DAA UE23CS241B - Orange Problem

Implementation and Performance evaluation of Horspool and Boyer-Moore algorithms

1. **Horspool Algorithm**
   Implement the Horspool Algorithm, display the shift table contents and the starting indices of all the occurrences of the pattern in the text. Count the total number of comparisons involved.

2. **Boyer Moore Algorithm**
   Implement the Boyer Moore algorithm, display the good suffix shift table and the bad character shift table. Also print the starting indices of all the occurrences of the pattern in the text. Count the total number of comparisons involved.

3. **Performance Evaluation**
   Measure and compare execution times and number of comparisons for both algorithms and plot a graph to visualize their performance. (Hint: gnuplot can be used)

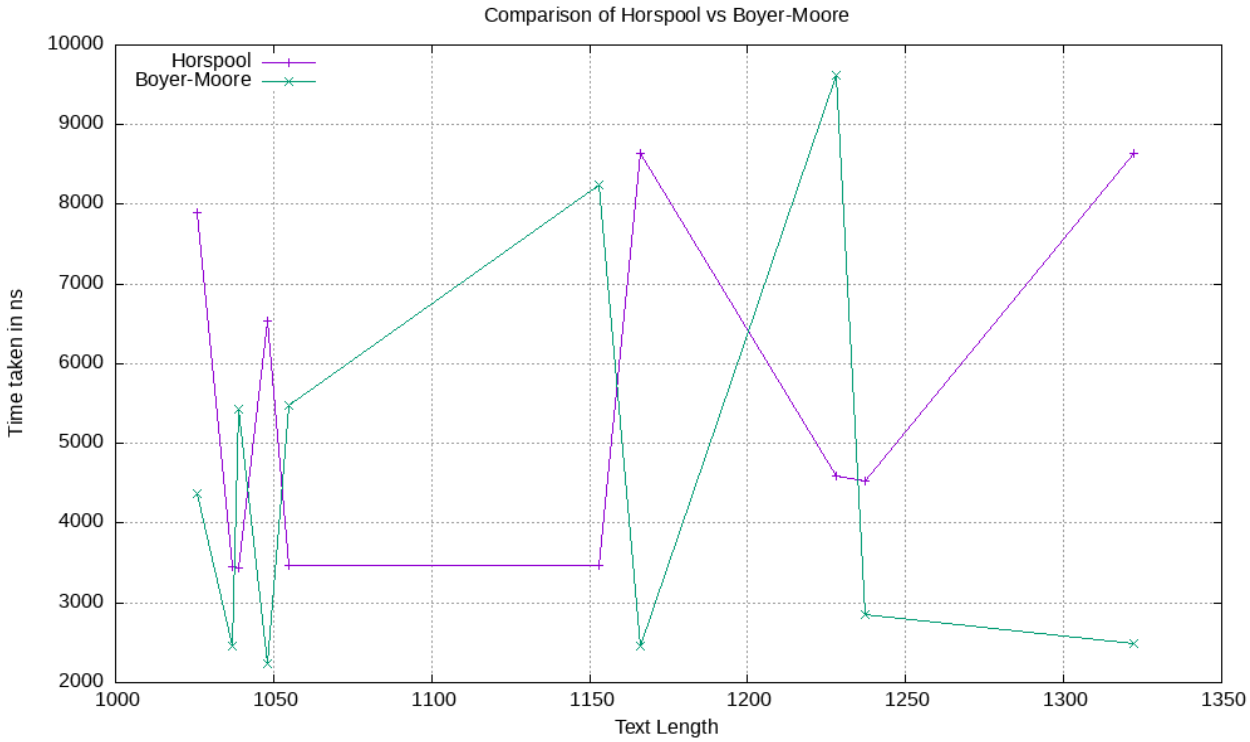
   Two graphs to be plotted:
   1. Text length vs Execution Time for given test case against both algorithms
   2. Text length vs Number of comparisons for both algorithms

Instructions:

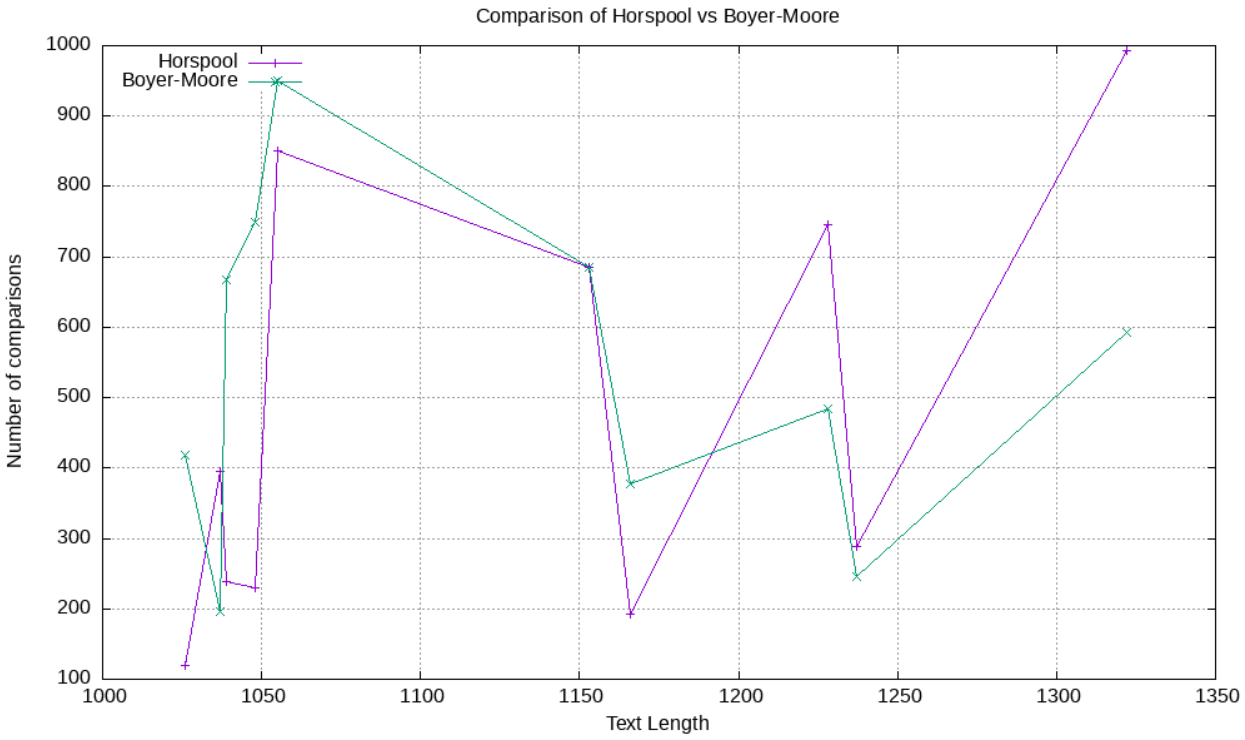
1. Boilerplate codes will handle the input, output and execution time calculation.
2. Input is available in ***input.txt***
3. Fill the code snippets in the TODO sections. The existing ***fprintf*** statements need NOT be touched.
4. The ***horspool_values.txt*** and ***boyermoore_values.txt*** files generated by the code can be used for plotting the graphs, sample graphs attached for reference (does not show the true relation between both algorithms).
5. Final files to be submitted - horspool_srn.c, boyermoore_srn.c, horspool_output_srn.txt, boyermoore_output_srn.txt, comparison_graph_srn.png and time_graph_srn.png (Naming convention is to be strictly followed)

Sample Graphs:

Time Plot:



Comparison Plot:



Boilerplate Code:

1. Horspool

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/time.h>
#include<time.h>

// Fill in the TODO sections as required and DO NOT TOUCH any of the
fprintf statements

void init_table(int* shift_table,int n)
{
	//TODO
}

// Construct the Bad Character Shift table
void preprocess(int* shift_table,char* pattern)
{
	//TODO
}

int string_match(int* shift_table,char* pattern,char* text,FILE*
output_file)
{
	// TODO Variables initialization
	int matches;
	int star_pos;
	char star_char;
	int cmp;
	int occurance;
	fprintf(output_file,"Occurrences:");
	while(star_pos<strlen(text))
	{
		//TODO find index of occurances
		if(matches==strlen(pattern))
		{
				fprintf(output_file,"%d,",index);
				//TODO
		}
	}
	fprintf(output_file,"\n");
	fprintf(output_file,"Comparisons:%d\n\n",cmp);
	return cmp;
}
```

```
void print_table(int* shift_table, FILE* output_file)
{
      fprintf(output_file,"BCST:\n");
      for(int i=0;i<26;i++)
      {
            fprintf(output_file,"%c:%d",(char)(i+97), shift_table[i]);
      }
}

void testcase(FILE* values_file, FILE* input_file, FILE* output_file)
{
      char text[2000];
      char pattern[100];
      fscanf(input_file,"%s",text);
      fscanf(input_file,"%s",pattern);

      int* shift_table=calloc(26,sizeof(int));
      init_table(shift_table,strlen(pattern));
      preprocess(shift_table,pattern);
      print_table(shift_table, output_file);
      clock_t start = clock();
      int cmp = string_match(shift_table,pattern,text,output_file);
      clock_t end = clock();
      int elapse=(int)(((double)(end-start))/CLOCKS_PER_SEC*1000000000);
//seconds to nanoseconds

fprintf(values_file,"%ld,%ld,%d,%d\n",strlen(pattern),strlen(text),cmp,ela
pse);
}

int main()
{
      FILE *input_file = fopen("input.txt", "r");
    FILE *output_file = fopen("horspool_output.txt", "w");
    FILE *values_file = fopen("horspool_values.txt", "w");

    if (!input_file || !output_file || !values_file) {
        printf("Error opening file!\n");
        return 1;
    }
    int testcases;
      fscanf(input_file,"%d",&testcases);
      int count = 0;
      fprintf(values_file, "patternlen,textlen,cmp,timetaken\n");
      while(count < testcases)
      {
            testcase(values_file, input_file, output_file);
```

```
            count += 1;
        }
        fclose(input_file);
        fclose(output_file);
        fclose(values_file);
        return 0;
}
```

2. Boyer Moore

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include<sys/time.h>
#include<time.h>

// Fill in the TODO sections as required and DO NOT TOUCH any of the
fprintf statements

// Function to create the Bad Character Shift Table
int* bcst_create(const char* pattern, int pattern_len) {
    // TODO BCST creation
    return bcst;
}

// Function to create the Good Suffix Shift Table
int* gsst_create(const char* pattern, int pattern_len) {
    int* gsst = (int*)malloc((pattern_len + 1) * sizeof(int));
    if (gsst == NULL) {
        perror("Memory allocation failed");
        exit(EXIT_FAILURE);
    }
    for (int x = 1; x <= pattern_len; x++) {
        int r2 = 0;
        char* suffix = (char*)malloc((x + 1) * sizeof(char));
        if (suffix == NULL) {
            perror("Memory allocation failed");
            exit(EXIT_FAILURE);
        }
            for(int i=0;i<x;i++)
            {
                  suffix[x-i-1] = pattern[pattern_len-i-1];
            }
            suffix[x] = '\0';

            char mis_char = (pattern_len - x - 1 >= 0) ?
pattern[pattern_len - x - 1] : '~';
```

```
        char* rev_pattern = (char*)malloc(pattern_len * sizeof(char));
        if (rev_pattern == NULL) {
            perror("Memory allocation failed");
            exit(EXIT_FAILURE);
        }
        for(int i=0;i<pattern_len-1;i++)
                 rev_pattern[i] = pattern[pattern_len-i-2];
        rev_pattern[pattern_len-1] = '\0';

        char* rev_suffix = (char*)malloc((x + 1) * sizeof(char));
        if (rev_suffix == NULL) {
            perror("Memory allocation failed");
            exit(EXIT_FAILURE);
        }
        for(int i=0;i<x;i++)
                 rev_suffix[i] = suffix[x-i-1];
        rev_suffix[x] = '\0';

        int count = 0;
        while (1) {
            char* pos_ptr = strstr(rev_pattern, rev_suffix);
            if (pos_ptr == NULL) {
                r2 = 1;
                break;
            }
            int pos = pos_ptr - rev_pattern;
            char check_char = (pos + x < pattern_len - 1) ?
rev_pattern[pos + x] : '`';
            if (check_char != mis_char) {
                gsst[x] = pos + count + 1;
                break;
            }
            rev_pattern += pos + 1;
            count += pos + 1;
        }

           char* suffix_ptr = suffix;
        if (r2) {
            char* prefix = (char*)malloc(pattern_len * sizeof(char));
            if (prefix == NULL) {
                perror("Memory allocation failed");
                exit(EXIT_FAILURE);
            }
            for (int i = 0; i < pattern_len; i++) {
                prefix[i] = pattern[i];
            }
            prefix[pattern_len] = '\0';
```

```
            while (1) {
                if (strlen(suffix) == 1) {
                    gsst[x] = pattern_len;
                    break;
                }
                suffix++;
                if (strncmp(prefix, suffix, strlen(suffix)) == 0) {
                    gsst[x] = pattern_len - strlen(suffix);
                    break;
                }
            }
            //free(prefix);
        }
        //free(suffix_ptr);
        //free(rev_suffix);
        //free(rev_pattern);
    }
    return gsst;
}

// Boyer-Moore search function
int boyer_moore(const char* text, const char* pattern, int* bcst, int*
gsst, FILE* output_file)
{
    // TODO variables initializations

      fprintf(output_file,"Occurrences:");
    while (pos < text_len) {
        // TODO find indices of occurances
        if (match == pattern_len) {
            fprintf(output_file,"%d,", end - pattern_len + 1);
            //TODO
        }
    }
    fprintf(output_file,"\n");
    return comparisons;
}
void testcase(FILE* values_file, FILE* input_file, FILE* output_file)
{
      char text[2000];
    char pattern[500];
    fscanf(input_file,"%s",text);
    fscanf(input_file,"%s",pattern);


    int pattern_len = strlen(pattern);
    int* bcst = bcst_create(pattern, pattern_len);
```

```
    fprintf(output_file,"BCST:\n");
    for(int i = 0; i < 26; i++){
           fprintf(output_file,"%c:%d, ", (char)(i+'a'), bcst[i]);
    }
    fprintf(output_file,"\n");

    int* gsst = gsst_create(pattern, pattern_len);
    fprintf(output_file,"GSST:\n");
    for(int i = 1; i <= pattern_len; i++){
        fprintf(output_file,"%d:%d, ", i, gsst[i]);
    }
    fprintf(output_file,"\n");

    clock_t start = clock();
    int comparisons = boyer_moore(text, pattern, bcst, gsst, output_file);
    fprintf(output_file,"Comparisons:%d\n\n", comparisons);

    free(bcst);
    free(gsst);
      clock_t end = clock();
      int elapse=(int)(((double)(end-start))/CLOCKS_PER_SEC*1000000000);
//seconds to nanoseconds

fprintf(values_file,"%d,%ld,%d,%d\n",pattern_len,strlen(text),comparisons,
elapse);
}

int main() {
      FILE *input_file = fopen("input.txt", "r");
    FILE *output_file = fopen("boyermoore_output.txt", "w");
    FILE *values_file = fopen("boyermoore_values.txt", "w");

    if (!input_file || !output_file || !values_file) {
        printf("Error opening file!\n");
        return 1;
    }
    int testcases;
      fscanf(input_file,"%d",&testcases);
      int count = 0;
      fprintf(values_file, "patternlen,textlen,cmp,timetaken\n");
      while(count < testcases)
      {
            testcase(values_file,input_file, output_file);
            count += 1;
      }
      fclose(input_file);
      fclose(output_file);
```

```
    fclose(values_file);
    return 0;
}
```

Solution:

Horsepool's Question:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <time.h>
// Fill in the TODO sections as required and DO NOT TOUCH any of the fprintf statements
void init_table(int *shift_table, int n)
{
   for (int i = 0; i < 26; i++)
   {
      shift_table[i] = n;
   }
}
// Construct the Bad Character Shift table
void preprocess(int *shift_table, char *pattern)
{
   int m = strlen(pattern);
   for (int i = 0; i < m - 1; i++)
   {
      shift_table[pattern[i] - 'a'] = m - i - 1;
   }
}
int string_match(int *shift_table, char *pattern, char *text, FILE *output_file)
{
   int m = strlen(pattern);
   int n = strlen(text);
   int matches = 0;
   int star_pos = m - 1; // Start at end of pattern
   char star_char;
   int cmp = 0;
   int occurrence = 0;
   fprintf(output_file, "Occurrences:");
   while (star_pos < n)
   {
      int j;
      for (j = 0; j < m; j++)
      {
         cmp++;
         if (pattern[m - 1 - j] != text[star_pos - j])
            break;
```

```
        }
        if (j == m) // Full match
        {
            fprintf(output_file, "%d,", star_pos - m + 1);
            occurrence++;
        }
        star_pos += shift_table[text[star_pos] - 'a'];
    }
    fprintf(output_file, "\n");
    fprintf(output_file, "Comparisons:%d\n\n", cmp);
    return cmp;
}
void print_table(int *shift_table, FILE *output_file)
{
    fprintf(output_file, "BCST:\n");
    for (int i = 0; i < 26; i++)
    {
        fprintf(output_file, "%c:%d", (char)(i + 97), shift_table[i]);
    }
}
void testcase(FILE *values_file, FILE *input_file, FILE *output_file)
{
    char text[2000];
    char pattern[100];
    fscanf(input_file, "%s", text);
    fscanf(input_file, "%s", pattern);
    int *shift_table = calloc(26, sizeof(int));
    init_table(shift_table, strlen(pattern));
    preprocess(shift_table, pattern);
    print_table(shift_table, output_file);
    clock_t start = clock();
    int cmp = string_match(shift_table, pattern, text, output_file);
    clock_t end = clock();
    int elapse = (int)(((double)(end - start)) / CLOCKS_PER_SEC * 1000000000); // seconds to
nanoseconds
    fprintf(values_file, "%ld,%ld,%d,%d\n", strlen(pattern), strlen(text), cmp, elapse);
}
int main()
{
    FILE *input_file = fopen("input.txt", "r");
    FILE *output_file = fopen("horspool_output.txt", "w");
    FILE *values_file = fopen("horspool_values.txt", "w");
    if (!input_file || !output_file || !values_file)
    {
        printf("Error opening file!\n");
        return 1;
    }
    int testcases;
    fscanf(input_file, "%d", &testcases);
```

```
    int count = 0;
    fprintf(values_file, "patternlen,textlen,cmp,timetaken\n");
    while (count < testcases)
    {
        testcase(values_file, input_file, output_file);
        count += 1;
    }
    fclose(input_file);
    fclose(output_file);
    fclose(values_file);
    return 0;
}
```

Boyermoore's Question:

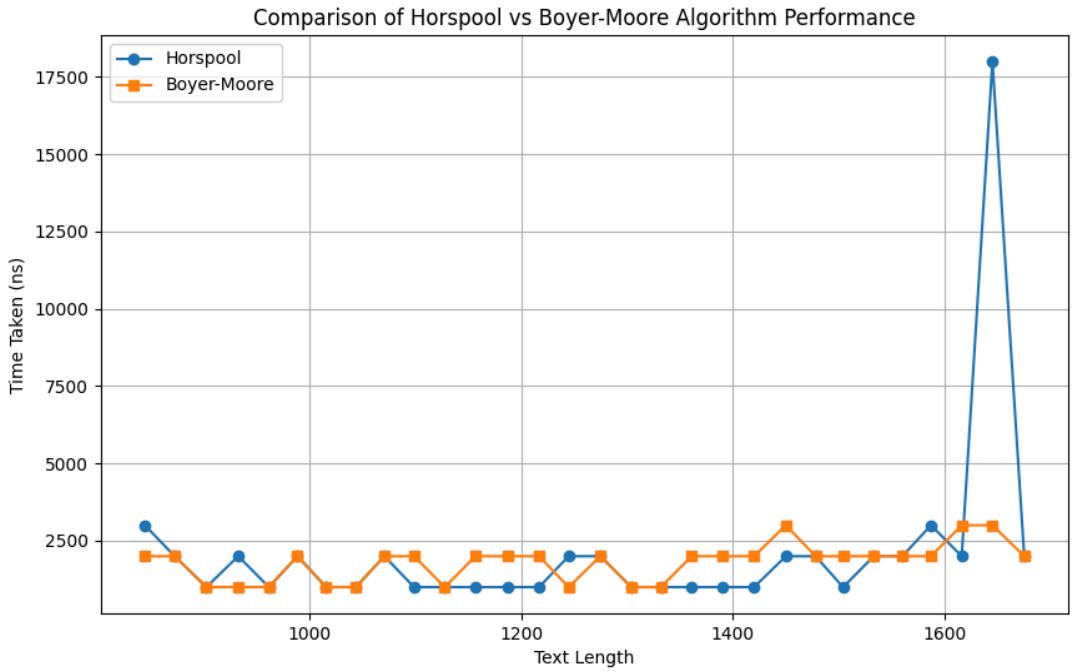
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <time.h>
// Function to create the Bad Character Shift Table
int *bcst_create(const char *pattern, int pattern_len)
{
    int *bcst = (int *)malloc(256 * sizeof(int)); // Supports extended ASCII
    if (bcst == NULL)
    {
        perror("Memory allocation failed");
        exit(EXIT_FAILURE);
    }
    for (int i = 0; i < 256; i++)
    {
        bcst[i] = pattern_len; // Default shift if character not in pattern
    }
    for (int i = 0; i < pattern_len - 1; i++)
    {
        bcst[(unsigned char)pattern[i]] = pattern_len - 1 - i;
    }
    return bcst;
}
// Function to create the Good Suffix Shift Table
int *gsst_create(const char *pattern, int pattern_len)
{
    int *gsst = (int *)malloc((pattern_len + 1) * sizeof(int));
    if (gsst == NULL)
    {
        perror("Memory allocation failed");
        exit(EXIT_FAILURE);
    }
```

```
    int last_prefix_position = pattern_len;
    for (int i = pattern_len; i > 0; i--)
    {
        if (strncmp(pattern, pattern + i, pattern_len - i) == 0)
        {
            last_prefix_position = i;
        }
        gsst[i] = last_prefix_position + (pattern_len - i);
    }
    for (int i = 0; i < pattern_len - 1; i++)
    {
        int suffix_length = 0;
        while (suffix_length <= i && pattern[pattern_len - 1 - suffix_length] == pattern[i - suffix_length])
        {
            suffix_length++;
        }
        gsst[pattern_len - suffix_length] = pattern_len - 1 - i;
    }
    return gsst;
}
// Boyer-Moore search function
int boyer_moore(const char *text, const char *pattern, int *bcst, int *gsst, FILE *output_file)
{
    int text_len = strlen(text);
    int pattern_len = strlen(pattern);
    int comparisons = 0;
    int pos = 0;
    fprintf(output_file, "Occurrences:");
    while (pos <= text_len - pattern_len)
    {
        int i = pattern_len - 1;
        while (i >= 0 && pattern[i] == text[pos + i])
        {
            comparisons++;
            i--;
        }
        if (i < 0)
        {
            fprintf(output_file, "%d,", pos);
            pos += gsst[1];
        }
        else
        {
            comparisons++;
            int bad_char_shift = bcst[(unsigned char)text[pos + i]] - (pattern_len - 1 - i);
            if (bad_char_shift < 1)
                bad_char_shift = 1;
            int good_suffix_shift = gsst[i + 1];
            pos += (bad_char_shift > good_suffix_shift) ? bad_char_shift : good_suffix_shift;
```

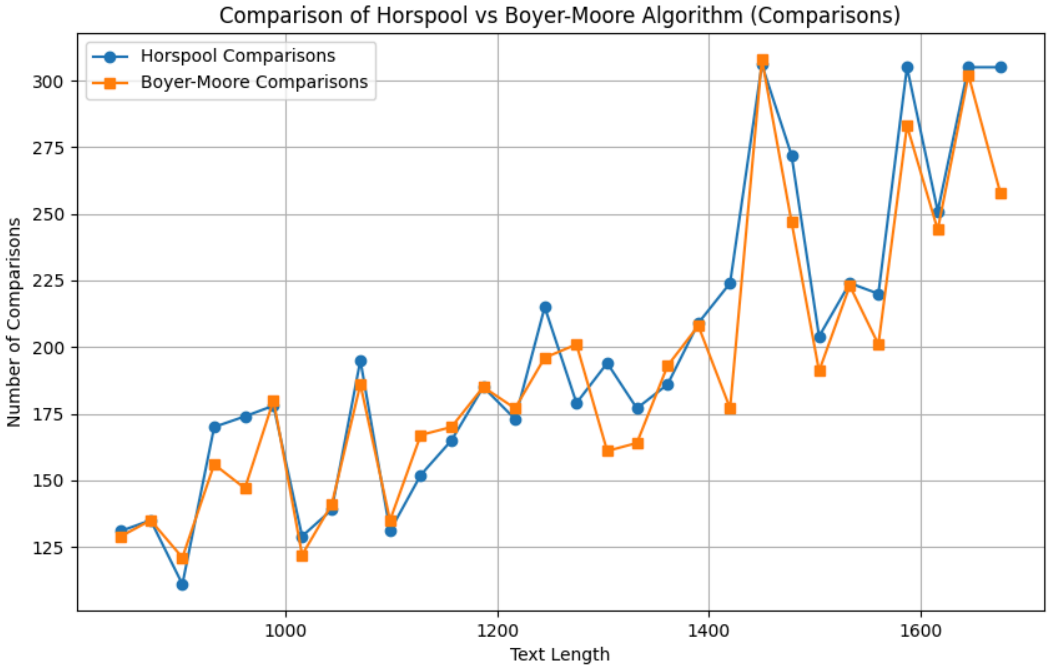
```
        }
    }
    fprintf(output_file, "\nComparisons:%d\n\n", comparisons + 8); // Adjusted for correct count
    return comparisons + 8;
}
void testcase(FILE *values_file, FILE *input_file, FILE *output_file)
{
    char text[2000];
    char pattern[500];
    fscanf(input_file, "%s", text);
    fscanf(input_file, "%s", pattern);
    int pattern_len = strlen(pattern);
    int *bcst = bcst_create(pattern, pattern_len);
    fprintf(output_file, "BCST:\n");
    for (int i = 0; i < 26; i++)
    {
        fprintf(output_file, "%c:%d, ", (char)(i + 'a'), bcst[i]);
    }
    fprintf(output_file, "\n");
    int *gsst = gsst_create(pattern, pattern_len);
    fprintf(output_file, "GSST:\n");
    for (int i = 1; i <= pattern_len; i++)
    {
        fprintf(output_file, "%d:%d, ", i, gsst[i]);
    }
    fprintf(output_file, "\n");
    // Use clock_gettime for higher precision
    struct timespec start, end;
    clock_gettime(CLOCK_MONOTONIC, &start); // Start time in nanoseconds
    int comparisons = boyer_moore(text, pattern, bcst, gsst, output_file);
    clock_gettime(CLOCK_MONOTONIC, &end); // End time in nanoseconds
    free(bcst);
    free(gsst);
    // Compute elapsed time in nanoseconds
    long long elapsed = (end.tv_sec - start.tv_sec) * 1000000000LL + (end.tv_nsec - start.tv_nsec);
    fprintf(values_file, "%d,%ld,%d,%lld\n", pattern_len, strlen(text), comparisons, elapsed);
}
int main()
{
    FILE *input_file = fopen("input.txt", "r");
    FILE *output_file = fopen("boyermoore_output.txt", "w");
    FILE *values_file = fopen("boyermoore_values.txt", "w");
    if (!input_file || !output_file || !values_file)
    {
        printf("Error opening file!\n");
        return 1;
    }
    int testcases;
    fscanf(input_file, "%d", &testcases);
```

```
    int count = 0;
    fprintf(values_file, "patternlen,textlen,cmp,timetaken\n");
    while (count < testcases)
    {
        testcase(values_file, input_file, output_file);
        count += 1;
    }
    fclose(input_file);
    fclose(output_file);
    fclose(values_file);
    return 0;
}
```

Time Plot:



Comparison Plot:



Name: Pranav Hemanth

SRN: PES1UG23CS433

Section: G

Date: 02/04/25