МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра вычислительной техники

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: Измерение временной сложности алгоритма

Студент гр. 5305 _____ Билькис П.П., Исмаилов Т.Э.

Преподаватель _____ Колинько П.Г.

Санкт-Петербург

2017

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

1. Цель работы

Экспериментально измерить временную сложность алгоритмов, реализованных в лабораторной работе №6.

2. Задание

Выполнить статистический эксперимент по измерению временной сложности алгоритма обработки данных, использующего стандартную библиотеку шаблонов (тема лабораторной работы №6).
Доработать программу таким образом, чтобы она генерировала множества различной мощности в некоторых пределах (например [10,200]), измеряла время выполнения цепочки операция над множествами и последовательностями и выводила результат в текстовой файл.
Для повышения надёжности эксперимента предусмотреть в программе перехват исключительных ситуаций таким образом, чтобы сбой сводился просто к пропуску очередного шага эксперимента.

3. Уточнение задания

Цепочка операций должна состоять из операций, реализованных при выполнении лабораторной работы №6, а именно:

- merge — слияние.
- concat — сцепление.
- change — замена.
- intersection — пересечение.
- difference — разность.
- union — объединение.
- symmetric difference — исключающее ИЛИ.

1. ПОДГОТОВКА К ЭКСПЕРИМЕНТУ

1.1. Способ представления данных

Данные хранятся в контейнерах: multiset для хранения множества и vector для поддержки последовательностей. Для удобства, контейнеры объединены в класс stud_set, где values – множество, a sequence вектор итераторов.

1.2. Последовательность операций

В массиве arrayofsets хранятся пять множеств.

```
stud_union(arrayofsets[0], arrayofsets[1], arrayofsets[2]);
stud_merge(arrayofsets[2], arrayofsets[1], arrayofsets[3]);
stud_concat(arrayofsets[2], arrayofsets[3], arrayofsets[0]);
stud_concat(arrayofsets[0], arrayofsets[1], arrayofsets[2]);
stud_change(arrayofsets[4], arrayofsets[2], arrayofsets[3], 15);
stud_xor(arrayofsets[1], arrayofsets[3], arrayofsets[1]);
stud_diff(arrayofsets[4], arrayofsets[0], arrayofsets[2]);
stud_and(arrayofsets[4], arrayofsets[2], arrayofsets[3]);
```

1.3. Теоретическая временная сложность операций

STL	Оценка временной сложности	
Операция	В худшем случае	В среднем
Вставка	$O(logn)$	$O(logn)$
Удаление	$O(logn)$	$O(logn)$
Поиск	$O(logn)$	$O(logn)$
$A \cup B$	$O(n)$	$O(n)$
$A \cap B$	$O(n)$	$O(n)$
$A \setminus B$	$O(n)$	$O(n)$
$A \oplus B$	$O(n)$	$O(n)$
MERGE	$O(n)$	$O(n)$
CONCAT	$O(n)$	$O(n)$
CHANGE	$O(n)$	$O(n)$

2. РЕЗУЛЬТАТ ЭКСПЕРИМЕНТА

2.1. Вывод программы RG32

Результаты статистической обработки

Вариант	D	S	K	(1) c0	(ln N) c1	(N) c2	(N ln N) c3	(N^2) c4	(N^3) c5	(N^4) c6		
1	5,16E+11	718413	1	2,82E+06	0	0	0	0	0	0	0	238
2	2,24E+11	473460	2	-1,05E+07	2,45E+06	0	0	0	0	0	461	0
3	2,17E+11	465731	2	220186	0	11042,6	0	0	0	0	357	0
4	2,17E+11	466121	3	4,18E+06	-901368	15025	0	0	0	0	921	0
5	2,17E+11	466120	3	1,14E+06	0	-15360,6	4100,26	0	0	0	863	0
6	2,18E+11	467114	4	2,24E+06	-323393	-4511,04	2637,27	0	0	0	1098	0
7	2,17E+11	466134	3	684263	0	6892,96	0	8,86088	0	0	859	0
8	2,18E+11	467170	4	-32676,2	0	41359,8	-6336,75	22,4017	0	0	888	0
9	2,19E+11	468166	5	33345,6	3937,99	37415,2	-5618,06	20,9045	0	0	1072	0
10	2,18E+11	467293	4	3596,73	0	15814,5	0	-28,8017	0,0514408	0	816	0
11	2,19E+11	468407	5	-79084,8	-54512,7	20187,3	0	-45,8265	0,0741108	0	1117	0
12	2,19E+11	468095	5	77752,5	0	69939,3	-14008,6	95,6653	-0,079478	0	1686	0
13	2,20E+11	469403	6	-424,754	3037,11	1978	3475,59	-57,8968	0,0813345	0	1024	0
14	2,19E+11	468107	5	-28130,9	0	21869,1	0	-103,794	0,360623	-0,000416	1534	0
15	2,20E+11	469120	6	-15668,4	0	6350,53	4280,13	-159,13	0,478777	-0,000526	1718	0
16	2,21E+11	470126	7	-9760,62	-29663,8	36721,4	-3602,87	-67,2665	0,288237	-0,000351	1849	0

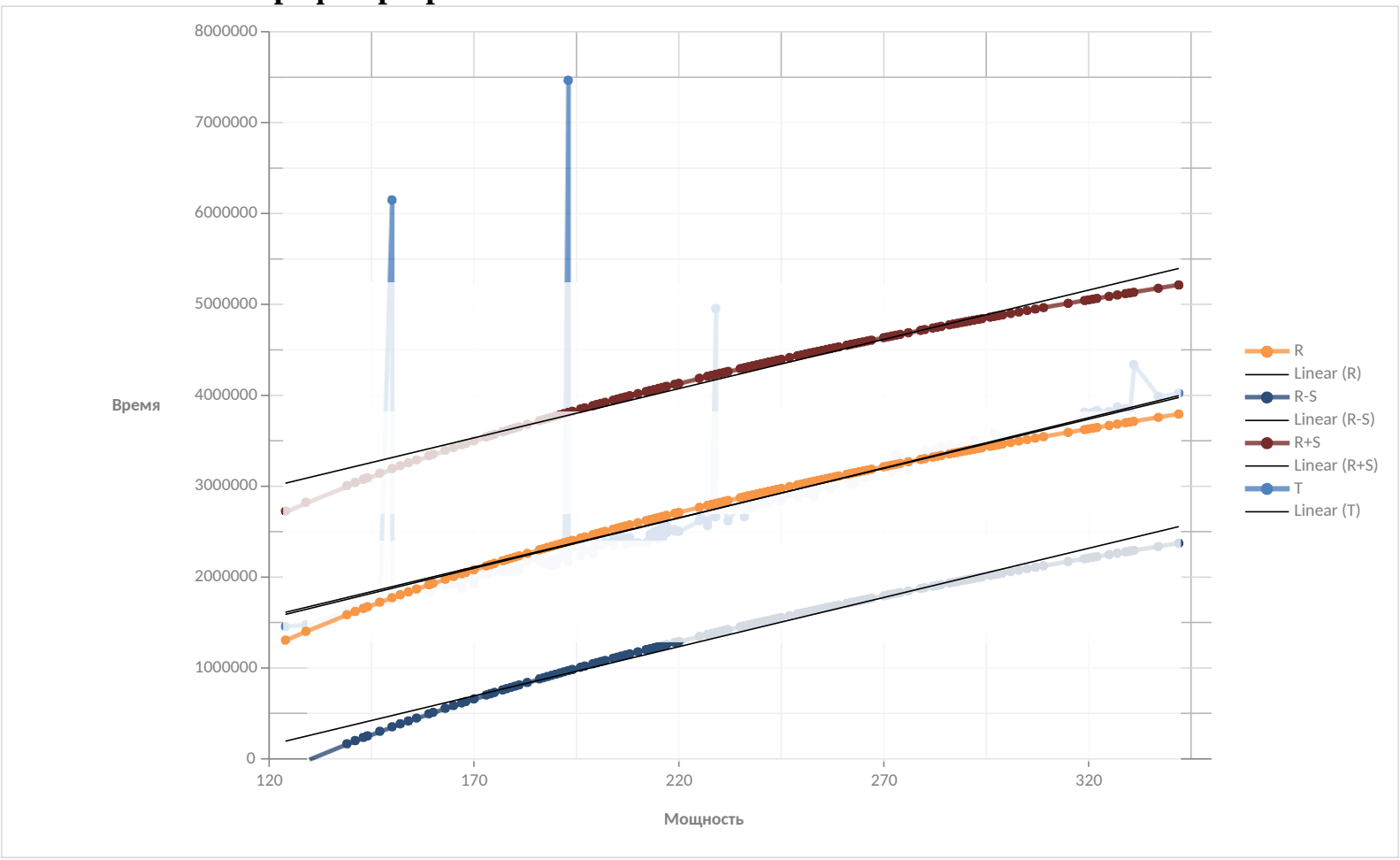
Вспомогательная таблица. Определение подходящего уравнения регрессии

Отношение дисперсий

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	1	0,43	0,42	0,42	0,42	0,42	0,42
2	2,30	1,00	0,97	0,97	0,97	0,97	0,97
3	2,38	1,03	1,00	1,00	1,00	1,01	1,00
4	2,38	1,03	1,00	1,00	1,00	1,00	1,00
5	2,38	1,03	1,00	1,00	1,00	1,00	1,00
6	2,37	1,03	0,99	1,00	1,00	1,00	1,00
7	2,38	1,03	1,00	1,00	1,00	1,00	1,00
8	2,36	1,03	0,99	1,00	1,00	1,00	1,00
9	2,35	1,02	0,99	0,99	0,99	1,00	0,99
10	2,36	1,03	0,99	0,99	0,99	1,00	1,00
11	2,35	1,02	0,99	0,99	0,99	0,99	0,99
12	2,36	1,02	0,99	0,99	0,99	1,00	0,99
13	2,34	1,02	0,98	0,99	0,99	0,99	0,99
14	2,36	1,02	0,99	0,99	0,99	1,00	0,99
15	2,35	1,02	0,99	0,99	0,99	0,99	0,99
16	2,34	1,01	0,98	0,98	0,98	0,99	0,98

(приведены первые 6 колонок таблицы отношений дисперсий)

2.2. График регрессии



2.3. Оценка временной сложности по результатам эксперимента

На основании отношений дисперсий нами была выбрана логарифмическая регрессия.

В нашей выборке 238 опытов, программа RG32.exe строит уравнения до 6 степени, следовательно, степень свободы выборки была больше чем для 200 опытов (1.26 при 5% погрешности).

Отношения первой и остальных дисперсий больше (~2.3), следовательно, сложность не константная.

Отношения же второй и остальных дисперсий 1.01-1.03 входят в нужный интервал, из чего можно сделать вывод о логарифмической временной сложности алгоритма.

ЗАКЛЮЧЕНИЕ

Нами был проведен эксперимент по оценке временной сложности выполнения цепочки операций над множествами, с использованием стандартной библиотеки шаблонов.

При теоретической оценке временной сложности как O(n), эксперимент показал логарифмическую временную сложность алгоритма — O(logn).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Алгоритмы и структуры данных : методические указания к лабораторным работам, практическим занятиям и курсовому проектированию. Ч. 2 Вып. 1702 / сост. П.Г.Колинько. — СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2017. - 52 с.:ил.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ

main.cpp

```
#include <iostream>
#include <fstream>
#include <ctime>
#include "stud_set.h"
using namespace std;

__inline__ uint64_t rdtsc() {
  uint64_t a, d;
  asm("rdtsc\n" : "=a" (a), "=d" (d));
  return (d<<32) | a;
}

int main(int argc, char** argv)
{
	srand(time(NULL));
	std::ofstream out;

	out.open("in.txt");

	if (!out.is_open()) return 1;

	int num_of_exper = (rand() % 150) + 100;
	unsigned long t1, t2;

	out << num_of_exper << endl;

	stud_set arrayofsets[5];
	for (int tempCount = 0; tempCount < num_of_exper; tempCount++) {
		try {
			int size = 0 , temp , sets = 4;

			for (int i = 0; i < 4; i++) {
				temp = (rand() % 250);
				size += temp;
				arrayofsets[i].gen(temp, rand() % 1000);
			}

			t1 = rdtsc();

			size += stud_union(arrayofsets[0], arrayofsets[1],
arrayofsets[2]); ++sets;
			size += stud_merge(arrayofsets[2], arrayofsets[1],
arrayofsets[3]); ++sets;
			size += stud_concat(arrayofsets[2], arrayofsets[3],
arrayofsets[0]); ++sets;
```

```
                  size += stud_concat(arrayofsets[0], arrayofsets[1],
arrayofsets[2]); ++sets;
                  size += stud_change(arrayofsets[4], arrayofsets[2],
arrayofsets[3], 15); ++sets;
                  size += stud_xor(arrayofsets[1], arrayofsets[3],
arrayofsets[1]); ++sets;
                  size += stud_diff(arrayofsets[4], arrayofsets[0],
arrayofsets[2]); ++sets;
                  size += stud_and(arrayofsets[4], arrayofsets[2],
arrayofsets[3]); ++sets;

                  t2 = rdtsc();
                  size /= sets;
                  out << size << ' ' << (t2 - t1) << std::endl;
            }
            catch (...) {
                  std::cout << "Unknown error occur during one of the
experiments, continue." << std::endl;
            }
      }

      out.close();

      return 0;
}
```

stud_set.h

```
#include <iostream>
#include <vector>
#include <set>
#include <iterator>
#include <algorithm>
using namespace std;

class stud_item {
public:
      int key;
      mutable int number;

      stud_item() { key = 0; number = 0; }
      stud_item(int key, int number) : key(key), number(number) { }
      stud_item(const stud_item& other) { key = other.key; number =
other.number; }
      stud_item& operator=(const stud_item& other) {
            key = other.key;
            number = other.number;
            return *this;
      }
```

```
	~stud_item() { }
	friend ostream& operator<<(ostream& os, const stud_item& item);
	friend bool operator<(const stud_item& item1, const stud_item& item2);
};

class stud_set {
private:
	char name;
	multiset<stud_item> values;
	vector<multiset<stud_item>::iterator> sequence;

public:
	stud_set () : name('?') { }
	stud_set (char name) : name(name) { }
	stud_set (char name, int n, int mod) : stud_set(name) {
		gen(n, mod);
	}

	void gen(int n, int mod) {
		values.clear();
		sequence.clear();
		for (int i = 0; i < n; ++i) {
			sequence.push_back(values.insert(stud_item(rand()%(1+mod),
sequence.size())));
		}
	}

	void setOut() {
		cout << "values " << name << " = { ";
		for (auto it = values.cbegin(); it != values.cend(); ++it)
			cout << *it << ' ';
		cout << '}' << endl;
	}

	void seqOut() {
		cout << "sequence " << name << " = { ";
		for (auto it = sequence.cbegin(); it != sequence.cend(); ++it)
			cout << *(*it) << ' ';
		cout << '}' << endl;
	}

	friend int stud_and(stud_set& A, stud_set& B, stud_set& C);
	friend int stud_diff(stud_set& A, stud_set& B, stud_set& C);
	friend int stud_union(stud_set& A, stud_set& B, stud_set& C);
	friend int stud_xor(stud_set& A, stud_set& B, stud_set& C);
	friend int stud_concat(stud_set& A, stud_set& B, stud_set& C);
	friend int stud_merge(stud_set& A, stud_set& B, stud_set& C);
	friend int stud_change(stud_set& A, stud_set& B, stud_set& C, int p);
```

```
};
```

stud_set.cpp

```
#include "stud_set.h"
bool operator<(const stud_item& item1, const stud_item& item2)
{
	return (item1.key < item2.key);
}

bool operator<=(const stud_item& item1, const stud_item& item2)
{
	return (item1.key <= item2.key);
}

ostream& operator<<(ostream& os, const stud_item& item)
{
	os << item.key << "(" << item.number << ")" << "";
	return os;
}
int stud_and(stud_set& A, stud_set& B, stud_set& C)
{
	C.values.clear();
	C.sequence.clear();

	set_intersection(A.values.cbegin(), A.values.cend(), B.values.cbegin(),
B.values.cend(), inserter(C.values, C.values.begin()));

	for (auto it = C.values.begin(); it != C.values.end(); ++it) {
		(*it).number = C.sequence.size();
		C.sequence.push_back(it);
	}

	return C.values.size();
}

int stud_diff(stud_set& A, stud_set& B, stud_set& C)
{
	C.values.clear();
	C.sequence.clear();

	set_difference(A.values.cbegin(), A.values.cend(), B.values.cbegin(),
B.values.cend(), inserter(C.values, C.values.begin()));

	for (auto it = C.values.begin(); it != C.values.end(); ++it) {
		(*it).number = C.sequence.size();
		C.sequence.push_back(it);
	}
```

```
	return C.values.size();
}

int stud_union(stud_set& A, stud_set& B, stud_set& C)
{
	C.values.clear();
	C.sequence.clear();

	set_union(A.values.cbegin(), A.values.cend(), B.values.cbegin(),
B.values.cend(), inserter(C.values, C.values.begin()));

	for (auto it = C.values.begin(); it != C.values.end(); ++it) {
		(*it).number = C.sequence.size();
		C.sequence.push_back(it);
	}

	return C.values.size();
}

int stud_xor(stud_set& A, stud_set& B, stud_set& C)
{
	C.values.clear();
	C.sequence.clear();

	set_symmetric_difference(A.values.cbegin(), A.values.cend(),
B.values.cbegin(), B.values.cend(), inserter(C.values, C.values.begin()));

	for (auto it = C.values.begin(); it != C.values.end(); ++it) {
		(*it).number = C.sequence.size();
		C.sequence.push_back(it);
	}

	return C.values.size();
}

int stud_concat(stud_set& A, stud_set& B, stud_set& C)
{
	std::vector<stud_item> vec1, vec2, result;
	vector<stud_item>::iterator x1, x2;

	C.values.clear();
	C.sequence.clear();

	// Формируем упорядоченный по возрастанию ключей массив из элементов
множества A
	for (auto it = A.values.begin(); it != A.values.end(); it++) {
		vec1.push_back(*it);
	}
```

```
      int p = vec1.size();

      // Формируем упорядоченный по возрастанию ключей массив из элементов
множества B
      for (auto it = B.values.begin(); it != B.values.end(); it++) {
            vec2.push_back(stud_item((*it).key, (*it).number + p));
      }

      x1 = vec1.begin();
  x2 = vec2.begin();
      // Конкатенация элементов последовательности vec1 и последовательности
vec2
  while (x1 != vec1.end() || x2 != vec2.end()) {
    if (x1 != vec1.end() && x2 != vec2.end()) {
      if (*x1 <= *x2) {
                        result.push_back(*x1);
                        ++x1;
      } else {
                        result.push_back(*x2);
                        ++x2;
      }
    } else if(x1 != vec1.end()) { // случай когда второй вектор закончился
      result.push_back(*x1);
      ++x1;
    } else { // случай когда первый вектор закончился
      result.push_back(*x2);
      ++x2;
    }
  }

      C.values.insert(result.begin(), result.end());
      C.sequence.resize(C.values.size());

      for (auto it = C.values.begin(); it != C.values.end(); it++) {
            C.sequence[(*it).number] = it;
      }

      return C.values.size();
}

int stud_merge(stud_set& A, stud_set& B, stud_set& C)
{
      stud_concat(A, B, C);

      C.sequence.clear();

      for(auto it = C.values.cbegin(); it != C.values.cend(); ++it) {
```

```
            (*it).number = C.sequence.size();
            C.sequence.push_back(it);
      }

      return C.values.size();
}

int stud_change(stud_set& A, stud_set& B, stud_set& C, int p)
{
      std::vector<stud_item> vec1,vec2,result_tmp,result;
      vector<stud_item>::iterator x1, x2;

      C.values.clear();
      C.sequence.clear();

      p = (A.values.size() < p) ? A.values.size() : p;

      // Формируем упорядоченный по возрастанию ключей массив из элементов
множества A
      for (auto it = A.values.begin(); it != A.values.end(); it++) {
            vec1.push_back(*it);
      }
      // Формируем упорядоченный по возрастанию ключей массив из элементов
множества B
      for (auto it = B.values.begin(); it != B.values.end(); it++) {
            vec2.push_back(stud_item((*it).key, (*it).number + p));
      }

  x1 = vec1.begin();
      // Создаём вектор элементов последовательности от 0 до p-1
  while (x1 != vec1.end()){
    if ((*x1).number < p){
                  result_tmp.push_back(*x1);
    }
    ++x1;
  }

      x1 = result_tmp.begin();
  x2 = vec2.begin();
    // Конкатенация первых p элементов последовательности vec1 и всей
последовательности vec2
  while (x1 != result_tmp.end() || x2 != vec2.end()) {
    if (x1 != result_tmp.end() && x2 != vec2.end()) {
      if (*x1 <= *x2) {
                        result.push_back(*x1);
                        ++x1;
      } else {
                        result.push_back(*x2);
```

```
                        ++x2;
      }
    } else if(x1 != result_tmp.end()) { // случай когда второй вектор закончился
      result.push_back(*x1);
      ++x1;
    } else { // случай когда первый вектор закончился
      result.push_back(*x2);
      ++x2;
    }
  }

      if (vec1.size() > vec2.size() + p) {
            vec2.clear();
            result_tmp.clear();

            x1 = vec1.begin();
    // Создаём вектор элементов последовательности vec1 от p+vec2.size() до
конца
    while (x1 != vec1.end()) {
      if ((*x1).number >= result.size()) {
                        vec2.push_back(*x1);
      }
      ++x1;
            }

            x1 = result.begin();
            x2 = vec2.begin();
            // Конкатенация первых p элементов последовательности vec1 и всей
последовательности vec2
            while (x1 != result.end() || x2 != vec2.end()) {
                  if (x1 != result.end() && x2 != vec2.end()) {
                        if (*x1 <= *x2) {
                              result_tmp.push_back(*x1);
                              ++x1;
                        } else {
                              result_tmp.push_back(*x2);
                              ++x2;
                        }
                  } else if(x1 != result.end()) { // случай когда второй вектор
закончился
                        result_tmp.push_back(*x1);
                        ++x1;
                  } else { // случай когда первый вектор закончился
                        result_tmp.push_back(*x2);
                        ++x2;
                  }
            }
```

```
            C.values.insert(result_tmp.begin(), result_tmp.end());
            C.sequence.resize(C.values.size());

            for (auto it = C.values.begin(); it != C.values.end(); it++) {
                  C.sequence[(*it).number] = it;
            }

      } else {
            C.values.insert(result.begin(), result.end());
            C.sequence.resize(C.values.size());

            for (auto it = C.values.begin(); it != C.values.end(); it++) {
                  C.sequence[(*it).number] = it;
            }
      }

      return C.values.size();
}
```