CSE 2046/2246

Analysis of Algorithms

Homework 1

Team Members

1) 150119683-Busenur Yılmaz
2) 150119858-Yasin Çörekci
3) 150119678-Asaf Talha Gültekin
4) 150119066-Ertan Karaoğlu

Introduction:

For the first experiment, we have chosen the web page
https://www.gutenberg.org/files/2554/2554-h/2554-h.htm

where there is the novel “Crime and Punishment” written by Fyodor Dostoyevsky.

We have selected 5 different patterns which have different lengths and they exists in the text.

The patterns are as follows:

- ee
- the
- serve
- Raskolnikov
- Later on, when he recalled that time and all that happened to him during

Outputs are saved as html files and examples of the marked html files are below:

“ee” marked:

TRANSLATOR’S PREFACE

A few words about Dostoevsky himself may help the English reader to understand his work.

Dostoevsky was the son of a doctor. His parents were very hard-working and deeply religious people, but so poor that they lived with their five children in only two rooms. The father and mother spent their evenings in reading aloud to their children, generally from books of a serious character.

Though always sickly and delicate Dostoevsky came out third in the final examination of the Petersburg school of Engineering. There he had already begun his first work, “Poor Folk.”

This story was published by the poet Nekrassov in his review and was received with acclamations. The shy, unknown youth found himself instantly something of a celebrity. A brilliant and successful career seemed to open before him, but those hopes were soon dashed. In 1849 he was arrested.

Though neither by temperament nor conviction a revolutionist, Dostoevsky was one of a little group of young men who met together to read Fourier and Proudhon. He was accused of “taking part in conversations against the censorship, of reading a letter from Byelinsky to Gogol, and of knowing of the intention to set up a printing press.” Under Nicholas I. (that “stern and just man,” as Maurice Baring calls him) this was enough, and he was condemned to death. After eight months’ imprisonment he was with twenty-one others taken out to the Semyonovsky Square to be shot. Writing to his brother Mihail, Dostoevsky says: “They snapped words over our heads, and they made us put on the white shirts worn by persons condemned to death. Thereupon we were bound in threes to stakes, to suffer execution. Being the third in the row, I concluded I had only a few minutes of life before me. I thought of you and your dear ones and I contrived to kiss Plestcheiev and Dourov, who were next to me, and to bid them farewell. Suddenly the troops beat a tattoo, we were unbound, brought back upon the scaffold, and informed that his Majesty had spared us our lives.” The sentence was commuted to hard labour.

One of the prisoners, Grigoryev, went mad as soon as he was untied, and never regained his sanity.

The intense suffering of this experience left a lasting stamp on Dostoevsky’s mind. Though his religious temper led him in the end to accept every suffering with resignation and to regard it as a blessing in his own case, he constantly recurs to the subject in his writings. He describes the awful agony of the condemned man and insists on the cruelty of inflicting such torture. Then followed four years of penal servitude, spent in the company of common criminals in Siberia, where he began the “Dead House,” and some years of service in a disciplinary battalion.

He had shown signs of some obscure nervous disease before his arrest and this now developed into violent attacks of epilepsy, from which he suffered for the rest of his life. The fits occurred three or four times a year and were more frequent in periods of great strain. In 1859 he was allowed to return to Russia. He started a journal—“Vremya,” which was forbidden by the Censorship through a misunderstanding. In 1864 he lost his first wife and his brother Mihail. He was in terrible poverty, yet he took upon himself the payment of his brother’s debts. He started another journal—“The Epoch,” which within a few months was also prohibited. He was weighed down by debt, his brother’s family was dependent on him, he was forced to write at heart-breaking speed, and is said never to have corrected his work. The later years of his life were much softened by the tenderness and devotion of his second wife.

In June 1880 he made his famous speech at the unveiling of the monument to Pushkin in Moscow and he was received with

"the" marked:

TRANSLATOR'S PREFACE

A few words about Dostoevsky himself may help the English reader to understand his work.

Dostoevsky was the son of a doctor. His parents were very hard-working and deeply religious people, but so poor that they lived with their five children in only two rooms. The father and mother spent their evenings in reading aloud to their children, generally from books of a serious character.

Though always sickly and delicate Dostoevsky came out third in the final examination of the Petersburg school of Engineering. There he had already begun his first work, "Poor Folk."

This story was published by the poet Nekrassov in his review and was received with acclamations. The shy, unknown youth found himself instantly something of a celebrity. A brilliant and successful career seemed to open before him, but those hopes were soon dashed. In 1849 he was arrested.

Though neither by temperament nor conviction a revolutionist, Dostoevsky was one of a little group of young men who met together to read Fourier and Proudhon. He was accused of "taking part in conversations against the censorship, of reading a letter from Byelinsky to Gogol, and of knowing of the intention to set up a printing press." Under Nicholas I. (that "stern and just man," as Maurice Baring calls him) this was enough, and he was condemned to death. After eight months' imprisonment he was with twenty-one others taken out to the Semyonovsky Square to be shot. Writing to his brother Mihail, Dostoevsky says: "They snapped words over our heads, and they made us put on the white shirts worn by persons condemned to death. Thereupon we were bound in threes to stakes, to suffer execution. Being the third in the row, I concluded I had only a few minutes of life before me. I thought of you and your dear ones and I contrived to kiss Plestcheiev and Dourov, who were next to me, and to bid them farewell. Suddenly the troops beat a tattoo, we were unbound, brought back upon the scaffold, and informed that his Majesty had spared us our lives." The sentence was commuted to hard labour.

One of the prisoners, Grigoryev, went mad as soon as he was untied, and never regained his sanity.

The intense suffering of this experience left a lasting stamp on Dostoevsky's mind. Though his religious temper led him in the end to accept every suffering with resignation and to regard it as a blessing in his own case, he constantly recurs to the subject in his writings. He describes the awful agony of the condemned man and insists on the cruelty of inflicting such torture. Then followed four years of penal servitude, spent in the company of common criminals in Siberia, where he began the "Dead House," and some years of service in a disciplinary battalion.

He had shown signs of some obscure nervous disease before his arrest and this now developed into violent attacks of epilepsy, from which he suffered for the rest of his life. The fits occurred three or four times a year and were more frequent in periods of great strain. In 1859 he was allowed to return to Russia. He started a journal—"Vremya," which was forbidden by the Censorship through a misunderstanding. In 1864 he lost his first wife and his brother Mihail. He was in terrible poverty, yet he took upon himself the payment of his brother's debts. He started another journal—"The Epoch," which within a few months was also prohibited. He was weighed down by debt, his brother's family was dependent on him, he was forced to write at heart-breaking speed, and is said never to have corrected his work. The later years of his life were much softened by the tenderness and devotion of his second wife.

In June 1880 he made his famous speech at the unveiling of the monument to Pushkin in Moscow and he was received with extraordinary demonstrations of love and honour.

A few months later Dostoevsky died. He was followed to the grave by a vast multitude of mourners, who "gave the hapless man the

"Raskolnikov" marked:

colourless, somewhat grizzled hair was thickly smeared with oil, and she wore no kerchief over it. Round her thin long neck, which looked like a hen's leg, was knotted some sort of flannel rag, and, in spite of the heat, there hung flapping on her shoulders, a mangy fur cape, yellow with age. The old woman coughed and groaned at every instant. The young man must have looked at her with a rather peculiar expression, for a gleam of mistrust came into her eyes again.

"Raskolnikov, a student, I came here a month ago," the young man made haste to mutter, with a half bow, remembering that he ought to be more polite.

"I remember, my good sir, I remember quite well your coming here," the old woman said distinctly, still keeping her inquiring eyes on his face.

"And here... I am again on the same errand," Raskolnikov continued, a little disconcerted and surprised at the old woman's mistrust. "Perhaps she is always like that though, only I did not notice it the other time," he thought with an uneasy feeling.

The old woman paused, as though hesitating; then stepped on one side, and pointing to the door of the room, she said, letting her visitor pass in front of her:

"Step in, my good sir."

The little room into which the young man walked, with yellow paper on the walls, geraniums and muslin curtains in the windows, was brightly lighted up at that moment by the setting sun.

"So the sun will shine like this *then* too!" flashed as it were by chance through Raskolnikov's mind, and with a rapid glance he scanned everything in the room, trying as far as possible to notice and remember its arrangement. But there was nothing special in the room. The furniture, all very old and of yellow wood, consisted of a sofa with a huge bent wooden back, an oval table in front of the sofa, a dressing-table with a looking-glass fixed on it between the windows, chairs along the walls and two or three half-penny prints in yellow frames, representing German damsels with birds in their hands—that was all. In the corner a light was burning before a small ikon. Everything was very clean; the floor and the furniture were brightly polished; everything shone.

"Lizaveta's work," thought the young man. There was not a speck of dust to be seen in the whole flat.

"It's in the houses of spiteful old widows that one finds such cleanliness," Raskolnikov thought again, and he stole a curious glance at the cotton curtain over the door leading into another tiny room, in which stood the old woman's bed and chest of drawers and into which he had never looked before. These two rooms made up the whole flat.

"What do you want?" the old woman said severely, coming into the room and, as before, standing in front of him so as to look him straight in the face.

"I've brought something to pawn here," and he drew out of his pocket an old-fashioned flat silver watch, on the back of which was engraved a globe; the chain was of steel.

"But the time is up for your last pledge. The month was up the day before yesterday."

"I will bring you the interest for another month; wait a little."

"But that's for me to do as I please, my good sir, to wait or to sell your pledge at once."

"How much will you give me for the watch, Alyona Ivanovna?"

"You come with such trifles, my good sir, it's scarcely worth anything. I gave you two roubles last time for your ring and one could buy it quite new at a jeweler's for a rouble and a half."

For the second experiment, we have created a totally random html file using 52 alphabetic characters.

```cpp
std::string Util::createRandomHtml()
{
    std::ostringstream ss;

    const static char alphabets[] = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
    size_t len = strlen(alphabets);

    std::srand(static_cast<unsigned int>(std::time(nullptr)));

    ss << "<html><head><title>Random Html</title><style type=\"text/css\" xml:space=\"preserve

    for (int i = 0; i < (1 << 21); ++i) {
        ss << alphabets[std::rand() % len];
        if (i % 120 == 0)
            ss << "\n";
    }

    ss << "</p></div></body></html>";

    return ss.str();
}
```

We have selected 5 different patterns which have different lengths (5, 10, 15, 20, 25) and they exists in the text.

The patterns are as follows:

- QhDdd
- QhDddLpgqM
- QhDddLpgqMXXODD
- QhDddLpgqMXXODDHqyfl
- QhDddLpgqMXXODDHqyflsTTLw

We have implemented the code in C++. The codes also attached at the end of the document.

We have search.cpp file in which we have implemented pattern search algorithms, util.cpp in which there are codes for file reading, writing and creating random html file. Besides we have timer.cpp file in which we have implemented a timer class to measure elapsed time for each algorithm.

Experiment 1:

Crime and Punishment HTML file

a)

Searched pattern: “ee”:

	Total Compares	Pattern	Pattern Length	Found	Elapsed
Brute Force	1473916	ee	2	3220	0.279835
Horspool	762165	ee	2	3220	0.215858
Boyer Moore	762165	ee	2	3220	0.373316

Bad character heuristic:
e: 1

Good character heuristic:
1, 1, 2

b)

Searched pattern: “the”:

	Total Compares	Pattern	Pattern Length	Found	Elapsed
Brute Force	1469113	the	3	11425	0.365834
Horspool	550961	the	3	11425	0.119033
Boyer Moore	536687	the	3	11425	0.138342

Bad character heuristic:
e: 2, h: 1, t: 0

Good character heuristic:
3, 3, 3, 1

c)

Searched pattern: “serve”:

	Total Compares	Pattern	Pattern Length	Found	Elapsed
Brute Force	1426988	serve	5	102	0.283211
Horspool	317572	serve	5	102	0.058398
Boyer Moore	315233	serve	5	102	0.0697822

Bad character heuristic:
e: 4, r: 2, s: 0, v: 3

Good character heuristic:
5, 5, 5, 5, 3, 1

d)

Searched pattern: “Raskolnikov”:

	Total Compares	Pattern	Pattern Length	Found	Elapsed
Brute Force	1377710	Raskolnikov	11	784	0.26069
Horspool	156586	Raskolnikov	11	784	0.0286669
Boyer Moore	156194	Raskolnikov	11	784	0.0373401

Bad character heuristic:
R: 0, a: 1, i: 7, k: 8, l: 5, n: 6, o: 9, s: 2, v: 10

Good character heuristic:
11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 1

e)

Searched pattern: “Later on, when he recalled that time and all that happened to him during”:

	Total Compares	Pattern	Pattern Length	Found	Elapsed
Brute Force	1369096	Later on, when he recalled that time and all that happened to him during	72	1	0.249067
Horspool	61289	Later on, when he recalled that time and all that happened to him during	72	1	0.01163
Boyer Moore	58257	Later on, when he recalled that time and all that happened to him during	72	1	0.0188612

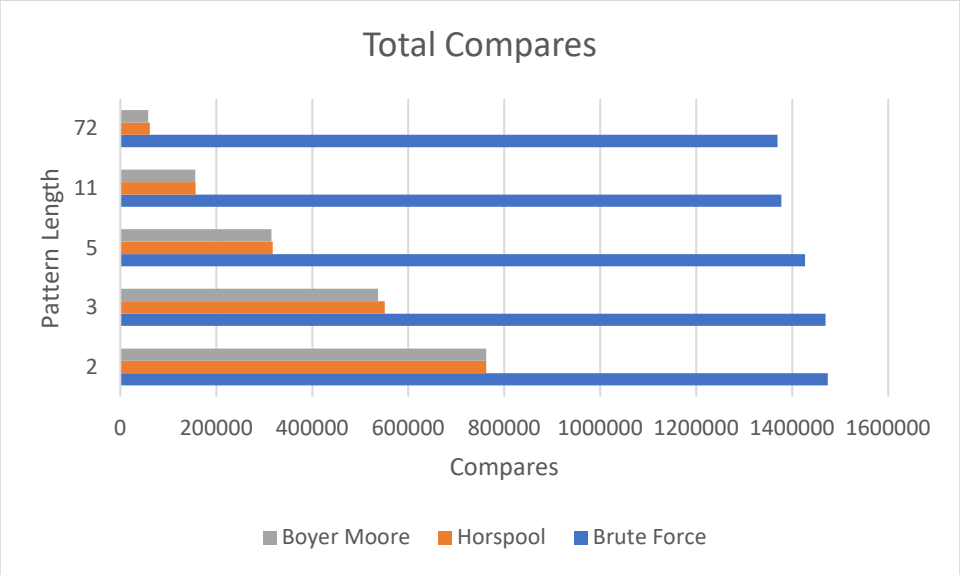
Bad character heuristic:
: 65, ,: 8, L: 0, a: 51, c: 20, d: 66, e: 56, g: 71, h: 62, i: 69, l: 43, m: 64, n: 70, o: 60, p: 53, r: 68, t: 59, u: 67, w: 10

Good character heuristic:
72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 72, 1

Analysis of experiment 1:

Total compares by pattern length:

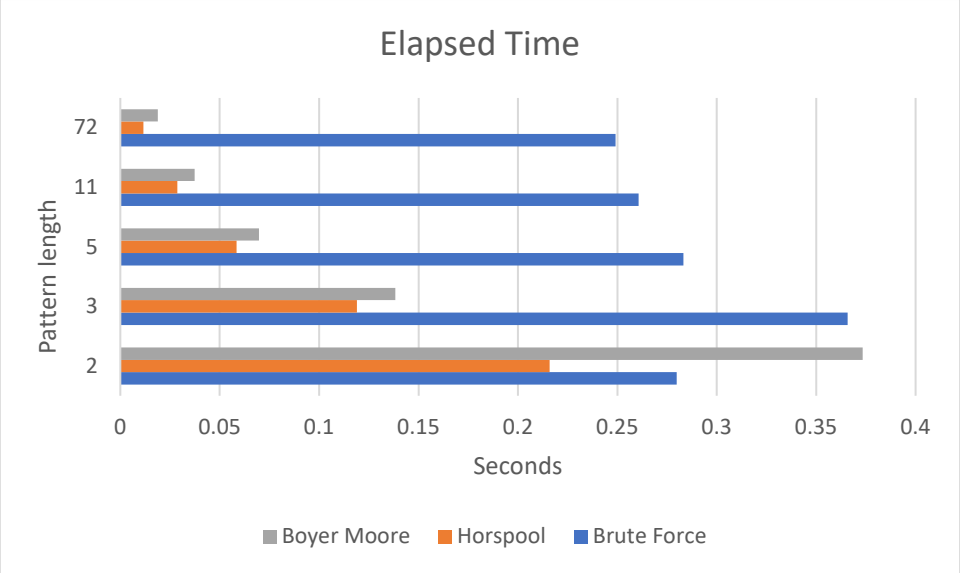
	Brute Force	Horspool	Boyer Moore
2	1473916	762165	762165
3	1469113	550961	536687
5	1426988	317572	315233
11	1377710	156586	156194
72	1369096	61289	58257



In the experiment we have seen that, Brute force algorithm has much more compares than both Horspool and Boyer Moore algorithms which agrees with the theoretical results. Because the text which we have chosen does not contains so much repeated patterns next to each other, Brute Force has the same even just a bit better performance when pattern length is increasing. When compared Horspool and Boyer Moore algorithms, Boyer Moore has less compares than Horspool. It is because Boyer Moore algorithm is both using bad character heuristic and good character heuristic. Besides we can see also better results (less compares) when the pattern length increasing. Bad heuristics and good heuristics made more shifts when there is a mismatched when began from the end of the pattern.

Elapsed Time by pattern length:

	Brute Force	Horspool	Boyer Moore
2	0,279835	0,215858	0,373316
3	0,365834	0,119033	0,138342
5	0,283211	0,058398	0,0697822
11	0,26069	0,0286669	0,0373401
72	0,249067	0,01163	0,0188612



In the experiment we have seen that, Brute force algorithm has much more elapsed time in general than both Horspool and Boyer Moore algorithms which agrees with the theoretical results. (Only exception is for the pattern length two which is “ee”). Because the text which we have chosen does not contains so much repeated patterns next to each other, Brute Force has nearly the performance when pattern length is increasing. When compared Horspool and Boyer Moore algorithms, Horspool has less running time than Boyer Moore. The reason can be Boyer Moore has more operation before beginning to search, such as creating good suffix table and we have used here some dynamic memory allocations for the tables. When the pattern length is increasing, the running time is decreasing for both Horspool and Boyer Moore algorithms. The reason for that is the pattern and text does not have so much repeating patterns and therefore as we have seen in the total number of compares, the shifts are larger when pattern length is increased.

Experiment 2:

Random generated HTML file

a)

Searched pattern: “QhDdd”:

	Total Compares	Pattern	Pattern Length	Found	Elapsed
Brute Force	2155464	QhDdd	5	1	0.415589
Horspool	443280	QhDdd	5	1	0.0765541
Boyer Moore	443193	QhDdd	5	1	0.0977836

Bad character heuristic:
D: 2, Q: 0, d: 4, h: 1

Good character heuristic:
5, 5, 5, 5, 1, 2

b)

Searched pattern: “QhDddLpgqM”:

	Total Compares	Pattern	Pattern Length	Found	Elapsed
Brute Force	2155464	QhDddLpgqM	10	1	0.344728
Horspool	234792	QhDddLpgqM	10	1	0.0472556
Boyer Moore	233959	QhDddLpgqM	10	1	0.0517536

Bad character heuristic:

D: 2, L: 5, M: 9, Q: 0, d: 4, g: 7, h: 1, p: 6, q: 8

Good character heuristic:

10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 1

c)

Searched pattern: “QhDddLpgqMXXODD”:

	Total Compares	Pattern	Pattern Length	Found	Elapsed
Brute Force	2155464	QhDddLpgqMXXODD	15	1	0.375579
Horspool	158934	QhDddLpgqMXXODD	15	1	0.0277546
Boyer Moore	158928	QhDddLpgqMXXODD	15	1	0.0439114

Bad character heuristic:
D: 14, L: 5, M: 9, O: 12, Q: 0, X: 11, d: 4, g: 7, h: 1, p: 6, q: 8

Good character heuristic:
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 1, 2

d)

Searched pattern: “QhDddLpgqMXXODDHqyfl”:

	Total Compares	Pattern	Pattern Length	Found	Elapsed
Brute Force	2155464	QhDddLpgqMXXODDHqyfl	20	1	0.419209
Horspool	126319	QhDddLpgqMXXODDHqyfl	20	1	0.0238254
Boyer Moore	125647	QhDddLpgqMXXODDHqyfl	20	1	0.0314799

Bad character heuristic:
D: 14, H: 15, L: 5, M: 9, O: 12, Q: 0, X: 11, d: 4, f: 18, g: 7, h: 1, l: 19, p: 6, q: 16, y: 17

Good character heuristic:
20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 1

e)

Searched pattern: “QhDddLpgqMXXODDHqyflsTTLw”:

	Total Compares	Pattern	Pattern Length	Found	Elapsed
Brute Force	2155464	QhDddLpgqMXXODDHqyflsTTLw	25	1	0.380229
Horspool	105379	QhDddLpgqMXXODDHqyflsTTLw	25	1	0.0199478
Boyer Moore	104738	QhDddLpgqMXXODDHqyflsTTLw	25	1	0.0237244

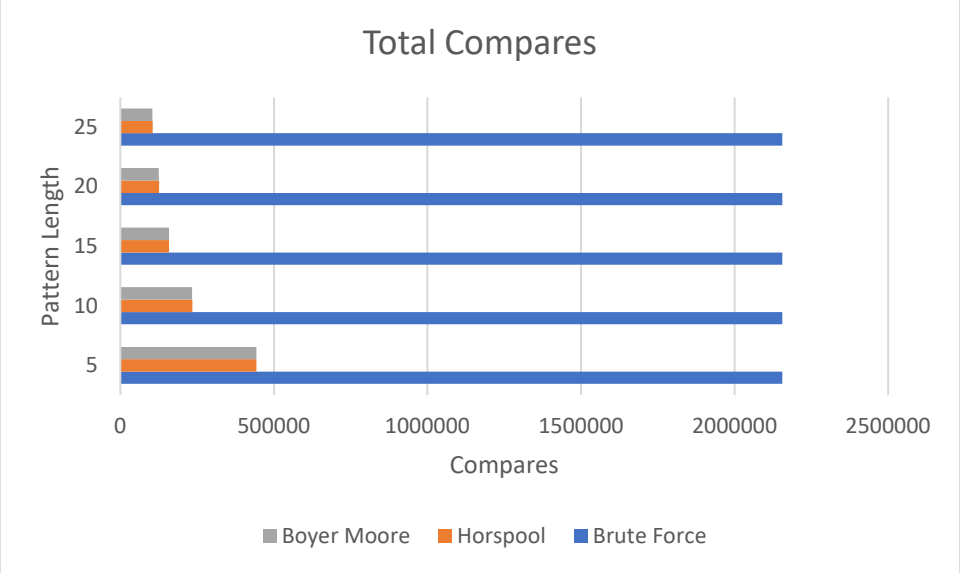
Bad character heuristic:
D: 14, H: 15, L: 23, M: 9, O: 12, Q: 0, T: 22, X: 11, d: 4, f: 18, g: 7, h: 1, l: 19, p: 6, q: 16, s: 20, w: 24, y: 17

Good character heuristic:
25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 1

Analysis of experiment 2:

Total compares by pattern length:

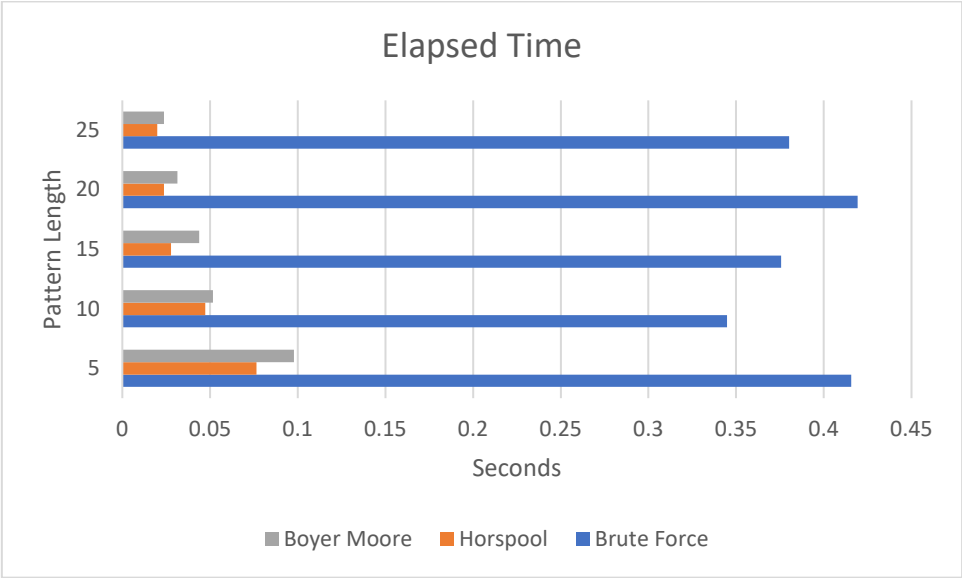
	Brute Force	Horspool	Boyer Moore
5	2155464	443280	443193
10	2155464	234792	233959
15	2155464	158934	158928
20	2155464	126319	125647
25	2155464	105379	104738



In the experiment we have seen that, Brute force algorithm has much more compares than both Horspool and Boyer Moore algorithms which agrees with the theoretical results. Because the text which we have chosen does not contains so much repeated patterns next to each other and also the html file is a random text, Brute Force has the performance when pattern length is increasing. When compared Horspool and Boyer Moore algorithms, Boyer Moore has less compares than Horspool. It is because Boyer Moore algorithm is both using bad character heuristic and good character heuristic. Besides we can see also better results (less compares) when the pattern length increasing. Bad heuristics and good heuristics made more shifts when there is a mismatched when began from the end of the pattern.

Elapsed Time by pattern length:

	Brute Force	Horspool	Boyer Moore
5	0,415589	0,0765541	0,0977836
10	0,344728	0,0472556	0,0517536
15	0,375579	0,0277546	0,0439114
20	0,419209	0,0238254	0,0314799
25	0,380229	0,0199478	0,0237244



In the experiment we have seen that, Brute force algorithm quite a lot more elapsed time in general than both Horspool and Boyer Moore algorithms which agrees with the theoretical results. Because the text which we have chosen does not contains so much repeated patterns next to each other (it is random), Brute Force has nearly the performance when pattern length is increasing. When compared Horspool and Boyer Moore algorithms, Horspool has less running time than Boyer Moore. The reason can be Boyer Moore has more operation before beginning to search, such as creating good suffix table and we have used here some dynamic memory allocations for the tables. When the pattern length is increasing, the running time is decreasing for both Horspool and Boyer Moore algorithms. The reason for that is the pattern and text does not have so much repeating patterns and therefore as we have seen in the total number of compares, the shifts are larger when pattern length is increased.

Results:

The table below shows that the time complexity and space complexity of each pattern search algorithms:

Algorithm	Search Type	Space	Average Case	Worst Case
Brute Force	Prefix	None	O(mn)	O(mn)
Horspool	Suffix	O(m)	O(m+n)	O(mn)
Boyer Moore	Suffix	O(m)	O(m+n)	O(mn)

In our experiment, we have seen from the total number of compares and also elapsed times average case time complexities holds.

Source code:

search.h

```
#ifndef __SEARCH_H
#define __SEARCH_H

#include <string>
#include <vector>

#define CHARACTERS_COUNT        256

class Search {
public:
	virtual std::vector<int> search(const std::string& text, const std::string&
pattern, int& totalCompares) = 0;
};

class BruteForceSearch : public Search {
public:
	std::vector<int> search(const std::string& text, const std::string&
pattern, int& totalCompares);
};

class HorspoolSearch : public Search {
public:
	std::vector<int> search(const std::string& text, const std::string&
pattern, int& totalCompares);
};

class BoyerMooreSearch : public Search {
public:
	std::vector<int> search(const std::string& text, const std::string&
pattern, int& totalCompares);

private:
	void createBadCharacterTable(const std::string& pattern, int* table, int
len);
	void createGoodSuffix(const std::string& pattern, int* borders, int*
shifts);
	void createPartialGoodSuffix(int* borders, int* shifts, size_t len);
};

#endif
```

search.cpp

```cpp
#include "search.h"

#include <iostream>

std::vector<int> BruteForceSearch::search(const std::string& text, const
std::string& pattern, int& totalCompares)
{
    std::vector<int> foundIndexes;
    totalCompares = 0;

      if (text.length() < pattern.length() || pattern.empty())
            return foundIndexes;

    for (int i = 0; i <= (int) (text.length() - pattern.length()); i++) {
        int k;

        for (k = 0; k < (int) pattern.length(); k++) {
            ++totalCompares;

            if (text[i + k] != pattern[k])
                break;
        }

        if (k == (int) pattern.length()) {
            foundIndexes.push_back(i);
        }
    }

    return foundIndexes;
}

std::vector<int> BoyerMooreSearch::search(const std::string& text, const
std::string& pattern, int& totalCompares)
{
      std::vector<int> foundIndexes;
      totalCompares = 0;

      if (text.length() < pattern.length() || pattern.empty())
            return foundIndexes;

      int* borders = new int[pattern.length()+1]();
      int* shifts = new int[pattern.length()+1]();
      int badCharacterTable[CHARACTERS_COUNT];

      createBadCharacterTable(pattern, badCharacterTable, CHARACTERS_COUNT);
      createGoodSuffix(pattern, borders, shifts);
      createPartialGoodSuffix(borders, shifts, pattern.length());

      int i = 0;
      int j;

      while (i <= (int) (text.length() - pattern.length())) {
            j = (int) (pattern.length() - 1);

            while (j >= 0) {
                  ++totalCompares;

                  if (pattern[j] != text[i + j])
                        break;
```

```
                --j;
            }

            if (j < 0) {
                foundIndexes.push_back(i);

                if ((i + pattern.length()) < text.length()) {
                    i += std::max(shifts[0], (int) pattern.length() -
badCharacterTable[(uint8_t) text[i + pattern.length()]]);
                }
                else {
                    i += shifts[0];
                }
            }
            else {
                i = i + std::max(shifts[j+1], j - badCharacterTable[(uint8_t)
text[i+j]]);
            }
        }

        delete[] borders;
        delete[] shifts;

        return foundIndexes;
}

void BoyerMooreSearch::createBadCharacterTable(const std::string& pattern, int*
table, int len)
{
    for (int i = 0; i < len; ++i)
        table[i] = -1;

    for (int i = 0; i < (int) pattern.length(); ++i) {
        table[(uint8_t) pattern[i]] = i;
    }

      std::string delim = "";

      std::cout << "Bad character heuristic: " << std::endl;

      for (int i = 0; i < len; ++i) {
            if (table[i] != -1) {
                std::cout << delim << (char)i << ": " << table[i];

                if (delim.empty())
                    delim = ", ";
            }
      }

      std::cout << std::endl;
}

void BoyerMooreSearch::createGoodSuffix(const std::string& pattern, int* borders,
int* shifts)
{
    int i = (int) pattern.length();
    int j = (int) (pattern.length() + 1);

      borders[i] = j;

      while (i > 0) {
            while ((j <= (int) pattern.length()) && (pattern[i-1] != pattern[j-
1])) {
```

```
                        if (shifts[j] == 0) {
                                shifts[j] = j-i;
                        }

                        j = borders[j];
                }

                --i;
                --j;
                borders[i] = j;
        }
}

void BoyerMooreSearch::createPartialGoodSuffix(int* borders, int* shifts, size_t
len)
{
        int j = borders[0];

        for (int i = 0; i <= (int) len; i++) {
                if (shifts[i] == 0) {
                        shifts[i] = j;
                }

                if (i == j) {
                        j = borders[j];
                }
        }

        std::string delim = "";

        std::cout << "Good character heuristic: " << std::endl;

        for (int i = 0; i <= (int) len; ++i) {
                if (shifts[i] != -1) {
                        std::cout << delim << shifts[i];

                        if (delim.empty())
                                delim = ", ";
                }
        }

        std::cout << std::endl;
}

std::vector<int> HorspoolSearch::search(const std::string& text, const
std::string& pattern, int& totalCompares)
{
        std::vector<int> foundIndexes;
        totalCompares = 0;

        if ((text.length() < pattern.length()) || pattern.empty())
                return foundIndexes;

        int i;

        int m = pattern.length();
        int n = text.length();

        int shifts[CHARACTERS_COUNT];

        for (i = 0; i < CHARACTERS_COUNT; ++i)
                shifts[i] = -1;
```

```
	for (i = 0; i < m; ++i)
		shifts[(uint8_t) pattern[i]] = i;

	i = 0;

	while (i <= (n - m)) {
		int j = m - 1;

		while (j >= 0) {
			++totalCompares;

			if (pattern[j] != text[i + j])
				break;

			--j;
		}

		if (j < 0) {
			foundIndexes.push_back(i);

			if ((i + pattern.length()) < text.length()) {
				i += (int) (pattern.length() - shifts[(uint8_t) text[i +
pattern.length()]]);
			}
			else {
				++i;
			}
		}
		else {
			i += std::max(1, j - shifts[(uint8_t) text[i + j]]);
		}
	}

	return foundIndexes;
}
```

timer.h

```
#ifndef __TIMER_H
#define __TIMER_H

#include <chrono>

class Timer
{
public:
    Timer();

    void start();
    void stop();
    double elapsed();
    double elapsedInMicroSeconds();
    double elapsedInMilliSeconds();
    double elapsedInSeconds();

private:
    std::chrono::time_point<std::chrono::system_clock> start_time;
    std::chrono::time_point<std::chrono::system_clock> end_time;
    bool is_running;
    bool is_started;
};

#endif
```

timer.cpp

```cpp
#include "timer.h"

Timer::Timer() : is_started(false), is_running(false)
{
}

void Timer::start()
{
    start_time = std::chrono::system_clock::now();
    is_running = true;
    is_started = true;
}

void Timer::stop()
{
    end_time = std::chrono::system_clock::now();
    is_running = false;
}

double Timer::elapsed()
{
    std::chrono::time_point<std::chrono::system_clock> time;

    if (!is_started)
        return 0.0;

    if (is_running) {
        time = std::chrono::system_clock::now();
    }
    else {
        time = end_time;
    }

    return (double) std::chrono::duration_cast<std::chrono::nanoseconds>(time -
start_time).count();
}

double Timer::elapsedInMicroSeconds()
{
    return elapsed() / 1000.0;
}

double Timer::elapsedInMilliSeconds()
{
    return elapsed() / 1000000.0;
}

double Timer::elapsedInSeconds()
{
    return elapsed() / 1000000000.0;
}
```

util.h

```
#ifndef __UTIL_H
#define __UTIL_H

#include <string>

class Util {
public:
	static std::string readFile(const std::string& filename);
	static bool writeFile(const std::string& filename, const std::string&
content);
	static std::string createRandomHtml();
};

#endif
```

util.cpp

```cpp
#include <fstream>
#include <sstream>
#include <random>
#include <ctime>

#include "util.h"

std::string Util::readFile(const std::string& filename)
{
    std::string content = "";

    std::ifstream ifs(filename);

    if (ifs.is_open()) {
        std::ostringstream ss;
        ss << ifs.rdbuf();
        content = ss.str();
        ifs.close();
    }

    return content;
}

bool Util::writeFile(const std::string& filename, const std::string& content)
{
    std::ofstream ofs(filename);

    if (ofs.is_open()) {
        ofs << content;
        ofs.close();
        return true;
    }

    return false;
}

std::string Util::createRandomHtml()
{
    std::ostringstream ss;

    const static char alphabets[] =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
    size_t len = strlen(alphabets);

    std::srand(static_cast<unsigned int>(std::time(nullptr)));

    ss << "<html><head><title>Random Html</title><style type=\"text/css\"
xml:space=\"preserve\">div.chapter {page-break-before: always; margin-top:
4em;}</style></head><body><div class=\"chapter\"><p>";

    for (int i = 0; i < (1 << 21); ++i) {
        ss << alphabets[std::rand() % len];
        if (i % 120 == 0)
            ss << "\n";
    }

    ss << "</p></div></body></html>";

    return ss.str();
}
```

main.cpp

```
#include <iostream>
#include <sstream>
#include <fstream>

#include "search.h"
#include "util.h"
#include "timer.h"

#define FILE_OUT

std::string markHtml(const std::string& html, const std::string& pattern,
std::vector<int>& positions)
{
	std::stringstream markedHtml;

	int m = (int) pattern.length();

	int last = 0;
	int begin = -1;
	bool cont = false;

	if (positions.size() == 0)
		return "";

	for (int i = 0; i < (int) (positions.size() - 1); ++i) {
		if (positions[i] + m > positions[i + 1]) {
			if (!cont) {
				begin = positions[i];
			}
			cont = true;
		}
		else {
			if (cont) {
				markedHtml << html.substr(last, begin - last);
				markedHtml << "<mark>";
				markedHtml << html.substr(begin, positions[i] + m -
begin);
				markedHtml << "</mark>";
			}
			else {
				markedHtml << html.substr(last, positions[i] - last);
				markedHtml << "<mark>";
				markedHtml << html.substr(positions[i], m);
				markedHtml << "</mark>";
			}

			last = positions[i] + m;
			cont = false;
		}
	}

	if (!positions.empty()) {
		if (cont) {
			markedHtml << html.substr(last, begin - last);
			markedHtml << "<mark>";
			markedHtml << html.substr(begin, positions[positions.size()-1]
+ m - begin);
			markedHtml << "</mark>";
		}
		else {
```

```
                        markedHtml << html.substr(last, positions[positions.size() -
1] - last);
                        markedHtml << "<mark>";
                        markedHtml << html.substr(positions[positions.size() - 1], m);
                        markedHtml << "</mark>";
                }

                last = positions[positions.size() - 1] + m;
        }

        if (last < (int) html.length()) {
                markedHtml << html.substr(last, html.length() - last);
        }

        return markedHtml.str();
}

void runTest(const std::string& text, const std::string& pattern)
{
        int totalCompares = 0;

        Timer timer;

        BruteForceSearch searchBruteForce;
        HorspoolSearch searchHorspool;
        BoyerMooreSearch searchBoyerMoore;

        std::cout << "#####################################" << std::endl;
        std::cout << "               Test                  " << std::endl;
        std::cout << "#####################################" << std::endl;

        std::cout << "Brute Force: " << std::endl;
        std::cout << "-------------------------------------" << std::endl;

        timer.start();
        auto v1 = searchBruteForce.search(text, pattern, totalCompares);
        timer.stop();

        std::string bruteForcedMarked = markHtml(text, pattern, v1);
        Util::writeFile("BruteForce_" + pattern + ".html", bruteForcedMarked);

        std::cout << "Total compares: " << totalCompares << ", " << "Found: " <<
v1.size() << ", Elapsed: " << timer.elapsedInSeconds() << std::endl;
        std::cout << std::endl;

        std::cout << "Horspool: " << std::endl;
        std::cout << "-------------------------------------" << std::endl;

        timer.start();
        auto v2 = searchHorspool.search(text, pattern, totalCompares);
        timer.stop();

        std::string horspoolMarked = markHtml(text, pattern, v2);
        Util::writeFile("Horspool_" + pattern + ".html", horspoolMarked);

        std::cout << "Total compares: " << totalCompares << ", " << "Found: " <<
v2.size() << ", Elapsed: " << timer.elapsedInSeconds() << std::endl;
        std::cout << std::endl;

        std::cout << "Boyer Moore: " << std::endl;
        std::cout << "-------------------------------------" << std::endl;

        timer.start();
```

```
	auto v3 = searchBoyerMoore.search(text, pattern, totalCompares);
	timer.stop();

	std::string boyerMooreMarked = markHtml(text, pattern, v3);
	Util::writeFile("BoyerMoore_" + pattern + ".html", boyerMooreMarked);

	std::cout << "Total compares: " << totalCompares << ", " << "Found: " <<
v3.size() << ", Elapsed: " << timer.elapsedInSeconds() << std::endl;
	std::cout << std::endl;

	std::cout << "Searched pattern: '" << pattern << "', Pattern length: " <<
pattern.length() << std::endl;

	std::cout << std::endl;
}

int main()
{
	std::string text = Util::readFile("CrimeAndPunishment.html");
	std::string pattern = "";

#ifdef FILE_OUT
	std::ofstream out("output.txt");
	std::streambuf* coutbuf = std::cout.rdbuf();
	std::cout.rdbuf(out.rdbuf());
#endif

	pattern = "ee";
	runTest(text, pattern);

	pattern = "the";
	runTest(text, pattern);

	pattern = "serve";
	runTest(text, pattern);

	pattern = "Raskolnikov";
	runTest(text, pattern);

	pattern = "Later on, when he recalled that time and all that happened to
him during";
	runTest(text, pattern);

	std::cout << "*****************************************" << std::endl;

	text = Util::createRandomHtml();

	for (int i = 1; i <= 20; ++i) {
		pattern = text.substr(text.length() / 2 + 5, 5*i);
		runTest(text, pattern);
	}

#ifdef FILE_OUT
	std::cout.rdbuf(coutbuf);
#endif

	return 0;
}
```

WORK DISTRIBUTION TABLE

- We had regular meetings with our team members throughout the project. As a result of these meetings, we distributed the workload of the project among ourselves. As a group, we worked together collaboratively. When implementing the codes and writing the report, each team member contributed equally. Further details can be found in the table.

WORK SHARING CHART					
No	**Task**	**Yasin Çörekci**	**Busenur Yılmaz**	**Asaf Talha Gültekin**	**Ertan Karaoğlu**
1)	Step 1: Designing the Experiment	✓	✓	✓	✓
2)	(a) Deciding on HTML files	✓			
3)	(b) Deciding on patterns		✓		
4)	Step 2: Coding and Running	✓	✓	✓	✓
5)	(a) Implementing algorithms		✓		✓
6)	(b) Parsing files			✓	
7)	(c) Generating highlighted file	✓	✓		
8)	(d) Measuring comparisons and time			✓	✓
9)	Step 3: Illustrating and Analyzing	✓	✓	✓	✓
10)	(a) Creating plots/tables		✓	✓	✓
11)	(b) Comparing algorithm performance	✓			
12)	(c) Providing detailed comments	✓	✓		✓