T.C.

MARMARA UNIVERSITY

FACULTY of ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

CSE 4065 Project I Report

Group Members

150118071- Lara Gülsüm ŞEN

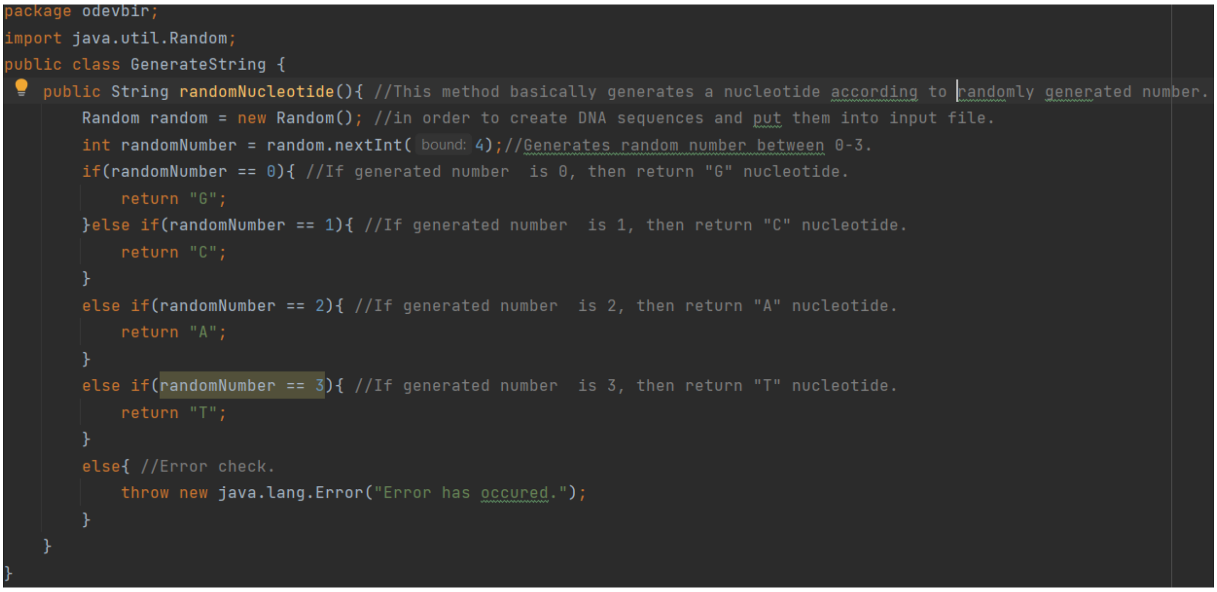
150117055 – Taylan Rojen DÖĞER

150115054 – Uğur Alp YAVUZ

1. Preparing Input File

1.1. GenerateString.java class:

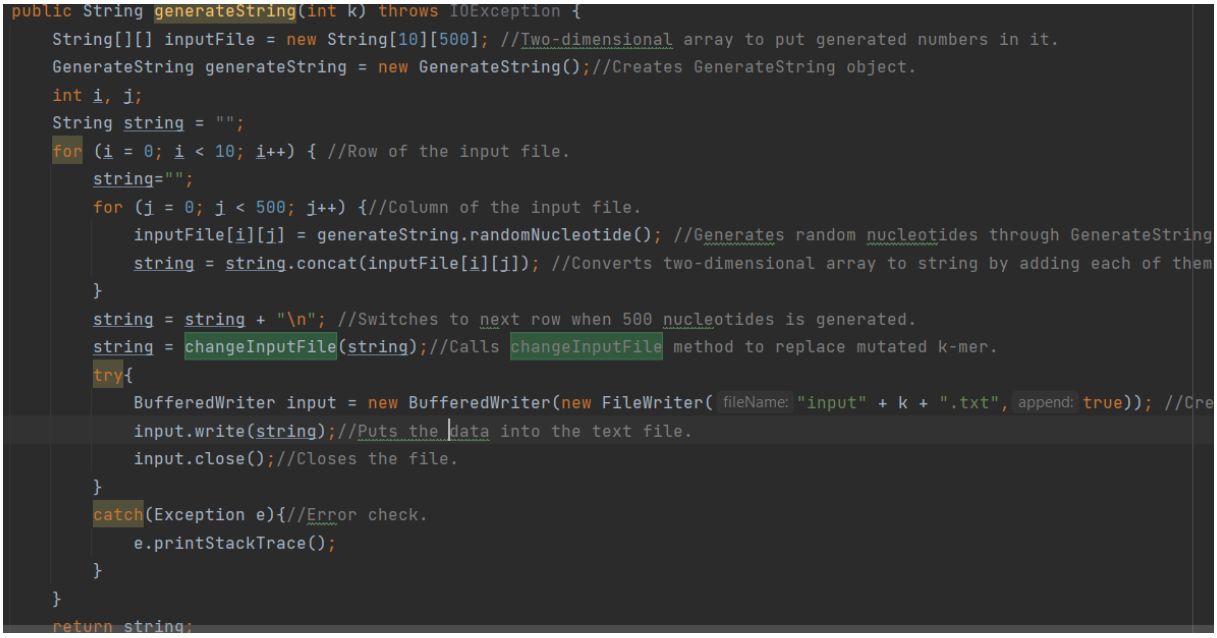
Firstly, we generated nucleotides randomly in our randomNucleotide() method:



```java
package odevbir;
import java.util.Random;
public class GenerateString {
    public String randomNucleotide(){ //This method basically generates a nucleotide according to randomly generated number.
        Random random = new Random(); //in order to create DNA sequences and put them into input file.
        int randomNumber = random.nextInt( bound: 4);//Generates random number between 0-3.
        if(randomNumber == 0){ //If generated number  is 0, then return "G" nucleotide.
            return "G";
        }else if(randomNumber == 1){ //If generated number  is 1, then return "C" nucleotide.
            return "C";
        }
        else if(randomNumber == 2){ //If generated number  is 2, then return "A" nucleotide.
            return "A";
        }
        else if(randomNumber == 3){ //If generated number  is 3, then return "T" nucleotide.
            return "T";
        }
        else{ //Error check.
            throw new java.lang.Error("Error has occured.");
        }
    }
}
```

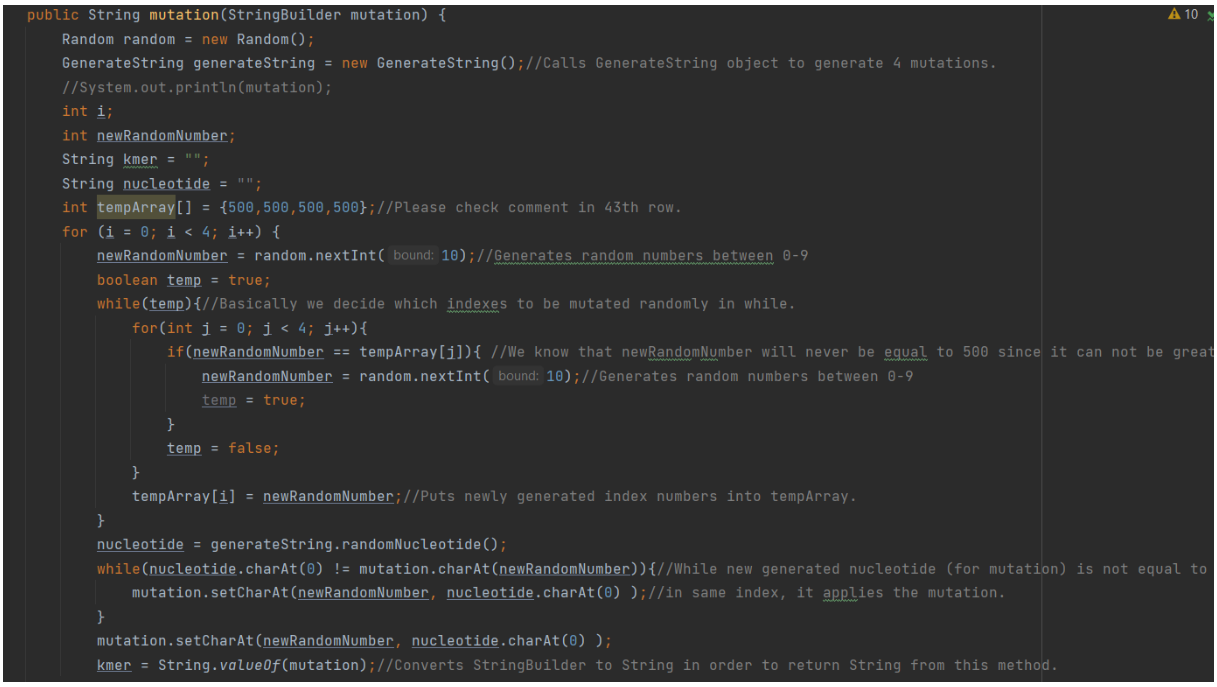
1.2. CreateInputFile.java class:

In generateString() method, we generated our DNA strings in 2-dimensional array which contains 500 nucleotides in each of 10 rows. Also, in this method, we called changeInputFile() method -which calls mutation() method inside of it- to apply mutation:



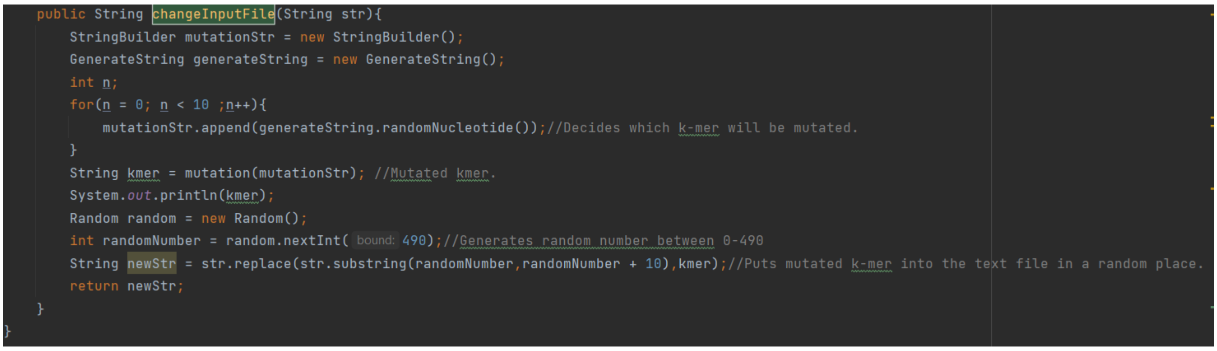
```java
public String generateString(int k) throws IOException {
    String[][] inputFile = new String[10][500]; //Two-dimensional array to put generated numbers in it.
    GenerateString generateString = new GenerateString();//Creates GenerateString object.
    int i, j;
    String string = "";
    for (i = 0; i < 10; i++) { //Row of the input file.
        string="";
        for (j = 0; j < 500; j++) {//Column of the input file.
            inputFile[i][j] = generateString.randomNucleotide(); //Generates random nucleotides through GenerateString
            string = string.concat(inputFile[i][j]); //Converts two-dimensional array to string by adding each of them
        }
        string = string + "\n"; //Switches to next row when 500 nucleotides is generated.
        string = changeInputFile(string);//Calls changeInputFile method to replace mutated k-mer.
        try{
            BufferedWriter input = new BufferedWriter(new FileWriter( fileName: "input" + k + ".txt", append: true)); //Cre
            input.write(string);//Puts the data into the text file.
            input.close();//Closes the file.
        }
        catch(Exception e){//Error check.
            e.printStackTrace();
        }
    }
    return string;
```

In mutation() method, basically we generate 4 new nucleotides to apply mutation to our k-mer.



```java
public String mutation(StringBuilder mutation) {
    Random random = new Random();
    GenerateString generateString = new GenerateString();//Calls GenerateString object to generate 4 mutations.
    //System.out.println(mutation);
    int i;
    int newRandomNumber;
    String kmer = "";
    String nucleotide = "";
    int tempArray[] = {500,500,500,500};//Please check comment in 43th row.
    for (i = 0; i < 4; i++) {
        newRandomNumber = random.nextInt( bound: 10);//Generates random numbers between 0-9
        boolean temp = true;
        while(temp){//Basically we decide which indexes to be mutated randomly in while.
            for(int j = 0; j < 4; j++){
                if(newRandomNumber == tempArray[j]){ //We know that newRandomNumber will never be equal to 500 since it can not be great
                    newRandomNumber = random.nextInt( bound: 10);//Generates random numbers between 0-9
                    temp = true;
                }
                temp = false;
            }
            tempArray[i] = newRandomNumber;//Puts newly generated index numbers into tempArray.
        }
        nucleotide = generateString.randomNucleotide();
        while(nucleotide.charAt(0) != mutation.charAt(newRandomNumber)){//While new generated nucleotide (for mutation) is not equal to
            mutation.setCharAt(newRandomNumber, nucleotide.charAt(0) );//in same index, it applies the mutation.
        }
        mutation.setCharAt(newRandomNumber, nucleotide.charAt(0) );
        kmer = String.valueOf(mutation);//Converts StringBuilder to String in order to return String from this method.
```

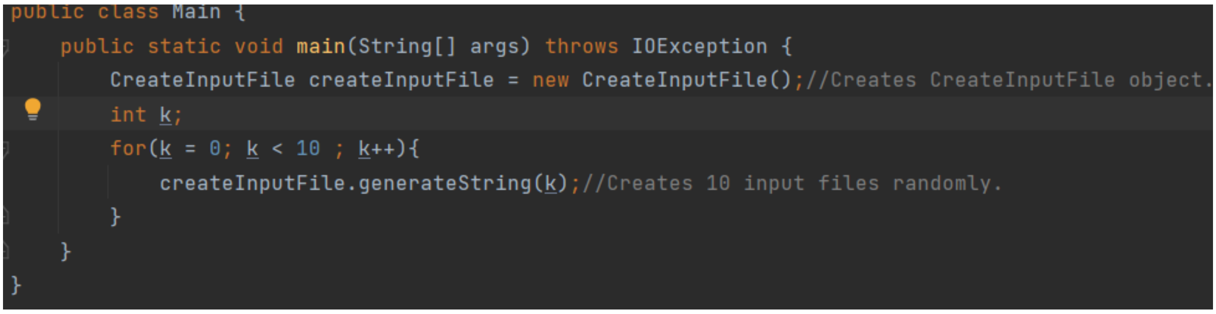
In changeInputFile method, we decided which k-mer is going to be mutated and fetch newly mutated k-mer from mutation() method. Finally, we added mutated k-mer to our text file.



```java
    public String changeInputFile(String str){
        StringBuilder mutationStr = new StringBuilder();
        GenerateString generateString = new GenerateString();
        int n;
        for(n = 0; n < 10 ;n++){
            mutationStr.append(generateString.randomNucleotide());//Decides which k-mer will be mutated.
        }
        String kmer = mutation(mutationStr); //Mutated kmer.
        System.out.println(kmer);
        Random random = new Random();
        int randomNumber = random.nextInt( bound: 490);//Generates random number between 0-490
        String newStr = str.replace(str.substring(randomNumber,randomNumber + 10),kmer);//Puts mutated k-mer into the text file in a random place.
        return newStr;
    }
}
```

1.3. Main.java class:

Main.java is our runner class where we instantiate a CreateInputFile object to generate 10 input files containing 10x500 nucleotides and our mutated k-mers inside of it.



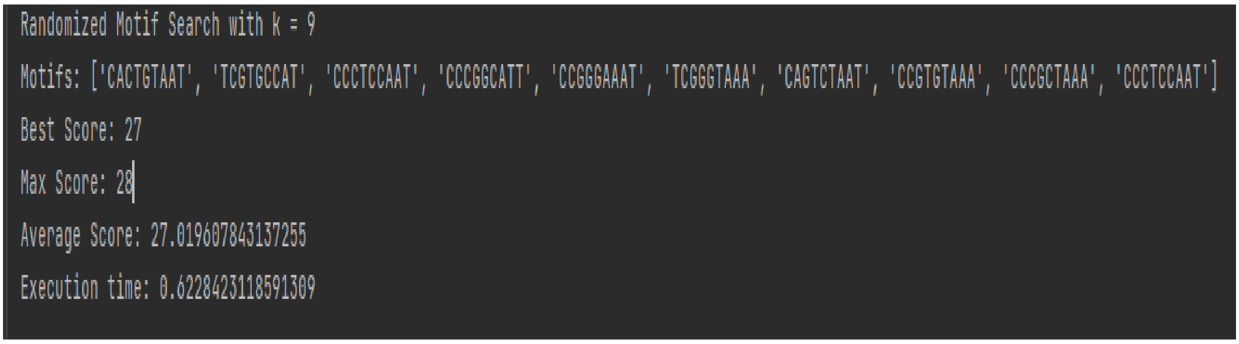
```java
public class Main {
    public static void main(String[] args) throws IOException {
        CreateInputFile createInputFile = new CreateInputFile();//Creates CreateInputFile object.
        int k;
        for(k = 0; k < 10 ; k++){
            createInputFile.generateString(k);//Creates 10 input files randomly.
        }
    }
}
```

2. Results

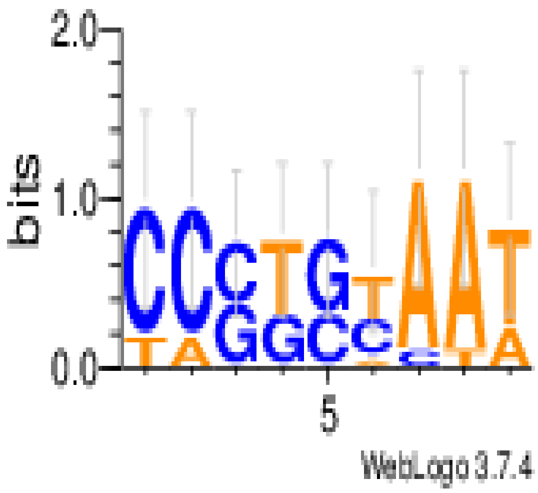
2.1. Randomized Motif Search

Here are some screenshots of example outputs and full tables for k=9,10,11

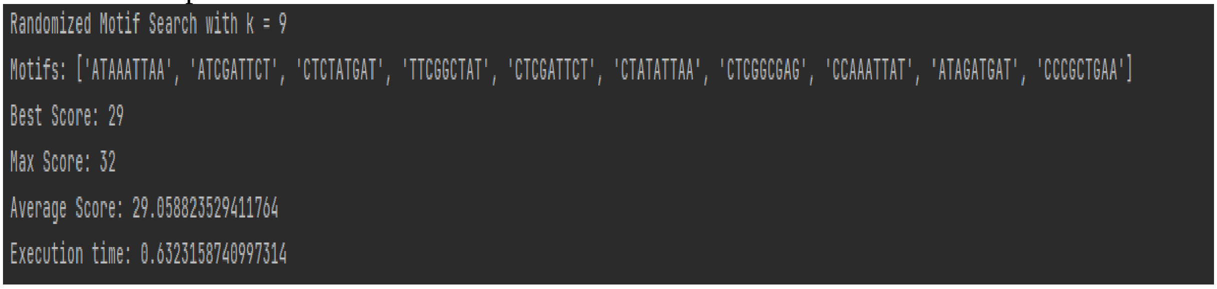
- k=9



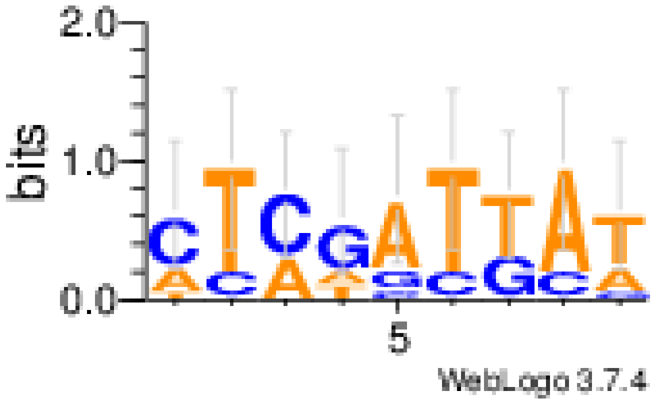
```
Randomized Motif Search with k = 9

Motifs: ['CACTGTAAT', 'TCGTGCCAT', 'CCCTCCAAT', 'CCCGGCATT', 'CCGGGAAAT', 'TCGGGTAAA', 'CAGTCTAAT', 'CCGTGTAAA', 'CCCGCTAAA', 'CCCTCCAAT']

Best Score: 27

Max Score: 28

Average Score: 27.019607843137255

Execution time: 0.6228423118591309
```



Another example:



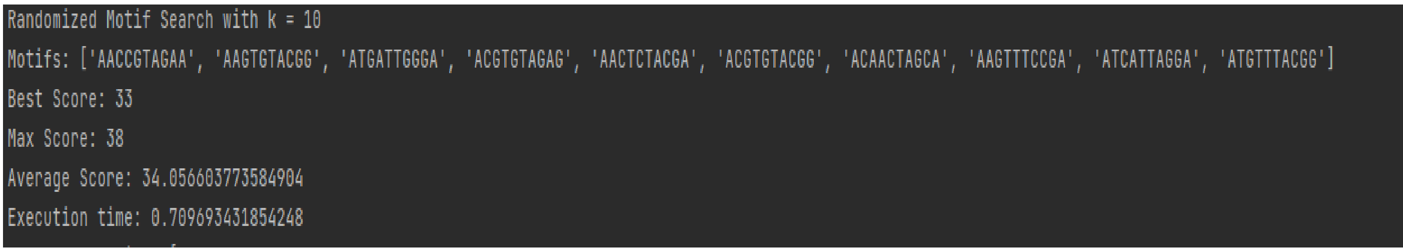
```
Randomized Motif Search with k = 9

Motifs: ['ATAAATTAA', 'ATCGATTCT', 'CTCTATGAT', 'TTCGGCTAT', 'CTCGATTCT', 'CTATATTAA', 'CTCGGCGAG', 'CCAAATTAT', 'ATAGATGAT', 'CCCGCTGAA']

Best Score: 29

Max Score: 32

Average Score: 29.058823529411764

Execution time: 0.6323158740997314
```



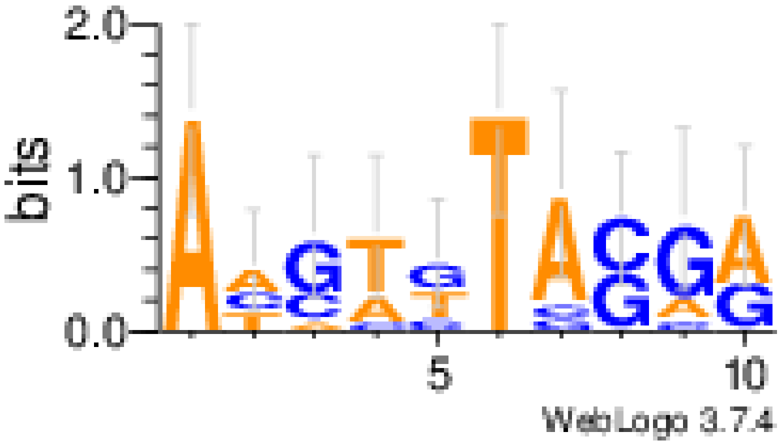
Here are 10 runs of Randomized Motif Search for k=9

K = 9	Best Score	Worst Score	Average Score	Consensus String
	27	28	27.019	CCCTGTAAT
	29	32	29.05	CTCGATTAT
	27	31	27.09	TAAGGGAAG
	35	38	35.05	TGCAATGGC
	28	30	28.03	GTCGTGGTT
	26	30	27.9	TTAAGAGAA
	25	29	25.16	TACGCATCC
	31	32	31.01	AATATGGGT
	29	32	29.09	ATGTTTACT
	28	31	28.05	TTGCTTTTC

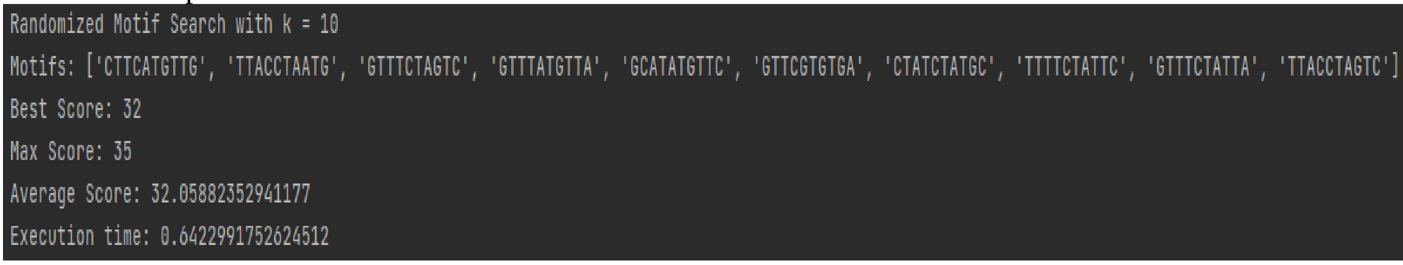
- k=10



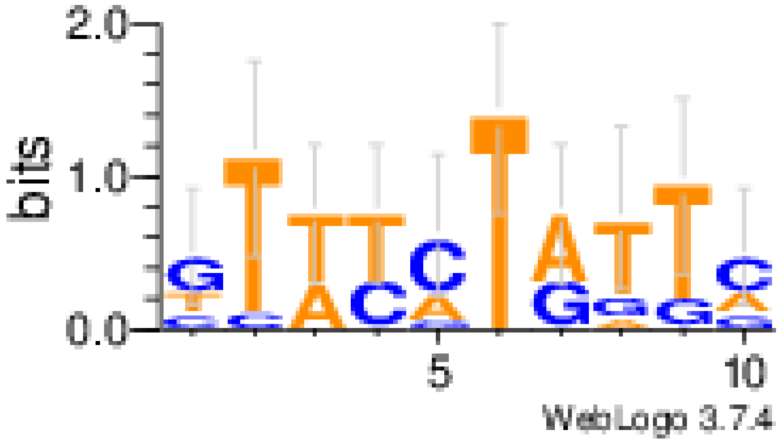
```
Randomized Motif Search with k = 10
Motifs: ['AACCGTAGAA', 'AAGTGTACGG', 'ATGATTGGGA', 'ACGTGTAGAG', 'AACTCTACGA', 'ACGTGTACGG', 'ACAACTAGCA', 'AAGTTTCCGA', 'ATCATTAGGA', 'ATGTTTACGG']
Best Score: 33
Max Score: 38
Average Score: 34.056603773584904
Execution time: 0.709693431854248
```



Another example:



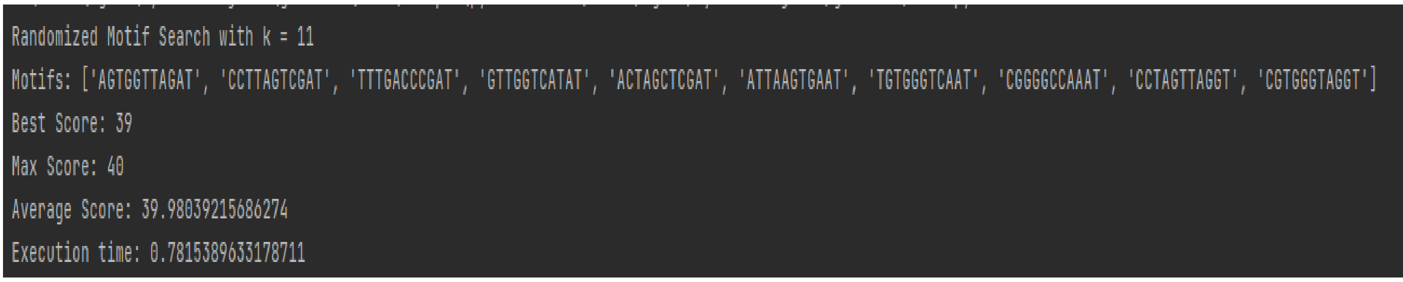
```
Randomized Motif Search with k = 10
Motifs: ['CTTCATGTTG', 'TTACCTAATG', 'GTTTCTAGTC', 'GTTTATGTTA', 'GCATATGTTC', 'GTTCGTGTGA', 'CTATCTATGC', 'TTTTCTATTC', 'GTTTCTATTA', 'TTACCTAGTC']
Best Score: 32
Max Score: 35
Average Score: 32.05882352941177
Execution time: 0.6422991752624512
```



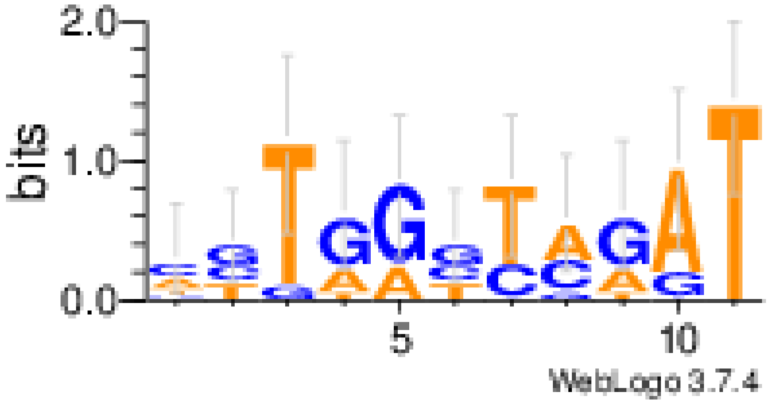
Here are 10 runs of Randomized Motif Search for k=10

K = 10	Best Score	Worst Score	Average Score	Consensus String
	33	38	34.05	AAGTGTACGA
	32	35	35.05	GTTTCTATTC
	35	37	35.03	GATAAATTAA
	36	37	36.93	TAGCGACGGG
	30	33	30.07	CTGTAAGTCG
	29	32	29.05	CAAAGACGAA
	30	31	30.01	GTAAGTGCAC
	33	35	33.03	CGGCCACGAA
	31	33	31.98	TGTAAATCTT
	32	33	32.98	CAGCGTACCA

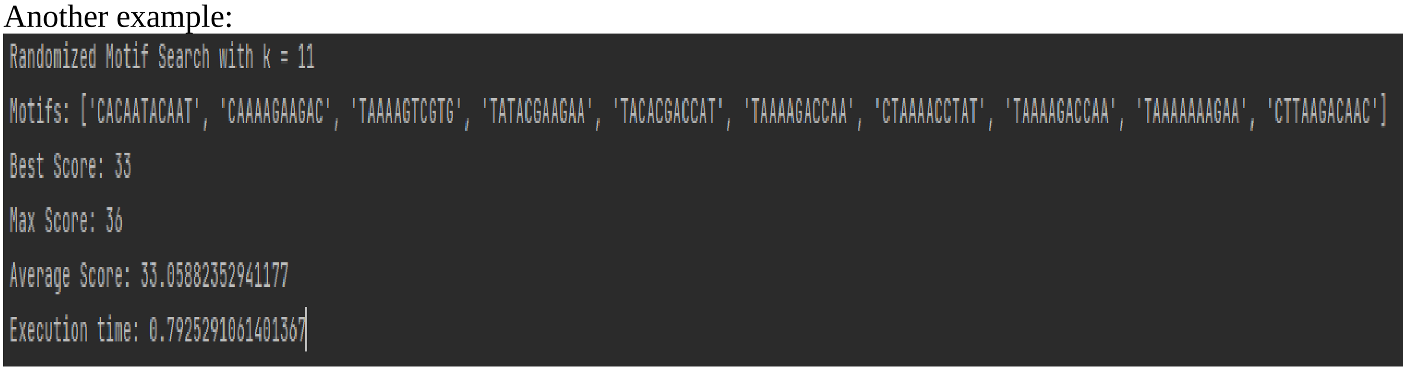
- k=11



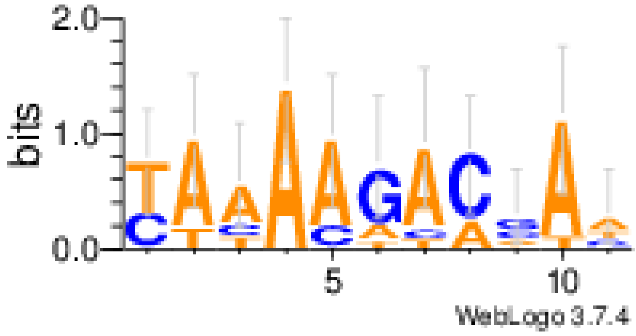
```
Randomized Motif Search with k = 11
Motifs: ['AGTGGTTAGAT', 'CCTTAGTCGAT', 'TTTGACCCGAT', 'GTTGGTCATAT', 'ACTAGCTCGAT', 'ATTAAGTGAAT', 'TGTGGGTCAAT', 'CGGGGCCAAAT', 'CCTAGTTAGGT', 'CGTGGGTAGGT']
Best Score: 39
Max Score: 40
Average Score: 39.98039215686274
Execution time: 0.7815389633178711
```



Another example:



```
Randomized Motif Search with k = 11

Motifs: ['CACAATACAAT', 'CAAAAGAAGAC', 'TAAAAGTCGTG', 'TATACGAAGAA', 'TACACGACCAT', 'TAAAAGACCAA', 'CTAAAACCTAT', 'TAAAAGACCAA', 'TAAAAAAAGAA', 'CTTAAGACAAC']

Best Score: 33

Max Score: 36

Average Score: 33.05882352941177

Execution time: 0.7925291061401367
```



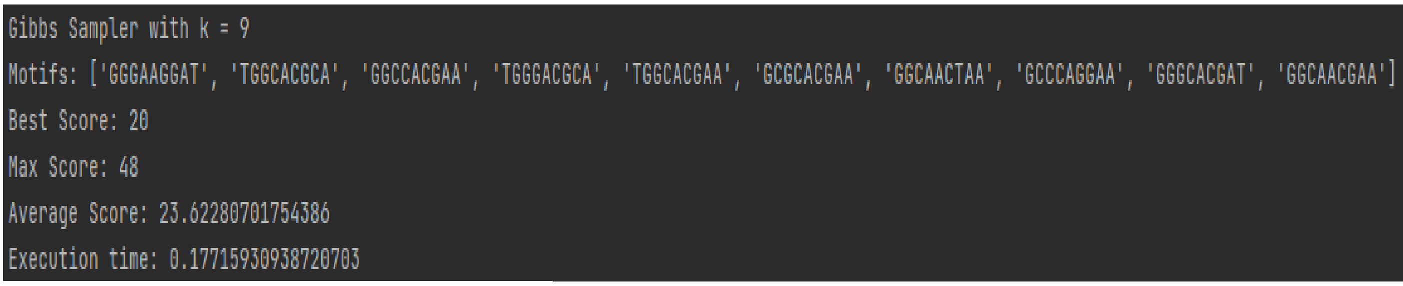
Here are 10 runs of Randomized Motif Search for k=11

K = 11	Best Score	Worst Score	Average Score	Consensus String
	39	40	39.98	CGTGGGTAGAT
	33	36	33.05	TAAAAGACGAA
	36	40	36.13	CTATTACATGC
	40	42	40.03	AGAGCTCGATA
	37	38	37.01	GCGCAAACGCA
	35	40	35.13	ACCTGTATTTC
	35	38	35.05	ATAACATGGTT
	33	42	33.17	GCATGACTCCC
	43	45	43.07	CGTCTAGCGGG
	38	42	38.07	ATCAGTGGAGT

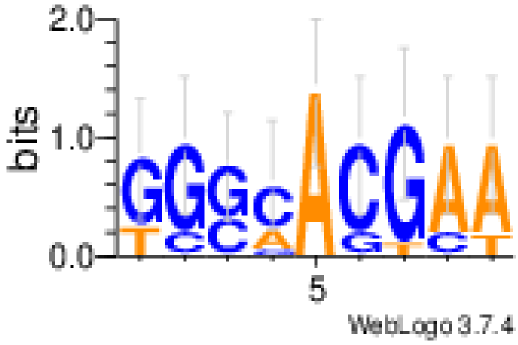
2.2. Gibbs Sampler

Here are some screenshots of example outputs and full tables for k=9,10,11

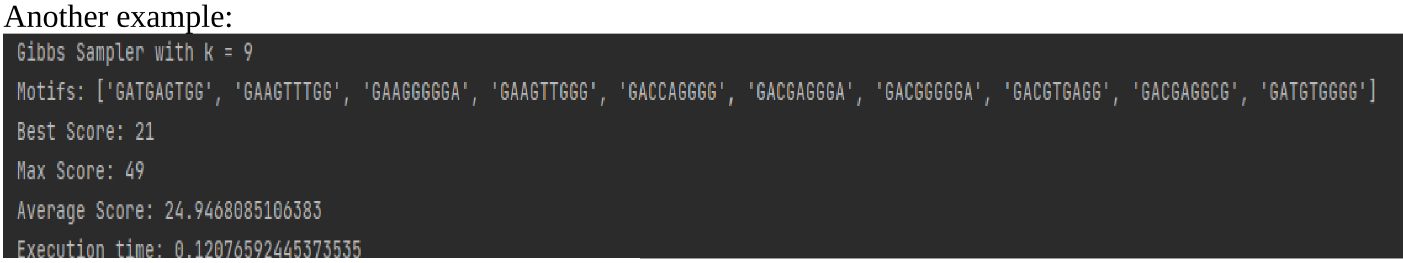
- k=9



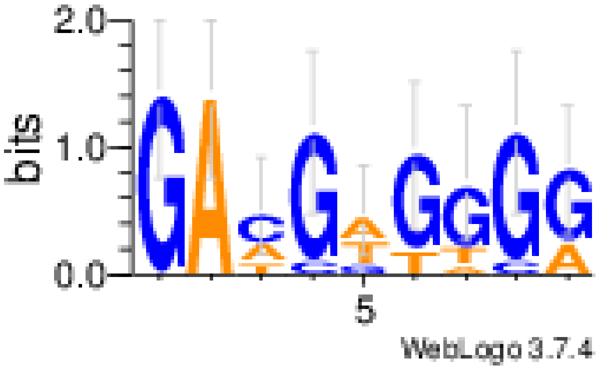
```
Gibbs Sampler with k = 9
Motifs: ['GGGAAGGAT', 'TGGCACGCA', 'GGCCACGAA', 'TGGGACGCA', 'TGGCACGAA', 'GCGCACGAA', 'GGCAACTAA', 'GCCCAGGAA', 'GGGCACGAT', 'GGCAACGAA']
Best Score: 20
Max Score: 48
Average Score: 23.62280701754386
Execution time: 0.17715930938720703
```



Another example:



```
Gibbs Sampler with k = 9
Motifs: ['GATGAGTGG', 'GAAGTTTGG', 'GAAGGGGGA', 'GAAGTTGGG', 'GACCAGGGG', 'GACGAGGGA', 'GACGGGGGA', 'GACGTGAGG', 'GACGAGGCG', 'GATGTGGGG']
Best Score: 21
Max Score: 49
Average Score: 24.9468085106383
Execution time: 0.12076592445373535
```

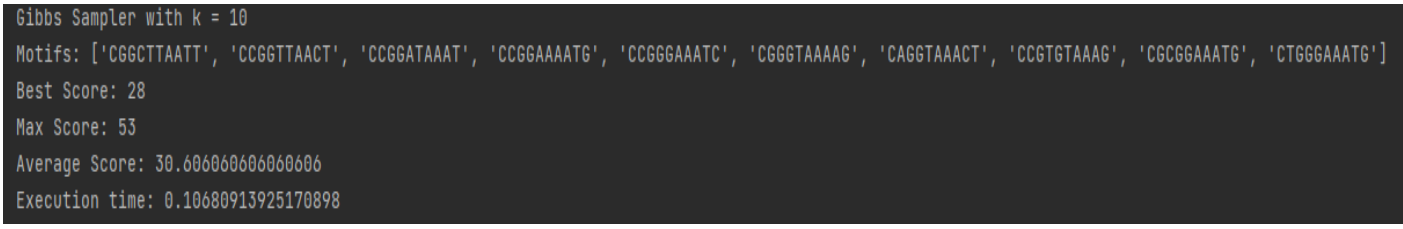


Here are 10 runs of Gibbs Sampler for k=9

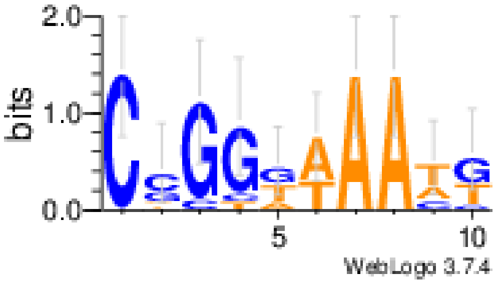
K = 9	Best Score	Worst Score	Average Score	Consensus String
	20	48	23.62	GGGCACGAA
	21	49	24.94	GGGCACGAA

	16	49	20.95	GTGAATAAC
	23	50	26.44	GGAAATCAG
	23	46	25.37	TTTGGCAAG
	24	49	27.5	CATTGAACA
	22	50	25.42	CATTTCTAA
	23	53	26.66	CGGGACGTC
	21	50	25.69	TAAAATAAG
	20	50	23.61	CAGAGTACC

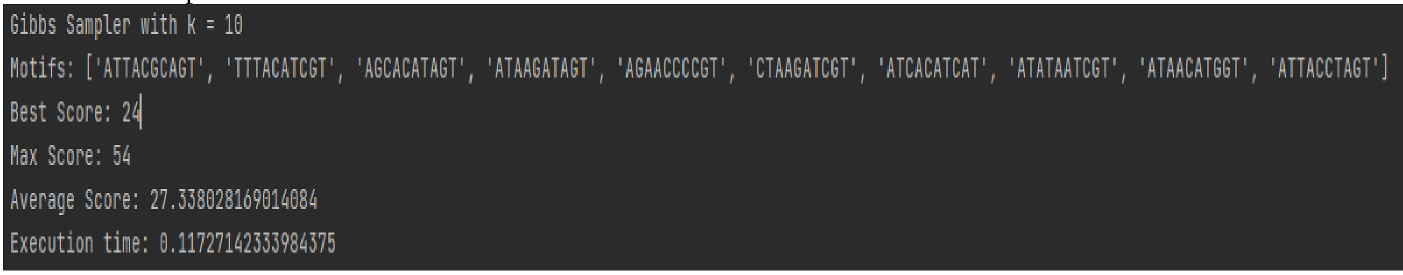
- k=10



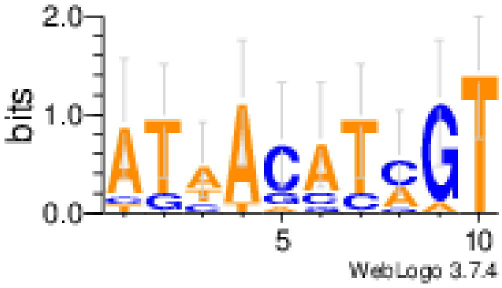
```
Gibbs Sampler with k = 10
Motifs: ['CGGCTTAATT', 'CCGGTTAACT', 'CCGGATAAAT', 'CCGGAAAATG', 'CCGGGAAATC', 'CGGGTAAAAG', 'CAGGTAAACT', 'CCGTGTAAAG', 'CGCGGAAATG', 'CTGGGAAATG']
Best Score: 28
Max Score: 53
Average Score: 30.606060606060606
Execution time: 0.10680913925170898
```



Another example:



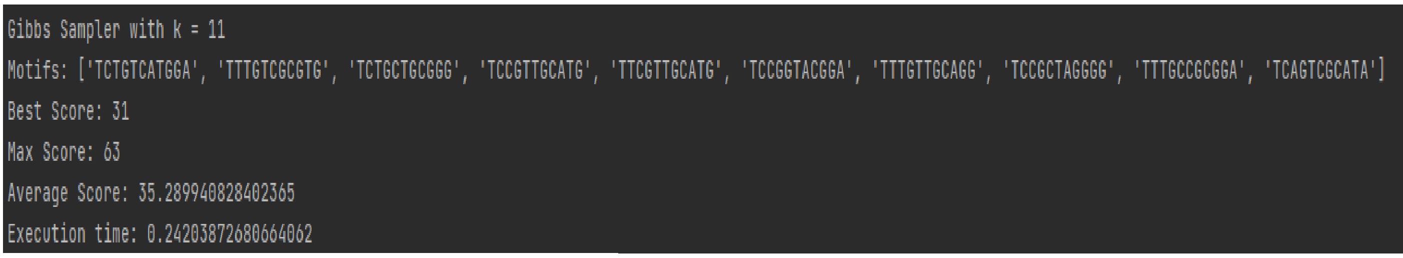
```
Gibbs Sampler with k = 10
Motifs: ['ATTACGCAGT', 'TTTACATCGT', 'AGCACATAGT', 'ATAAGATAGT', 'AGAACCCCGT', 'CTAAGATCGT', 'ATCACATCAT', 'ATATAATCGT', 'ATAACATGGT', 'ATTACCTAGT']
Best Score: 24
Max Score: 54
Average Score: 27.338028169014084
Execution time: 0.11727142333984375
```



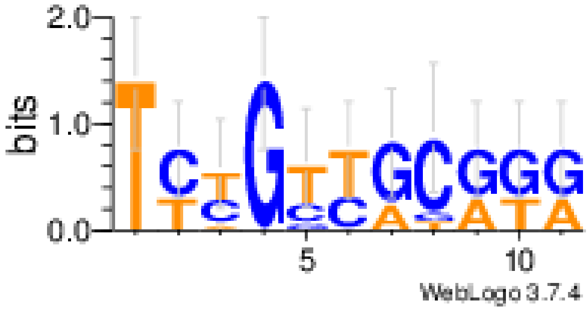
Here are 10 runs of Gibbs Sampler for k=10

K = 10	Best Score	Worst Score	Average Score	Consensus String
	28	53	30.6	CCGGGAAATG
	24	54	27.33	ATAACATCGT
	24	57	28.74	TCTTTGATTA
	28	56	31.09	ACGTGTGGGT
	27	54	31.13	AAAGTTGGCG
	25	54	28.69	GCAACGTCCG
	22	49	26.18	ATCGTACCCC
	25	59	27.91	CTAGCTGGGT
	26	54	30.15	ATTCTCCGCC
	24	53	29.08	GAGGGATGGG

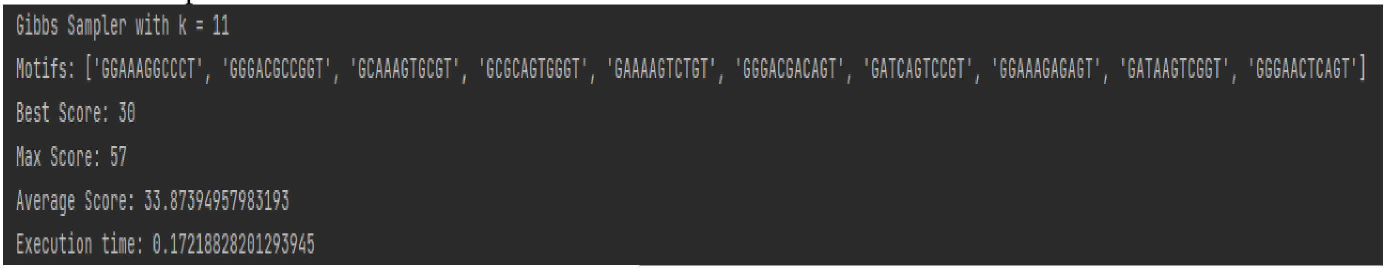
- k=11



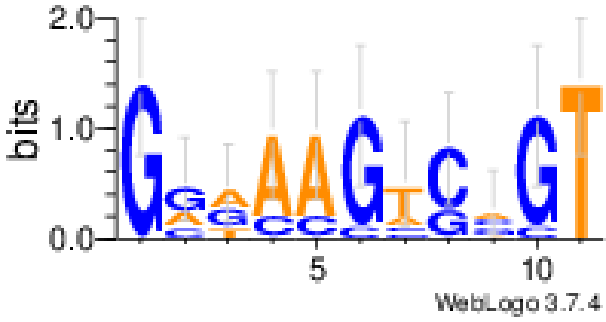
```
Gibbs Sampler with k = 11
Motifs: ['TCTGTCATGGA', 'TTTGTCGCGTG', 'TCTGCTGCGGG', 'TCCGTTGCATG', 'TTCGTTGCATG', 'TCCGGTACGGA', 'TTTGTTGCAGG', 'TCCGCTAGGGG', 'TTTGCCGCGGA', 'TCAGTCGCATA']
Best Score: 31
Max Score: 63
Average Score: 35.289940828402365
Execution time: 0.24203872680664062
```



Another example:



```
Gibbs Sampler with k = 11
Motifs: ['GGAAAGGCCCT', 'GGGACGCCGGT', 'GCAAAGTGCGT', 'GCGCAGTGGGT', 'GAAAAGTCTGT', 'GGGACGACAGT', 'GATCAGTCCGT', 'GGAAAGAGAGT', 'GATAAGTCGGT', 'GGGAACTCAGT']
Best Score: 30
Max Score: 57
Average Score: 33.87394957983193
Execution time: 0.17218828201293945
```



K = 11	Best Score	Worst Score	Average Score	Consensus String
	31	63	35.28	TCTGTTGCGGG
	30	57	33.87	GGAAAGTCAGT
	30	58	33.03	GTCCGGTATTA
	30	59	34.54	TAATCTTCATT
	31	55	34.65	TCTTACACGGG
	27	63	30.7	CAAATGAAGAG
	30	56	33.45	GAGGGGGTTCT
	27	58	30.55	TGTGTCTAGAT
	30	60	33.63	ATTTGACCCGC
	27	61	32.26	ATTAGGAACGG

Conclusion:

As a result of our repeated runs, we observed that the scores increased in parallel when the k value increased. In addition, we observed that at smaller k values, the consensus string is closer to the original string. When we compared the two algorithms, we found that Gibbs Sampler gave better results compared to Randomized Motif Search. Our best score was 16 in the Gibbs Sampler, and 27 in Randomized Motif Search.