*­Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования*

|  |  |
| --- | --- |
| Gerb-BMSTU_01 | ***«Московский государственный технический университет  имени Н.Э. Баумана (национальный исследовательский институт)»***  ***(МГТУ им. Н.Э. Баумана)*** |

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

**Отчёт**

**по лабораторной работе 3**

**Дисциплина: Анализ Алгоритмов**

**Тема лабораторной работы работы: Трудеомкость алгоритмов умножения матриц**

Студентки гр. ИУ7-51б **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сушина А.Д.**

Преподаватель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Волкова Л.Л.**

­

Москва, 2019г

Оглавление

[Введение 3](#__RefHeading___Toc2490_614358252)

[1. Аналитическая часть 4](#__RefHeading___Toc2492_614358252)

[1.1. Описание алгоритмов 4](#__RefHeading___Toc2494_614358252)

[2. Конструкторская часть 5](#__RefHeading___Toc2496_614358252)

[2.1. Разработка алгоритмов 5](#__RefHeading___Toc2498_614358252)

[2.2. Анализ трудоемкости алгоритмов 7](#__RefHeading___Toc2502_614358252)

[Модель вычислений: 7](#__RefHeading___Toc2504_614358252)

[Сортировка вставками 8](#__RefHeading___Toc2506_614358252)

[Сортировка Шейкер 8](#__RefHeading___Toc2508_614358252)

[Сортировка слиянием 8](#__RefHeading___Toc2510_614358252)

[3. Технологическая часть 10](#__RefHeading___Toc2512_614358252)

[3.1. Требования к программному обеспечению 10](#__RefHeading___Toc2514_614358252)

[3.2. Средства реализации 10](#__RefHeading___Toc2516_614358252)

[3.3. Листинг кода 10](#__RefHeading___Toc2518_614358252)

[Сортировка вставками 10](#__RefHeading___Toc2520_614358252)

[Сортировка шейкер 10](#__RefHeading___Toc2522_614358252)

[Сортировка слиянием 11](#__RefHeading___Toc2524_614358252)

[3.4. Описание тестирования 12](#__RefHeading___Toc2526_614358252)

[4. Экспериментальная часть 13](#__RefHeading___Toc2528_614358252)

[4.1. Примеры работы 13](#__RefHeading___Toc2530_614358252)

[4.2. Результаты тестирования 14](#__RefHeading___Toc2532_614358252)

[4.3. Постановка эксперимента по замеру времени 15](#__RefHeading___Toc2534_614358252)

[Заключение 17](#__RefHeading___Toc2536_614358252)

[Литература 18](#__RefHeading___Toc2538_614358252)

# Введение

*Сортировкой* (англ. *sorting*) называется процесс упорядочивания множества объектов по какому-либо признаку.

**Алгоритм сортировки** — это алгоритм для упорядочивания элементов в списке.

**Целью** данной лабораторной работы является изучение алгоритмов сортировки и их трудоемкости.

**Задачи** лабораторной работы:

* реализовать 3 выбранных алгоритма сортировки;
* рассчитать трудоемкость каждого из алгоритма сортировки;
* провести временное тестирование алгоритмов сортировки.

# 1. Аналитическая часть

Существует огромное количество разнообразных алгоритмов сортировки. Они все отличаются методами, трудоемкостью, скоростью работы.

В данной лабораторной работе были выбраны следующие алгоритмы сортировки:

* сортировка вставками;
* сортировка "шейкер";
* сортировка слиянием.

## 1.1. Описание алгоритмов

**Сортировка вставками**

Суть алгоритма заключается в том что, на каждом шаге берется один из элементов массива, находим позицию для вставки и вставляем. При этом левая часть массива всегда остается отсортированной и для каждого элемента ищется подходящее место в отсортированном массиве. Массив из 1-го элемента считается отсортированным.

**Сортировка "шейкер"**

Шейкер-сортировка является разновидностью пузырьковой сортировки. На каждой итерации самый "тяжелый" элемент опускается вниз, а самый "легкий" поднимается вверх. За счет этого можно на каждой итерации уменьшать правую и левую границу сортируемой части массива. Таким образом, уже отсортированные части массива исключаются из рассмотрения.

**Сортировка слиянием**

Алгоритм использует принцип «разделяй и властвуй»: задача разбивается на подзадачи меньшего размера, которые решаются по отдельности, после чего их решения комбинируются для получения решения исходной задачи. Конкретно процедуру сортировки слиянием можно описать следующим образом:

* Если в рассматриваемом массиве один элемент, то он уже отсортирован — алгоритм завершает работу.
* Иначе массив разбивается на две части, которые сортируются рекурсивно.
* После сортировки двух частей массива к ним применяется процедура слияния, которая по двум отсортированным частям получает исходный отсортирован.

# 2. Конструкторская часть

## 2.1. Разработка алгоритмов

В данном разделе будут представлены схемы алгоритмов сортировки вставками(рис.1), шейкер сортировки(рис.2), сортировки слиянием(рис.3).

|  |
| --- |
|  |

Рис 1. Алгоритм сортировки вставками

|  |
| --- |
|  |

Рис 2. Алгоритм сортировки шейкер

|  |
| --- |
| Рис 3. Алгоритм сортировки слиянием |

## 2.2. Анализ трудоемкости алгоритмов

### Модель вычислений:

1. Трудоемкость базовых операций.

Следующие операции стоят +1: +, -, \*, %, =, <, >, <=, >=, ==, !=, [], +=, -=

2. Трудоемкость условного перехода.

Стоимость условного перехода 0. При этом сам расчет условия оцениваем.

3. Трудоемкость цикла for.

### Сортировка вставками

|  |  |
| --- | --- |
| void **insertionSort**(int \*Array, int n)  { |  |
| for (size\_t i = 0; i < n - 1; i++)  { | 4 |
| int key = i + 1; | 2 |
| int tmp = Array[key]; | 2 |
| int j = i + 1; | 2 |
| while (j > 0 && tmp < Array[j - 1]) | 5 |
| {  Array[j] = Array[j - 1]; | 4 |
| key = j - 1; | 2 |
| j = j — 1;  } | 2 |
| Array[key] = tmp;  }  } | 2 |

Лучший случай(отсортированный массив:

Худший случай:

### Сортировка Шейкер

Лучший случай для этой сортировки — отсортированный массив (O(n)), худший — отсортированный в обратном порядке (O(n\*n)).

Наименьшее число сравнений в алгоритме Шейкер-сортировки . Это соответствует единственному проходу по упорядоченному массиву (лучший случай) [1]

### Сортировка слиянием

Чтобы оценить время работы этого алгоритма, составим рекуррентное соотношение. Пускай *T*(*n*) — время сортировки массива длины *n*, тогда для сортировки слиянием справедливо *T*(*n*)=2*T*(*n*/2)+*O*(*n*)

*O*(*n*) — время, необходимое на то, чтобы слить два массива длины *n*

Распишем это соотношение:

*T*(*n*)=2*T*(*n*/2)+*O*(*n*)=4*T*(*n*/4)+2*O*(*n*)=⋯=*T*(1)+log(*n*)*O*(*n*)=*O*(*n*log(*n*))

Сложность алгоритма O(n\*log n).

[2]

# 3. Технологическая часть

## 3.1. Требования к программному обеспечению

На вход в программу поступает массив некоторой длины. На выходе необходимо получить три отсортированных массива — результаты работы трех алгоритмов.

Также необходимо реализовать функцию для замеров времени и функцию тестирования.

## 3.2. Средства реализации

Для реализации программы был выбран язык С++. Этот язык позволяет решить

задачу с минимальными затратами по памяти. Этот язык работает

быстрее аналогов, он удобен, а так же знаком мне. Среда разработки —

Qt creator.

## 3.3. Листинг кода

### Сортировка вставками

|  |
| --- |
| void **insertionSort**(int \*Array, int n)  {  for (size\_t i = 0; i < n - 1; i++)  {  int key = i + 1;  int tmp = Array[key];  int j = i + 1;  while (j > 0 && tmp < Array[j - 1])  {  Array[j] = Array[j - 1];  key = j - 1;  j = j - 1;  }  Array[key] = tmp;  }  } |

### Сортировка шейкер

|  |
| --- |
| void **shekerSort**(int \*mass, int count)  {  int left = 0, right = count - 1;  int flag = 1;  while ((left < right) && flag > 0)  {  flag = 0;  for (int i = left; i<right; i++)  {  if (mass[i]>mass[i + 1])  {  double t = mass[i];  mass[i] = mass[i + 1];  mass[i + 1] = t;  flag = 1;  }  }  right--;  for (int i = right; i>left; i--)  {  if (mass[i - 1]>mass[i])  {  double t = mass[i];  mass[i] = mass[i - 1];  mass[i - 1] = t;  flag = 1;  }  }  left++;  }  } |

### Сортировка слиянием

|  |
| --- |
| int **cmp**(int a, int b) {  return a > b;  }  void **Merge**(int \*a, int len\_a, int \*b, int len\_b, int \*c, int (\*cmp)(int, int) = cmp) {  int i = 0;  int ai = 0;  int bi = 0;  while ( ai != len\_a && bi != len\_b)  {  if (cmp(a[ai],b[bi]) < 0)  {  c[i] = a[ai];  ai++;  }  else  {  c[i] = b[bi];  bi++;  }  i++;  }  while (bi != len\_b)  {  c[i] = b[bi];  bi++;  i++;  }  while (ai != len\_a)  {  c[i] = a[ai];  ai++;  i++;  } }  void **myMergeSort**(int \*a, int alen)  {  if( alen <= 1 ) return;  int firstLen = (alen) / 2;  int secondLen = (alen) - firstLen;  myMergeSort( a, firstLen );  myMergeSort( a + firstLen, secondLen );  int \*c = new int[alen];  Merge( a, firstLen, a + firstLen, secondLen, c,cmp);  for (int i = 0; i < alen; i++)  {  a[i] = c[i];  }  delete[] c;  } |

## 3.4. Описание тестирования

Тестирование будет реализовано в виде отдельной функции в программе, которую можно запустить по желанию пользователя. Для каждого алгоритма реализована своя функция с тестами, однако тесты для всех алгоритмов одинаковые.

Тестирование по следующим данным:

1. Проверка работы с массивом длины 1.

2. Проверка работы с массивами длины 2, отсортированными и несортированными.

3. Проверка работы с отсортированным массивом.

4. Проверка работы с обратно отсортированным массивом.

5. Проверка работы с рандомно заполненным массивом.

6. Проверка работы с массивами, в которых встречаются повторяющиеся числа.

# 4. Экспериментальная часть

## 4.1. Примеры работы

|  |
| --- |
| Рис 4. Пример работы программы на отсортированном массиве |

|  |
| --- |
| Рис 5. Пример работы программы на обратно отсортированном массиве |
| Рис 6. Пример работы программы на рандомно заполненном массиве |

## 4.2. Результаты тестирования

Все тесты прошли успешно. Результаты тестирования представлены на рисунке 7.

Данные для тестов представлены в таблице 1.

Таблица 1.

Тесты для алгоритмов сортировки.

|  |  |  |
| --- | --- | --- |
| № теста | Вход | Выход |
| 1 | [1] | [1] |
| 2 | [1,2] | [1,2] |
| 3 | [2,1] | [1,2] |
| 4 | [5,4,3,2,1] | [1,2,3,4,5] |
| 5 | [1,2,3,4,5] | [1,2,3,4,5] |
| 6 | [4,5,2,1,3] | [1,2,3,4,5] |
| 7 | [3,1,3,4,1,2,5,7,3] | [1,1,2,3,3,3,4,5,7] |

## 4.3. Постановка эксперимента по замеру времени

Были проведены временные эксперименты для массивов от 100 до 2000 элементов с шагом 100. Для каждого замера взят средний результат из 50 замеров. Замеры проведены для трех случаев: отсортированного массива, обратно отсортированного массива и радномно заполненного массива. Результаты экспериментов представлены на рисунках 7-9.

|  |
| --- |
| Рис 7. Работа алгоритмов на отсортированном массиве  На рисунке 7 видно, что алгоритм сортировки слиянием работает хуже остальных алгоритмов на отсортированном массиве. Это можно объяснить тем, что алгоритмы сортировки вставками и шейкер проверяют ситуацию отсортированного массива сразу. |
| Рис 8. Работа алгоритмов на обратно отсортированном массиве  По рисунку 8 можно сделать вывод, что сортировка слиянием в худшем случае работает быстрее двух других. Хуже всех работает сортировка Шейкер. |
| Рис 9. Работа алгоритмов на рандомно заполненном массиве  По рисунку 9 можно сказать, что алгоритмы при рандомном заполнении массива работают так же, как в худшем случае, однако по времени немного быстрее. |

## Заключение

В ходе лабораторной работы были изучены и реализованы алгоритмы сортировки: вставками, шейкер и слиянием. Были оценены трудоемкости данных алгоритмов.

## Литература

1. Wikipedia[электронный ресурс]   
URL: <https://ru.wikipedia.org/wiki/сортировка_перемешиванием>

2. Университет ИТМО Викиконспекты[элекронный ресурс]  
URL: [https://neerc.ifmo.ru/wiki/index.php?title=%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0\_%D1%81%D0%BB%D0%B8%D1%8F%D0%BD%D0%B8%D0%B5%D0%BC](https://neerc.ifmo.ru/wiki/index.php?title=Сортировка_слиянием)