Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего профессионального образования "Национальный исследовательский университет "Высшая школа экономики"

Московский институт электроники и математики Национального исследовательского университета "Высшая школа экономики"

Департамент прикладной математики

ОТЧЕТ

По лабораторной работе №1

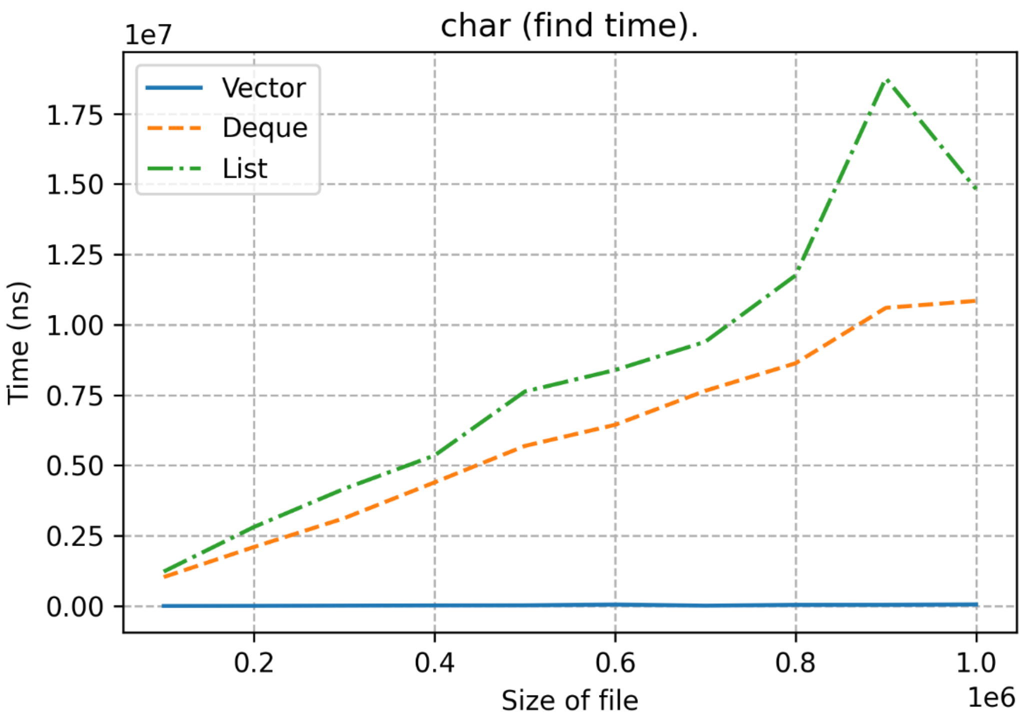
По курсу «Алгоритмизация и программирование»

ФИО студента	Номер группы	Дата
Колодин Матвей Алексеевич	БПМ213	21.10.22

Москва – 2022 г.

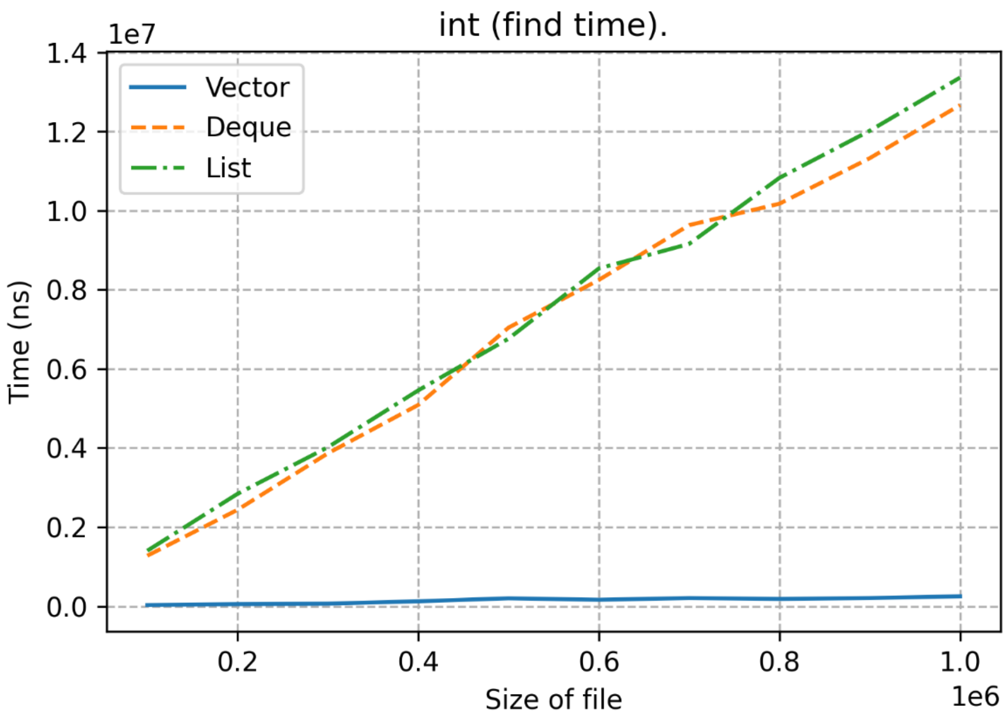
Результаты

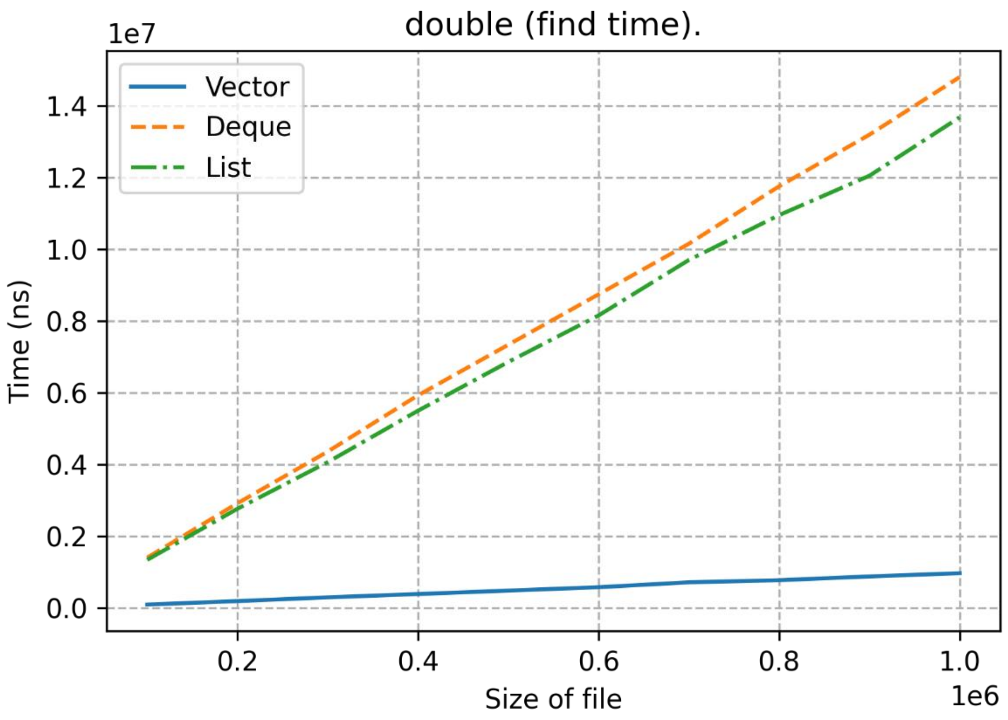
1) Давайте сравним алгоритм find на следующих контейнерах: vector, deque и list. Данные будут типа char, а файлы, в которых хранятся данные, разных размеров.

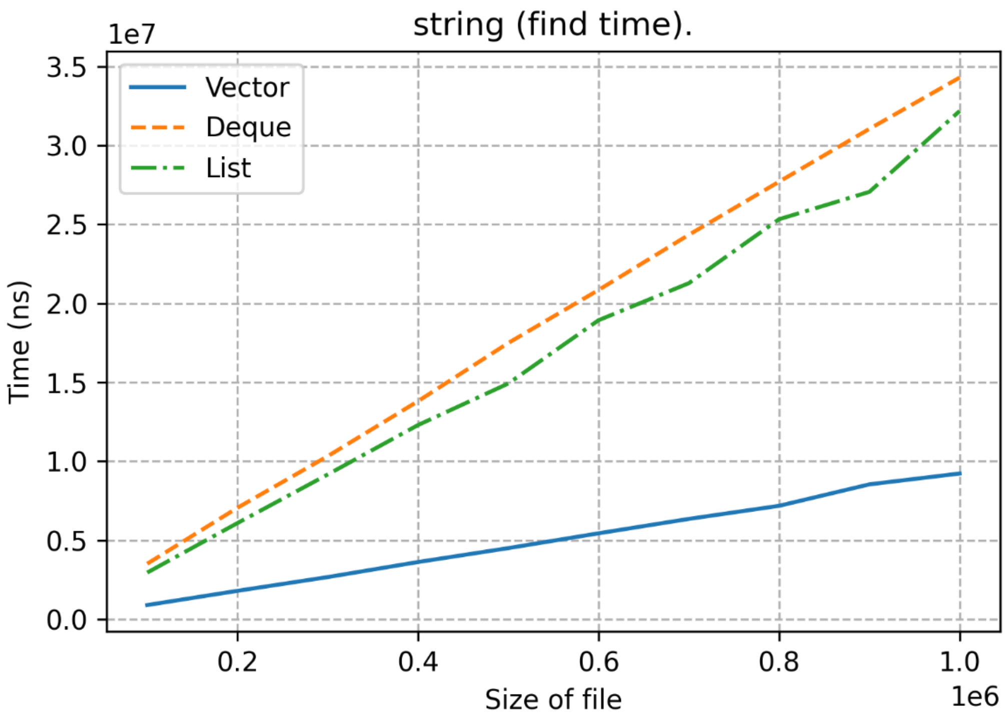


Можно видеть, что наилучшее время у vector, а худшее у list.
Объясняется это их хранением в памяти и тем, как ходят итераторы по их элементам.
Есть некоторая аномалия с листом (возможно какое-то провисание операционной системы)

2) Для int мы видим, что Deque и List очень близки по результату, а вот Vector сильно лучше их.

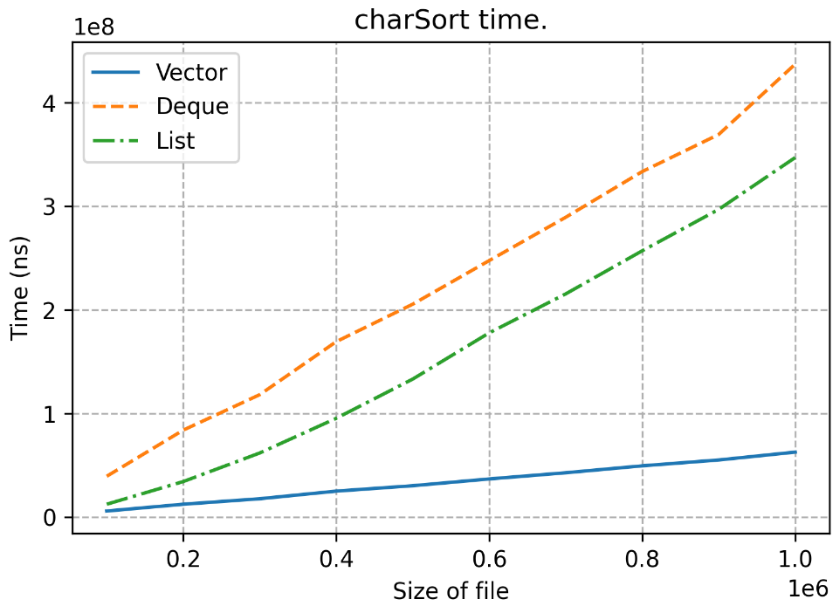


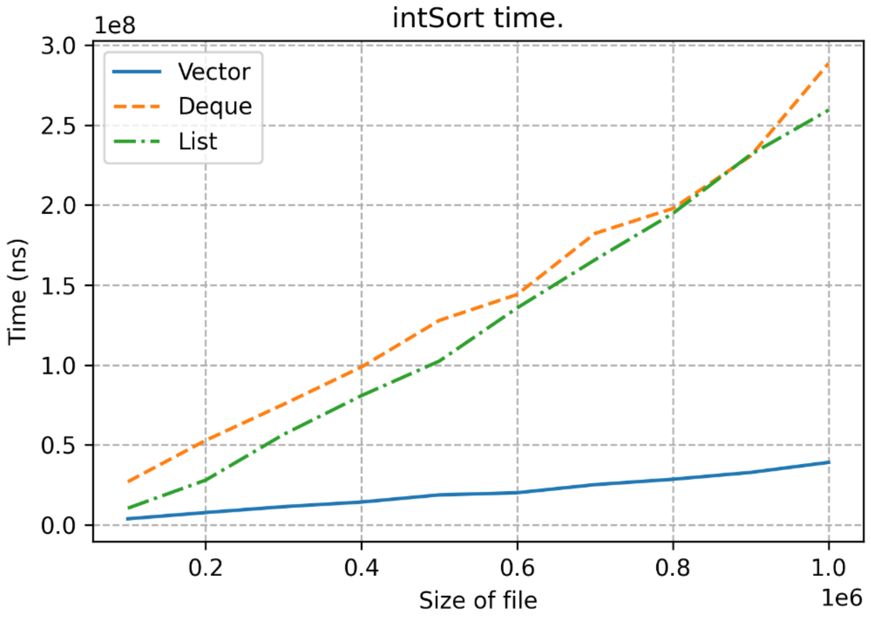


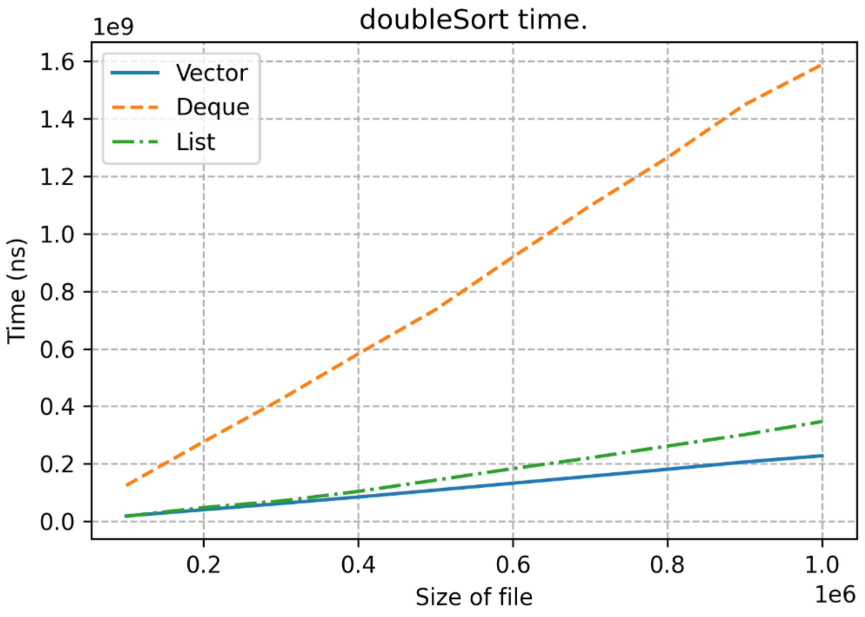


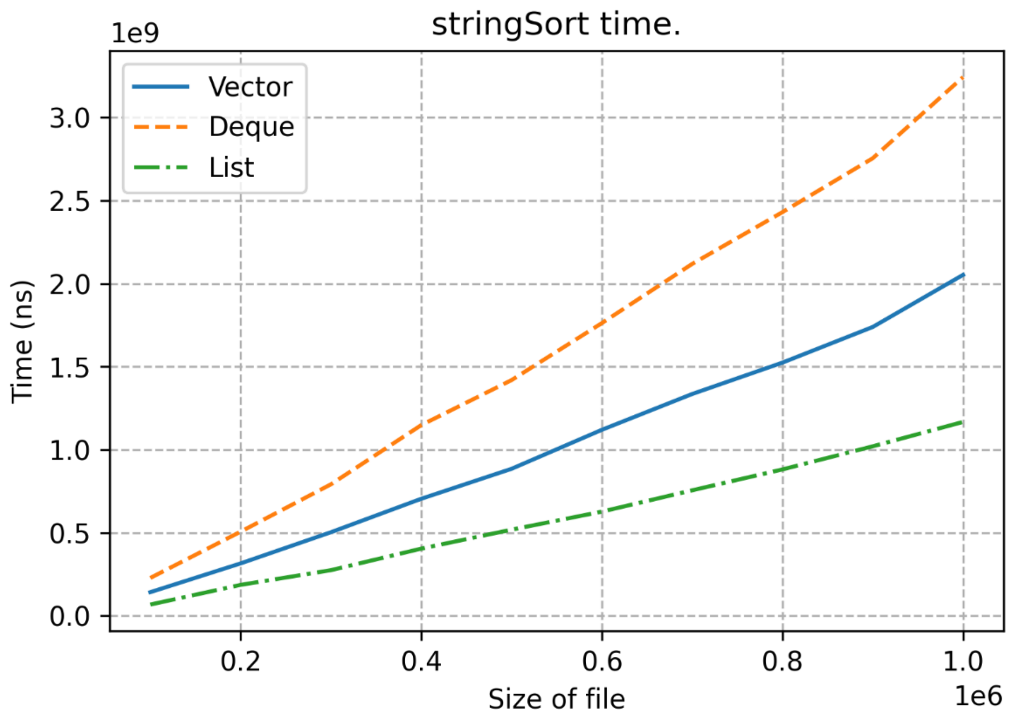
Для double ситуация схожая, а вот для string, Vector начинает тратить больше времени.

3) Переходим к sort – алгоритму.


charSort time.
1e8
Vector
Deque
List
Time (ns)
0
1
2
3
4
0.2
0.4
0.6
0.8
1.0
Size of file
1e6

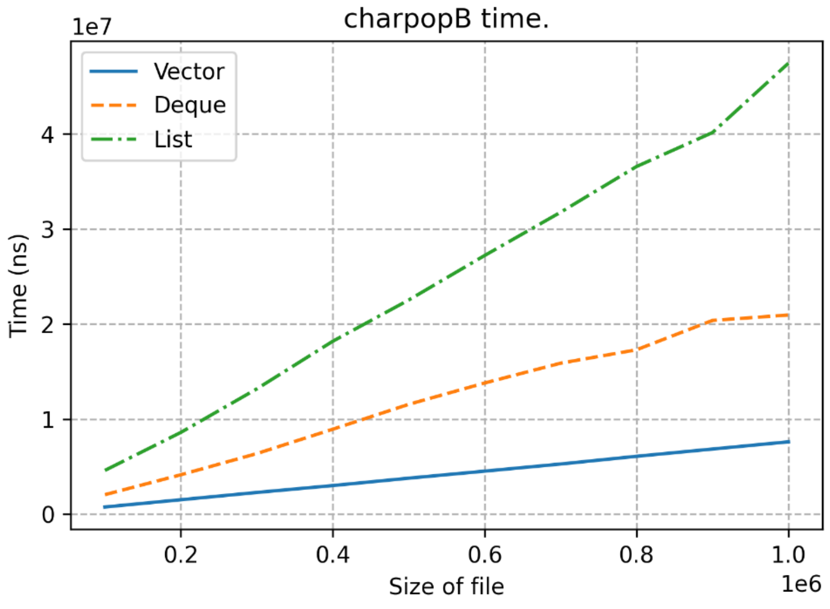

intSort time.
1e8
Vector
Deque
List
Time (ns)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0.2
0.4
0.6
0.8
1.0
Size of file
1e6

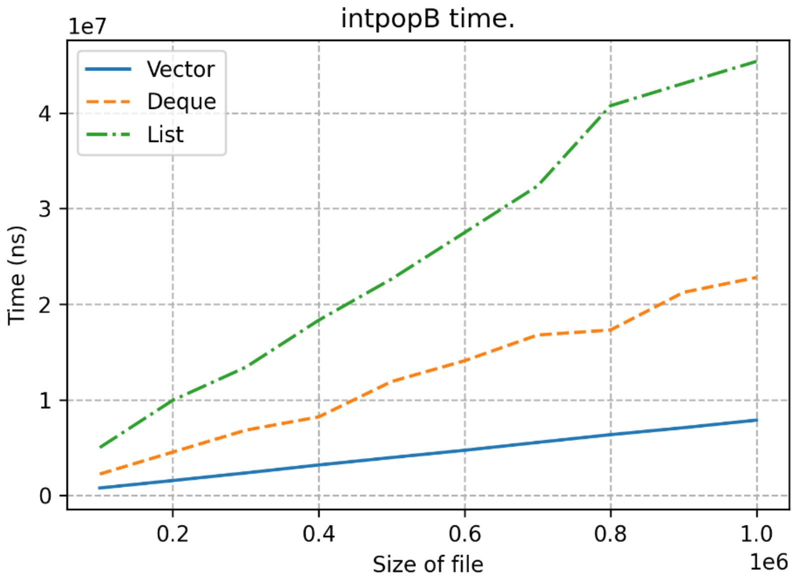

doubleSort time.
1e9
Vector
Deque
List
Time (ns)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
0.2
0.4
0.6
0.8
1.0
Size of file
1e6

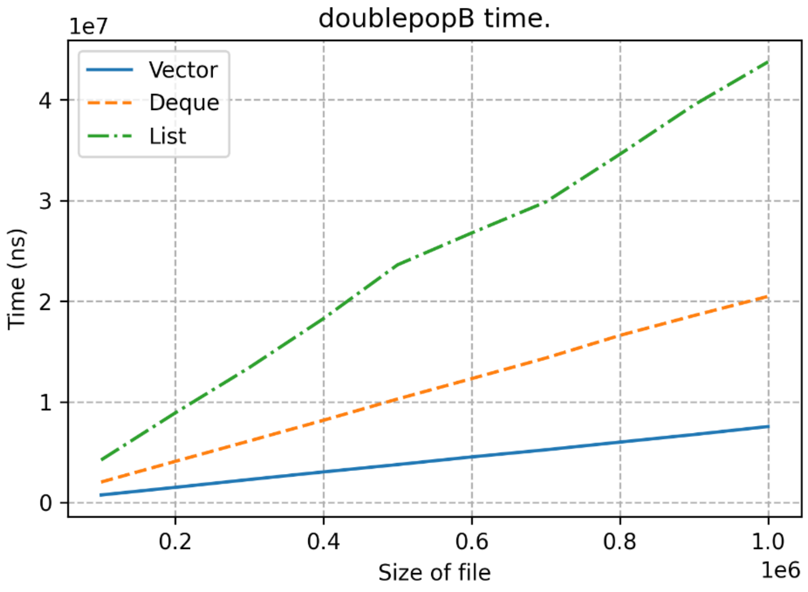


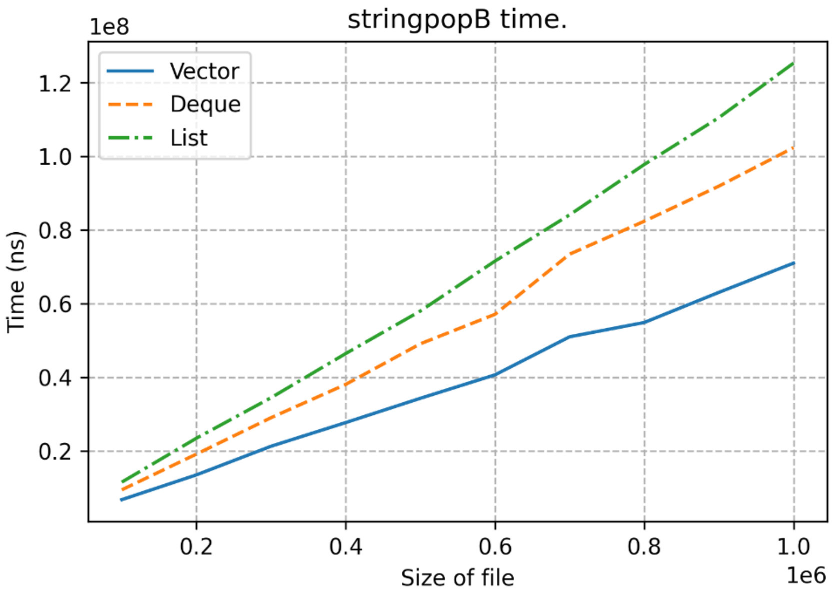
Ситуация, в целом, для char и int схожая, а вот в double к победителям по скорости присоединяется List, а для string и вовсе становится самым быстрым методом.
У List свой метод сортировки, поскольку там не random-access итератор.

4) Переходим к выкидыванию элементов! Pop_back



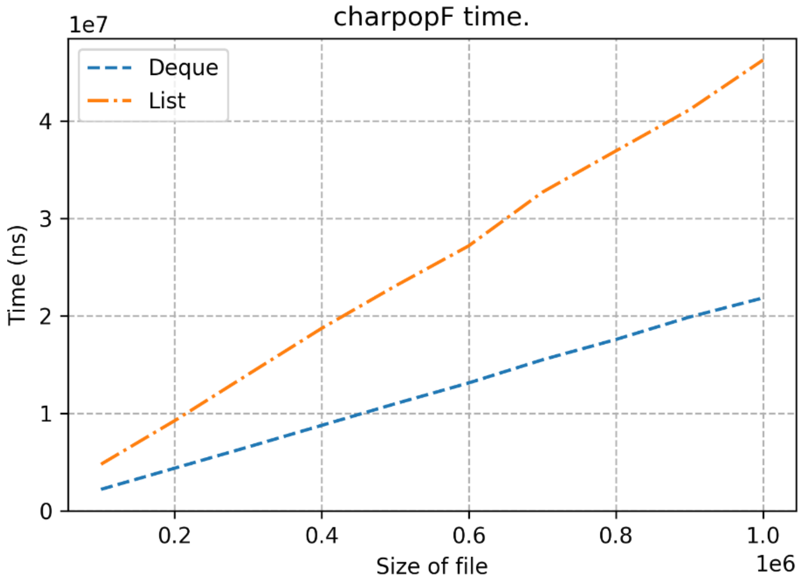


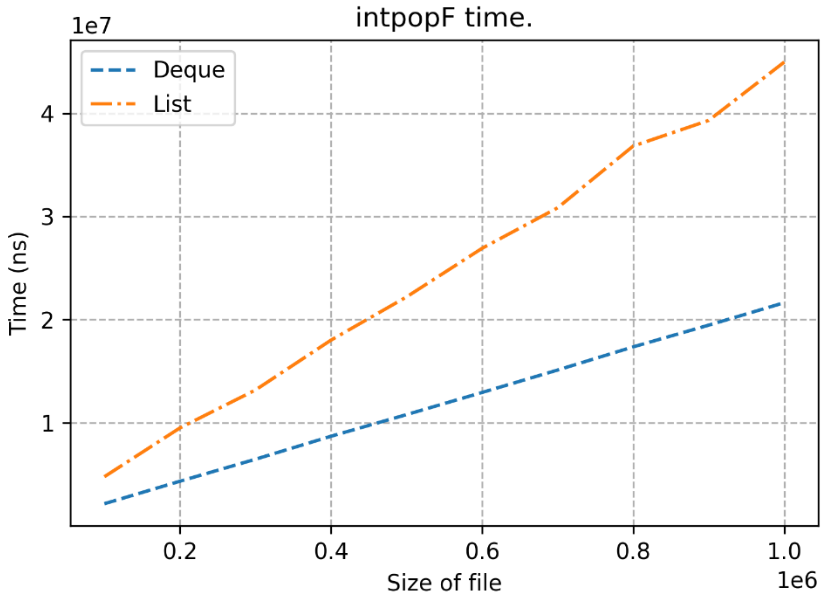


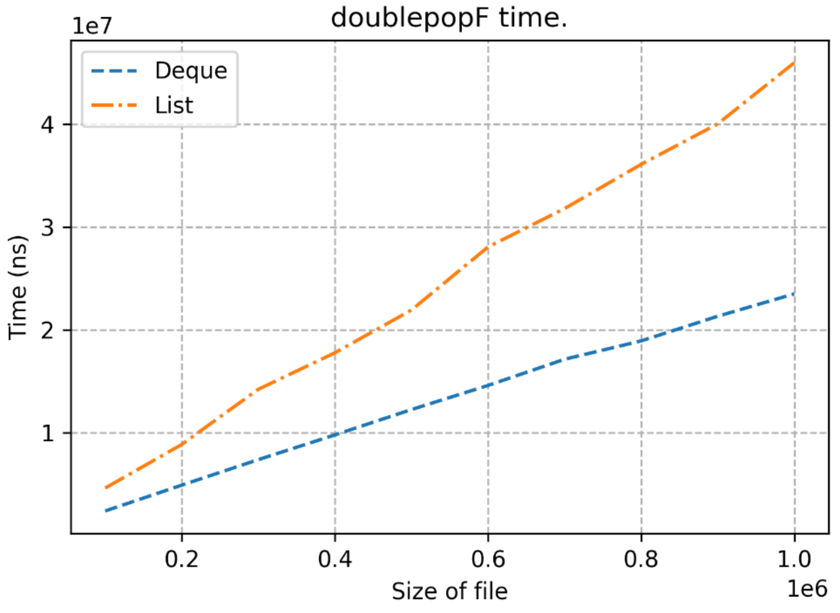


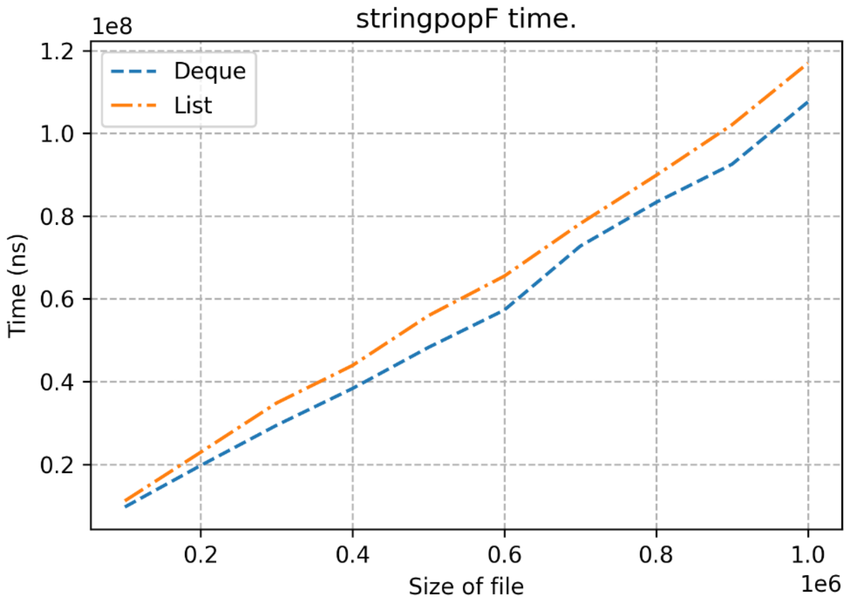
Тут для всех ситуаций тройка по скорости работы определяется однозначно.

5) Теперь pop_front!


charpopF time.
1e7
Deque
List
Time (ns)
0
1
2
3
4
0.2
0.4
0.6
0.8
1.0
Size of file
1e6

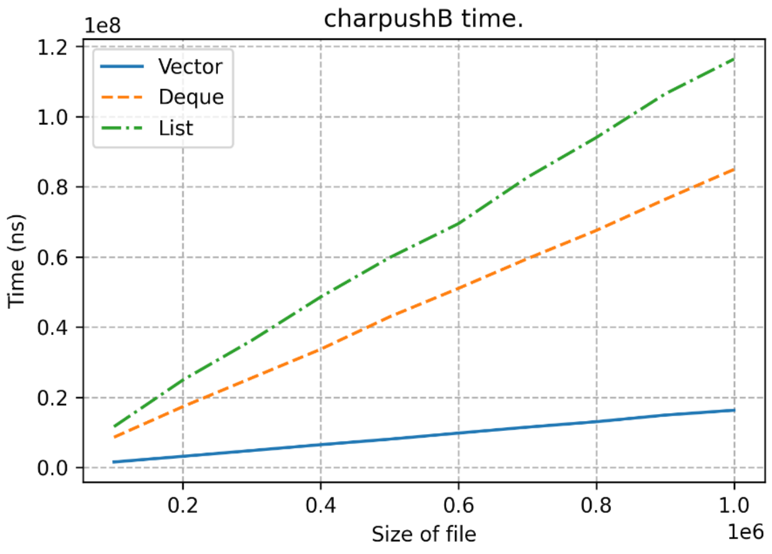

intpopF time.
1e7
Deque
List
Time (ns)
1
2
3
4
0.2
0.4
0.6
0.8
1.0
Size of file
1e6

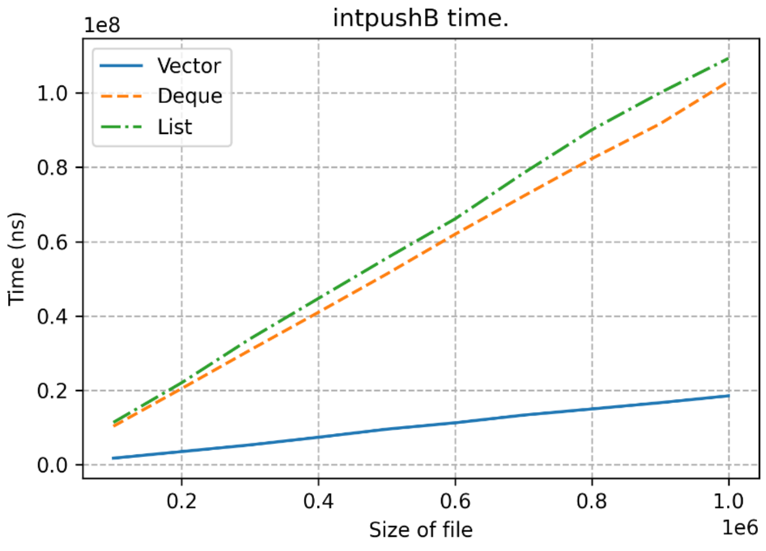

doublepopF time.
1e7
Deque
List
Time (ns)
1
2
3
4
0.2
0.4
0.6
0.8
1.0
Size of file
1e6

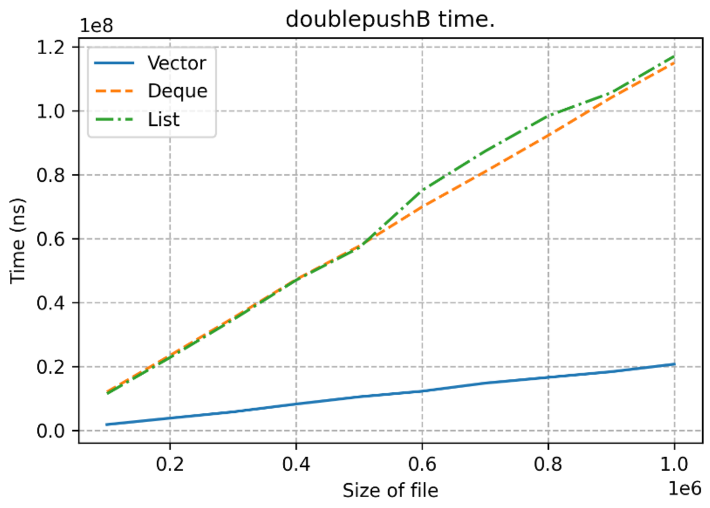


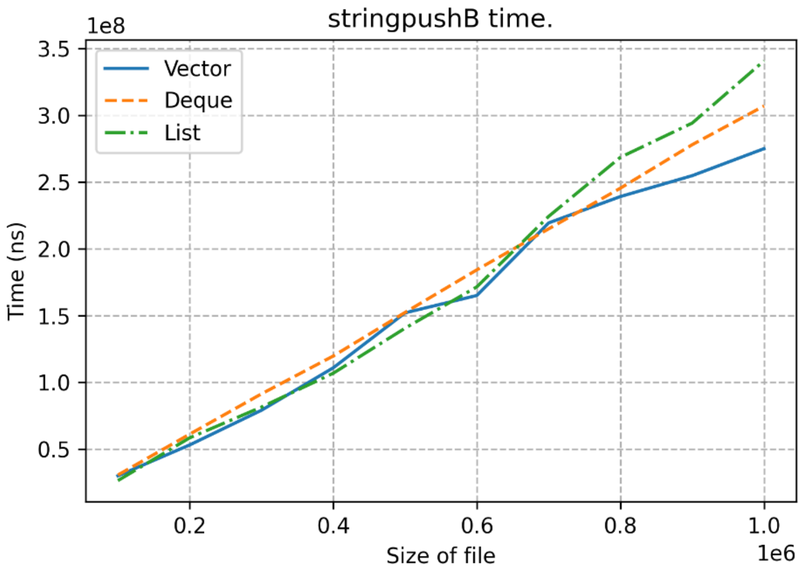
Контейнера vector нет, поскольку в нем не реализован метод pop_front (но костыльный вариант есть в коде). Работает долго(по расчетам около 6 часов уйдет)., смысла смотреть нет Тут по скорости все также однозначно!

6) Переходим к последним функциям – push! На очереди pushB!



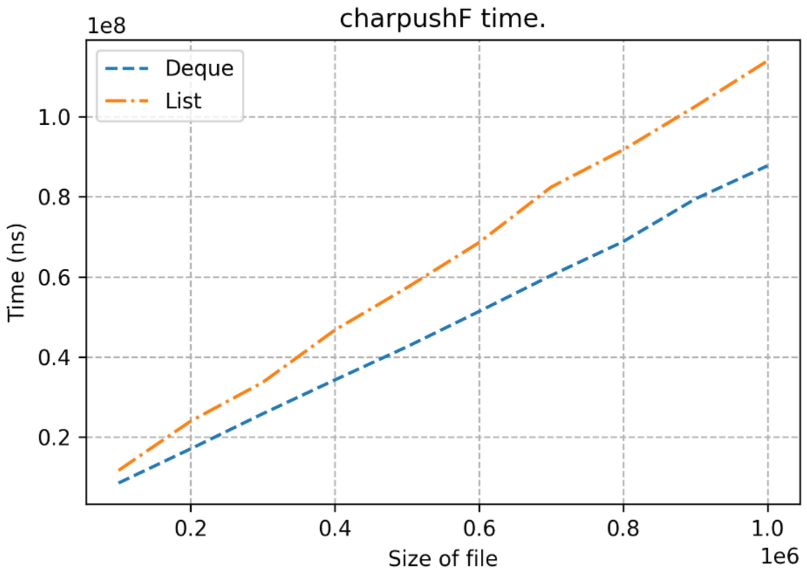


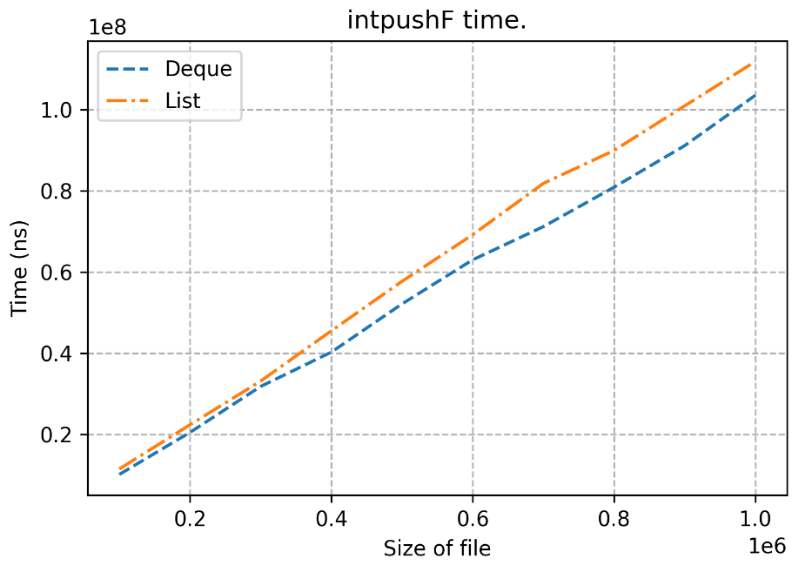


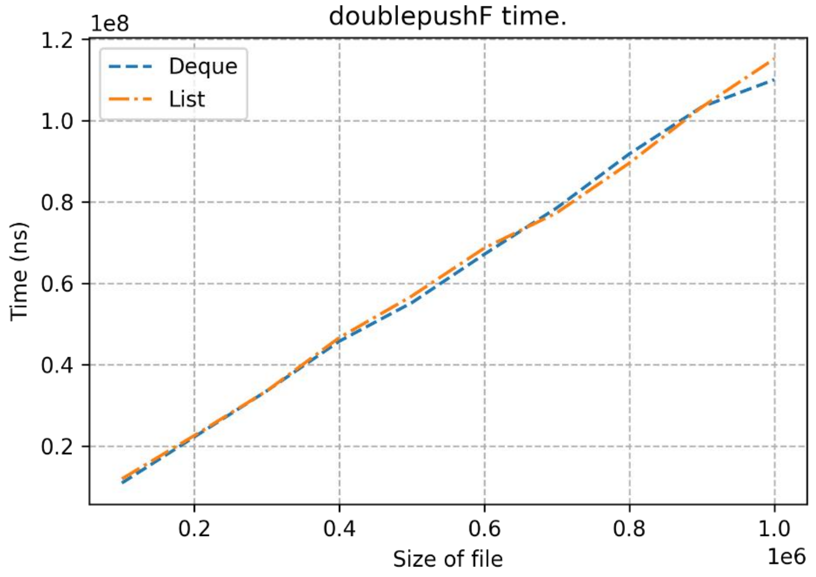


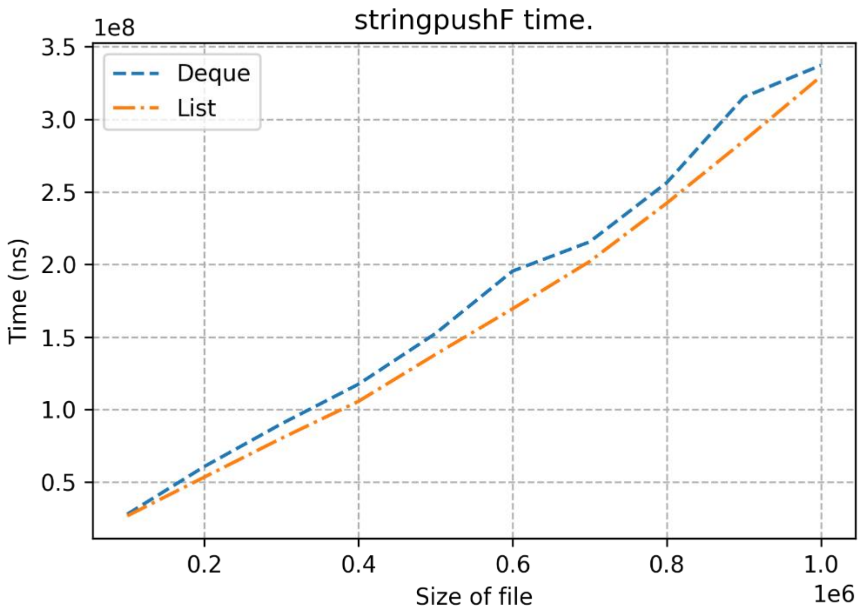
В целом, ситуации схожи! Только, в очередной раз, выделился случай со string – данными.

7) И последнее – pushF!









Результаты очень схожи! На string чуть лучше Deque, а на char и int – List.
Vector – контейнер опять отсутствует по причине того, что метод не реализован, а костыль работает долго

Код

Main.cpp
```
#include "TimeLogger.h"
#include "Load.h"
#include "Experiments.h"
int main() {
	experiments::ex_all();
}
```

TimeLogger.h
```
#pragma once

#include <iostream>
#include <fstream>
#include <chrono>
#include <string>

namespace time_log {

	class TimeLogger {
	public:
		TimeLogger(const std::string& path_to_out_file);

		using clock = std::chrono::high_resolution_clock;
		using time_point = std::chrono::time_point<clock>;

		void reset_start();
		void log_duration(int cont_size, const char* type, const char*
opt_message);
		void log_duration(int cont_size, const char* type);

	private:
		time_point m_start;
		std::ofstream m_fout;
	};
}
#pragma once

#include <vector>
#include <list>
#include <deque>
#include <string>
#include <iostream>
#include <fstream>

namespace data {
	std::vector<char> load_vector_char(const std::string& path_file);
	std::vector<int> load_vector_int(const std::string& path_file);
	std::vector<double> load_vector_double(const std::string& path_file);
	std::vector<std::string> load_vector_string(const std::string& path_file);
	std::list<char> load_list_char(const std::string& path_file);
	std::list<int> load_list_int(const std::string& path_file);
	std::list<double> load_list_double(const std::string& path_file);
	std::list<std::string> load_list_string(const std::string& path_file);
	std::deque<char> load_deque_char(const std::string& path_file);
	std::deque<int> load_deque_int(const std::string& path_file);
	std::deque<double> load_deque_double(const std::string& path_file);
	std::deque<std::string> load_deque_string(const std::string& path_file);
}
```

Load.h
```
#pragma once

#include <vector>
#include <list>
```

```
#include <deque>
#include <string>
#include <iostream>
#include <fstream>

namespace data {
	std::vector<char> load_vector_char(const std::string& path_file);
	std::vector<int> load_vector_int(const std::string& path_file);
	std::vector<double> load_vector_double(const std::string& path_file);
	std::vector<std::string> load_vector_string(const std::string& path_file);
	std::list<char> load_list_char(const std::string& path_file);
	std::list<int> load_list_int(const std::string& path_file);
	std::list<double> load_list_double(const std::string& path_file);
	std::list<std::string> load_list_string(const std::string& path_file);
	std::deque<char> load_deque_char(const std::string& path_file);
	std::deque<int> load_deque_int(const std::string& path_file);
	std::deque<double> load_deque_double(const std::string& path_file);
	std::deque<std::string> load_deque_string(const std::string& path_file);
}
```

Experiments.h

```
#pragma once

namespace experiments {
	void ex_all();

	void ex_find_test();
	void ex_sort_test();
	void ex_pop_bf_test();
	void ex_push_bf_test();

	void find_test_char();
	void find_test_int();
	void find_test_double();
	void find_test_string();

	void sort_test_char();
	void sort_test_int();
	void sort_test_double();
	void sort_test_string();

	void popB_test_char();
	void popB_test_int();
	void popB_test_double();
	void popB_test_string();

	void popF_test_char();
	void popF_test_int();
	void popF_test_double();
	void popF_test_string();

	void pushB_test_char();
	void pushB_test_int();
	void pushB_test_double();
	void pushB_test_string();

	void pushF_test_char();
	void pushF_test_int();
	void pushF_test_double();
	void pushF_test_string();

}
```

TimeLogger.cpp

```
#include "TimeLogger.h"

time_log::TimeLogger::TimeLogger(const std::string& path_to_out_file)
	: m_start(clock::now()), m_fout(path_to_out_file)
{
	if (!m_fout.is_open()) {
		std::cout << "The file can't be opened" << "\n";
		std::cout << "Name of file is:" << path_to_out_file << "\n";
		return;
	}
	m_fout << "sep=;\n";
	m_fout << "Duration (ns); Size; Type; Message\n";
}

void time_log::TimeLogger::reset_start() {
	m_start = clock::now();
}

void time_log::TimeLogger::log_duration(int cont_size, const char* type, const char* opt_message)
{
	std::chrono::nanoseconds duration = clock::now() - m_start;
	m_fout << duration.count() << ';' << cont_size << ';' << type << ';' << opt_message << '\n';
}

void time_log::TimeLogger::log_duration(int cont_size, const char* type) {
	std::chrono::nanoseconds duration = clock::now() - m_start;
	m_fout << duration.count() << ';' << cont_size << ';' << type << ';' << '\n';
}
```

Load.cpp

```
#include "Load.h"

// Сначала выпишем блок для векторов.
std::vector<char> data::load_vector_char(const std::string& path_file) {
	std::vector<char> vec;
	std::ifstream fin(path_file);

	if (!fin.is_open()) {
		std::cout << "The file can't be opened" << "\n";
		std::cout << "Name of file is:" << path_file << "\n";
		return vec;
	}

	for (char el; !fin.eof();) {
		if (fin >> el) {
			vec.push_back(el);
		}
	}

	fin.close();
	return vec;
}

std::vector<int> data::load_vector_int(const std::string& path_file) {
	std::vector<int> vec;
	std::ifstream fin(path_file);

	if (!fin.is_open()) {
		std::cout << "The file can't be opened";
		std::cout << "Name of file is:" << path_file << "\n";
		return vec;
	}

	for (int el; !fin.eof();) {
		if (fin >> el) {
			vec.push_back(el);
		}
	}

	fin.close();
	return vec;
}

std::vector<double> data::load_vector_double(const std::string& path_file) {
	std::vector<double> vec;
	std::ifstream fin(path_file);

	if (!fin.is_open()) {
		std::cout << "The file can't be opened";
		std::cout << "Name of file is:" << path_file << "\n";
		return vec;
	}

	for (double el; !fin.eof();) {
		if (fin >> el) {
			vec.push_back(el);
		}
	}

	fin.close();
	return vec;
}
```

```
std::vector<std::string> data::load_vector_string(const std::string& path_file) {
	std::vector<std::string> vec;
	std::ifstream fin(path_file);

	if (!fin.is_open()) {
		std::cout << "The file can't be opened";
		std::cout << "Name of file is:" << path_file << "\n";
		return vec;
	}

	for (std::string el; !fin.eof();) {
		if (fin >> el) {
			vec.push_back(el);
		}
	}

	fin.close();
	return vec;
}

// Теперь определим функции для листа.
std::list<char> data::load_list_char(const std::string& path_file) {
	std::list<char> lst;
	std::ifstream fin(path_file);

	if (!fin.is_open()) {
		std::cout << "The file can't be opened";
		std::cout << "Name of file is:" << path_file << "\n";
		return lst;
	}

	for (char el; !fin.eof();) {
		if (fin >> el) {
			lst.push_back(el);
		}
	}

	fin.close();
	return lst;
}

std::list<int> data::load_list_int(const std::string& path_file) {
	std::list<int> lst;
	std::ifstream fin(path_file);

	if (!fin.is_open()) {
		std::cout << "The file can't be opened";
		std::cout << "Name of file is:" << path_file << "\n";
		return lst;
	}

	for (int el; !fin.eof();) {
		if (fin >> el) {
			lst.push_back(el);
		}
	}

	fin.close();
	return lst;
}

std::list<double> data::load_list_double(const std::string& path_file) {
	std::list<double> lst;
	std::ifstream fin(path_file);

	if (!fin.is_open()) {
		std::cout << "The file can't be opened";
		std::cout << "Name of file is:" << path_file << "\n";
		return lst;
	}

	for (double el; !fin.eof();) {
		if (fin >> el) {
			lst.push_back(el);
		}
	}

	fin.close();
	return lst;
}

std::list<std::string> data::load_list_string(const std::string& path_file) {
	std::list<std::string> lst;
	std::ifstream fin(path_file);

	if (!fin.is_open()) {
		std::cout << "The file can't be opened";
		std::cout << "Name of file is:" << path_file << "\n";
		return lst;
	}

	for (std::string el; !fin.eof();) {
		if (fin >> el) {
			lst.push_back(el);
		}
	}

	fin.close();
	return lst;
}

// И, наконец, напишем реализацию для deque.
std::deque<char> data::load_deque_char(const std::string& path_file) {
	std::deque<char> deq;
	std::ifstream fin(path_file);

	if (!fin.is_open()) {
		std::cout << "The file can't be opened";
		std::cout << "Name of file is:" << path_file << "\n";
		return deq;
	}

	for (char el; !fin.eof();) {
		if (fin >> el) {
			deq.push_back(el);
		}
	}

	fin.close();
	return deq;
}

std::deque<int> data::load_deque_int(const std::string& path_file) {
	std::deque<int> deq;
	std::ifstream fin(path_file);

	if (!fin.is_open()) {
		std::cout << "The file can't be opened";
		std::cout << "Name of file is:" << path_file << "\n";
		return deq;
	}

	for (int el; !fin.eof();) {
		if (fin >> el) {
			deq.push_back(el);
		}
	}

	fin.close();
	return deq;
}

std::deque<double> data::load_deque_double(const std::string& path_file) {
	std::deque<double> deq;
	std::ifstream fin(path_file);

	if (!fin.is_open()) {
		std::cout << "The file can't be opened";
		std::cout << "Name of file is:" << path_file << "\n";
		return deq;
	}

	for (double el; !fin.eof();) {
		if (fin >> el) {
			deq.push_back(el);
		}
	}

	fin.close();
	return deq;
}

std::deque<std::string> data::load_deque_string(const std::string& path_file) {
	std::deque<std::string> deq;
	std::ifstream fin(path_file);

	if (!fin.is_open()) {
		std::cout << "The file can't be opened";
		std::cout << "Name of file is:" << path_file << "\n";
		return deq;
	}

	for (std::string el; !fin.eof();) {
		if (fin >> el) {
			deq.push_back(el);
		}
	}

	fin.close();
	return deq;
}
```

Experiments.cpp

```
#include "Experiments.h"
#include "Load.h"
#include "TimeLogger.h"
#include <string>
#include <algorithm>
#include <iterator>

std::string data_path = "C:\\Users\\ender\\Desktop\\LP1\\Generator\\Data\\";
std::string reults_path = "C:\\Users\\ender\\Desktop\\LP1\\Results\\";

void experiments::ex_all() {
	ex_find_test();
	ex_sort_test();
	ex_pop_bf_test();
	ex_push_bf_test();
}

void experiments::ex_find_test() {
	find_test_char();
	find_test_int();
	find_test_double();
	find_test_string();
}

void experiments::ex_sort_test() {
	sort_test_char();
	sort_test_int();
	sort_test_double();
	sort_test_string();
}

void experiments::ex_pop_bf_test() {
	popB_test_char();
	popB_test_int();
	popB_test_double();
	popB_test_string();

	popF_test_char();
	popF_test_int();
	popF_test_double();
	popF_test_string();
}

void experiments::ex_push_bf_test() {
	pushB_test_char();
	pushB_test_int();
	pushB_test_double();
	pushB_test_string();

	pushF_test_char();
	pushF_test_int();
	pushF_test_double();
	pushF_test_string();
}


// 1 блок - поиск.
void experiments::find_test_char() {
	time_log::TimeLogger timer = time_log::TimeLogger(reults_path + 
"find_test_char.csv");
	std::string TD = "char_";

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::vector<char> arr = data::load_vector_char(data_path + TD + data_size 
+ ".txt");
		timer.reset_start();
		std::find(arr.begin(), arr.end(), '/');
		timer.log_duration(i, "Vector");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::deque<char> arr = data::load_deque_char(data_path + TD + data_size + 
".txt");
		timer.reset_start();
		std::find(arr.begin(), arr.end(), '/');
		timer.log_duration(i, "Deque");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::list<char> arr = data::load_list_char(data_path + TD + data_size + 
".txt");
		timer.reset_start();
		std::find(arr.begin(), arr.end(), '/');
		timer.log_duration(i, "List");
	}
}

void experiments::find_test_int() {
	time_log::TimeLogger timer = time_log::TimeLogger(reults_path + 
"find_test_int.csv");
	std::string TD = "int_";

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::vector<int> arr = data::load_vector_int(data_path + TD + data_size + 
".txt");
		timer.reset_start();
		std::find(arr.begin(), arr.end(), '/');
		timer.log_duration(i, "Vector");
	}
	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::deque<int> arr = data::load_deque_int(data_path + TD + data_size +
".txt");
		timer.reset_start();
		std::find(arr.begin(), arr.end(), '/');
		timer.log_duration(i, "Deque");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::list<int> arr = data::load_list_int(data_path + TD + data_size +
".txt");
		timer.reset_start();
		std::find(arr.begin(), arr.end(), '/');
		timer.log_duration(i, "List");
	}
}

void experiments::find_test_double() {
	time_log::TimeLogger timer = time_log::TimeLogger(reults_path +
"find_test_double.csv");
	std::string TD = "double_";

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::vector<double> arr = data::load_vector_double(data_path + TD +
data_size + ".txt");
		timer.reset_start();
		std::find(arr.begin(), arr.end(), '/');
		timer.log_duration(i, "Vector");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::deque<double> arr = data::load_deque_double(data_path + TD +
data_size + ".txt");
		timer.reset_start();
		std::find(arr.begin(), arr.end(), '/');
		timer.log_duration(i, "Deque");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::list<double> arr = data::load_list_double(data_path + TD + data_size
+ ".txt");
		timer.reset_start();
		std::find(arr.begin(), arr.end(), '/');
		timer.log_duration(i, "List");
	}
}

void experiments::find_test_string() {
	time_log::TimeLogger timer = time_log::TimeLogger(reults_path +
"find_test_string.csv");
	std::string TD = "string_";

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::vector<std::string> arr = data::load_vector_string(data_path + TD +
data_size + ".txt");
		timer.reset_start();
		std::find(arr.begin(), arr.end(), "/");
		timer.log_duration(i, "Vector");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
```

```
            std::deque<std::string> arr = data::load_deque_string(data_path + TD +
data_size + ".txt");
            timer.reset_start();
            std::find(arr.begin(), arr.end(), "/");
            timer.log_duration(i, "Deque");
      }

      for (int i = 100000; i <= 1000000; i += 100000) {
            std::string data_size = std::to_string(i);
            std::list<std::string> arr = data::load_list_string(data_path + TD +
data_size + ".txt");
            timer.reset_start();
            std::find(arr.begin(), arr.end(), "/");
            timer.log_duration(i, "List");
      }
}

// 2 блок - сортировка.
void experiments::sort_test_char() {
      time_log::TimeLogger timer = time_log::TimeLogger(reults_path +
"sort_test_char.csv");
      std::string TD = "char_";

      for (int i = 100000; i <= 1000000; i += 100000) {
            std::string data_size = std::to_string(i);
            std::vector<char> arr = data::load_vector_char(data_path + TD + data_size
+ ".txt");
            timer.reset_start();
            std::sort(arr.begin(), arr.end());
            timer.log_duration(i, "Vector");
      }

      for (int i = 100000; i <= 1000000; i += 100000) {
            std::string data_size = std::to_string(i);
            std::deque<char> arr = data::load_deque_char(data_path + TD + data_size +
".txt");
            timer.reset_start();
            std::sort(arr.begin(), arr.end());
            timer.log_duration(i, "Deque");
      }

      for (int i = 100000; i <= 1000000; i += 100000) {
            std::string data_size = std::to_string(i);
            std::list<char> arr = data::load_list_char(data_path + TD + data_size +
".txt");
            timer.reset_start();
            arr.sort();
            timer.log_duration(i, "List");
      }
}

void experiments::sort_test_int() {
      time_log::TimeLogger timer = time_log::TimeLogger(reults_path +
"sort_test_int.csv");
      std::string TD = "int_";

      for (int i = 100000; i <= 1000000; i += 100000) {
            std::string data_size = std::to_string(i);
            std::vector<int> arr = data::load_vector_int(data_path + TD + data_size +
".txt");
            timer.reset_start();
            std::sort(arr.begin(), arr.end());
            timer.log_duration(i, "Vector");
      }

      for (int i = 100000; i <= 1000000; i += 100000) {
            std::string data_size = std::to_string(i);
            std::deque<int> arr = data::load_deque_int(data_path + TD + data_size +
".txt");
```

```
		timer.reset_start();
		std::sort(arr.begin(), arr.end());
		timer.log_duration(i, "Deque");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::list<int> arr = data::load_list_int(data_path + TD + data_size +
".txt");
		timer.reset_start();
		arr.sort();
		timer.log_duration(i, "List");
	}
}

void experiments::sort_test_double() {
	time_log::TimeLogger timer = time_log::TimeLogger(reults_path +
"sort_test_double.csv");
	std::string TD = "double_";

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::vector<double> arr = data::load_vector_double(data_path + TD +
data_size + ".txt");
		timer.reset_start();
		std::sort(arr.begin(), arr.end());
		timer.log_duration(i, "Vector");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::deque<double> arr = data::load_deque_double(data_path + TD +
data_size + ".txt");
		timer.reset_start();
		std::sort(arr.begin(), arr.end());
		timer.log_duration(i, "Deque");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::list<double> arr = data::load_list_double(data_path + TD + data_size
+ ".txt");
		timer.reset_start();
		arr.sort();
		timer.log_duration(i, "List");
	}
}

void experiments::sort_test_string() {
	time_log::TimeLogger timer = time_log::TimeLogger(reults_path +
"sort_test_string.csv");
	std::string TD = "string_";

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::vector<std::string> arr = data::load_vector_string(data_path + TD +
data_size + ".txt");
		timer.reset_start();
		std::sort(arr.begin(), arr.end());
		timer.log_duration(i, "Vector");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::deque<std::string> arr = data::load_deque_string(data_path + TD +
data_size + ".txt");
		timer.reset_start();
		std::sort(arr.begin(), arr.end());
		timer.log_duration(i, "Deque");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::list<std::string> arr = data::load_list_string(data_path + TD + 
data_size + ".txt");
		timer.reset_start();
		arr.sort();
		timer.log_duration(i, "List");
	}
}

// 3.1 блок - popB.
void experiments::popB_test_char() {
	time_log::TimeLogger timer = time_log::TimeLogger(reults_path + 
"popB_test_char.csv");
	std::string TD = "char_";

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::vector<char> arr = data::load_vector_char(data_path + TD + data_size 
+ ".txt");
		timer.reset_start();
		while (arr.size() > 0)
			arr.pop_back();
		timer.log_duration(i, "Vector");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::deque<char> arr = data::load_deque_char(data_path + TD + data_size + 
".txt");
		timer.reset_start();
		while (arr.size() > 0)
			arr.pop_back();
		timer.log_duration(i, "Deque");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::list<char> arr = data::load_list_char(data_path + TD + data_size + 
".txt");
		timer.reset_start();
		while (arr.size() > 0)
			arr.pop_back();
		timer.log_duration(i, "List");
	}
}

void experiments::popB_test_int() {
	time_log::TimeLogger timer = time_log::TimeLogger(reults_path + 
"popB_test_int.csv");
	std::string TD = "int_";

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::vector<int> arr = data::load_vector_int(data_path + TD + data_size + 
".txt");
		timer.reset_start();
		while (arr.size() > 0)
			arr.pop_back();
		timer.log_duration(i, "Vector");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::deque<int> arr = data::load_deque_int(data_path + TD + data_size + 
".txt");
		timer.reset_start();
		while (arr.size() > 0)
			arr.pop_back();
		timer.log_duration(i, "Deque");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::list<int> arr = data::load_list_int(data_path + TD + data_size +
".txt");
		timer.reset_start();
		while (arr.size() > 0)
			arr.pop_back();
		timer.log_duration(i, "List");
	}
}

void experiments::popB_test_double() {
	time_log::TimeLogger timer = time_log::TimeLogger(reults_path +
"popB_test_double.csv");
	std::string TD = "double_";

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::vector<double> arr = data::load_vector_double(data_path + TD +
data_size + ".txt");
		timer.reset_start();
		while (arr.size() > 0)
			arr.pop_back();
		timer.log_duration(i, "Vector");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::deque<double> arr = data::load_deque_double(data_path + TD +
data_size + ".txt");
		timer.reset_start();
		while (arr.size() > 0)
			arr.pop_back();
		timer.log_duration(i, "Deque");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::list<double> arr = data::load_list_double(data_path + TD + data_size
+ ".txt");
		timer.reset_start();
		while (arr.size() > 0)
			arr.pop_back();
		timer.log_duration(i, "List");
	}
}

void experiments::popB_test_string() {
	time_log::TimeLogger timer = time_log::TimeLogger(reults_path +
"popB_test_string.csv");
	std::string TD = "string_";

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::vector<std::string> arr = data::load_vector_string(data_path + TD +
data_size + ".txt");
		timer.reset_start();
		while (arr.size() > 0)
			arr.pop_back();
		timer.log_duration(i, "Vector");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::deque<std::string> arr = data::load_deque_string(data_path + TD +
data_size + ".txt");
		timer.reset_start();
		while (arr.size() > 0)
			arr.pop_back();
		timer.log_duration(i, "Deque");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::list<std::string> arr = data::load_list_string(data_path + TD +
data_size + ".txt");
		timer.reset_start();
		while (arr.size() > 0)
			arr.pop_back();
		timer.log_duration(i, "List");
	}
}

// 3.2 блок - popF;
void experiments::popF_test_char() {
	time_log::TimeLogger timer = time_log::TimeLogger(reults_path +
"popF_test_char.csv");
	std::string TD = "char_";

	/* Долго работает(!!!). Мне передали, что можно написать, но не использовать.
	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::vector<char> arr = data::load_vector_char(data_path + TD + data_size
+ ".txt");
		timer.reset_start();
		while (arr.size() > 0)
			arr.erase(arr.begin());
		timer.log_duration(i, "Vector");
	}
	*/

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::deque<char> arr = data::load_deque_char(data_path + TD + data_size +
".txt");
		timer.reset_start();
		while (arr.size() > 0)
			arr.pop_front();
		timer.log_duration(i, "Deque");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::list<char> arr = data::load_list_char(data_path + TD + data_size +
".txt");
		timer.reset_start();
		while (arr.size() > 0)
			arr.pop_front();
		timer.log_duration(i, "List");
	}
}

void experiments::popF_test_int() {
	time_log::TimeLogger timer = time_log::TimeLogger(reults_path +
"popF_test_int.csv");
	std::string TD = "int_";

	/* Долго работает(!!!). Мне передали, что можно написать, но не использовать.
	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::vector<int> arr = data::load_vector_int(data_path + TD + data_size +
".txt");
		timer.reset_start();
		while (arr.size() > 0)
			arr.erase(arr.begin());
		timer.log_duration(i, "Vector");
	}
	*/

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::deque<int> arr = data::load_deque_int(data_path + TD + data_size +
".txt");
		timer.reset_start();
		while (arr.size() > 0)
			arr.pop_front();
		timer.log_duration(i, "Deque");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::list<int> arr = data::load_list_int(data_path + TD + data_size +
".txt");
		timer.reset_start();
		while (arr.size() > 0)
			arr.pop_front();
		timer.log_duration(i, "List");
	}
}

void experiments::popF_test_double() {
	time_log::TimeLogger timer = time_log::TimeLogger(reults_path +
"popF_test_double.csv");
	std::string TD = "double_";

	/* Долго работает(!!!). Мне передали, что можно написать, но не использовать.
	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::vector<double> arr = data::load_vector_double(data_path + TD +
data_size + ".txt");
		timer.reset_start();
		while (arr.size() > 0)
			arr.erase(arr.begin());
		timer.log_duration(i, "Vector");
	}
	*/

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::deque<double> arr = data::load_deque_double(data_path + TD +
data_size + ".txt");
		timer.reset_start();
		while (arr.size() > 0)
			arr.pop_front();
		timer.log_duration(i, "Deque");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::list<double> arr = data::load_list_double(data_path + TD + data_size
+ ".txt");
		timer.reset_start();
		while (arr.size() > 0)
			arr.pop_front();
		timer.log_duration(i, "List");
	}
}

void experiments::popF_test_string() {
	time_log::TimeLogger timer = time_log::TimeLogger(reults_path +
"popF_test_string.csv");
	std::string TD = "string_";
	/* Долго работает(!!!). Мне передали, что можно написать, но не использовать.
	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::vector<std::string> arr = data::load_vector_string(data_path + TD +
data_size + ".txt");
		timer.reset_start();
		while (arr.size() > 0)
			arr.erase(arr.begin());
		timer.log_duration(i, "Vector");
	}
	*/

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::deque<std::string> arr = data::load_deque_string(data_path + TD +
data_size + ".txt");
		timer.reset_start();
		while (arr.size() > 0)
			arr.pop_front();
		timer.log_duration(i, "Deque");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::list<std::string> arr = data::load_list_string(data_path + TD +
data_size + ".txt");
		timer.reset_start();
		while (arr.size() > 0)
			arr.pop_front();
		timer.log_duration(i, "List");
	}
}

// 4.1 блок - pushB.
void experiments::pushB_test_char() {
	time_log::TimeLogger timer = time_log::TimeLogger(reults_path +
"pushB_test_char.csv");
	std::string TD = "char_";

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::vector<char> arr = data::load_vector_char(data_path + TD + data_size
+ ".txt");
		std::vector<char> tmp;
		timer.reset_start();
		for (int i = 0; i < arr.size(); ++i) {
			tmp.push_back(arr[i]);
		}
		timer.log_duration(i, "Vector");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::deque<char> arr = data::load_deque_char(data_path + TD + data_size +
".txt");
		std::deque<char> tmp;
		timer.reset_start();
		for (int i = 0; i < arr.size(); ++i) {
			tmp.push_back(arr[i]);
		}
		timer.log_duration(i, "Deque");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::list<char> arr = data::load_list_char(data_path + TD + data_size +
".txt");
		std::list<char> tmp;
		std::list<char>::iterator it = arr.begin();
		timer.reset_start();
		while (it != arr.end()) {
			tmp.push_back(*it);
			++it;
		}
		timer.log_duration(i, "List");
	}
}

void experiments::pushB_test_int() {
	time_log::TimeLogger timer = time_log::TimeLogger(reults_path +
"pushB_test_int.csv");
	std::string TD = "int_";

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::vector<int> arr = data::load_vector_int(data_path + TD + data_size +
".txt");
		std::vector<int> tmp;
		timer.reset_start();
		for (int i = 0; i < arr.size(); ++i) {
			tmp.push_back(arr[i]);
		}
		timer.log_duration(i, "Vector");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::deque<int> arr = data::load_deque_int(data_path + TD + data_size +
".txt");
		std::deque<int> tmp;
		timer.reset_start();
		for (int i = 0; i < arr.size(); ++i) {
			tmp.push_back(arr[i]);
		}
		timer.log_duration(i, "Deque");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::list<int> arr = data::load_list_int(data_path + TD + data_size +
".txt");
		std::list<int> tmp;
		std::list<int>::iterator it = arr.begin();
		timer.reset_start();
		while (it != arr.end()) {
			tmp.push_back(*it);
			++it;
		}
		timer.log_duration(i, "List");
	}
}

void experiments::pushB_test_double() {
	time_log::TimeLogger timer = time_log::TimeLogger(reults_path +
"pushB_test_double.csv");
	std::string TD = "double_";

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::vector<double> arr = data::load_vector_double(data_path + TD +
data_size + ".txt");
		std::vector<double> tmp;
		timer.reset_start();
		for (int i = 0; i < arr.size(); ++i) {
			tmp.push_back(arr[i]);
		}
		timer.log_duration(i, "Vector");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::deque<double> arr = data::load_deque_double(data_path + TD +
data_size + ".txt");
		std::deque<double> tmp;
		timer.reset_start();
		for (int i = 0; i < arr.size(); ++i) {
			tmp.push_back(arr[i]);
		}
		timer.log_duration(i, "Deque");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::list<double> arr = data::load_list_double(data_path + TD + data_size
+ ".txt");
		std::list<double> tmp;
		std::list<double>::iterator it = arr.begin();
		timer.reset_start();
		while (it != arr.end()) {
			tmp.push_back(*it);
			++it;
		}
		timer.log_duration(i, "List");
	}
}

void experiments::pushB_test_string() {
	time_log::TimeLogger timer = time_log::TimeLogger(reults_path +
"pushB_test_string.csv");
	std::string TD = "string_";

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::vector<std::string> arr = data::load_vector_string(data_path + TD +
data_size + ".txt");
		std::vector<std::string> tmp;
		timer.reset_start();
		for (int i = 0; i < arr.size(); ++i) {
			tmp.push_back(arr[i]);
		}
		timer.log_duration(i, "Vector");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::deque<std::string> arr = data::load_deque_string(data_path + TD +
data_size + ".txt");
		std::deque<std::string> tmp;
		timer.reset_start();
		for (int i = 0; i < arr.size(); ++i) {
			tmp.push_back(arr[i]);
		}
		timer.log_duration(i, "Deque");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::list<std::string> arr = data::load_list_string(data_path + TD +
data_size + ".txt");
		std::list<std::string> tmp;
		std::list<std::string>::iterator it = arr.begin();
		timer.reset_start();
		while (it != arr.end()) {
			tmp.push_back(*it);
			++it;
		}
```

```
            timer.log_duration(i, "List");
        }
}

// 4.2 блок - pushF.
void experiments::pushF_test_char() {
        time_log::TimeLogger timer = time_log::TimeLogger(reults_path +
"pushF_test_char.csv");
        std::string TD = "char_";

        /* Долго работает(!!!). Мне передали, что можно написать, но не использовать.
        for (int i = 100000; i <= 1000000; i += 100000) {
                std::string data_size = std::to_string(i);
                std::vector<char> arr = data::load_vector_char(data_path + TD + data_size
+ ".txt");
                std::vector<char> tmp;
                timer.reset_start();
                for (int i = 0; i < arr.size(); ++i) {
                        tmp.insert(tmp.begin(), arr[i]);
                }
                timer.log_duration(i, "Vector");
        }
        */

        for (int i = 100000; i <= 1000000; i += 100000) {
                std::string data_size = std::to_string(i);
                std::deque<char> arr = data::load_deque_char(data_path + TD + data_size +
".txt");
                std::deque<char> tmp;
                timer.reset_start();
                for (int i = 0; i < arr.size(); ++i) {
                        tmp.push_front(arr[i]);
                }
                timer.log_duration(i, "Deque");
        }

        for (int i = 100000; i <= 1000000; i += 100000) {
                std::string data_size = std::to_string(i);
                std::list<char> arr = data::load_list_char(data_path + TD + data_size +
".txt");
                std::list<char> tmp;
                std::list<char>::iterator it = arr.begin();
                timer.reset_start();
                while (it != arr.end()) {
                        tmp.push_front(*it);
                        ++it;
                }
                timer.log_duration(i, "List");
        }
}

void experiments::pushF_test_int() {
        time_log::TimeLogger timer = time_log::TimeLogger(reults_path +
"pushF_test_int.csv");
        std::string TD = "int_";

        /* Долго работает(!!!). Мне передали, что можно написать, но не использовать.
        for (int i = 100000; i <= 1000000; i += 100000) {
                std::string data_size = std::to_string(i);
                std::vector<int> arr = data::load_vector_int(data_path + TD + data_size +
".txt");
                std::vector<int> tmp;
                timer.reset_start();
                for (int i = 0; i < arr.size(); ++i) {
                        tmp.insert(tmp.begin(), arr[i]);
                }
                timer.log_duration(i, "Vector");
        }
        */
```

```
	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::deque<int> arr = data::load_deque_int(data_path + TD + data_size +
".txt");
		std::deque<int> tmp;
		timer.reset_start();
		for (int i = 0; i < arr.size(); ++i) {
			tmp.push_front(arr[i]);
		}
		timer.log_duration(i, "Deque");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::list<int> arr = data::load_list_int(data_path + TD + data_size +
".txt");
		std::list<int> tmp;
		std::list<int>::iterator it = arr.begin();
		timer.reset_start();
		while (it != arr.end()) {
			tmp.push_front(*it);
			++it;
		}
		timer.log_duration(i, "List");
	}
}

void experiments::pushF_test_double() {
	time_log::TimeLogger timer = time_log::TimeLogger(reults_path +
"pushF_test_double.csv");
	std::string TD = "double_";

	/* Долго работает(!!!). Мне передали, что можно написать, но не использовать.
	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::vector<double> arr = data::load_vector_double(data_path + TD +
data_size + ".txt");
		std::vector<double> tmp;
		timer.reset_start();
		for (int i = 0; i < arr.size(); ++i) {
			tmp.insert(tmp.begin(), arr[i]);
		}
		timer.log_duration(i, "Vector");
	}
	*/

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::deque<double> arr = data::load_deque_double(data_path + TD +
data_size + ".txt");
		std::deque<double> tmp;
		timer.reset_start();
		for (int i = 0; i < arr.size(); ++i) {
			tmp.push_front(arr[i]);
		}
		timer.log_duration(i, "Deque");
	}

	for (int i = 100000; i <= 1000000; i += 100000) {
		std::string data_size = std::to_string(i);
		std::list<double> arr = data::load_list_double(data_path + TD + data_size
+ ".txt");
		std::list<double> tmp;
		std::list<double>::iterator it = arr.begin();
		timer.reset_start();
		while (it != arr.end()) {
			tmp.push_front(*it);
			++it;
```

```
            }
            timer.log_duration(i, "List");
      }
}

void experiments::pushF_test_string() {
      time_log::TimeLogger timer = time_log::TimeLogger(reults_path +
"pushF_test_string.csv");
      std::string TD = "string_";

      /* Долго работает(!!!). Мне передали, что можно написать, но не использовать.
      for (int i = 100000; i <= 1000000; i += 100000) {
            std::string data_size = std::to_string(i);
            std::vector<std::string> arr = data::load_vector_string(data_path + TD +
data_size + ".txt");
            std::vector<std::string> tmp;
            timer.reset_start();
            for (int i = 0; i < arr.size(); ++i) {
                  tmp.insert(tmp.begin(), arr[i]);
            }
            timer.log_duration(i, "Vector");
      }
      */

      for (int i = 100000; i <= 1000000; i += 100000) {
            std::string data_size = std::to_string(i);
            std::deque<std::string> arr = data::load_deque_string(data_path + TD +
data_size + ".txt");
            std::deque<std::string> tmp;
            timer.reset_start();
            for (int i = 0; i < arr.size(); ++i) {
                  tmp.push_front(arr[i]);
            }
            timer.log_duration(i, "Deque");
      }

      for (int i = 100000; i <= 1000000; i += 100000) {
            std::string data_size = std::to_string(i);
            std::list<std::string> arr = data::load_list_string(data_path + TD +
data_size + ".txt");
            std::list<std::string> tmp;
            std::list<std::string>::iterator it = arr.begin();
            timer.reset_start();
            while (it != arr.end()) {
                  tmp.push_front(*it);
                  ++it;
            }
            timer.log_duration(i, "List");
      }
}
```