In [14]:

# helper function #1

def get_random_kmers(sequences, k):
    # input is a list of sequences and value k
    # return from each sequence substring with length k

In [17]:

# helper function #2

def compile_kmer_profile(kmers, k, i):
    # input is a list of kmers, value k (for convenience) and index value i
    # return compiled position frequency matrix (PFM) over all kmers, except kmer with index i

In [157]:

# helper function #3

def update_ith_kmer(profile, k, sequence):
    # input is PFM, value k and ith sequence from the list of sequences
    # calculate profile probability for PFM in each position in the sequence
    # return profile-random substring from sequence

In [211]:

# helper function #4

def mot_score(kmers, k):
    # input is a list of kmers and value k (for convenience)
    # return sum of mismatches against median string over PFM

In [ ]:

# helper function #5

def median_motif(kmers, k):
    # input is a list of kmers and value k (for convenience)
    # return a string where every letter corresponds to one of most occurring letter in the list of kmers for give position
def gibbs_sampler(sequences, k, N, R=1):
    from random import random
    kmers = get_random_kmers(sequences, k)
    best_motifs = [kmer for kmer in kmers]
    for r in range(R):
        for n in range(N):
            i = int(len(sequences) * random())
            profile = compile_kmer_profile(kmers, k, i)
            kmers[i] = update_ith_kmer(profile, k, sequences[i])
            # express progress
            if (n + 1) % 100 == 0:
                print('{:.2%}'.format((N * r + n + 1) / (N * R)), end='\r')
            # update global best, if found
            if mot_score(kmers, k) < mot_score(best_motifs, k):
                print('        best motif score:' , '{:.2f} ['.format(mot_score(kmers, k)), r, n, ']', end='\r')
                best_motifs = [kmer for kmer in kmers]
        kmers = get_random_kmers(sequences, k)
        print()  # add end of the line to progress
    return best_motifs

from timeit import default_timer as timer

# read in test data
with open('/data/rosalind_ba2g.txt', 'r') as ifh:
    for line in ifh.readlines():
        test_seqs.append(line.rstrip())

k = 15
N = 2000
R = 20

time_start = timer()
mots = gibbs_sampler(test_seqs, k, N, R)
time_elapsed = '{:.2f}'.format(timer() - time_start)
final_score = '{:.2f}'.format(mot_score(mots, k))
median_motif = '{:.2f}'.format(median_motif(mots, k))