Homework #2

Name: Purnabhishek Sripathi

Email: ps747@nau.edu

Userid: 6274051

Problem #1 (of 1)

1. A default constructor (that zeroes everything out)

```
 99  class Queries_AR
100  {
101  public:
102      vector<array<char,32>> queries; // Vector for query storage, with each query being a 32-character array
103      long long int size; //Declaring the variable to store the size
104
105      Queries_AR() // Default Constructor
106      {
107
108      }
109      Queries_AR(char *path) //Custom Constructor that takes file path as input and retrieves queries from a text document
110
111      {
112          long long int fileSize = get_size(path); // getting the size of the file
113          cout<<"queries filesize:"<<fileSize<<endl; // printing the size of the file
114          // fflush(stdout);
115          queries.reserve(fileSize/45); //Reserving the space in the vector by file size & expected entry length
116          // cout<<"check1 \n";
117          readDataSet(path);
118      }
```

2. At least one custom constructor (e.g. one taking a file path or ifstream as input)

```
 99  class Queries_AR
100  {
101  public:
102      vector<array<char,32>> queries; // Vector for query storage, with each query being a 32-character array
103      long long int size; //Declaring the variable to store the size
104
105      Queries_AR() // Default Constructor
106      {
107
108      }
109      Queries_AR(char *path) //Custom Constructor that takes file path as input and retrieves queries from a text document
110
111      {
112          long long int fileSize = get_size(path); // getting the size of the file
113          cout<<"queries filesize:"<<fileSize<<endl; // printing the size of the file
114          // fflush(stdout);
115          queries.reserve(fileSize/45); //Reserving the space in the vector by file size & expected entry length
116          // cout<<"check1 \n";
117          readDataSet(path);
118      }
```

3. A function to read the query dataset file.

```
    Queries_AR(char *path) //Custom Constructor that takes file path as input and retrieves queries from a text document

    {
        long long int fileSize = get_size(path); // getting the size of the file
        cout<<"queries filesize:"<<fileSize<<endl; // printing the size of the file
        // fflush(stdout);
        queries.reserve(fileSize/45); //Reserving the space in the vector by file size & expected entry length
        // cout<<"check1 \n";
        readDataSet(path);
    }
    void readDataSet(char* path){ // Function to read the query dataset file

        ifstream queryFile(path); // opening the file with given path
        int fileSize;
        string currLine; //Declaring the string currLine to store each line read from the file
        if (!queryFile.is_open())
        cout << "err: unable to open the file"; //print the error if the file cannot be opened
        bool val=false; //Declaring the boolean variable to alternate reading lines
        // cout<<"check2 \n";
        while (getline(queryFile, currLine)){ // while loop to read the lines from the file
            val=!val;
            if(val)
            continue;
            array<char,32> arr; // array to store the current query
            fileSize=33;
            while(--fileSize){
                arr[32-fileSize]=currLine[32-fileSize]; //Copying the chars from currLine to array
            }
            queries.emplace_back(arr); //Adding query to vector of queries
        }
        queries.shrink_to_fit(); //Shrink the vector to fit the actual number of elements
        // cout<<"check3 \n";
        fflush(stdout);
        cout << "\nqueries read successfully: " << queries.size();// Printing the no:of queries read
        queryFile.close();// Closing the file
    }
```

4. A search function designed to find a sequence fragment within class's data.

```
    }
    bool linear_search(char *c){ // Linear search for a query
        int temp;
        for(int i=0;i<queries.size();++i){ // Iterating through the queries
            temp=-1;
            while((++temp)<32&&queries[i][temp]==c[temp]); // Comparing the each character of the query
            if(temp==32)
            return true; // Returning true if all characters match
        }
        return false; // Returning false if no match is found
    }
    bool binary_search(char *c){ // Binary search for a query in the vector
        long long int start=0,end=queries.size();
        int temp; //declaring the temp variable
        while(start<=end){
            long long int mid= (end-start)/2+start; // calculating the middle index
            temp=-1; // initializing the temp to -1
            while((++temp)<32&&queries[mid][temp]==c[temp]); // while loop to compare each character of the query
            if(temp==32) // if statement to return true if all characters match
            return true;
            else if(queries[mid][temp]<c[temp]) // Move to the right half
            start=mid+1;
            else // Move to the left half
            end=mid-1;
        }
        return false; //return false if no match is found
    }
```

5. A function to sort the fragments of the Queries_AR object.

```
    }
    void sortQueries(){ // function to sort the fragments of the Queries_AR object.
        // cout<<"before sort"<<endl;
        // for(int i=0;i<10;++i){
        //     cout<<string(queries[i].begin(),queries[i].end())<<endl;
        // }
        //sort(queries.begin(),queries.end());
        // cout<<"after sort"<<endl;
        // for(int i=0;i<10;++i){
        //     cout<<string(queries[i].begin(),queries[i].end())<<endl;
        // }
        merge_sort(0, queries.size() - 1, queries); // merge sort method calling
    }
    ~Queries_AR(){} // queries_ar destructor
};

```

6. A destructor

```
    ~Queries_AR(){} // queries_ar destructor
};

class GNOME
{
    public:
    char *data = nullptr; // data pointer to store the gnome data
    long long int size = 0; // size variable to store the total size of the data
    GNOME(char *path) // Constructor that reads gnome data from a file
    {
        unsigned long long fileSize = get_size(path); // Getting size of the file and storing it in filesize
        cout<<"\ngnome size"<< fileSize<<endl; // Printing the size of the file
        fflush(stdout); // Flush the output buffer
        data = (char *)malloc(fileSize); // Allocating memory for the gnome data
        ifstream queryFile(path); // Open the file with the given path
        string currLine; // declaring string to store the each line read from the file
        currLine.reserve(100); // reversing the space for 100 characters
        if (!queryFile.is_open())  // condition to check if the file failed to open
            cout << "err: unable to open the file"; // print error message
        while (getline(queryFile, currLine)) //reading the file line by line until end of the file
        {
            if (currLine[0] != '>'){ // condition to check is the line doesnot start with >
                memcpy(data + size, currLine.c_str(), currLine.size()); //copying the current line content
                size+=currLine.size(); // incrementing the size
            }
        }
        cout<<"total chars read from gnome file:"<<size<<endl; // printing the total chars read from the gnome file
        queryFile.close(); // closing the file
    }
    ~GNOME() // destructor to free the allocated memory for gnome
    {
        free(data); // free the allocated memory
    }
};
```

A. (30 pts)

1. How long did it take you to search for the first 10K, 100K, and 1M 32-character long fragments of the *subject dataset* within the *query dataset*?

   Ans:

   Time taken to search for the first 10K:
   18624 seconds
   Approximately it takes 5.1733333 hours.

   Time taken to search for the first 100K:
   181741 seconds
   Approximately it takes 50.4836111 hours.

   Time taken to search for the first 1M: 504.836111 hours approx**.**

   I have calculated the approximate calculation for 1M is as below:

   Time taken to search for the first 1M = (Time taken for the first 100K fragments) * (1M / 100K)

   Time taken to search for first 1M fragments = 50.4836111 hours * (1,000,000 / 100,000)

   Time taken to search for first 1M fragments = 50.4836111 hours * 10

   Time taken to search for first 1M fragments = 504.836111 (**21 days approximately**)

2. **How long would it take to search for every possible 32-character long fragment of the *subject dataset* within the *query dataset*? Please note that depending on the efficiency of your algorithm, this step may take a long time. If the total time is greater than 24 CPU hours, provide an estimate rather than an exact number.**

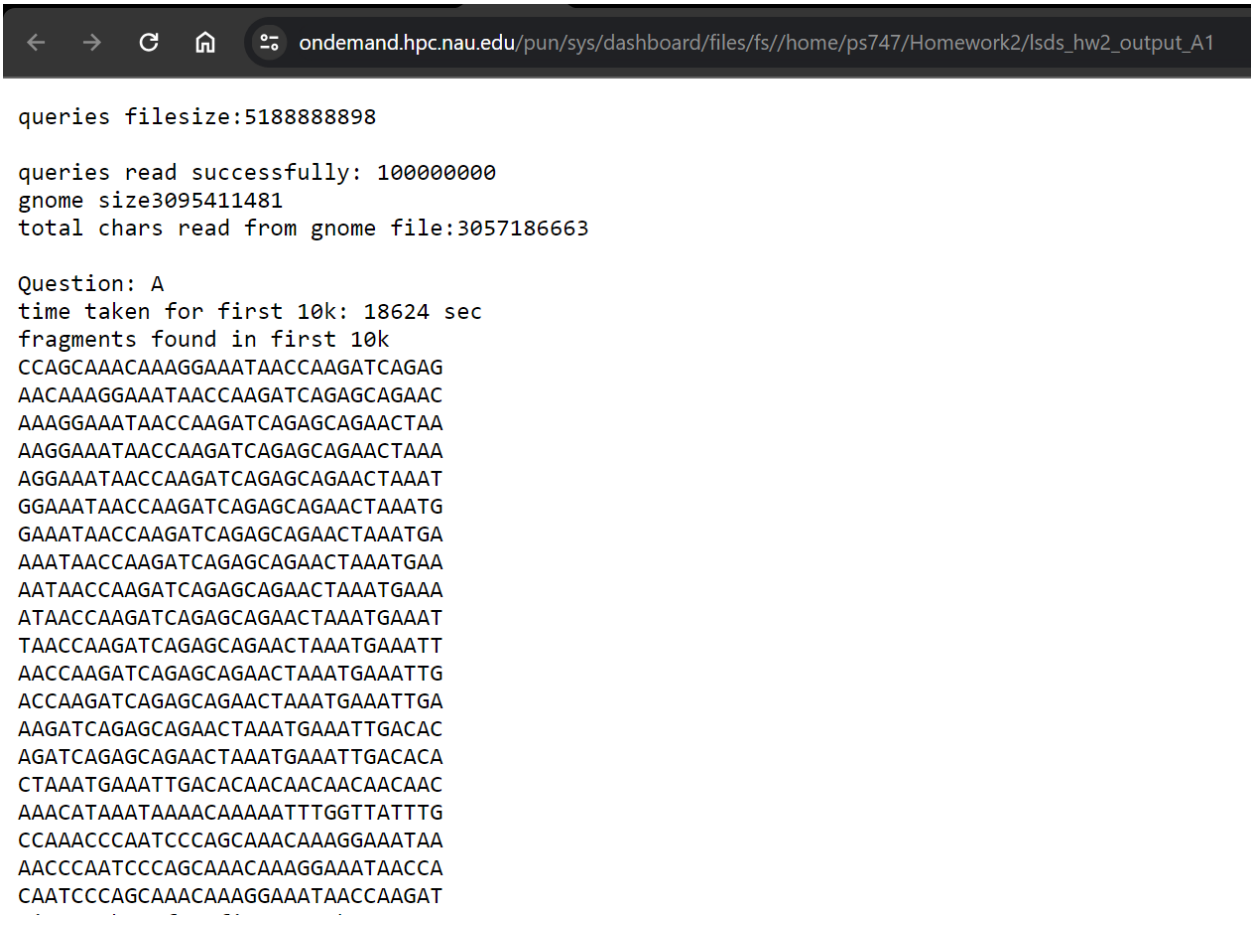
   Ans:
   The time required for every possible 32-character long fragment within the query data set, it takes approximately **1543378.21 hours**.

   To translate it in days it would take **64307.43 days** and if we converted it into years, it would take **176.18 years** to compute.

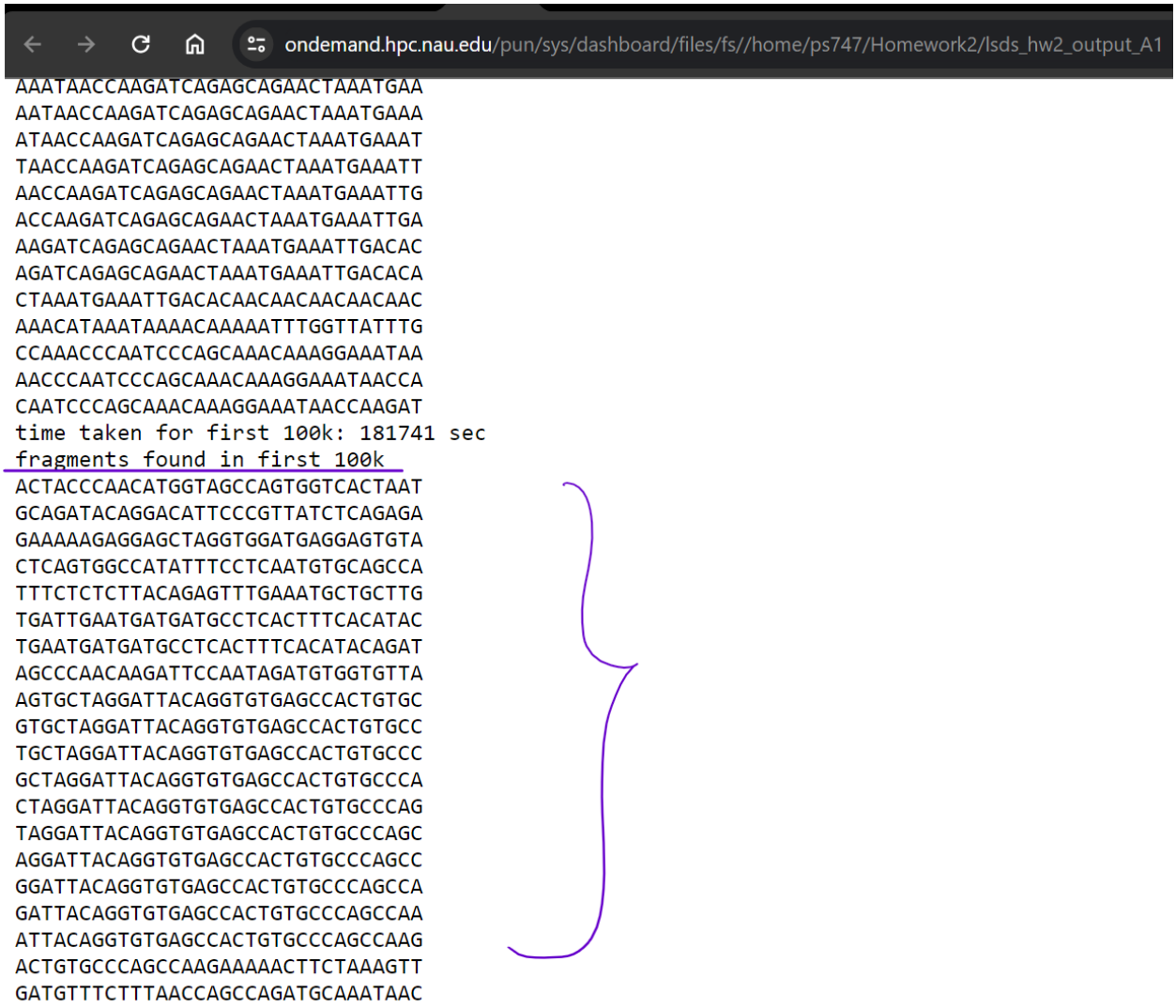
   As I have used Linear search here takes these many years to run the program.

3. **Print the 20 query fragments found within the *subject dataset* that have the largest indices (i.e. found later in the subject) for the first 10K, 100K, and 1M 32-character long fragments.**

   Screenshot that shows the last 20 fragments for 10k:



Screenshot that shows the last 20 fragments for 10k:



For 1M last 20 fragments:

The program is still running for the 1M 32-character long fragments. Due to this reason, I do not have the last 20 fragments for the 1M character long fragments.

B. (30 pts)

1: How long did it take you to search for the first 10K, 100K, and 1M 32-character long fragments of the *subject dataset* within the *query dataset*?

Ans:

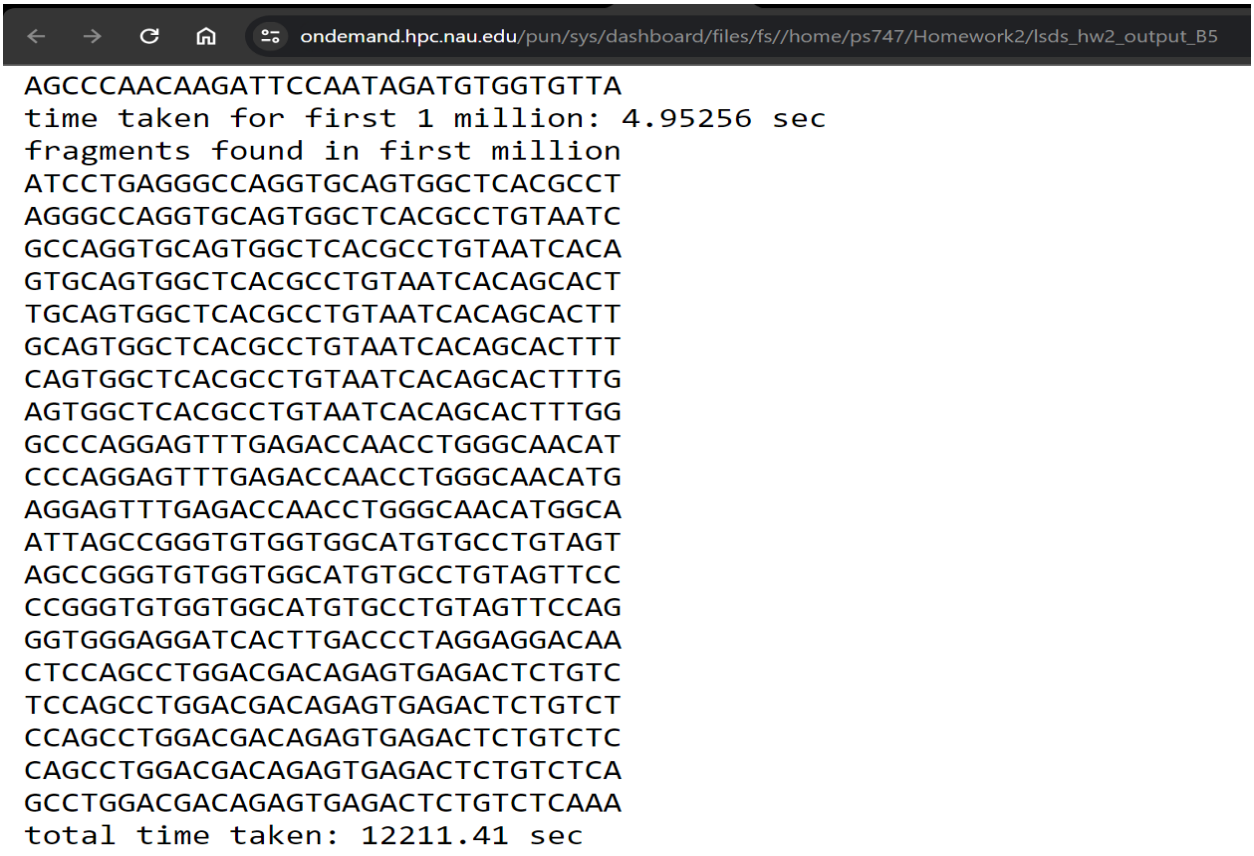
Time taken to search for the first 10K: 0.074871 seconds.

Time taken to search for the first 100K: 0.625749 seconds.

Time taken to search for the first 1M: 4.95256 seconds.

2: How long would it take to search for every possible 32-character long fragment of the *subject dataset* within the *query dataset*? Please note that depending on the efficiency of your algorithm, this step may take a long time. If the total time estimate is greater than 24 CPU hours, provide an estimate rather than exact number.

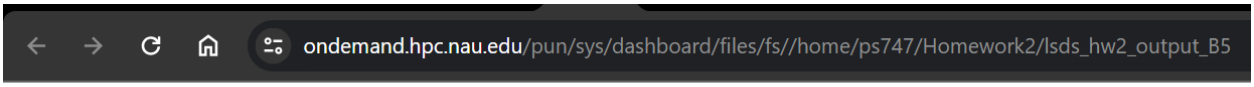
Total time taken is: 12211.41 seconds and it is approximately **3.39 hours.**



3: Print the 20 query fragments found within the *subject dataset* that have the largest indices (i.e. found later in the subject) for the first 10K, 100K, and 1M 32-character long fragments.

Ans:

Below is the screenshot for the 20 query fragments found within the first 10K.

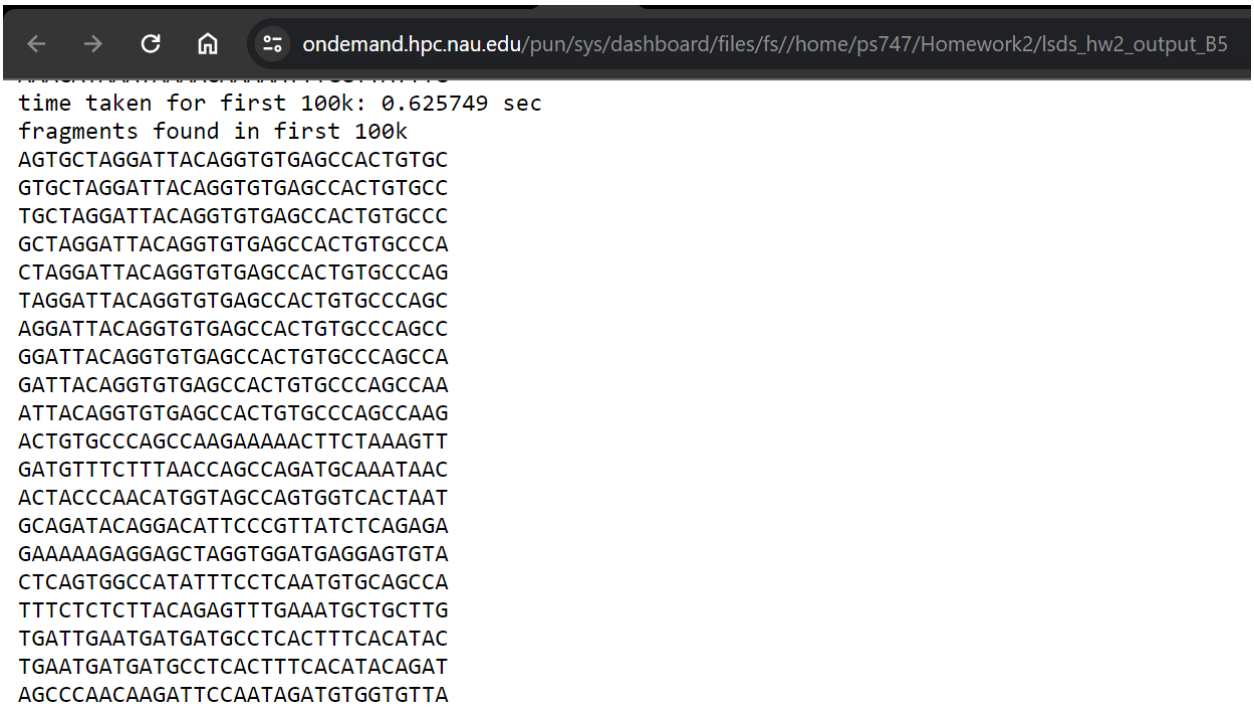



```
queries filesize:5188888898

queries read successfully: 100000000
gnome size3095411481
total chars read from gnome file:3057186663
started sorting
Queries Sorted

Question: B
time taken for first 10k: 0.074871 sec
fragments found in first 10k
CCAAACCCAATCCCAGCAAACAAAGGAAATAA
AACCCAATCCCAGCAAACAAAGGAAATAACCA
CAATCCCAGCAAACAAAGGAAATAACCAAGAT
CCAGCAAACAAAGGAAATAACCAAGATCAGAG
AACAAAGGAAATAACCAAGATCAGAGCAGAAC
AAAGGAAATAACCAAGATCAGAGCAGAACTAA
AAGGAAATAACCAAGATCAGAGCAGAACTAAA
AGGAAATAACCAAGATCAGAGCAGAACTAAAT
GGAAATAACCAAGATCAGAGCAGAACTAAATG
GAAATAACCAAGATCAGAGCAGAACTAAATGA
AAATAACCAAGATCAGAGCAGAACTAAATGAA
AATAACCAAGATCAGAGCAGAACTAAATGAAA
ATAACCAAGATCAGAGCAGAACTAAATGAAAT
TAACCAAGATCAGAGCAGAACTAAATGAAATT
AACCAAGATCAGAGCAGAACTAAATGAAATTG
ACCAAGATCAGAGCAGAACTAAATGAAATTGA
AAGATCAGAGCAGAACTAAATGAAATTGACAC
AGATCAGAGCAGAACTAAATGAAATTGACACA
CTAAATGAAATTGACACAACAACAACAACAAC
AAACATAAATAAAACAAAAATTTGGTTATTTG
```

Below is the screenshot for the 20 query fragments found within the first 100K.



Below is the screenshot for the 20 query fragments found within the first 1M.