Table of Content:

1.) Project Information:

We have learnt that in the problem of aligning alignments, inputs are two multi-alignment blocks, each with $k1$ and $k2$ species respectively, and output is the multi-alignment of ($k1$+$k2$) species. Using sum-of-scores scoring strategy, the time complexity of aligning alignments is $O(k1 \times k2 \times m \times n)$ where m and n are the numbers of columns of each input alignment block. It limits the efficiency of multi-alignments when the number of sequenced genomes grows significantly. In this project, we will explore the profile scoring strategy to reduce the time cost to $O((k1+k2) \times m \times n)$. We will be implementing both algorithms and compare them with simulated alignments.

2.) What we have learnt so far:

So far in the previous three projects, we have learned what is Genome. Genome is encoded into four types of DNA bases and we use the first letter of each to represent them. The four Alphabets are A, C, G and T. So, genomes are long strings containing these four alphabets.

We have learnt that all species have experienced a long history of evolution. And couple of these evolutionary events have occurred because of insertion / deletion (indel; as we cannot establish for a fact whether there was an insertion or deletion and hence, we call it indel) substitution and few more like duplication, translocation and transposition. But in this project, we are just concentrating on substitution.

Substitution is a type where one base is transformed to another (A – G, A – T, A –C, G – A, G – C, G – T, C – A, C – G, C – T, T – A, T – C and T – G). There are two types in substitution:

Transition: When substitution between A – G and C – T occurs.

Transversion: Rest all the types of substitution.

Also, we have learnt that an indel is often referred as a gap and it is represented using '-'.
A sequence of contiguous '-' characters is counted as one gap. The gap length refers to the number of '-' characters in the gap.

We have learnt there are 7 regions Exon, Intron, Intergenic, Coding, 5'UTR, 3'UTR and Promotor. And how based on Promotor base the values of match, mismatch, gap rate varies. We also computed Sensitivity and Specificity by taking various values for parameters *'K' i.e. "Alignment Score Cut off"* and *'O' i.e.* "*Gap Open Penalty*".

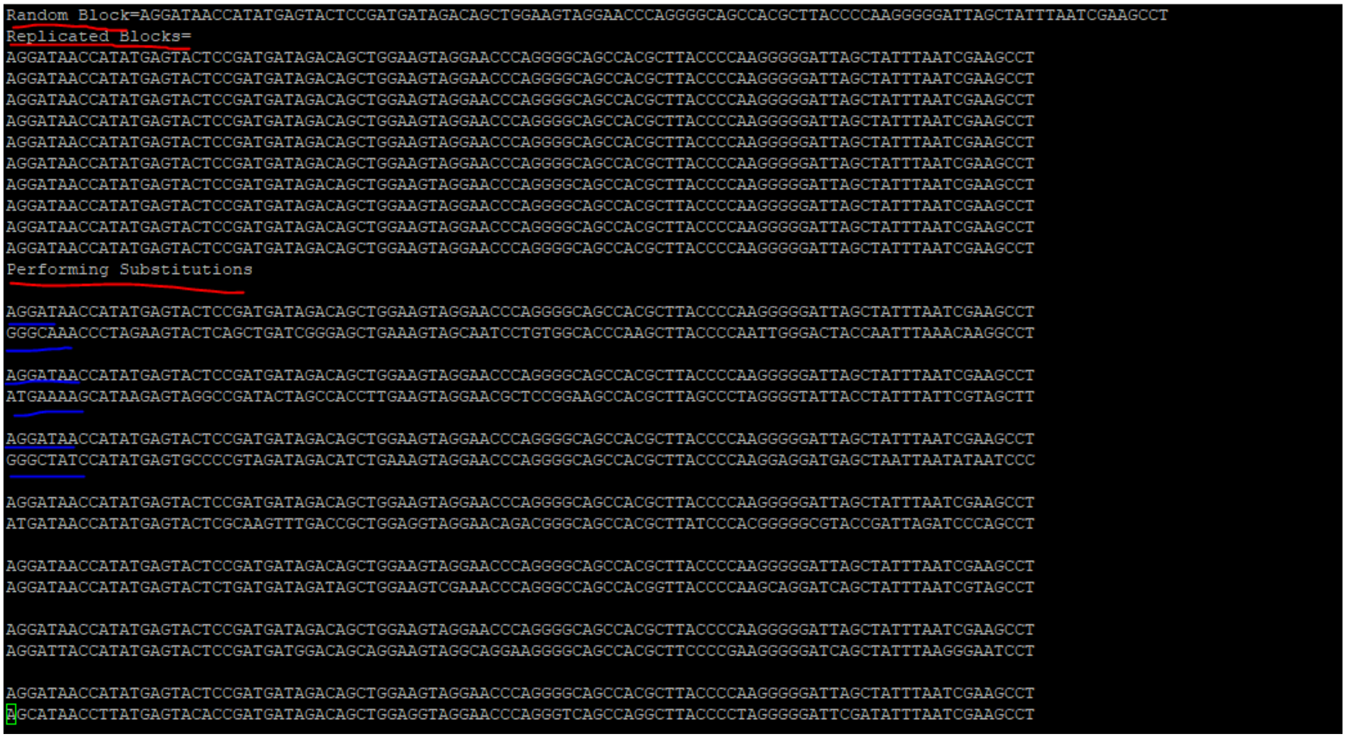
3.) What we are implementing in this Project:

- We are trying implement a program to simulate alignment blocks by considering two parameters *'N'* i.e. *"Number of Strings" (String consists of A, C, G, and T)* and *'L'* i.e. *"Length of String"* (*String consists of A, C, G, and T*).

The 1st part of the project consists of the following:

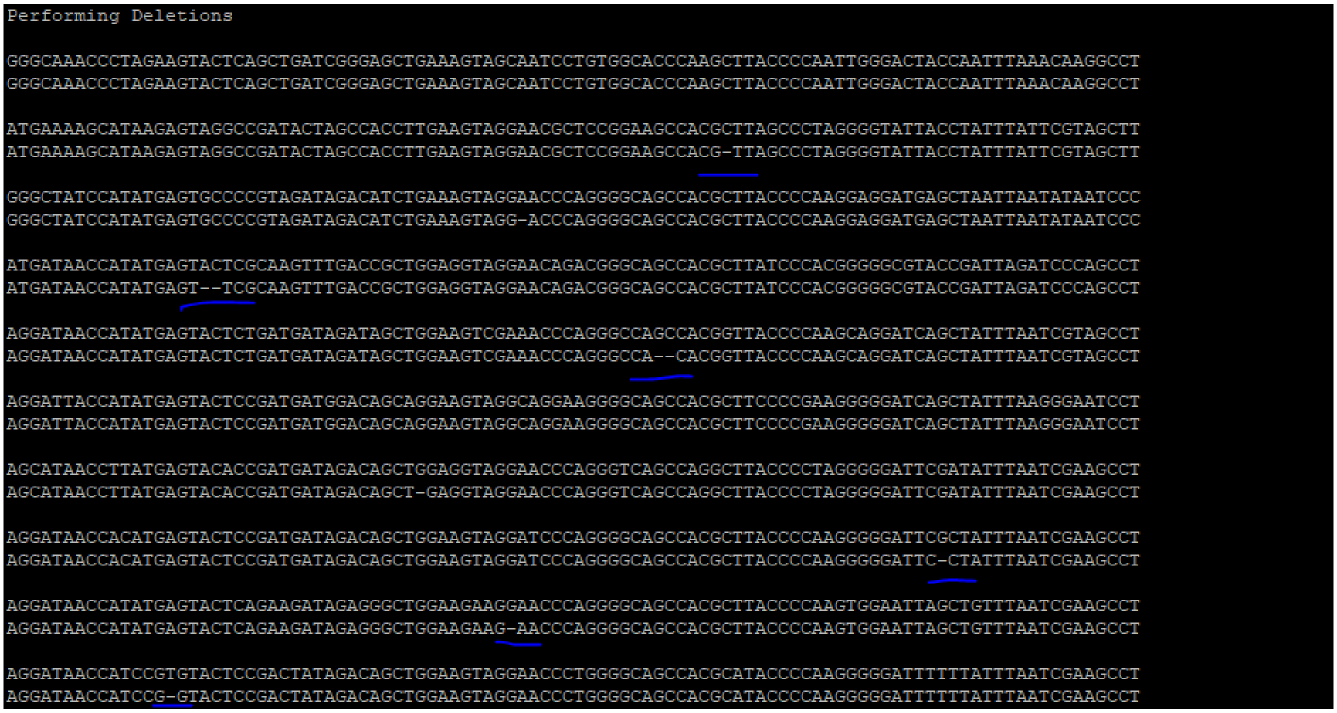
- Randomly produce a string of length *L* composed of letters A, C, G, and T, each of 25%.
- Replicate *N*-1 copies of above string. Now you have *N* strings in total. Line them up as in an alignment block.
- Simulate substitutions: for each string, randomly generate 1%~30% substitutions. Substitutions are uniformly distributed in the string.
- Simulate deletions: for each string, randomly generate 0.2%~3.6% deletions by replacing a contiguous segment of letters by '- 's. Gaps are uniformly distributed in the string. For simplification, assume the gap length is uniformly distributed between 1 and 10.
- Divide above alignment block into two, each of *N*/2 rows. The number of columns of each sub-alignment block currently is still *L*. In each sub-alignment block, randomly generate 1% insertions:

    - Randomly pick a row and a random position.
    - Insert a sequence of length *x*, *x* is a random number between 1 and 5. The sequence is randomly generated from {A, C, G, T}.
    - Insert a segment of '- 's of length *x* in all other rows at the same position to keep all rows of the alignment block with the same number of columns.
    - For the below example screenshots I have used N=10, L=100



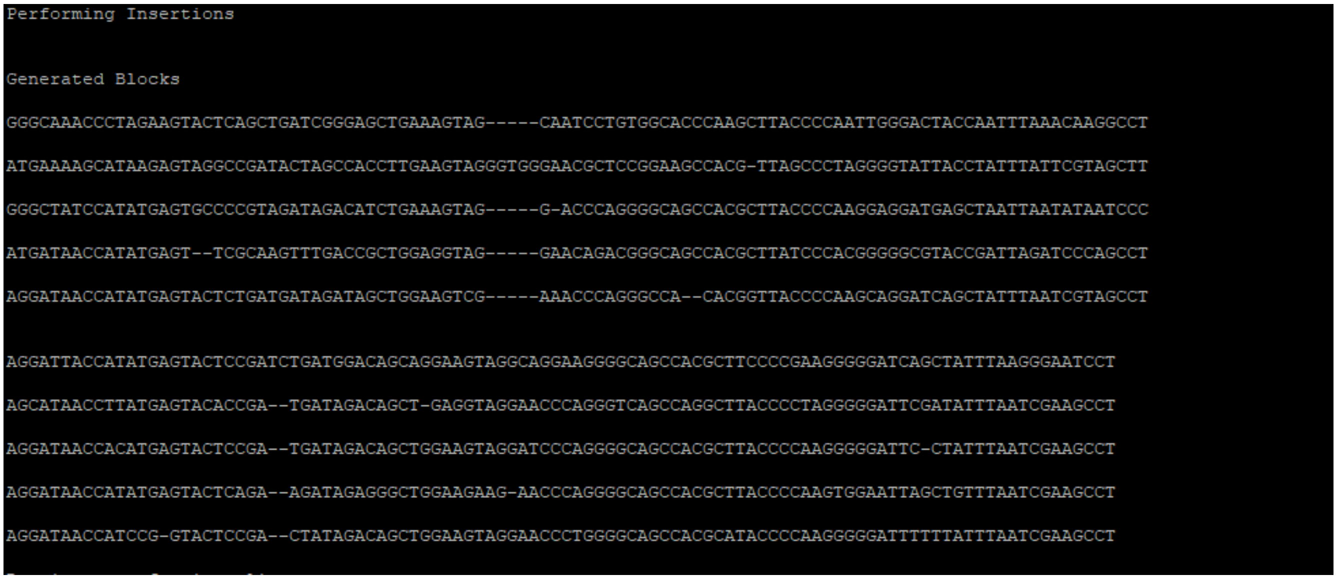
```
Random Block=AGGATAACCATATGAGTACTCCGATGATAGACAGCTGGAAGTAGGAACCCAGGGGCAGCCACGCTTACCCCAAGGGGGATTAGCTATTTAATCGAAGCCT
Replicated Blocks=
AGGATAACCATATGAGTACTCCGATGATAGACAGCTGGAAGTAGGAACCCAGGGGCAGCCACGCTTACCCCAAGGGGGATTAGCTATTTAATCGAAGCCT
AGGATAACCATATGAGTACTCCGATGATAGACAGCTGGAAGTAGGAACCCAGGGGCAGCCACGCTTACCCCAAGGGGGATTAGCTATTTAATCGAAGCCT
AGGATAACCATATGAGTACTCCGATGATAGACAGCTGGAAGTAGGAACCCAGGGGCAGCCACGCTTACCCCAAGGGGGATTAGCTATTTAATCGAAGCCT
AGGATAACCATATGAGTACTCCGATGATAGACAGCTGGAAGTAGGAACCCAGGGGCAGCCACGCTTACCCCAAGGGGGATTAGCTATTTAATCGAAGCCT
AGGATAACCATATGAGTACTCCGATGATAGACAGCTGGAAGTAGGAACCCAGGGGCAGCCACGCTTACCCCAAGGGGGATTAGCTATTTAATCGAAGCCT
AGGATAACCATATGAGTACTCCGATGATAGACAGCTGGAAGTAGGAACCCAGGGGCAGCCACGCTTACCCCAAGGGGGATTAGCTATTTAATCGAAGCCT
AGGATAACCATATGAGTACTCCGATGATAGACAGCTGGAAGTAGGAACCCAGGGGCAGCCACGCTTACCCCAAGGGGGATTAGCTATTTAATCGAAGCCT
AGGATAACCATATGAGTACTCCGATGATAGACAGCTGGAAGTAGGAACCCAGGGGCAGCCACGCTTACCCCAAGGGGGATTAGCTATTTAATCGAAGCCT
AGGATAACCATATGAGTACTCCGATGATAGACAGCTGGAAGTAGGAACCCAGGGGCAGCCACGCTTACCCCAAGGGGGATTAGCTATTTAATCGAAGCCT
AGGATAACCATATGAGTACTCCGATGATAGACAGCTGGAAGTAGGAACCCAGGGGCAGCCACGCTTACCCCAAGGGGGATTAGCTATTTAATCGAAGCCT
Performing Substitutions

AGGATAACCATATGAGTACTCCGATGATAGACAGCTGGAAGTAGGAACCCAGGGGCAGCCACGCTTACCCCAAGGGGGATTAGCTATTTAATCGAAGCCT
GGGCAAACCCTAGAAGTACTCAGCTGATCGGGAGCTGAAAGTAGCAATCCTGTGGCACCCAAGCTTACCCCAATTGGGACTACCAATTTAAACAAGGCCT

AGGATAACCATATGAGTACTCCGATGATAGACAGCTGGAAGTAGGAACCCAGGGGCAGCCACGCTTACCCCAAGGGGGATTAGCTATTTAATCGAAGCCT
ATGAAAAGCATAAGAGTAGGCCGATACTAGCCACCTTGAAGTAGGAACGCTCCGGAAGCCACGCTTAGCCCTAGGGGTATTACCTATTTATTCGTAGCTT

AGGATAACCATATGAGTACTCCGATGATAGACAGCTGGAAGTAGGAACCCAGGGGCAGCCACGCTTACCCCAAGGGGGATTAGCTATTTAATCGAAGCCT
GGGCTATCCATATGAGTGCCCCGTAGATAGACATCTGAAAGTAGGAACCCAGGGGCAGCCACGCTTACCCCAAGGAGGATGAGCTAATTAATATAATCCC

AGGATAACCATATGAGTACTCCGATGATAGACAGCTGGAAGTAGGAACCCAGGGGCAGCCACGCTTACCCCAAGGGGGATTAGCTATTTAATCGAAGCCT
ATGATAACCATATGAGTACTCGCAAGTTTGACCGCTGGAGGTAGGAACAGACGGGCAGCCACGCTTATCCCACGGGGGCGTACCGATTAGATCCCAGCCT

AGGATAACCATATGAGTACTCCGATGATAGACAGCTGGAAGTAGGAACCCAGGGGCAGCCACGCTTACCCCAAGGGGGATTAGCTATTTAATCGAAGCCT
AGGATAACCATATGAGTACTCTGATGATAGATAGCTGGAAGTCGAAACCCAGGGCCAGCCACGGTTACCCCAAGCAGGATCAGCTATTTAATCGTAGCCT

AGGATAACCATATGAGTACTCCGATGATAGACAGCTGGAAGTAGGAACCCAGGGGCAGCCACGCTTACCCCAAGGGGGATTAGCTATTTAATCGAAGCCT
AGGATTACCATATGAGTACTCCGATGATGGACAGCAGGAAGTAGGCAGGAAGGGGCAGCCACGCTTCCCCGAAGGGGGATCAGCTATTTAAGGGAATCCT

AGGATAACCATATGAGTACTCCGATGATAGACAGCTGGAAGTAGGAACCCAGGGGCAGCCACGCTTACCCCAAGGGGGATTAGCTATTTAATCGAAGCCT
AGCATAACCTTATGAGTACACCGATGATAGACAGCTGGAGGTAGGAACCCAGGGTCAGCCAGGCTTACCCCTAGGGGGATTCGATATTTAATCGAAGCCT
```

Figure 1: Random Block Generation, Replicated Blocks and Performing Substitutions



```
Performing Deletions

GGGCAAACCCTAGAAGTACTCAGCTGATCGGGAGCTGAAAGTAGCAATCCTGTGGCACCCAAGCTTACCCCAATTGGGACTACCAATTTAAACAAGGCCT
GGGCAAACCCTAGAAGTACTCAGCTGATCGGGAGCTGAAAGTAGCAATCCTGTGGCACCCAAGCTTACCCCAATTGGGACTACCAATTTAAACAAGGCCT

ATGAAAAGCATAAGAGTAGGCCGATACTAGCCACCTTGAAGTAGGAACGCTCCGGAAGCCACGCTTAGCCCTAGGGGTATTACCTATTTATTCGTAGCTT
ATGAAAAGCATAAGAGTAGGCCGATACTAGCCACCTTGAAGTAGGAACGCTCCGGAAGCCACG-TTAGCCCTAGGGGTATTACCTATTTATTCGTAGCTT

GGGCTATCCATATGAGTGCCCCGTAGATAGACATCTGAAAGTAGGAACCCAGGGGCAGCCACGCTTACCCCAAGGAGGATGAGCTAATTAATATAATCCC
GGGCTATCCATATGAGTGCCCCGTAGATAGACATCTGAAAGTAGG-ACCCAGGGGCAGCCACGCTTACCCCAAGGAGGATGAGCTAATTAATATAATCCC

ATGATAACCATATGAGTACTCGCAAGTTTGACCGCTGGAGGTAGGAACAGACGGGCAGCCACGCTTATCCCACGGGGGCGTACCGATTAGATCCCAGCCT
ATGATAACCATATGAGT--TCGCAAGTTTGACCGCTGGAGGTAGGAACAGACGGGCAGCCACGCTTATCCCACGGGGGCGTACCGATTAGATCCCAGCCT

AGGATAACCATATGAGTACTCTGATGATAGATAGCTGGAAGTCGAAACCCAGGGCCAGCCACGGTTACCCCAAGCAGGATCAGCTATTTAATCGTAGCCT
AGGATAACCATATGAGTACTCTGATGATAGATAGCTGGAAGTCGAAACCCAGGGCCA--CACGGTTACCCCAAGCAGGATCAGCTATTTAATCGTAGCCT

AGGATTACCATATGAGTACTCCGATGATGGACAGCAGGAAGTAGGCAGGAAGGGGCAGCCACGCTTCCCCGAAGGGGGATCAGCTATTTAAGGGAATCCT
AGGATTACCATATGAGTACTCCGATGATGGACAGCAGGAAGTAGGCAGGAAGGGGCAGCCACGCTTCCCCGAAGGGGGATCAGCTATTTAAGGGAATCCT

AGCATAACCTTATGAGTACACCGATGATAGACAGCTGGAGGTAGGAACCCAGGGTCAGCCAGGCTTACCCCTAGGGGGATTCGATATTTAATCGAAGCCT
AGCATAACCTTATGAGTACACCGATGATAGACAGCT-GAGGTAGGAACCCAGGGTCAGCCAGGCTTACCCCTAGGGGGATTCGATATTTAATCGAAGCCT

AGGATAACCACATGAGTACTCCGATGATAGACAGCTGGAAGTAGGATCCCAGGGGCAGCCACGCTTACCCCAAGGGGGATTCGCTATTTAATCGAAGCCT
AGGATAACCACATGAGTACTCCGATGATAGACAGCTGGAAGTAGGATCCCAGGGGCAGCCACGCTTACCCCAAGGGGGATTC-CTATTTAATCGAAGCCT

AGGATAACCATATGAGTACTCAGAAGATAGAGGGCTGGAAGAAGGAACCCAGGGGCAGCCACGCTTACCCCAAGTGGAATTAGCTGTTTAATCGAAGCCT
AGGATAACCATATGAGTACTCAGAAGATAGAGGGCTGGAAGAAG-AACCCAGGGGCAGCCACGCTTACCCCAAGTGGAATTAGCTGTTTAATCGAAGCCT

AGGATAACCATCCGTGTACTCCGACTATAGACAGCTGGAAGTAGGAACCCTGGGGCAGCCACGCATACCCCAAGGGGGATTTTTTATTTAATCGAAGCCT
AGGATAACCATCCG-GTACTCCGACTATAGACAGCTGGAAGTAGGAACCCTGGGGCAGCCACGCATACCCCAAGGGGGATTTTTTATTTAATCGAAGCCT
```

Figure 2: Performing Deletions



```
Performing Insertions


Generated Blocks

GGGCAAACCCTAGAAGTACTCAGCTGATCGGGAGCTGAAAGTAG-----CAATCCTGTGGCACCCAAGCTTACCCCAATTGGGACTACCAATTTAAACAAGGCCT

ATGAAAAGCATAAGAGTAGGCCGATACTAGCCACCTTGAAGTAGGGTGGGAACGCTCCGGAAGCCACG-TTAGCCCTAGGGGTATTACCTATTTATTCGTAGCTT

GGGCTATCCATATGAGTGCCCCGTAGATAGACATCTGAAAGTAG-----G-ACCCAGGGGCAGCCACGCTTACCCCAAGGAGGATGAGCTAATTAATATAATCCC

ATGATAACCATATGAGT--TCGCAAGTTTGACCGCTGGAGGTAG-----GAACAGACGGGCAGCCACGCTTATCCCACGGGGGCGTACCGATTAGATCCCAGCCT

AGGATAACCATATGAGTACTCTGATGATAGATAGCTGGAAGTCG-----AAACCCAGGGCCA--CACGGTTACCCCAAGCAGGATCAGCTATTTAATCGTAGCCT


AGGATTACCATATGAGTACTCCGATCTGATGGACAGCAGGAAGTAGGCAGGAAGGGGCAGCCACGCTTCCCCGAAGGGGGATCAGCTATTTAAGGGAATCCT

AGCATAACCTTATGAGTACACCGA--TGATAGACAGCT-GAGGTAGGAACCCAGGGTCAGCCAGGCTTACCCCTAGGGGGATTCGATATTTAATCGAAGCCT

AGGATAACCACATGAGTACTCCGA--TGATAGACAGCTGGAAGTAGGATCCCAGGGGCAGCCACGCTTACCCCAAGGGGGATTC-CTATTTAATCGAAGCCT

AGGATAACCATATGAGTACTCAGA--AGATAGAGGGCTGGAAGAAG-AACCCAGGGGCAGCCACGCTTACCCCAAGTGGAATTAGCTGTTTAATCGAAGCCT

AGGATAACCATCCG-GTACTCCGA--CTATAGACAGCTGGAAGTAGGAACCCTGGGGCAGCCACGCATACCCCAAGGGGGATTTTTTATTTAATCGAAGCCT
```

Figure 3: Performing Insertion

The 2nd part of the project consists of the following:

- We are then implementing a multi-alignment procedure with the traditional sum-of-pairs scoring strategy. To keep it simple we are not going to use affine gap penalty. We are using the provided scoring matrix T.
- The algorithm being used for **"Sum of Scores" Scoring Strategy** is as follows:

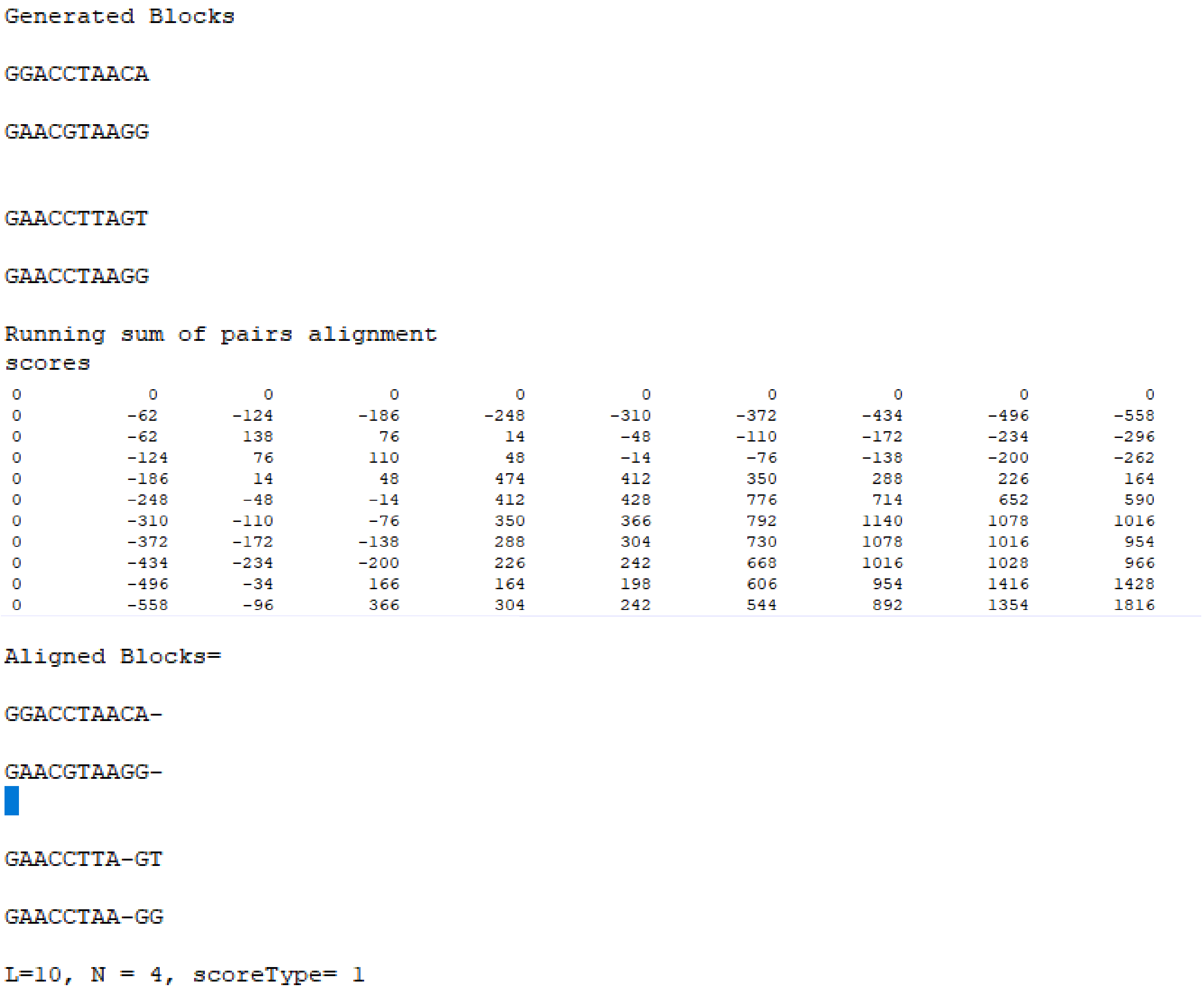
$$s_{i,j} = max \begin{cases} s_{i\text{-}1,j\text{-}1} + \delta\,(v_i, w_j) \\ s_{\,i\text{-}1,j} + \delta\,(v_i, \text{-}) \\ s_{\,i,j\text{-}1} + \delta\,(\text{-}, w_j) \end{cases}$$

where v_i is the ith column of the first alignment block. w_j is the jth column of the second alignment block. $\delta\ (v_i, w_j)$ is the sum of all pairs of scores where the two letters of a pair are from v_i and w_j respectively.

$$\delta\,(v_i, w_j) = \sum_{x \in v_i, y \in w_j} T[x, y]$$

For example, *vi* is composed of {x1, x2, x3} and *vj* is composed of {y1, y2, y3, y4}. Then $\delta\ (v_i, w_j)$=T[x1,y1] + T[x1,y2] + T[x1,y3] + T[x1,y4] + T[x2,y1] + T[x2, y2] + T[x2, y3] + T[x2,y4] + T[x3,y1] + T[x3,y2] + T[x3,y2] + T[x3,y3] + T[x3,y4].
$\delta\ (v_i, \text{-})$ is calculated similarly to $\delta\ (v_i, w_j)$, just the letters in the whole column of w_j are all '- 's. $\delta\ (\text{-}, w_j)$ is calculated similarly to $\delta\ (v_i, w_j)$, just the letters in the whole column of v_i are all '- 's.

- But the time complexity for this is $O(k1 \times k2 \times m \times n)$ where m and n are the numbers of columns of each input alignment block.



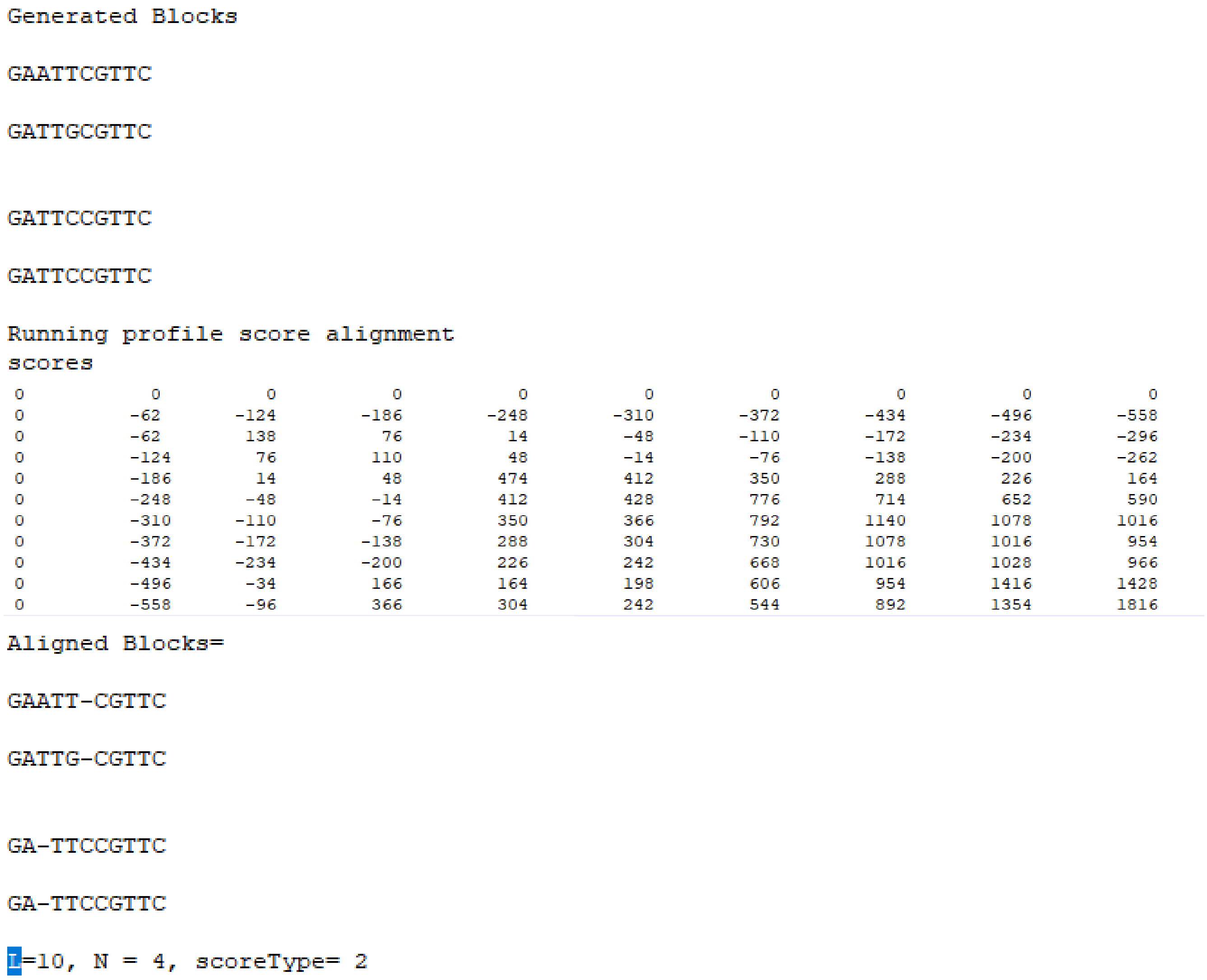
```
Generated Blocks

GGACCTAACA

GAACGTAAGG


GAACCTTAGT

GAACCTAAGG

Running sum of pairs alignment
scores
0      0     0     0     0     0     0     0     0     0
0    -62  -124  -186  -248  -310  -372  -434  -496  -558
0    -62   138    76    14   -48  -110  -172  -234  -296
0   -124    76   110    48   -14   -76  -138  -200  -262
0   -186    14    48   474   412   350   288   226   164
0   -248   -48   -14   412   428   776   714   652   590
0   -310  -110   -76   350   366   792  1140  1078  1016
0   -372  -172  -138   288   304   730  1078  1016   954
0   -434  -234  -200   226   242   668  1016  1028   966
0   -496   -34   166   164   198   606   954  1416  1428
0   -558   -96   366   304   242   544   892  1354  1816

Aligned Blocks=

GGACCTAACA-

GAACGTAAGG-


GAACCTTA-GT

GAACCTAA-GG

L=10, N = 4, scoreType= 1
```

3rd Part of the Project is as follows:

- So, in order to reduce the time complexity, we are going to implement profile scoring strategy using the same recurrence and the same scoring matrix as above.
- Given the counts of A, C, G, T, and '- 'in v_i: $c^v{}_A$, $c^v{}_C$, $c^v{}_G$, $c^v{}_T$, $c^v{}_-$ and such counts in w_j: $c^w{}_A$, $c^w{}_C$, $c^w{}_G$, $c^w{}_T$, $c^w{}_-$, we have the sum-of-pairs scoring:

$$\delta\ (v_i, w_j)\ = \sum_{x,y\in\{A,C,G,T,-\}} c_x^v \times c_y^w \times T[x,y]$$

- For the same pair of columns *vi* and *wj*, the traditional and profile sum-of-pairs scoring strategies should give the same score.
- With this algorithm we are going to reduce the time complexity to $O((k1+k2)\times m\times n)$



```
Generated Blocks

GAATTCGTTC

GATTGCGTTC


GATTCCGTTC

GATTCCGTTC

Running profile score alignment
scores
0          0         0         0         0         0         0         0         0         0
0        -62      -124      -186      -248      -310      -372      -434      -496      -558
0        -62       138        76        14       -48      -110      -172      -234      -296
0       -124        76       110        48       -14       -76      -138      -200      -262
0       -186        14        48       474       412       350       288       226       164
0       -248       -48       -14       412       428       776       714       652       590
0       -310      -110       -76       350       366       792      1140      1078      1016
0       -372      -172      -138       288       304       730      1078      1016       954
0       -434      -234      -200       226       242       668      1016      1028       966
0       -496       -34       166       164       198       606       954      1416      1428
0       -558       -96       366       304       242       544       892      1354      1816
Aligned Blocks=

GAATT-CGTTC

GATTG-CGTTC


GA-TTCCGTTC

GA-TTCCGTTC

L=10, N = 4, scoreType= 2
```

4.) Programming Language Used:

Used the Command g++ -o <execution file name> <program name.cc> to compile the program.

I have modified the previous project's shell script so, I can execute for all the parameters at a time, instead of compiling it every time, by changing the parameter. I will attach the shell script as well when uploading the source code.

To execute, I use the command time <executable file name> <N Parameter value> <L Parameter Value> <option1 / option 2> <D or No D >

Option1 is Sum of Score, Option 2 is Profile Score, D is debug where we get the score matrix, if we don't use D then we will just the time and not the generated string and score matrix.

5.) Findings:

When I executed the following findings were made:

a.) For "Sum of Scores" Scoring Strategy:

Parameters	Real Time	User Time	System Time
N = 10, L = 800	0m3.298s	0m3.244s	0m0.012s
N = 10, L = 900	0m4.195s	0m4.128s	0m0.024s
N = 10, L = 1000	0m5.295s	0m5.244s	0m0.032s
N = 20, L = 800	0m10.368s	0m10.312s	0m0.026s
N = 20, L = 900	0m13.189s	0m13.128s	0m0.041s
N = 20, L = 1000	0m16.981s	0m16.884s	0m0.045s
N = 30, L = 800	0m21.370s	0m21.308s	0m0.048s
N = 30, L = 900	0m27.196s	0m27.108s	0m0.054s
N = 30, L = 1000	0m35.162s	0m35.104s	0m0.059s
N = 40, L = 800	0m36.107s	0m36.044s	0m0.053s
N = 40, L = 900	0m46.136s	0m46.080s	0m0.074s
N = 40, L = 1000	0m59.721s	0m59.648s	0m0.088s
N = 50, L = 800	0m58.271s	0m58.468s	0m0.086s
N = 50, L = 900	1m12.010s	1m11.928s	0m0.093s
N = 50, L = 1000	1m30.008s	1m29.932s	0m0.097s
N = 60, L = 800	1m17.708s	1m17.628s	0m0.096s
N = 60, L = 900	1m40.815s	1m40.740s	0m0.112s
N = 60, L = 1000	2m8.211s	2m8.108s	0m0.127s

N = 70, L = 800	1m44.491s	1m44.424s	0m0.118s
N = 70, L = 900	2m15.262s	2m15.188s	0m0.128s
N = 70, L = 1000	2m50.745s	2m50.676s	0m0.145s
N = 80, L = 800	2m26.614s	2m26.536s	0m0.122s
N = 80, L = 900	2m46.618s	2m46.314s	0m0.150s
N = 80, L = 1000	3m52.876s	3m52.776s	0m1.015s
N = 90, L = 800	2m52.327s	2m52.260s	0m0.129s
N = 90, L = 900	3m37.158s	3m37.072s	0m0.203s
N = 90, L = 1000	4m41.398s	4m41.264s	0m1.544s
N = 100, L = 800	4m4.719s	4m4.616s	0m0.206s
N = 100, L = 900	4m30.150s	4m30.092s	0m0.256s
N = 100, L = 1000	5m46.084s	5m45.924s	0m2.264s

b.) For "Profile Score" Scoring Strategy:

Parameters	**Real Time**	**User Time**	**System Time**
N = 10, L = 800	0m1.760s	0m1.716s	0m0.006s
N = 10, L = 900	0m2.264s	0m2.172s	0m0.011s
N = 10, L = 1000	0m2.778s	0m2.720s	0m0.020s
N = 20, L = 800	0m3.210s	0m3.152s	0m0.016s
N = 20, L = 900	0m4.077s	0m4.012s	0m0.028s
N = 20, L = 1000	0m5.108s	0m5.076s	0m0.037s
N = 30, L = 800	0m4.633s	0m4.584s	0m0.037s
N = 30, L = 900	0m5.791s	0m5.744s	0m0.046s

N = 30, L = 1000	0m7.466s	0m7.412s	0m0.050s
N = 40, L = 800	0m5.990s	0m5.964s	0m0.047s
N = 40, L = 900	0m7.823s	0m7.740s	0m0.054s
N = 40, L = 1000	0m9.917s	0m9.840s	0m0.063s
N = 50, L = 800	0m7.735s	0m7.684s	0m0.066s
N = 50, L = 900	0m9.778s	0m9.620s	0m0.072s
N = 50, L = 1000	0m12.164s	0m12.096s	0m0.077s
N = 60, L = 800	0m8.946s	0m8.840s	0m0.074s
N = 60, L = 900	0m11.396s	0m11.316s	0m0.083s
N = 60, L = 1000	0m14.684s	0m14.476s	0m0.092s
N = 70, L = 800	0m10.329s	0m10.284s	0m0.096s
N = 70, L = 900	0m13.796s	0m13.744s	0m0.102s
N = 70, L = 1000	0m16.779s	0m16.716s	0m0.118s
N = 80, L = 800	0m11.830s	0m11.780s	0m0.109s
N = 80, L = 900	0m15.150s	0m15.100s	0m0.126s
N = 80, L = 1000	0m20.876s	0m20.816s	0m0.182s
N = 90, L = 800	0m13.361s	0m13.296s	0m0.114s
N = 90, L = 900	0m17.218s	0m17.164s	0m0.142s
N = 90, L = 1000	0m23.287s	0m23.224s	0m0.232s
N = 100, L = 800	0m14.683s	0m14.628s	0m0.162s
N = 100, L = 900	0m19.020s	0m18.968s	0m0.198s
N = 100, L = 1000	0m24.196s	0m24.132s	0m0.328s

5.) Observation / Summary:

- We observe that for both the algorithms when we increase the N and L value the time taken to compute increases.

- But the time taken to compute the scores by using **“Profile Score” Scoring Strategy** is way less compared to **“Sum of Scores” Scoring Strategy.**

- This is ideal because the time complexity increases ***linearly*** when using ***“Profile Score” Scoring Strategy***, whereas the time complexity increases **quadratically** when using **“Sum of Scores” Scoring Strategy.**

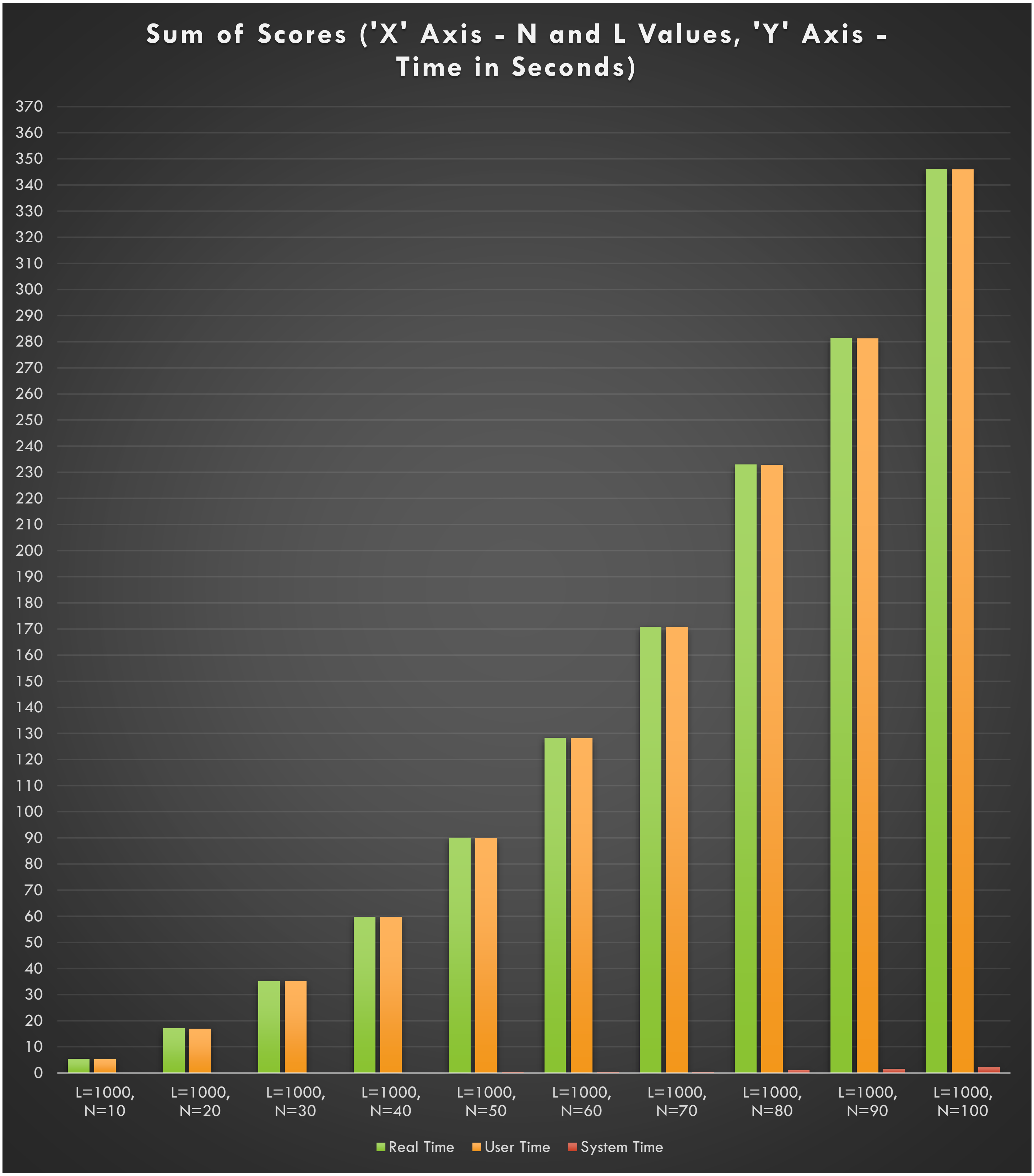
- The time complexity for “**Profile Score” Scoring Strategy** is $O((k1+k2) \times m \times n)$

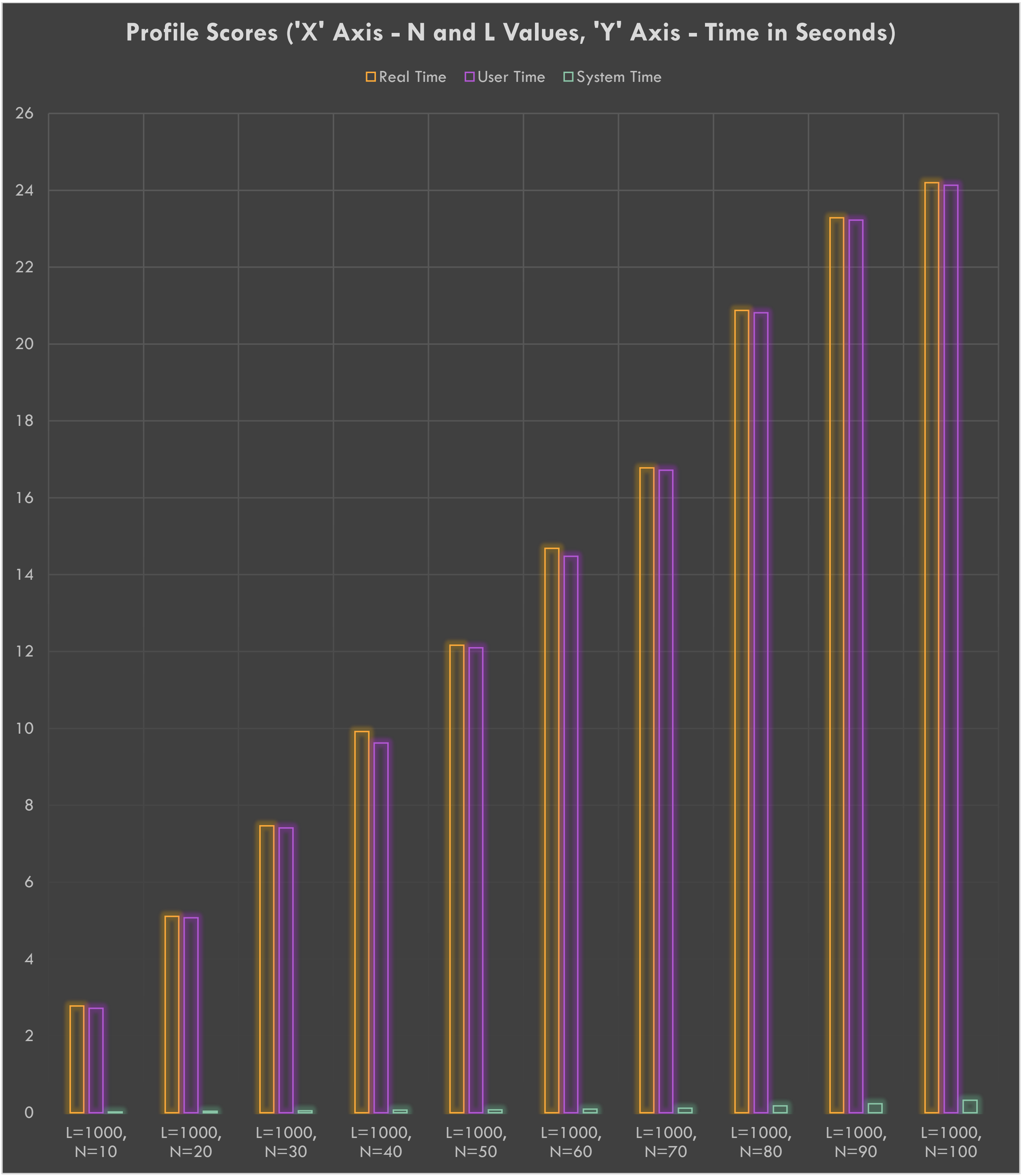
- The time complexity for **“Sum of Scores” Scoring Strategy** it is $O(k1 \times k2 \times m \times n)$.

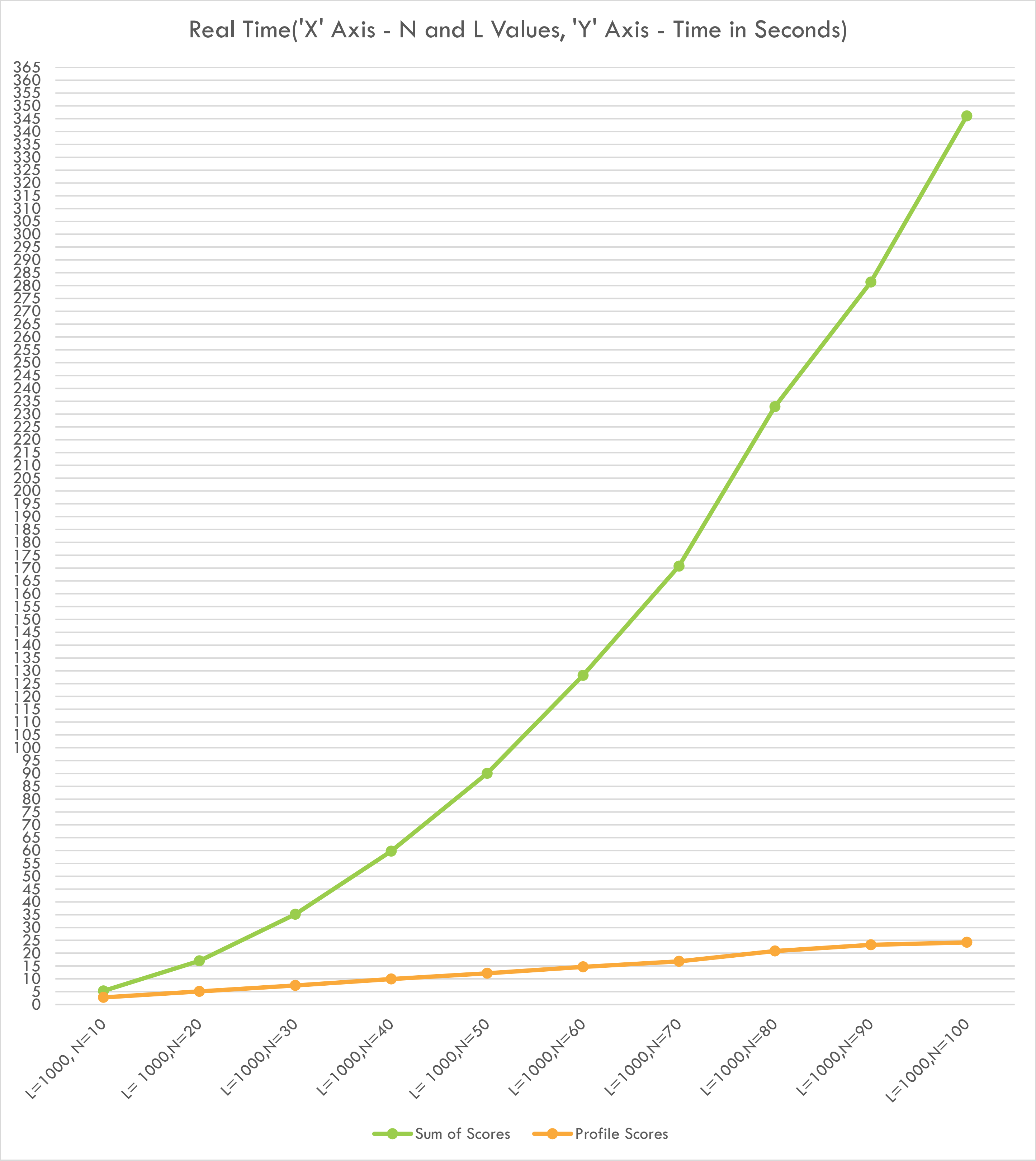
- For *“Sum of Scores” Scoring Strategy* we can see a steep curve up in the value of system time between N = 70 and N = 80.

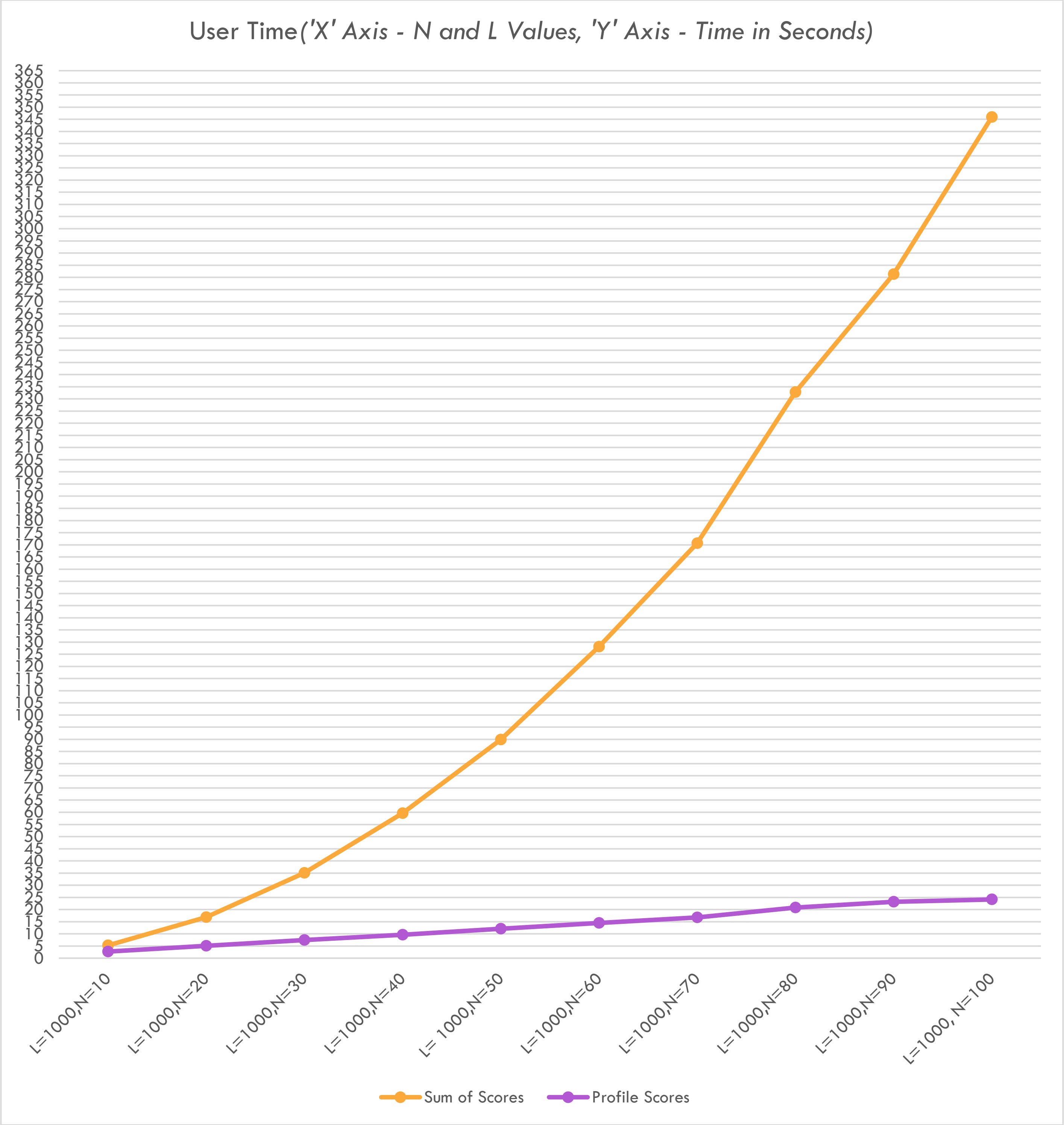
- Below are the graphical representations:

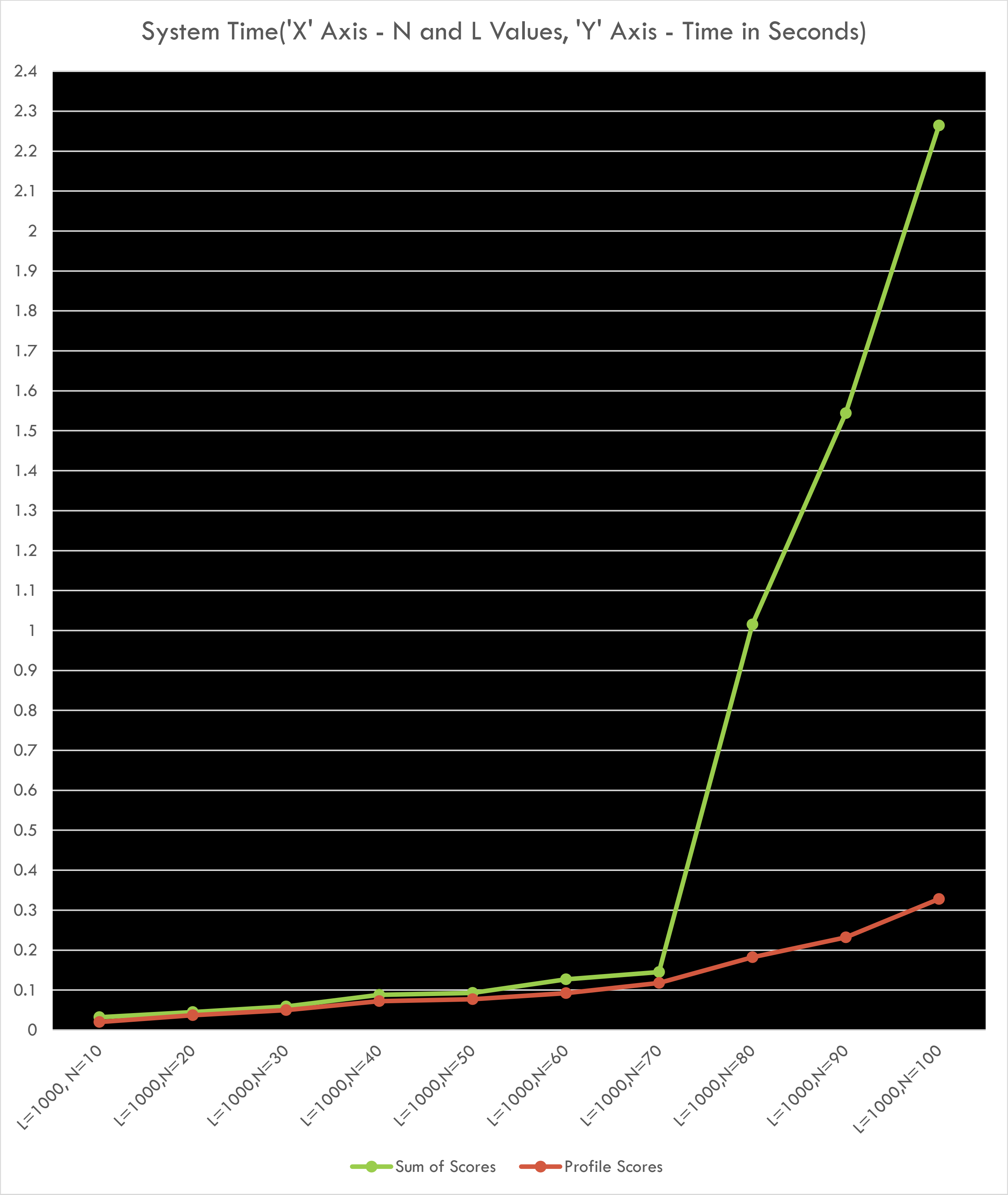
- I have graphically represented values only for L = 1000, so we can easily identify the amount of time the algorithm takes to output the scores. Also, by taking the max values, we can get the desired curve in the graph.


Sum of Scores ('X' Axis - N and L Values, 'Y' Axis - Time in Seconds)
370
360
350
340
330
320
310
300
290
280
270
260
250
240
230
220
210
200
190
180
170
160
150
140
130
120
110
100
90
80
70
60
50
40
30
20
10
0
L=1000, N=10
L=1000, N=20
L=1000, N=30
L=1000, N=40
L=1000, N=50
L=1000, N=60
L=1000, N=70
L=1000, N=80
L=1000, N=90
L=1000, N=100
🟩 Real Time
🟧 User Time
🟥 System Time


Profile Scores ('X' Axis - N and L Values, 'Y' Axis - Time in Seconds)
Real Time
User Time
System Time
26
24
22
20
18
16
14
12
10
8
6
4
2
0
L=1000, N=10
L=1000, N=20
L=1000, N=30
L=1000, N=40
L=1000, N=50
L=1000, N=60
L=1000, N=70
L=1000, N=80
L=1000, N=90
L=1000, N=100


Real Time('X' Axis - N and L Values, 'Y' Axis - Time in Seconds)
365
360
355
350
345
340
335
330
325
320
315
310
305
300
295
290
285
280
275
270
265
260
255
250
245
240
235
230
225
220
215
210
205
200
195
190
185
180
175
170
165
160
155
150
145
140
135
130
125
120
115
110
105
100
95
90
85
80
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
L=1000, N=10
L= 1000,N=20
L=1000,N=30
L=1000,N=40
L= 1000,N=50
L=1000,N=60
L=1000,N=70
L=1000,N=80
L=1000,N=90
L=1000,N=100
Sum of Scores
Profile Scores


User Time('X' Axis - N and L Values, 'Y' Axis - Time in Seconds)
365
360
355
350
345
340
335
330
325
320
315
310
305
300
295
290
285
280
275
270
265
260
255
250
245
240
235
230
225
220
215
210
205
200
195
190
185
180
175
170
165
160
155
150
145
140
135
130
125
120
115
110
105
100
95
90
85
80
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
L=1000,N=10
L=1000,N=20
L=1000,N=30
L=1000,N=40
L= 1000,N=50
L=1000,N=60
L=1000,N=70
L=1000,N=80
L=1000,N=90
L=1000, N=100
Sum of Scores
Profile Scores


System Time('X' Axis - N and L Values, 'Y' Axis - Time in Seconds)
2.4
2.3
2.2
2.1
2
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
L=1000, N=10
L=1000,N=20
L=1000,N=30
L=1000,N=40
L=1000,N=50
L=1000,N=60
L=1000,N=70
L=1000,N=80
L=1000,N=90
L=1000,N=100
Sum of Scores
Profile Scores