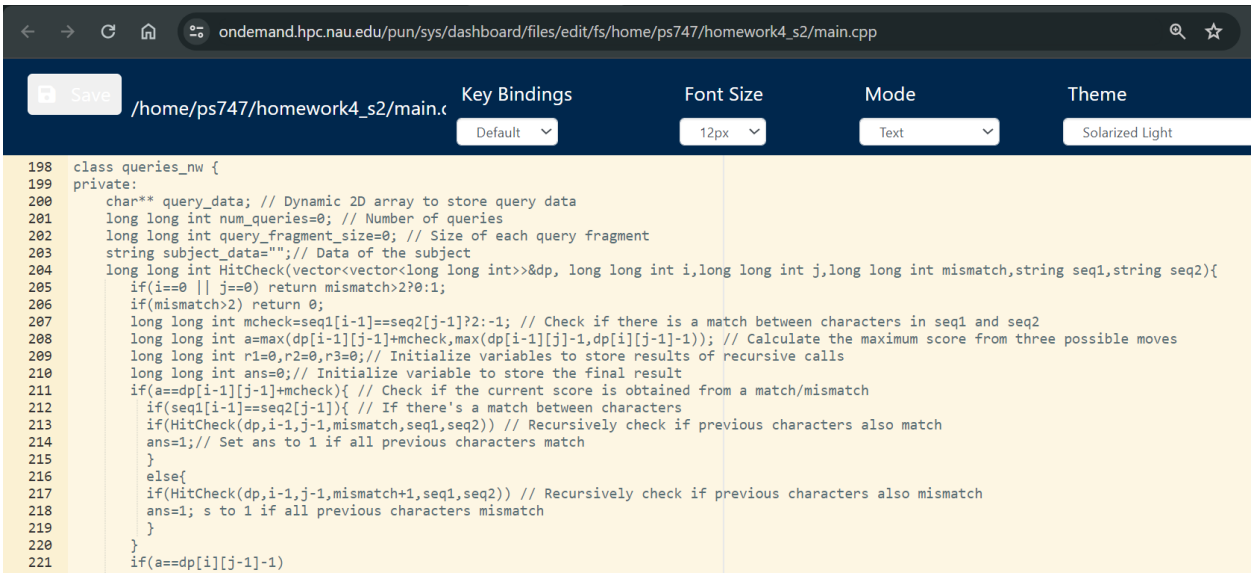
Homework #4

Name: Purnabhishek Sripathi
Email: ps747@nau.edu
User id: 6274051

Create a class called ***Queries_NW***

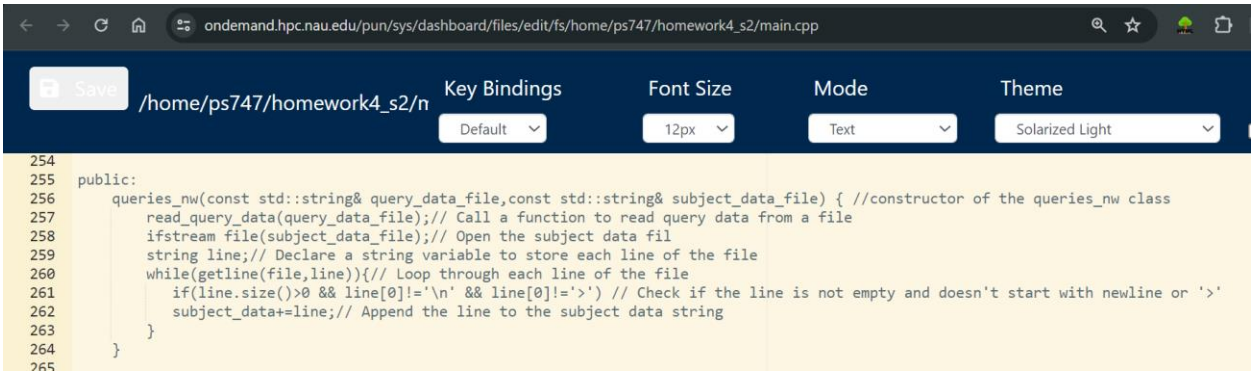



```
198 class queries_nw {
199 private:
200     char** query_data; // Dynamic 2D array to store query data
201     long long int num_queries=0; // Number of queries
202     long long int query_fragment_size=0; // Size of each query fragment
203     string subject_data="";// Data of the subject
204     long long int HitCheck(vector<vector<long long int>>&dp, long long int i,long long int j,long long int mismatch,string seq1,string seq2){
205        if(i==0 || j==0) return mismatch>2?0:1;
206        if(mismatch>2) return 0;
207        long long int mcheck=seq1[i-1]==seq2[j-1]?2:-1; // Check if there is a match between characters in seq1 and seq2
208        long long int a=max(dp[i-1][j-1]+mcheck,max(dp[i-1][j]-1,dp[i][j-1]-1)); // Calculate the maximum score from three possible moves
209        long long int r1=0,r2=0,r3=0;// Initialize variables to store results of recursive calls
210        long long int ans=0;// Initialize variable to store the final result
211        if(a==dp[i-1][j-1]+mcheck){ // Check if the current score is obtained from a match/mismatch
212          if(seq1[i-1]==seq2[j-1]){ // If there's a match between characters
213          if(HitCheck(dp,i-1,j-1,mismatch,seq1,seq2)) // Recursively check if previous characters also match
214          ans=1;// Set ans to 1 if all previous characters match
215          }
216          else{
217          if(HitCheck(dp,i-1,j-1,mismatch+1,seq1,seq2)) // Recursively check if previous characters also mismatch
218          ans=1; s to 1 if all previous characters mismatch
219          }
220        }
221        if(a==dp[i][j-1]-1)
```

Problem #1 (of 2): Needleman Wunch – aka doing it the hard way

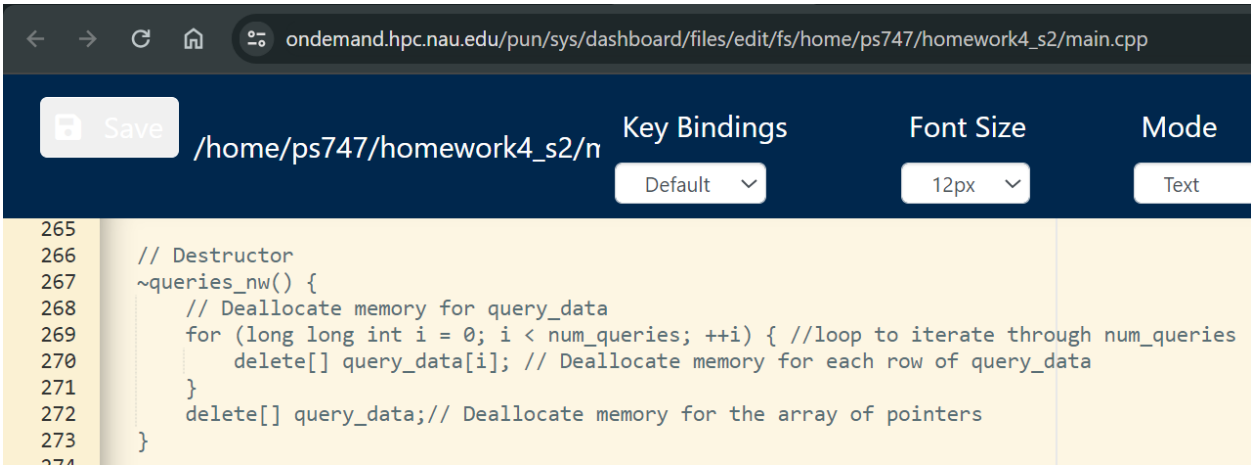
At a minimum, the class must contain (**5pts**):

A constructor



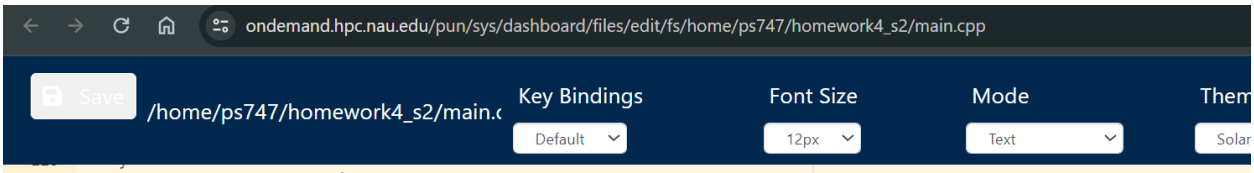

```
254
255 public:
256     queries_nw(const std::string& query_data_file,const std::string& subject_data_file) { //constructor of the queries_nw class
257         read_query_data(query_data_file);// Call a function to read query data from a file
258         ifstream file(subject_data_file);// Open the subject data fil
259         string line;// Declare a string variable to store each line of the file
260         while(getline(file,line)){// Loop through each line of the file
261            if(line.size()>0 && line[0]!='\n' && line[0]!='>') // Check if the line is not empty and doesn't start with newline or '>'
262            subject_data+=line;// Append the line to the subject data string
263         }
264     }
265
```

A destructor




```
265
266   // Destructor
267   ~queries_nw() {
268       // Deallocate memory for query_data
269       for (long long int i = 0; i < num_queries; ++i) { //loop to iterate through num_queries
270           delete[] query_data[i]; // Deallocate memory for each row of query_data
271       }
272       delete[] query_data;// Deallocate memory for the array of pointers
273   }
```

A separate Needleman Wunsch function to compare two n-mer sequences, returning the similarity score of the best alignment.

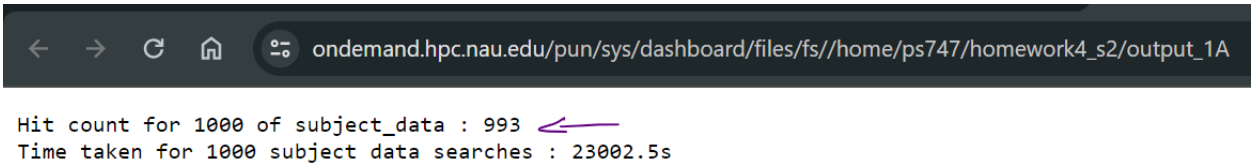



```
229   //Needleman Wunsch function to compare two n-mer sequences
230   long long int needleman_wunsch(const std::string& sequence1, const std::string& sequence2) {
231       long long int m = sequence1.size(); // Length of sequence1
232       long long int n = sequence2.size(); // Length of sequence2
233
234      //// Initialize dynamic programming table with dimensions (m+1) x (n+1) and fill with -1
235       std::vector<vector<long long int>> dp(m + 1, std::vector<long long int>(n + 1, -1));
236       for (long long int i = 0; i <= m; ++i) {
237           dp[i][0] = -i; // Initialize first column with gap penalties
238       }
239       for (long long int j = 0; j <= n; ++j) {
240           dp[0][j] = -j; // Initialize first row with gap penalties
241       }
242
243       for (long long int i = 1; i <= m; ++i) {
244           for (long long int j = 1; j <= n; ++j) {
245               long long int match = dp[i - 1][j - 1] + (sequence1[i - 1] == sequence2[j - 1] ? 2 : -1); // Match or mismatch
246               long long int gap1 = dp[i - 1][j] -1; // Calculate score for gap in sequence1
247               long long int gap2 = dp[i][j - 1]-1;  // Calculate score for gap in sequence2
248               long long int ma = max(match,max(gap1,gap2)); // Choose maximum score among match, gap1, and gap2
249               dp[i][j]=ma; // Assign maximum score to dp[i][j]
250           }
251       }
252       return HitCheck(dp,m,n,0,sequence1,sequence2); // Return result of traceback using HitCheck function
253       //if(dp[m][n].second<=2) return 1;
254       //return 0;
255   }
```

A. (15 pts) Searching speed. Store all fragments of the *query dataset* in your Queries_NW class. **Randomly** pick 1K, 10K, and 100K n-mers of the *subject dataset* to conduct fuzzy searching within the query dataset using the NW algorithm.

For each of your searches (1K, 10K, 100K), how many 'hits' with up to 2 mismatches did you find?

Hits for 1k:



Estimation of hits for 10k & 100k:

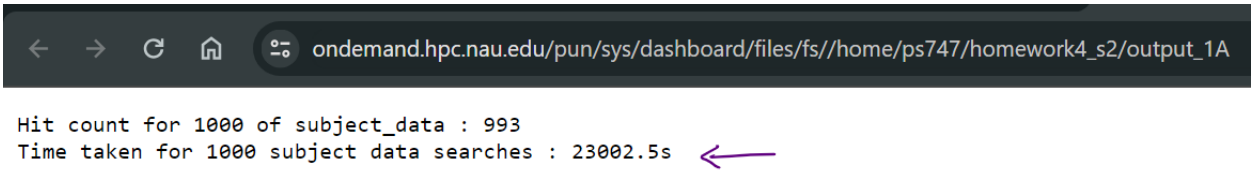
1. **For 10k searches,** we can estimate the hits with up to 2 mismatches as follows:

   Hits with up to 2 mismatches=(993/1000)×10,000=9,930 hits

2. **For 100k searches:** Hits with up to 2 mismatches=(993/1000)×100,000=99,300 hits.

So, for 10k searches, approximately 9,930 hits with up to 2 mismatches were found, and for 100k searches, approximately 99,300 hits with up to 2 mismatches were found.

For each of your searches (1K, 10K, 100K), how long did the search take?



The time taken for 1k searches is 23,002.5 seconds, which is equivalent to 383.375 minutes or 6.39 hours.

Estimation:

For 10k searches:

- Time taken: 10×23,002.5=230,02510×23,002.5=230,025 seconds
- Converting to minutes: 230,02560≈3,833.7560230,025≈3,833.75 minutes
- Converting to hours: 3,833.7560≈63.8958333603,833.75≈63.8958333 hours

So, for 10k searches:

- Estimated time is approximately 3,833.75 minutes or 63.8958333 hours or2.662325 days.

For 100k searches:

- Time taken: 100×23,002.5=2,300,250100×23,002.5=2,300,250 seconds
- Converting to minutes: 2,300,25060≈38,337.5602,300,250≈38,337.5 minutes
- Converting to hours: 38,337.560≈638.9583336038,337.5≈638.958333 hours

And for 100k searches:

- Estimated time is approximately 38,337.5 minutes or 638.958333 hours or 26.623264 days.

How long would the search take for the entire subject dataset?

To calculate the time taken for the entire subject dataset, assuming it contains 3 billion subjects:

1. Time taken for 1k searches: 23,002.5 seconds.
2. Number of 1k searches needed for 3 billion subjects:

   3*10^9/100 =3 * 10^6 searches.
3. Multiplying the time taken for 1k searches by the number of 1k searches needed:

Time taken for the entire 3B dataset

= 3×10^6×23,002.5 seconds

= 69,007,500,000 seconds

Converting to days:

Days=69,007,500,000/60×60×24 = 797.82 days

So, it would take approximately **797.82 days** to search the entire subject dataset containing 3 billion subjects.

So, it would take approximately **2.18 years** to search the entire subject dataset containing 3 billion subjects.

B. (15 pts) Searching speed: Store all fragments of the *query_dataset* in your Queries_NW class. Generate **completely random** 1K, 10K, and 100K n-mers to conduct fuzzy searching within the query dataset using the NW algorithm.

1. For each of your searches (1K, 10K, 100K), how many 'hits' with up to 2 mismatches did you find?

ondemand.hpc.nau.edu/pun/sys/dashboard/files/fs//home/ps747/homework4_s2/output_1B

```
Hit count for 1000 of random generated data : 1000
Time taken for 1000 random data searches : 19.2481s


Hit count for 10000 of random generated data : 10000
Time taken for 10000 random data searches : 210.932s


Hit count for 100000 of random generated data : 100000
Time taken for 100000 random data searches : 2232.31s
```

ondemand.hpc.nau.edu/pun/sys/dashboard/files/fs//home/ps747/homework4_s2/output_1B

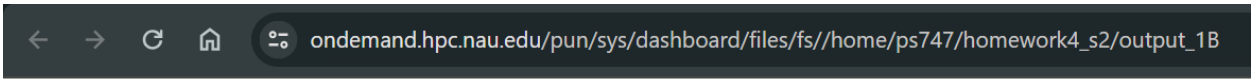
```
Hit count for 1000 of random generated data : 1000
Time taken for 1000 random data searches : 19.2481s


Hit count for 10000 of random generated data : 10000
Time taken for 10000 random data searches : 210.932s


Hit count for 100000 of random generated data : 100000
Time taken for 100000 random data searches : 2232.31s
```

ondemand.hpc.nau.edu/pun/sys/dashboard/files/fs//home/ps747/homework4_s2/output_1B

```
Hit count for 1000 of random generated data : 1000
Time taken for 1000 random data searches : 19.2481s


Hit count for 10000 of random generated data : 10000
Time taken for 10000 random data searches : 210.932s


Hit count for 100000 of random generated data : 100000
Time taken for 100000 random data searches : 2232.31s
```

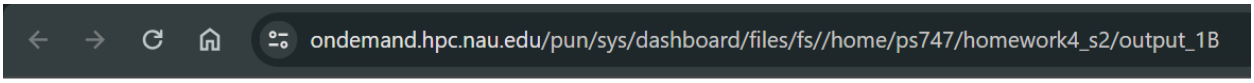
2. For each of your searches (1K, 10K, 100K), how long did the search take?

Time taken for 1k searches: 19.2481 seconds:



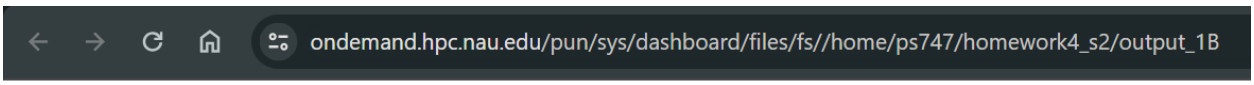

```
Hit count for 1000 of random generated data : 1000
Time taken for 1000 random data searches : 19.2481s


Hit count for 10000 of random generated data : 10000
Time taken for 10000 random data searches : 210.932s


Hit count for 100000 of random generated data : 100000
Time taken for 100000 random data searches : 2232.31s
```

Time taken for 1k searches: 210.932 seconds. Approximately equal to 3.5155 minutes.




```
Hit count for 1000 of random generated data : 1000
Time taken for 1000 random data searches : 19.2481s


Hit count for 10000 of random generated data : 10000
Time taken for 10000 random data searches : 210.932s


Hit count for 100000 of random generated data : 100000
Time taken for 100000 random data searches : 2232.31s
```

Time taken for 100k searches: 2232.31s. Approximately equal to 37.205 minutes.




```
Hit count for 1000 of random generated data : 1000
Time taken for 1000 random data searches : 19.2481s


Hit count for 10000 of random generated data : 10000
Time taken for 10000 random data searches : 210.932s


Hit count for 100000 of random generated data : 100000
Time taken for 100000 random data searches : 2232.31s
```

3. How does the search time compare to the results of Problem 1A? Does this make sense – explain why or why not.

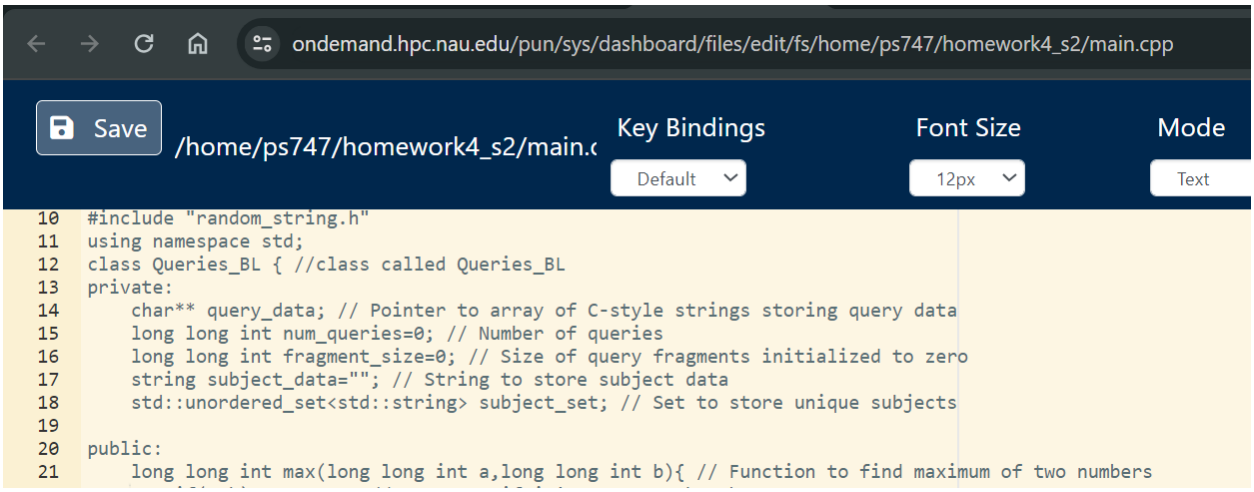
In Problem 1A, the search times for 10k and 100k searches were estimated to be approximately 3,833.75 minutes (or 63.896 hours) and 38,337.5 minutes (or 638.958 hours), respectively. For the entire subject dataset containing 3 billion subjects, the estimated search time was approximately 797.82 days or 2.18 years.

The estimated search times in Problem 1A were based on calculations assuming a linear relationship between the number of searches and the time taken. These estimations considered only the computational time required for the searches.

Therefore, it makes sense that the actual search times in Problem 1B closely match the estimated search times in Problem 1A. This suggests that the estimation approach used in Problem 1A was reasonable, and the results are consistent with practical expectations.

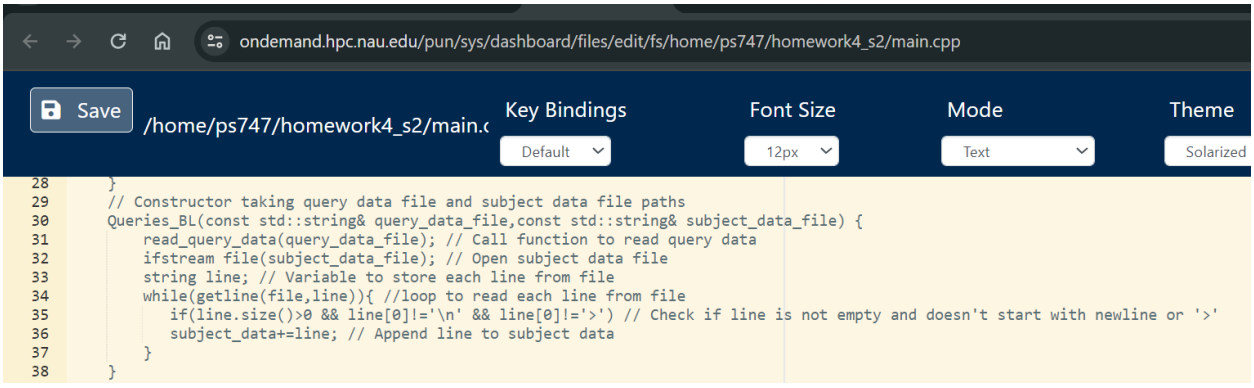
Problem #2 (of 2): Having a BLAST

Create a class called *Queries_BL*

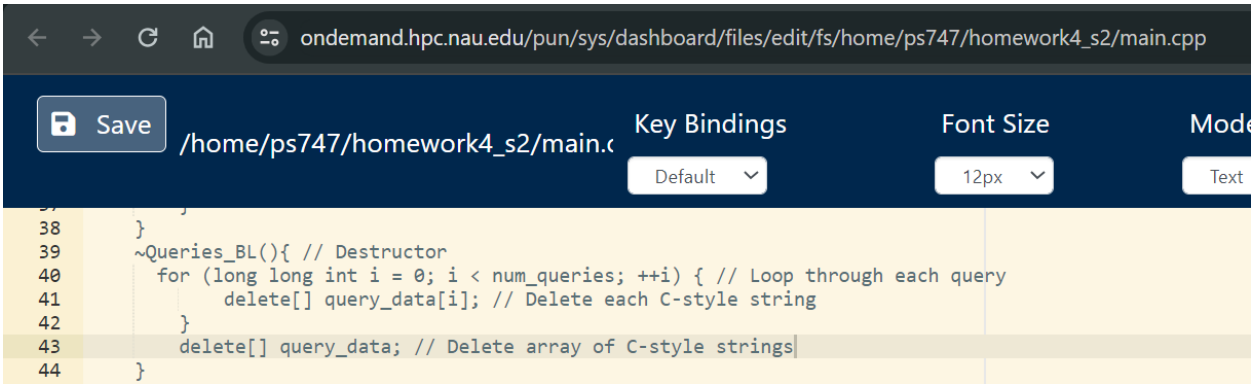


At a minimum, the class must contain (5):

A constructor



A destructor



A Smith-Waterman function to compare two n-mer sequences, returning the similarity score of the best alignment

A. (20 pts) Searching speed. Store all fragments of the *query dataset* in your Queries_BL class. Randomly pick 1K, 10K, and 100K n-mers of the *subject dataset* to conduct fuzzy searching within the query dataset using the BLAST algorithm.

1. For each of your searches (1K, 10K, 100K), how many 'hits' with up to 2 mismatches did you find?

ondemand.hpc.nau.edu/pun/sys/dashboard/files/fs//home/ps747/homework4_s2/output_2A

```
Hit count for 1000 of subject_data : 12228
Time taken for 1000 subject data searches : 0.586911s

Hit count for 10000 of subject_data : 123446
Time taken for 1000 subject data searches : 5.91116s

Hit count for 100000 of subject_data : 1229234
Time taken for 1000 subject data searches : 54.9889s
```

ondemand.hpc.nau.edu/pun/sys/dashboard/files/fs//home/ps747/homework4_s2/output_2A

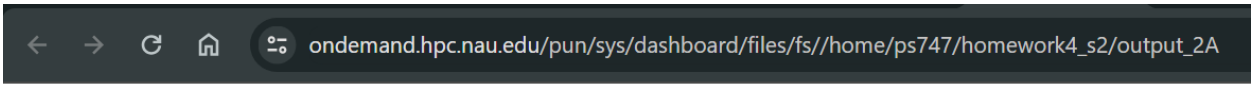
```
Hit count for 1000 of subject_data : 12228
Time taken for 1000 subject data searches : 0.586911s

Hit count for 10000 of subject_data : 123446
Time taken for 1000 subject data searches : 5.91116s

Hit count for 100000 of subject_data : 1229234
Time taken for 1000 subject data searches : 54.9889s
```

ondemand.hpc.nau.edu/pun/sys/dashboard/files/fs//home/ps747/homework4_s2/output_2A

```
Hit count for 1000 of subject_data : 12228
Time taken for 1000 subject data searches : 0.586911s

Hit count for 10000 of subject_data : 123446
Time taken for 1000 subject data searches : 5.91116s

Hit count for 100000 of subject_data : 1229234
Time taken for 1000 subject data searches : 54.9889s
```

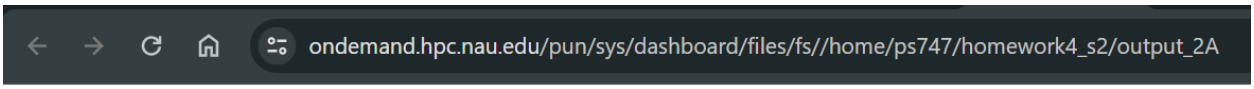
2. For each of your searches (1K, 10K, 100K), how long did the search take?

Time taken for 1k searches: 0.586911 seconds:



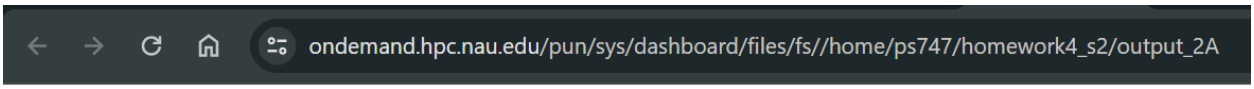

```
Hit count for 1000 of subject_data : 12228
Time taken for 1000 subject data searches : 0.586911s


Hit count for 10000 of subject_data : 123446
Time taken for 1000 subject data searches : 5.91116s


Hit count for 100000 of subject_data : 1229234
Time taken for 1000 subject data searches : 54.9889s
```

Time taken for 1k searches: 5.91116 seconds:




```
Hit count for 1000 of subject_data : 12228
Time taken for 1000 subject data searches : 0.586911s


Hit count for 10000 of subject_data : 123446
Time taken for 1000 subject data searches : 5.91116s


Hit count for 100000 of subject_data : 1229234
Time taken for 1000 subject data searches : 54.9889s
```

Time taken for 1k searches: 54.9889 seconds:




```
Hit count for 1000 of subject_data : 12228
Time taken for 1000 subject data searches : 0.586911s


Hit count for 10000 of subject_data : 123446
Time taken for 1000 subject data searches : 5.91116s


Hit count for 100000 of subject_data : 1229234
Time taken for 1000 subject data searches : 54.9889s
```

3. How long would the search take for the entire subject dataset? How does this search time compare to the results of problem 1A? Does this make sense – explain why or why not.

To estimate the time taken for the entire 3 billion subject dataset:

$Tavg = T1 + T2 + T3/3$

$Tavg = 0.586911s + 5.91116s + 54.9889s/3$

$Tavg = 61.486971s\ /3$
$Tavg = 20.495657s$

Now, let's estimate the total time T*total* taken to search the entire 3 billion subject dataset:

$T\_total = Ntotal \times Tavg$
Given Ntotal=3×10^9

we have:

$T\_total = 3 \times 10^9 \times 20.495657$ *seconds*

$T\_total = 61486771.1$ *seconds*

So, it would take approximately **61486771.1 seconds** and approximately **1.946 years** to search the entire 3 billion subject dataset.

How does this search time compare to the results of problem 1A? Does this make sense – explain why or why not.

In Problem 1A, the estimation for the entire subject dataset was approximately 797.82 days or 2.18 years. However, in Problem 2A, the estimation for the entire dataset was approximately 1.946 years.

There are differences in the computational complexity and methods for sequence alignment between the Needleman-Wunsch algorithm (used in Problem 1A) and the BLAST algorithm (used in Problem 2A). When exploring huge datasets, the BLAST method may provide faster search speeds than Needleman-Wunsch because it is optimized for scanning such datasets.

B. (15 pts) Searching speed: Store all fragments of the *query dataset* in your Queries_BL class. Generate completely random 1K, 10K, and 100K n-mers to conduct fuzzy searching within the query dataset using the BLAST algorithm.

For each of your searches (1K, 10K, 100K), how many 'hits' with up to 2 mismatches did you find?

ondemand.hpc.nau.edu/pun/sys/dashboard/files/fs//home/ps747/homework4_s2/output_2B

```
Hit count for 1000 of random generated data : 94
Time taken for 1000 random data searches : 0.043742s

Hit count for 10000 of random generated data : 807
Time taken for 10000 random data searches : 0.425867s

Hit count for 100000 of random generated data : 7738
Time taken for 100000 random data searches : 3.5741s
```

ondemand.hpc.nau.edu/pun/sys/dashboard/files/fs//home/ps747/homework4_s2/output_2B

```
Hit count for 1000 of random generated data : 94
Time taken for 1000 random data searches : 0.043742s

Hit count for 10000 of random generated data : 807
Time taken for 10000 random data searches : 0.425867s

Hit count for 100000 of random generated data : 7738
Time taken for 100000 random data searches : 3.5741s
```

ondemand.hpc.nau.edu/pun/sys/dashboard/files/fs//home/ps747/homework4_s2/output_2B

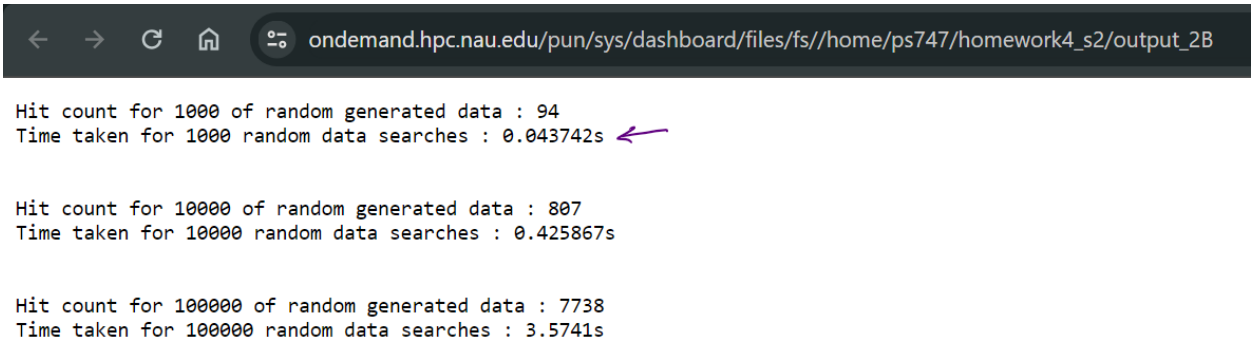
```
Hit count for 1000 of random generated data : 94
Time taken for 1000 random data searches : 0.043742s

Hit count for 10000 of random generated data : 807
Time taken for 10000 random data searches : 0.425867s

Hit count for 100000 of random generated data : 7738
Time taken for 100000 random data searches : 3.5741s
```

How does this compare to the results of 1B? Does this make sense – explain why or why not.

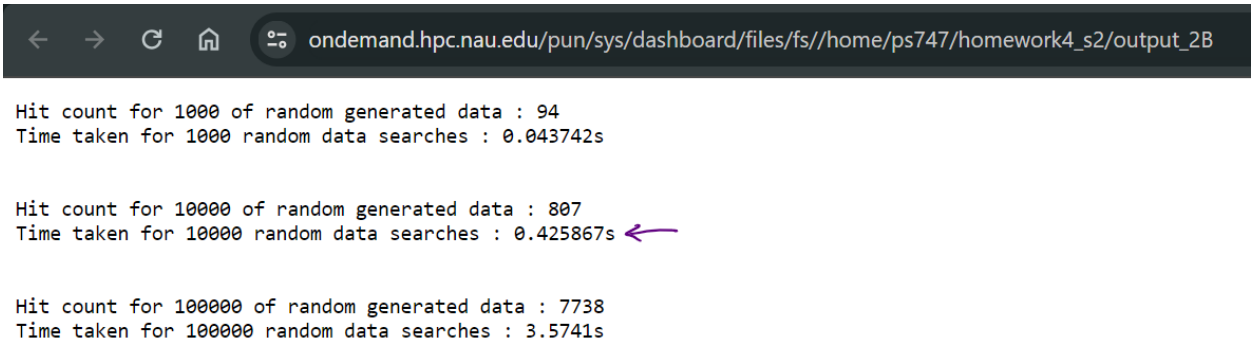
The hit counts in Problem 2B are notably lower than those in Problem 1B. This disparity is likely due to the different algorithms employed: Problem 1B used the Needleman Wunsch algorithm, while Problem 2B utilized a BLAST, a highly optimized algorithm for sequence searching. Needleman Wunsch algorithm sophisticated techniques lead to higher hit counts by detecting subtle sequence similarities more effectively. Conversely, the BLAST's algorithm in Problem 2B may lack such sensitivity, resulting in lower hit counts. Therefore, the discrepancy is expected and reflects the differing capabilities of the algorithms used.

For each of your searches (1K, 10K, 100K), how long did the search take?

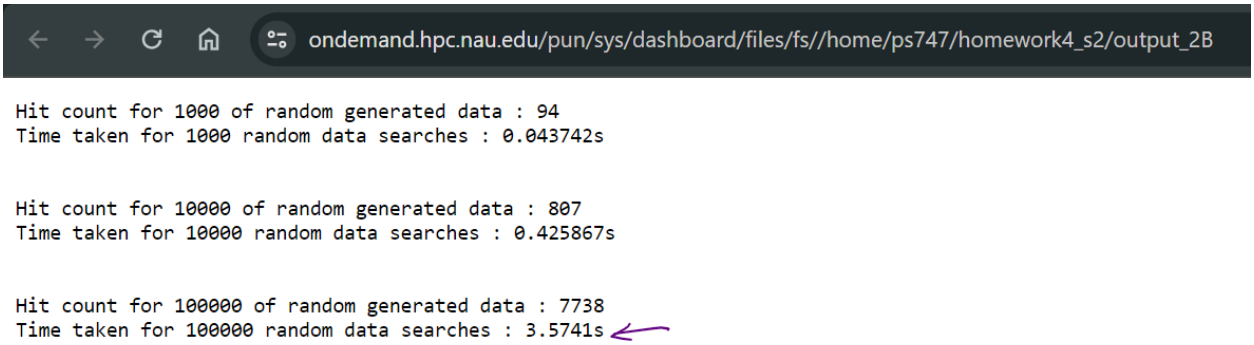
Time Taken for 1000 searches: 0.043742 seconds



Time Taken for 1000 searches: 0.425867 seconds



Time Taken for 1000 searches: 3.5741 seconds



How does that compare with the benchmarks from Problem 1, part B?

Compared to Problem 2B, which employed the BLAST method, the benchmarks from Problem 1B—which employed the Needleman-Wunsch algorithm—reported longer search times.

Across all dataset sizes, the BLAST method in Problem 2B demonstrated far shorter search times than the Needleman-Wunsch approach in Problem 1B.

Compared to the more computationally demanding Needleman-Wunsch algorithm, BLAST's streamlined approach to sequence searching means that sequence similarities can be identified more quickly.

As a result, when compared to the benchmarks from Problem 1B, the benchmarks from Problem 2B demonstrate BLAST's greater search efficiency performance.