

*Государственное образовательное учреждение высшего профессионального  
образования*

**«Московский государственный технический университет  
имени Н. Э. Баумана»  
(МГТУ им. Н.Э. Баумана)**

---

ФАКУЛЬТЕТ

«Информатика и системы управления»

КАФЕДРА

«Программное обеспечение ЭВМ и информационные технологии»

**РАСЧЁТНО - ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**  
к лабораторной работе на тему:

Оценка трудоемкости сортировок

Студент

\_\_\_\_\_ Киселев А.М.  
(Подпись, дата)

Преподаватель

\_\_\_\_\_ Волкова Л.Л.  
(Подпись, дата)

Москва 2018

## Содержание

1	Аналитический раздел . . . . .	3
1.1	Сортировка пузырьком . . . . .	3
1.2	Сортировка простыми вставками . . . . .	3
1.3	Сортировка слиянием . . . . .	4
1.4	Расчет трудоемкости алгоритма . . . . .	5
1.4.1	Модель вычислений . . . . .	5
2	Конструкторский раздел . . . . .	6
2.1	Разработка алгоритмов . . . . .	6
2.1.1	Алгоритм сортировки пузырька . . . . .	6
2.1.2	Алгоритм сортировки слиянием . . . . .	6
2.1.3	Алгоритм сортировки простыми вставками . . . . .	7
3	Технологический раздел . . . . .	8
3.1	Требования к программному ПО . . . . .	8
3.2	Средства реализации . . . . .	8
3.3	Листинг . . . . .	8
4	Экспериментальный раздел . . . . .	10
	Заключение . . . . .	13

## 1 Аналитический раздел

Задача упорядочивания элементов имеет широкое применения в совершенно разных сферах: упорядочивание слов по алфавиту, ряда чисел или структур данных. Многие алгоритмы нуждаются в упорядоченных рядах чисел для продолжения вычислений.

Сортировка ряда чисел сводится к получению упорядоченного ряда. Пусть задан ряд  $\{5, 3, 2, 6, 1\}$ , тогда результат сортировки будет  $\{1, 2, 3, 5, 6\}$ . Существует множество методов сортировки, которые позволяют получить упорядоченный массив чисел. Одними из таких алгоритмов являются: метод пузырька, метод слияния и метод простыми вставками.

### 1.1 Сортировка пузырьком

Пусть дан массив  $\{3, 2, 6, 7, 9, 4\}$ . Дана задача отсортировать его по убыванию. При решении этой задачи, алгоритм пройдет несколько раз по этому массиву, каждый раз проходя все меньше количество элементов. Первый проход представлен в таблице 1.1.

Таблица 1.1 — Процесс обработки массива на первой итерации алгоритма пузырька

Нет обмена	$2 \leftrightarrow 6$	$2 \leftrightarrow 7$	$2 \leftrightarrow 9$	$2 \leftrightarrow 4$
4	4	4	4	<b>2</b>
9	9	9	<b>2</b>	<b>4</b>
7	7	<b>2</b>	<b>9</b>	9
6	<b>2</b>	<b>7</b>	7	7
<b>2</b>	<b>6</b>	6	6	6
<b>3</b>	3	3	3	3

Далее последний элемент массива фиксируется и действия сортировки продолжают, но уже не до последнего элемента массива, а до предпоследнего. Количество обрабатываемых элементов на каждой итерации изменяется как  $N = N - 1$ , где  $N$  изначально является длиной массива. Полный результат работы алгоритма на каждой итерации представлен в таблице.

### 1.2 Сортировка простыми вставками

В сортировке вставками последовательно обрабатываются отрезки массива, начиная с первого элемента и затем последовательно расширяя подмассив, вставляя на своё место очередной неотсортированный элемент.

Для вставки используется буферная область памяти, в которой хранится элемент, ещё не вставленный на своё место (так называемый ключевой элемент). В подмассиве, начиная с конца отрезка, перебираются элементы, которые сравниваются с ключевым. Если эти элементы больше ключевого, то они сдвигаются на одну позицию вправо, освобождая место для последующих элементов. Если в результате этого перебора попадает элемент, меньший или равный ключевому, то значит в текущую свободную ячейку можно вставить ключевой элемент.

### 1.3 Сортировка слиянием

Это рекурсивный алгоритм, который постоянно разбивает список пополам. Если список пуст или состоит из одного элемента, то он отсортирован по определению (базовый случай). Если в списке больше, чем один элемент, мы разбиваем его и рекурсивно вызываем сортировку слиянием для каждой из половин. После того, как обе они уже отсортированы, выполняется основная операция, называемая слиянием. Слияние - это процесс комбинирования двух меньших отсортированных списков в один новый, но тоже отсортированный.

Таким образом задача разбивается на подзадачи, которые в свою очередь рекурсивно доходят до базовых случаев. Предположим, что нужно отсортировать массив  $A$ . Подзадачей будет сортировка подраздела этого массива начиная с индекса  $p$  и заканчивая индексом  $r$ , обозначенного как  $A[p..r]$ .

Если  $q$  – точка на пути между  $p$  и  $r$ , то подмассив  $A[p..r]$  можно разделить на  $A[p..q]$  и  $A[q+1..r]$ . Далее оба массива сортируются и если базовый случай не был достигнут, то повторяем те же действия для каждого из подмассивов.

Таблица 1.2 — Результат работы алгоритма пузырька на каждой итерации

4	<b>2</b>	<b>2</b>	<b>2</b>	<b>2</b>	<b>2</b>
9	4	<b>3</b>	<b>3</b>	<b>3</b>	<b>3</b>
7	9	4	4	4	4
6	7	9	9	<b>6</b>	<b>6</b>
2	6	7	7	9	<b>7</b>
3	3	6	6	7	9
<b>i=0</b>	<b>i=1</b>	<b>i=2</b>	<b>i=3</b>	<b>i=4</b>	<b>i=5</b>

## 1.4 Расчет трудоемкости алгоритма

Расчет трудоемкости алгоритма опирается на модель вычисления, в которой нужно определить свод правил : оценка операций, циклов, стоимость перехода по условиям.

### 1.4.1 Модель вычислений

Стоимость каждой операции из следующего множества будем считать 1:  $\{+, -, *, /, \%, <, <=, !=, >=, >, =, :=, \&, ||, []\}$

Стоимость условного перехода if-else – 0 и будет учитывать лишь стоимость выражения условия.

Стоимость цикла будет рассчитываться следующим образом:

Пусть цикл задан так:

$$for(i = 0, i < N, i++)\{body\} \quad (1.1)$$

, где N – количество итераций

Тогда  $i=0, i < N$  – операции, выполняемые перед началом выполнения тела цикла и их общая стоимость равна 2,  $i++, i < N$  – операции, которые выполняются каждую итерацию(их стоимость в общем тоже равна 2)

$$f_{for} = 2 + N * (2 + f_{body}) \quad (1.2)$$

## 2 Конструкторский раздел

### 2.1 Разработка алгоритмов

#### 2.1.1 Алгоритм сортировки пузырька

В листенге 2.1 представлен алгоритм сортировки пузырьком.

Листинг 2.1 — Pseudocode of bubble sort

```
1 цикл для j от min до max
2   цикл для i от min до max-1
3     если A[i] больше чем A[i+1] то:
4       обменять местами A[i] и A[i+1]
```

#### 2.1.2 Алгоритм сортировки слиянием

В листинге 2.2 представлен алгоритм сортировки слиянием.

Листинг 2.2 — Pseudocode of merge sort

```
1 функция mergesort( массив A )
2   Если n=1 то:
3     вернуть A
4
5   l1 = A[0] .. A[n/2]
6   l2 = A[n/2+1] .. A[n]
7
8   l1 = mergesort( l1 )
9   l2 = mergesort( l2 )
10
11   вернуть merge( l1 , l2 )
12 конец функции
13
14 функция merge( массив A, массив B )
15   пока в A и в B есть элементы:
16     если A[0] > B[0]:
17       добавить B[0] в конец C
18       удалить B[0] из B
19     иначе
20       добавить A[0] в конец C
21       удалить A[0] из A
22   пока в A есть элементы:
23     добавить A[0] в конец C
24     удалить A[0] из A
25   пока в B есть элементы:
26     добавить B[0] в конец C
```

```
27         удалить B[0] из B
28     вернуть C
29 конец функции
```

### 2.1.3 Алгоритм сортировки простыми вставками

Листинг алгоритма сортировки простыми вставками представлен в 2.3

#### Листинг 2.3 — Pseudocode of insertions algorithm

```
1  цикл для i от 0 до n - 1
2      j = i - 1
3      пока j >= 0 и a[j] > a[j + 1]
4          поменять местами a[j] и a[j + 1]
5          j—
```

## 3 Технологический раздел

Данный раздел содержит информацию о реализации ПО и листингах программ.

### 3.1 Требования к программному ПО

Программы реализованы в двух вариациях. Первая – каждая программа реализована поотдельности и обладает своим собственным функционалом – возможность определять размеры массива и получать вывод в поток вывода. Вторая – программа, содержащая в себе функции с реализованными алгоритмами, в которой производится тестирование времени алгоритмов и вывод отправляется в поток вывода информации.

### 3.2 Средства реализации

Данные реализации разрабатывались на языке C, использовался компилятор gcc-8.2.1. Замер времени производится с помощью библиотеки *time.h*. Данная библиотека позволяет производить замер тиков, откуда можно получать время в секундах. Программы компилировались с выключенной оптимизацией с флагом -O0

### 3.3 Листинг

Алгоритм сортировки пузырьком представлен в листинге 3.1

```
1 void bubble(int* a, int size)
2 {
3     for(int i = 0; i < size; i++)
4         for(int j = 0; j < size-j-i; j++)
5             if(a[j] > a[j+1])
6                 swap(&a[j], &a[j+1]);
7 }
```

Листинг 3.1 — bubble sort

Алгоритм сортировки слиянием представлен в листинге 3.2

```
1 void merge(int* a, int* l, int leftCount, int* r, int rightCount)
2 {
3     int i = 0,
4         j = 0,
5         k = 0;
6
7     while(i < leftCount && j < rightCount) {
8         if(l[i] < r[j]) a[k++] = l[i++];
```



```

9         else a[k++] = r[j++];
10    }
11    while(i < leftCount) a[k++] = l[i++];
12    while(j < rightCount) a[k++] = r[j++];
13 }
14
15 void mergeSort(int* a, int n)
16 {
17     int mid, i, *l, *r;
18     if(n < 2) return;
19
20     mid = n/2;
21
22     l = (int*) malloc(mid*sizeof(int));
23     r = (int*) malloc((n - mid)*sizeof(int));
24
25     for(i = 0; i<mid; i++) l[i] = a[i]; // creating left subarray
26     for(i = mid; i<n; i++) r[i-mid] = a[i]; // creating right subarray
27
28     mergeSort(l, mid);
29     mergeSort(r, n-mid);
30     merge(a, l, mid, r, n-mid);
31     free(l);
32     free(r);
33 }

```

Листинг 3.2 — merge sort

Листинг алгоритма сортировки простыми вставками представлен в 3.3

```

1 void insert(int* a, int size)
2 {
3     int j, tmp;
4
5     for(int i = 1; i < size; i++) {
6         tmp = a[i];
7         for (j = i - 1; j >= 0 && a[j] > tmp; j--)
8             a[j + 1] = a[j];
9         a[j + 1] = tmp;
10    }
11 }

```

Листинг 3.3 — Insertions sort

## 4 Экспериментальный раздел

Результаты работы алгоритмов представлены в таблицах 4.1 и 4.2 (для четных и нечетных размерностей матриц). Полученные данные представлены в виде диаграмм изображенных на рисунках 4.1 и 4.2

1 – Классический алгоритм Винограда (на графиках представлен черным цветом)

2 – Оптимизированный алгоритм Винограда (на графиках представлен красным цветом)

3 – Последовательное перемножение матриц (на графиках представлен синим цветом)

Таблица 4.1 — Таблица для сравнения результатов работы алгоритма для четной размерности в микросекундах.

Размер	1	2	3
100 X 100	20.077	20.859	25.865
200 X 200	258.613	154.072	201.215
300 X 300	536.736	516.737	687.302
400 X 400	1277.363	1229.492	1657.234
500 X 500	2508.538	2408.629	3294.885
600 X 600	4878.377	4685.910	6107.689
700 X 700	7778.614	7495.057	9871.546
800 X 800	11597.817	11170.562	14822.127
900 X 900	16712.889	16070.812	21545.400
1000 X 1000	23102.137	22225.149	29986.602

По полученным данным можно сделать вывод, что стандартный алгоритм перемножения матриц работает медленнее, чем классический алгоритм Винограда и улучшенный алгоритм Винограда. Улучшенный алгоритм работает быстрее по времени, чем классический. Если обратиться к рассчитанным трудоемкостям алгоритмов, можно заметить, что стандартный алгоритм менее трудоемкий, чем два других, но время, затрачиваемое на расчет матрицы стандартным методом все равно очень большое. Так как это зависит от модели вычислений, где умножение и обращения к индексам и сложение были взяты за 1, можно сказать, что следует учитывать некоторые нюансы при выборе весов операций, чтобы вычисленная трудоемкость была более точной.

Таблица 4.2 — Таблица для сравнения результатов работы алгоритма для нечетной размерности в микросекундах.

Размер	1	2	3
101 X 101	20.809	21.282	26.734
201 X 201	161.322	156.745	204.425
301 X 301	541.239	525.954	692.841
401 X 401	1284.754	1236.969	1685.261
501 X 501	2528.653	2422.614	3316.100
601 X 601	4918.674	4727.957	6148.991
701 X 701	7833.712	7528.168	9911.066
801 X 801	11725.979	11282.570	15007.543
901 X 901	16744.922	16120.746	21658.227
1001 X 1001	23189.963	22264.232	30098.645

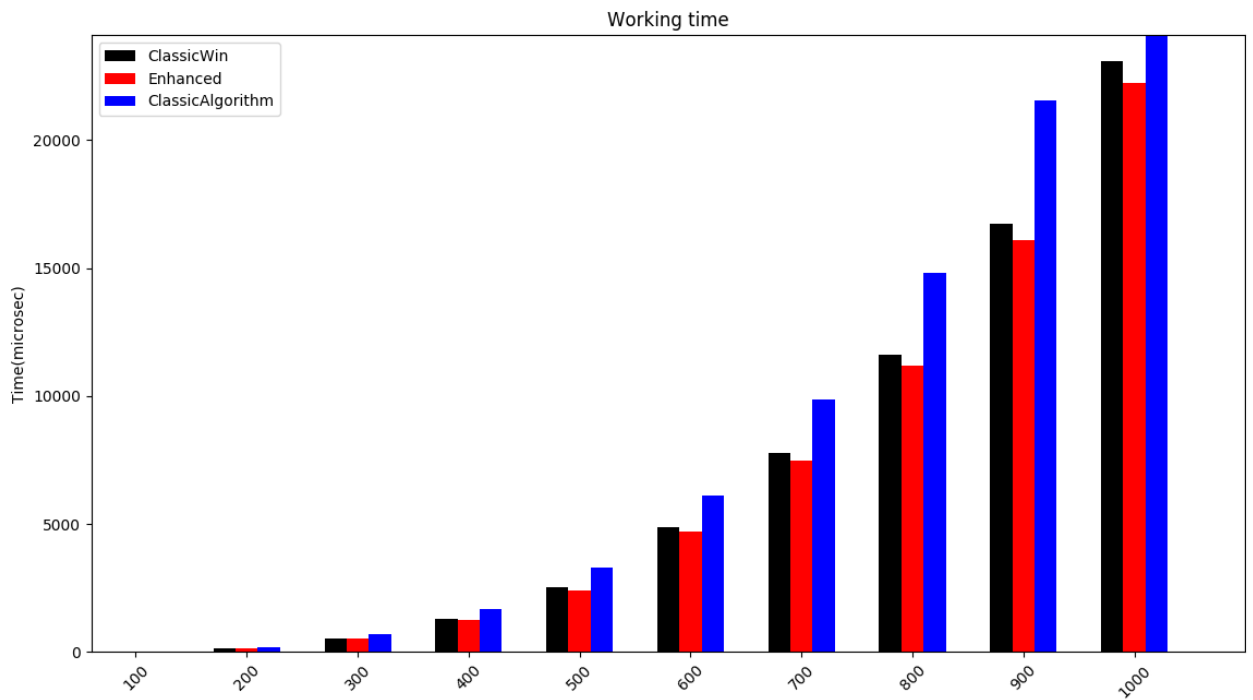


Рисунок 4.1 — График для квадратных матриц четной размерности

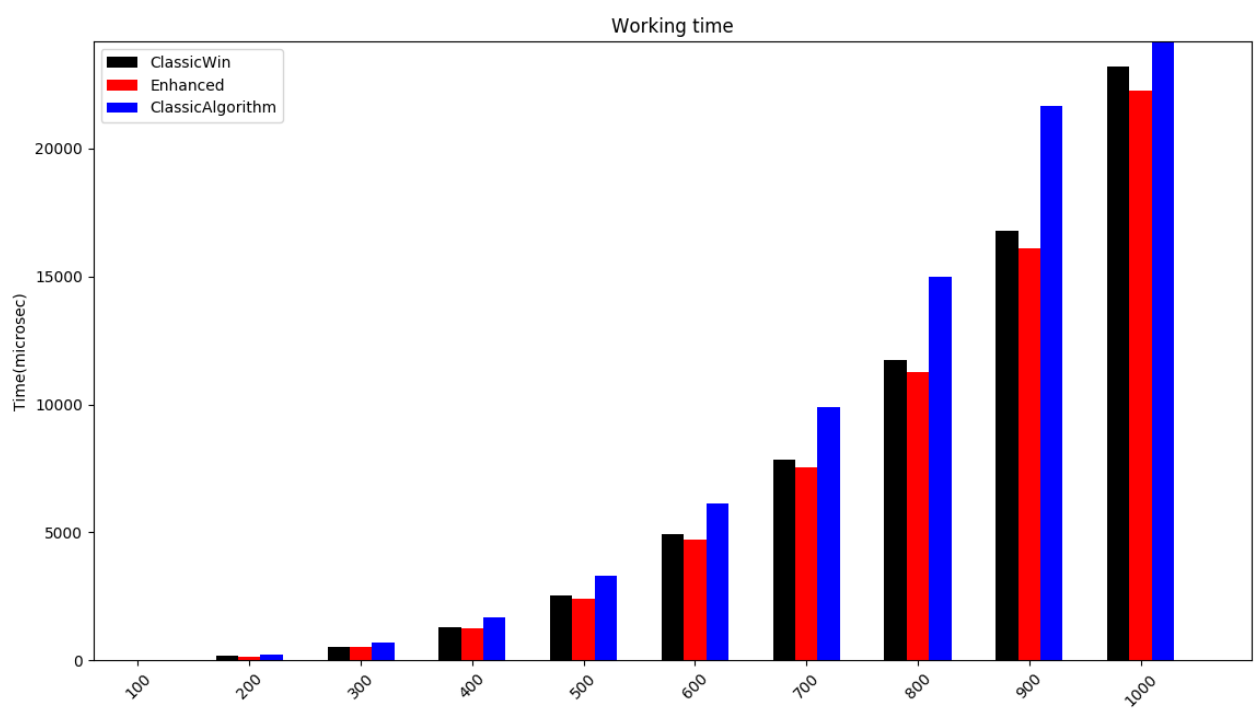


Рисунок 4.2 — График для квадратных матриц нечетных размерностей

## Заключение

В результате выполнения лабораторной работы были получены следующие основные навыки:

- а) изучены теоритические понятия в алгоритмах перемножения матриц;
- б) рассмотрены различные способы оптимизации алгоритмов;
- в) проведено сравнение четырех реализаций заданного алгоритма Винограда, выявлены их слабые места;