Sequences

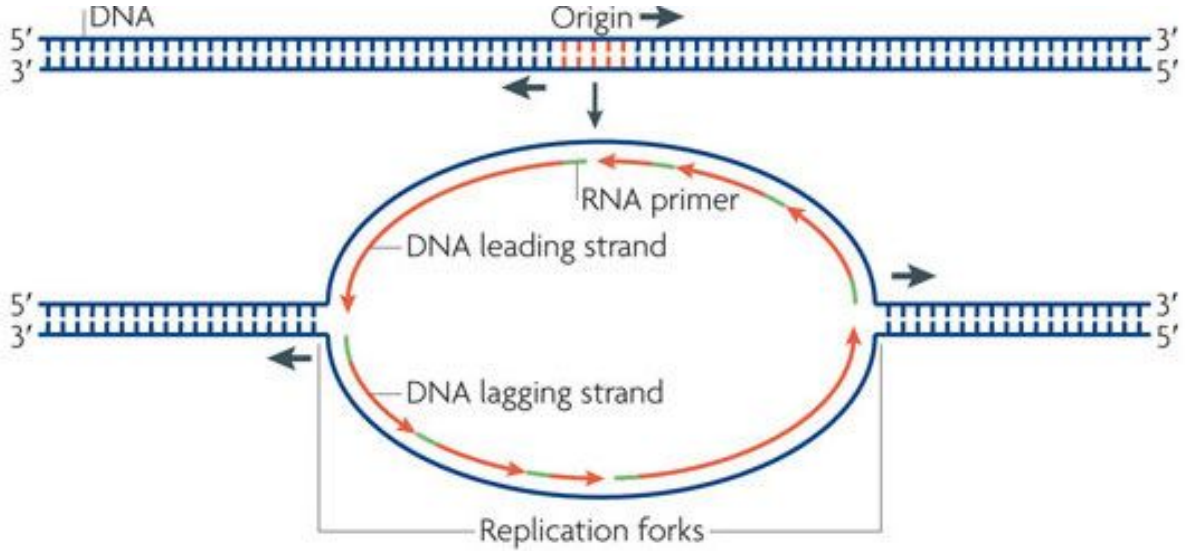


Life ≡ Reproduction ≡ Replicating a Genome

One of the most incredible things about DNA is that it provides instructions for replicating itself. Today, we consider how the replication process initiates.

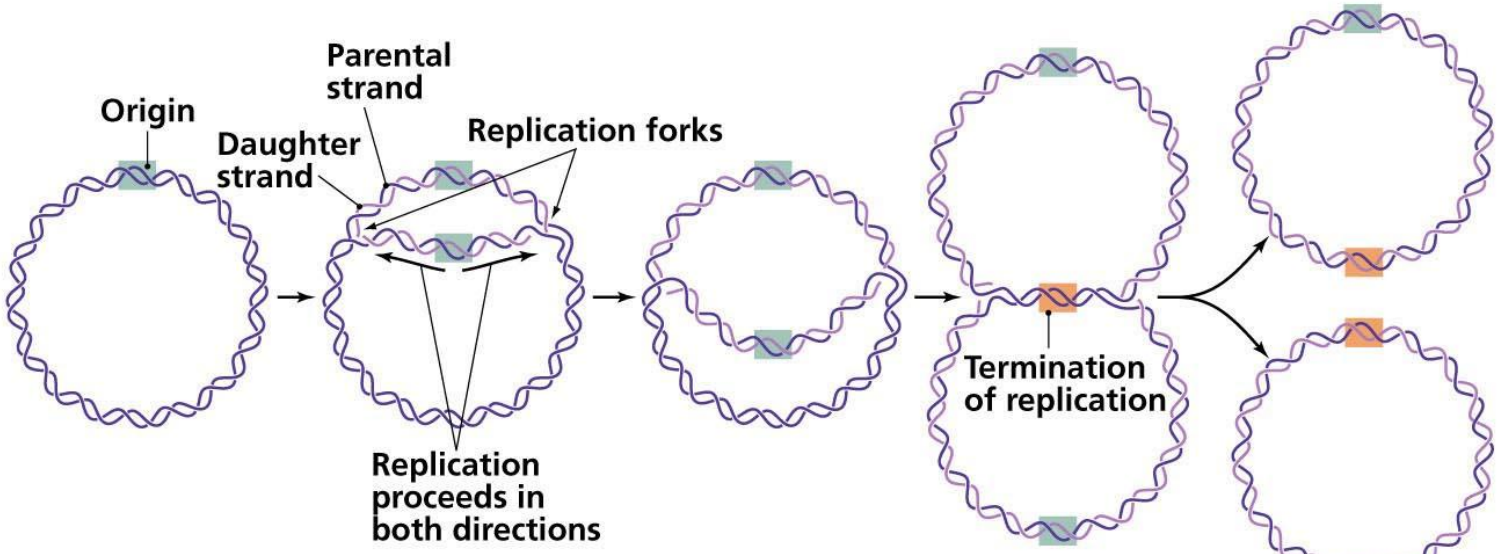


Where Does Replication Begin?



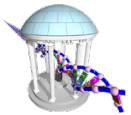
The DNA replication process begins reliably at a regions of the genome called the *origins of replication* or *oriC*. Today we explore the sequence properties of these regions to gain insight into how they might be identified?

A cartoon of the DNA replication process



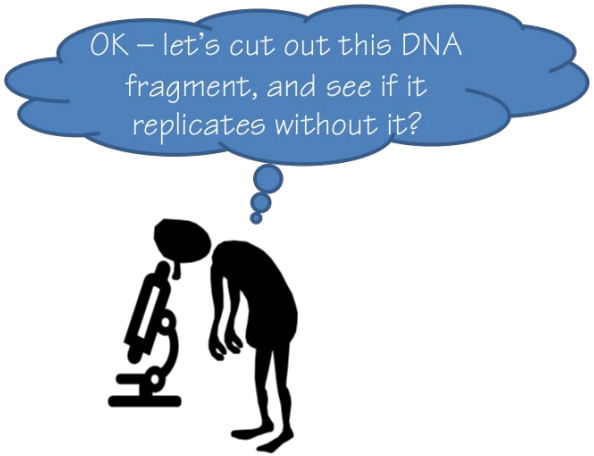
We seek to find the DNA sequence pattern at the point of origin, which is consistent.

The *oriC* finding Problem



Given a genome, find its oriC region or regions

Wet lab Approach:

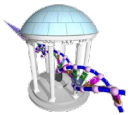


Advantage: You can start immediately
Disadvantage: It can take a long time

Computational Approach:

Advantage: It can be fast, and general
Disadvantage: Problem is not adequately specified

Let's look at an example *oriC*

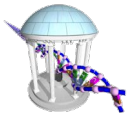


The replication origin of Vibrio Cholerae:

```
atcaatgatcaacgtaagcttctaagcatgatcaaggtgctcacacagtttatccacaac
ctgagtggatgacatcaagataggtcgttgtatctccttcctctcgtactctcatgacca
cggaaagatgatcaagagaggatgatttcttggccatatcgcaatgaatacttgtgactt
gtgcttccaattgacatcttcagcgccatattgcgctggccaaggtgacggagcgggatt
acgaaagcatgatcatggctgttgttctgtttatcttgttttgactgagacttgttagga
tagacggtttttcatcactgactagccaaagccttactctgcctgacatcgaccgtaaat
tgataatgaatttacatgcttccgcgacgatttacctcttgatcatcgatccgattgaag
atcttcaattgttaattctcttgcctcgactcatagccatgatgagctcttgatcatgtt
tccttaaccctctatttttacggaagaatgatcaagctgctgctcttgatcatcgtttc
```

Is there some pattern which might help us to develop an algorithm?

Where is it?



From before, seq[0], is chromosome 1 from our FASTA file.

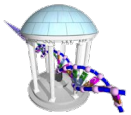
Here we print a 540 base region of the genome after 151,887, known to be near *oriR*. See any patterns?

In [25]:

```
genome = seq[0]
print("oriC:")
OriCStart = 151887
oriC = genome[OriCStart:OriCStart+540]
for i in range(9):
    print("    %s" % oriC[60*i:60*(i+1)].lower())
```

```
oriC:
    atcaatgatcaacgtaagcttctaagcatgatcaaggtgctcacacagtttatccacaac
    ctgagtggatgacatcaagataggtcgttgtatctccttcctctcgtactctcatgacca
    cggaaagatgatcaagagaggatgatttcttggccatatcgcaatgaatacttgtgactt
    gtgcttccaattgacatcttcagcgccatattgcgctggccaaggtgacggagcgggatt
    acgaaagcatgatcatggctgttgttctgtttatcttgttttgactgagacttgttagga
    tagacggtttttcatcactgactagccaaagccttactctgcctgacatcgaccgtaaat
    tgataatgaatttacatgcttccgcgacgatttacctcttgatcatcgatccgattgaag
    atcttcaattgttaattctcttgcctcgactcatagccatgatgagctcttgatcatgtt
    tccttaaccctctatttttttacggaagaatgatcaagctgctgctcttgatcatcgtttc
```

How to Look for Interesting Patterns



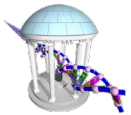
- So let's look at our example *oriC* region to see if we can find any interesting patterns
- Still not sure what "interesting" means yet
- So let's consider every pattern of a given length, k

A new *well-specified* problem: Find the frequency of all subsequences of length k, k-mers

```
atcaatgatcaacgtaagcttctaagcatgatcaaggtgctcacacagtttatccacaac
atca      caacg      ttctaa      atcaagg      acacagtt
 tcaa      aacgt      tctaag      tcaaggt      cacagttt
  caat      acgta      ctaagc      caaggtg      acagttta
   aatg      cgtaa      taagca      aaggtgc      cagtttat
    atga      gtaag      aagcat      aggtgct      agtttatc
     tgat      taagc      agcatg      ggtgctc      gtttatcc
   4-mers    5-mers     6-mers       7-mers       8-mers
```

- Let's count the occurence of every k-mer in the sequence, given a value for k.

Example k-mer counts



This genome example from before was a little unwieldy. Let's look at some smaller examples.

In [26]:
```
print(kmerCounts("TAGACAT",3))
print(kmerCounts("missmississippi",3))
```

```
{'AGA': 1, 'GAC': 1, 'ACA': 1, 'CAT': 1}
{'iss': 3, 'ssm': 1, 'smi': 1, 'mis': 1, 'ssi': 2, 'sis': 1, 'sip': 1, 'ipp': 1, 'ppi': 1}
```

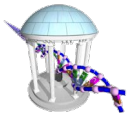
Now lets look at a k-mer counts for a range of k-mers sizes in the given oriC region

In [32]:
```
def mostFreqKmer(start, end, sequence):
    for k in range(start,end):
        kmerStats = kmerCounts(sequence,k)
        kmerOrder = sorted(kmerStats, reverse=True, key=kmerStats.get)
        mostFreq = [(kmer, kmerStats[kmer]) for kmer in kmerOrder[0:6]]
        print(k, mostFreq)

mostFreqKmer(1,10,oriC)
```

```
1 [('T', 174), ('A', 135), ('C', 122), ('G', 108)]
2 [('TT', 55), ('AT', 53), ('TC', 48), ('TG', 47), ('GA', 47), ('CT', 44)]
3 [('TGA', 25), ('GAT', 21), ('ATC', 20), ('TCA', 17), ('CTT', 17), ('TTG', 17)]
4 [('ATGA', 12), ('TGAT', 11), ('GATC', 10), ('ATCA', 10), ('CTTG', 9), ('TGAC', 8)]
5 [('TGATC', 8), ('GATCA', 8), ('ATGAT', 7), ('TCTTG', 6), ('ATCAA', 5), ('AATGA', 4)]
6 [('TGATCA', 8), ('ATGATC', 5), ('GATCAA', 4), ('ATCAAG', 4), ('GATCAT', 4), ('CTCTTG', 4)]
7 [('ATGATCA', 5), ('TGATCAA', 4), ('TGATCAT', 4), ('GATCAAG', 3), ('TGACATC', 3), ('CTCTTGA', 3)]
8 [('ATGATCAA', 4), ('TGATCAAG', 3), ('CTCTTGAT', 3), ('TCTTGATC', 3), ('CTTGATCA', 3), ('TTGATCAT', 3)]
9 [('ATGATCAAG', 3), ('CTCTTGATC', 3), ('TCTTGATCA', 3), ('CTTGATCAT', 3), ('AATGATCAA', 2), ('AAGCATGAT', 2)]
```

k-mer Likelihoods



Are two 5-mers repeated 8 times interesting? Surprizing? How about four 9-mers repeated 3 times?

Under the assumption that all k-mers are equally likely, we'd expect a given k-mer to occur:

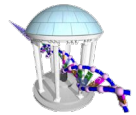
$$p(k) = 1/4^k$$

So we expect a specific 5-mer once per 1024 bases, so having 8 in 535 (540 - 5) bases is more likely than expected. We also expect a specific 9-mer once per 262,144 bases, so having 3 in 531 (540 - 9) is much more than expected.

Moreover, is their any relationship between the 9-mers ATGATCAAG and CTTGATCAT?

```
atcaatgatcaacgtaagcttctaagcATGATCAAGgtgctcacacagtttatccacaac
ctgagtggatgacatcaagataggtcgttgtatctccttcctctcgtactctcatgacca
cggaaagATGATCAAGagaggatgatttcttggccatatcgcaatgaatacttgtgactt
gtgcttccaattgacatcttcagcgccatattgcgctggccaaggtgacggagcgggatt
acgaaagcatgatcatggctgttgttctgtttatcttgttttgactgagacttgttagga
tagacggtttttcatcactgactagccaaagccttactctgcctgacatcgaccgtaaat
tgataatgaatttacatgcttccgcgacgatttacctCTTGATCATcgatccgattgaag
atcttcaattgttaattctcttgcctcgactcatagccatgatgagctCTTGATCATgtt
tccttaaccctctatttttacggaagaATGATCAAGctgctgctCTTGATCATcgtttc
```

Biological Insights



- Replication is performed by a DNA polymerase, and the initiation of replication is mediated by a protein called *DnaA*.
- DnaA binds to short (≈ 9 nucleotides long) segments within the replication origin known as a *DnaA* box (≈ 500 bases).
- A DnaA box is a signal telling DnaA to “bind here!”
- DnaA can bind to either strand. Thus, both the DnaA box and its reverse-complement are equal targets.
- For reliablity “Life” wants to see multiple nearby DnaA boxes.
- Sequences used by DnaA tend to be "AT-rich" (rich in adenine and thymine bases), because AT base pairs have two hydrogen bonds (rather than the three formed in a CG pair) which makes them easier to unzip. (Recall A and T are the most common bases with 174 and 135)
- Once the origin has been located, these initiators recruit other proteins and form the pre-replication complex, which unzips the double-stranded DNA.