main.cpp

```
/**
*       FileName: main.cpp
*       Author: James Muoghalu (j286m692@ku.edu)
*       Date: 3/11/18
*       Description: driver file for program
*/
#include <string>
#include <iostream>
#include <fstream>
#include "Hakimi.h"


int main(int argc, char** argv)
{

	Hakimi* program = new Hakimi();

	bool continue_loop = true;
	int input_choice = 0;
	while(continue_loop)
	{

		std::cout << "\n.....\n"
		<< "Pick One of the Following Options for Running Hakimi-Havel's Algorithm:"
		<< "\n1) Read from a Text File"
		<< "\n2) Exit Program"
		<< "\nYour Choice (1 or 2): ";
		std::cin >> input_choice;

		std::cout << std::endl;
		if(input_choice < 1 || input_choice > 2)
		{
			std::cout << "Invalid Input" << std::endl;
		}

		else if(input_choice == 1)
		{

			std::string file_name = "";
			std::string seq_input = "";

			std::cout << "Provide the Name of the Input File: ";
			std::cin >> file_name;
```

```
std::ifstream input_file(file_name);
while(std::getline(input_file, seq_input))
{

	// if there are empty lines in the input file
	if(seq_input == "\n" || seq_input == "\r")
	{
		continue;
	}

	delete program->sequence;
	program->sequence = new std::vector<int>();
	delete program->original_sequence;
	program->original_sequence = new std::vector<int>();

	bool valid_sequence = true;
	std::string sub = "";

	for(std::size_t i = 0; i < seq_input.length() ; i++)
	{

		// read the next digit character
		if(std::isdigit(seq_input[i]))
		{
			sub += seq_input[i];
		}

		// a number has been read for the sequence
		else if(seq_input[i] == ' ' && sub != "")
		{
			program->original_sequence->push_back(std::stoi(sub));
			program->sequence->push_back(std::stoi(sub));
			sub = "";
		}

		// an invalid string appears in the file
		else if(seq_input[i] != '\n' && seq_input[i] != '\0' && seq_input[i] !=
'\r')

		{
			valid_sequence = false;
			break;
		}
	}
	if(valid_sequence && sub != "")
	{
		program->original_sequence->push_back(std::stoi(sub));
```

```
                        program->sequence->push_back(std::stoi(sub));
                    }

                    if(!valid_sequence)
                    {
                        std::cout << "\n-----------------------------------------------------" <<
std::endl;
                        std::cout << "Error: All values in the sequence must be non-negative
integers" << std::endl;
                        std::cout << "-----------------------------------------------------\n" <<
std::endl;
                    }

                    // this line of input is valid, so the algorithm will run
                    else
                    {
                        program->quickSort(program->sequence, 0, (program->sequence-
>size()-1));
                        program->quickSort(program->original_sequence, 0, (program-
>original_sequence->size()-1));

                        std::cout << "\n-----------------------------------------------------" <<
std::endl;
                        std::cout << "Original Sequence: " << program->original_sequence-
>at(0);
                        for(std::size_t i = 1; i < program->original_sequence->size(); i++)
                        {
                            std::cout << " " << program->original_sequence->at(i);
                        }
                        std::cout << std::endl;

                        // check if the degree of the highest-degree vertex is greater than the
order of the graph
                        if( program->original_sequence->at(0) > (int)program-
>original_sequence->size() )
                        {
                            std::cout << "Result: The sequence is not graphical." << std::endl;
                        }
                        else
                        {
                            bool answer = program->runAlgorithm();
                            if(answer)
                            {
                                std::cout << "Result: The sequence is graphical.\n" <<
std::endl;
                            }
```

```
                                        else
                                        {
                                                std::cout << "Result: The sequence is not graphical." <<
std::endl;
                                        }
                                }

                                std::cout << "-------------------------------------------------------\n" <<
std::endl;
                        }
                }

        }

        else if(input_choice == 2)
        {
                std::cout << "Goodbye." << std::endl;
                continue_loop = false;
        }
        std::cout << std::endl;

    }



    delete program;
    return(0);

}
```

Hakimi.cpp

```
/**
*       FileName: Hakimi.cpp
*       Author: James Muoghalu (j286m692@ku.edu)
*       Date: 3/11/18
*       Description: implementation file for Hakimi class
*/

#include "Hakimi.h"
#define PDEBUG 0 // set to 0 or 1
/*
      #if PDEBUG == 1
      #endif
*/
Hakimi::Hakimi()
{
      this->sequence = nullptr;
      this->original_sequence = nullptr;
}


Hakimi::~Hakimi()
{
      delete this->sequence;
      delete this->original_sequence;
}


bool Hakimi::runAlgorithm()
{
      bool answer = true;
      bool end = false;

      #if (PDEBUG == 1)
            int reduction_number = 1;
      #endif

      while(!end)
      {
            int first = this->sequence->at(0);
            std::vector<int>* next_iteration = new std::vector<int>();

            std::size_t i = 1;

            // perform the algorithm arithmetic
```

```
            for(; i <= (std::size_t) first; i++)
            {
                  next_iteration->push_back((this->sequence->at(i))-1);
            }
            for(std::size_t j = i; j < this->sequence->size(); j++)
            {
                  next_iteration->push_back(this->sequence->at(j));
            }

            quickSort(next_iteration, 0, (next_iteration->size()-1));

            bool found_non_zero = false;
            for(std::size_t k = 0; k < next_iteration->size(); k++)
            {
                  if(next_iteration->at(k) > 0)
                  {
                        found_non_zero = true;
                        break;
                  }
            }
            if(found_non_zero) // continue to the next iteration of the while loop
            {
                  delete this->sequence;
                  this->sequence = next_iteration;
                  #if (PDEBUG == 1)
                        std::cout << "\nReduction " << reduction_number++ << ": " << this-
>sequence->at(0);
                        for(std::size_t i = 1; i < this->sequence->size(); i++)
                        {
                              std::cout << " " << this->sequence->at(i);
                        }
                        std::cout << std::endl;
                  #endif

            }

            else // else, ignore the new reduction and determine if this->sequence is graphical
            {
                  end = true;
                  delete next_iteration;
            }
      }

      if(this->sequence->at(0) >= (int) this->sequence->size())
      {
            return false;
```

```
	}

	int sum = 0;
	for(std::size_t i = 0; i < this->sequence->size(); i++)
	{
		sum +=  this->sequence->at(i);
	}
	if(sum % 2)
	{
		return false;
	}
	else
	{
		return true;
	}


	return answer;
}


// sort the given sequence
void Hakimi::quickSort(std::vector<int>* seq_to_sort, std::size_t low, std::size_t high)
{
	if(low < high && low >= 0 && high < seq_to_sort->size())
	{
		int pivot_index = partition(seq_to_sort, low, high);
		#if (PDEBUG == 1)
			//std::cout << "\t\t\tPivot Index: " << pivot_index << std::endl;
		#endif
		quickSort(seq_to_sort, low, (pivot_index-1));
		quickSort(seq_to_sort, (pivot_index+1), high);
	}
}

// quickSort helper function
int Hakimi::partition(std::vector<int>* seq_to_sort, std::size_t low, std::size_t high)
{
	int pivot = seq_to_sort->at(high);

	std::size_t i = low-1;
	for(std::size_t j = low; j < high; j++)
	{
		if( seq_to_sort->at(j) > pivot)
		{
			i++;
```

```
                int temp = seq_to_sort->at(i);
                seq_to_sort->at(i) = seq_to_sort->at(j);
                seq_to_sort->at(j) = temp;
            }
        }
        int temp = seq_to_sort->at(i+1);
        seq_to_sort->at(i+1) = seq_to_sort->at(high);
        seq_to_sort->at(high) = temp;

        return i+1;
}
```

Hakimi.h

```
/**
*       FileName: Hakimi.h
*       Author: James Muoghalu (j286m692@ku.edu)
*       Date: 3/11/18
*       Description: header file for Hakimi class
*/

#ifndef HAKIMI_H
#define HAKIMI_H

#include <iostream>
#include <vector>
#include <ctype.h>
#include <fstream>

class Hakimi
{

    public:

            /*
            * @brief
            * @param
            * @return
            */
        Hakimi();


        ~Hakimi();


        bool runAlgorithm();


        void quickSort(std::vector<int>* seq_to_sort, std::size_t low, std::size_t high);


        int partition(std::vector<int>* seq_to_sort, std::size_t low, std::size_t high);


        std::vector<int>* sequence;
        std::vector<int>* original_sequence;

};
```

```
#endif
```

Makefile

```
program := hakimi
directory := HomeworkExtra
submission := Muoghalu_ExtraCredit


$(program): main.o Hakimi.o
	g++ -std=c++11 -g -Wall $^ -o $(program)

main.o: main.cpp
	g++ -std=c++11 -g -Wall -c main.cpp

Hakimi.o: Hakimi.cpp
	g++ -std=c++11 -g -Wall -c Hakimi.cpp

clean:
	rm *.o $(program)
	echo clean done

clean2:
	rm *.zip *.tar.gz
	echo clean done

tar:
	mkdir $(submission)
	rsync --exclude=$(submission) --exclude=*.tar.gz --exclude=*.zip ../$(directory)/*
$(submission)
	tar cvzf $(submission).tar.gz $(submission)
	rm -rf $(submission)

zip:
	mkdir $(submission)
	rsync --exclude=$(submission) --exclude=*.tar.gz --exclude=*.zip ../$(directory)/*
$(submission)
	zip -r $(submission).zip $(submission)
	rm -rf $(submission)
```

data.txt

6 4 4 4 4 3 3 3 2 2 1

4 4 3 2 1 2 6 3 4 4 3

4 4 4 4 4 4 4

1 2 2 1 4 1 1

1 3 1 1   1 1 4

 1 1 1 1 6 1 1

2 2 3 3 3 3 2 2

3 3 2 2 4 2

2 1 -1
8 1 5 3 a 4 33

50 9 8 1