Лабораторная работа

«Сортировки»

Содержание

[1 Разработка эффективных алгоритмов. Подход «Разделяй и властвуй» 1](#_Toc2880863)

[2 Задача сортировки 2](#_Toc2880864)

[3 Методы сортировки 3](#_Toc2880865)

[3.1 Сортировка пузырьком / Bubble sort 3](#_Toc2880866)

[3.2 Шейкерная сортировка / Shaker sort 3](#_Toc2880867)

[3. 3 Сортировка расческой / Comb sort 3](#_Toc2880868)

[3.4 Сортировка вставками / Insertion sort 4](#_Toc2880869)

[3.5 Сортировка Шелла / Shellsort 4](#_Toc2880870)

[3.6 Сортировка деревом / Tree sort 4](#_Toc2880871)

[3.7 Гномья сортировка / Gnome sort 5](#_Toc2880872)

[3.8 Сортировка выбором / Selection sort 5](#_Toc2880873)

[3.9 Пирамидальная сортировка / Heapsort 5](#_Toc2880874)

[3.10 Быстрая сортировка / Quicksort 5](#_Toc2880875)

[3.11 Сортировка слиянием / Merge sort 5](#_Toc2880876)

[3.12 Сортировка подсчетом / Counting sort 6](#_Toc2880877)

[3.13 LSD (least significant digit) 6](#_Toc2880878)

[3.14 Битонная сортировка / Bitonic sort: 6](#_Toc2880879)

[4 Самостоятельная работа 7](#_Toc2880880)

[Контрольные вопросы 8](#_Toc2880881)

[Литература 8](#_Toc2880882)

[Приложение А: Пример оформления 9](#_Toc2880883)

[Приложение B: Варианты 10](#_Toc2880884)

# 1 Разработка эффективных алгоритмов. Подход «Разделяй и властвуй»

Многие полезные алгоритмы имеют рекурсивную структуру: для решения данной задачи они рекурсивно вызывают сами себя один или несколько раз, чтобы решить вспомогательную задачу, имеющую непосредственное отношение к поставленной задаче.

Такие алгоритмы зачастую разрабатываются с помощью *метода декомпозиции*, или *разбиения*: сложная задача разбивается на несколько более простых, которые подобны исходной задаче, но имеют меньший объем; далее эти вспомогательные задачи решаются рекурсивным методом, после чего полученные решения комбинируются с целью получить решение исходной задачи.

Парадигма, лежащая в основе метода декомпозиции «разделяй и властвуй», на каждом уровне рекурсии включает в себя три этапа.

1. *Разделение* задачи на несколько подзадач (непересекающихся).
2. *Покорение* – рекурсивное решение этих подзадач. Когда объем подзадачи достаточно мал, выделенные подзадачи решаются непосредственно.
3. *Комбинирование* решения исходной задачи из решений вспомогательных задач.

При разбиении задачи на подзадачи полезен ***принцип балансировки***, который предполагает, что задача разбивается на подзадачи приблизительно равных размерностей, т.е. идет поддержание равновесия. Обычно такая стратегия приводит к разделению исходной задачи пополам и обработке каждой из ее частей тем же способом до тех пор, пока части не станут настолько малыми, что их можно будет обрабатывать непосредственно.

Часто такой процесс приводит к логарифмическому множителю в формуле, описывающей трудоемкость алгоритма.

# 2 Задача сортировки

***Частичным порядком*** на множестве *S* называется такое бинарное отношение *R,* что для любых *а, b* и *с* из S

1. *aRa (R* рефлексивно),
2. *aRb*и *bRc =>aRc (R* транзитивно),
3. *aRb*и *bRa => a=b (R* антисимметрично).

***Линейным****,* или ***полным порядком*** на множестве *S* называется такой частичный порядок *R* на *S*, что для любых двух элементов *a*, *b* выполняется либо *aRb,* либо *bRa (другими словами, элементы a*, *b сравнимы).*

***Задача***: Пусть дана последовательность из *п* элементов {*а*1*, а*2*, …* ,*ап*} выбранных из множества, на котором задан линейный порядок. элемент *аi*назовем записью, линейный порядок будем обозначать ≤.

Каждая запись *аi*имеет ключ *ki*, который управляет процессом сортировки, помимо ключа, запись может иметь некоторую дополнительную информацию, которая не влияет на процесс сортировки, но всегда присутствует в этой записи.

Требуется найти такую перестановку π = (π(1), π(2), …, π(*n*)) этих *п* записей, после которой ключи расположились бы в неубывающем порядке: *k* π (1) ≤ *k* π(2) ≤… ≤ *k* π(n).

Алгоритм сортировки называется ***устойчивым***, если в процессе сортировки относительное расположение элементов одинаковыми ключами не изменяется (предполагается, что элементы уже были отсортированы по некоторому вторичному ключу).

π(*i*) <π(*j*), если*k* π (*i*) ≤ *k* π(*j*) и*i*<*j.*

# 3 Методы сортировки

## 3.1 Сортировка пузырьком / Bubblesort

Требуется пройти по массиву слева направо. Если текущий элемент больше следующего, менять их местами. Делается так, пока массив не будет отсортирован. Следует отметить, что после первой итерации самый большой элемент будет находиться в конце массива, на правильном месте. После двух итераций на правильном месте будут стоять два наибольших элемента, и так далее.

Очевидно, не более чем после n итераций массив будет отсортирован. Таким образом, асимптотика в худшем и среднем случае – O(n2), в лучшем случае – O(n).

## 3.2 Шейкерная сортировка / Shakersort

Также известна как сортировка перемешиванием и коктейльная сортировка.

Следует отметить, что сортировка пузырьком работает медленно на тестах, в которых маленькие элементы стоят в конце (их еще называют «*черепахами*»). Такой элемент на каждом шаге алгоритма будет сдвигаться всего на одну позицию влево. Поэтому следует идти не только слева направо, но и справа налево. Требуется поддерживать два указателя begin и end, обозначающих, какой отрезок массива еще не отсортирован.

На очередной итерации при достижении end нужно вычитать из него единицу и двигаться справа налево, аналогично, при достижении begin добавлять единицу и двигаться слева направо.

Асимптотика у алгоритма такая же, как и у сортировки пузырьком, однако реальное время работы лучше.

## 3. 3 Сортировка расческой / Combsort

Еще одна модификация сортировки пузырьком. Для того, чтобы избавиться от «черепах», следует переставлять элементы, стоящие на расстоянии. Зафиксировать его и идти слева направо, сравнивая элементы, стоящие на этом расстоянии, переставляя их, если необходимо.

Очевидно, это позволит «черепахам» быстро добраться в начало массива. Оптимально изначально взять расстояние равным длине массива, а далее делить его на некоторый коэффициент, равный примерно 1.247.

Когда расстояние станет равно единице, выполняется сортировка пузырьком. В лучшем случае асимптотика равна O(nlogn), в худшем – O(n2).

Более подробно об этих сортировках (пузырьком, шейкерной и расческой) можно почитать в [статье [3]](https://habr.com/ru/post/204600/).

## 3.4 Сортировка вставками / Insertionsort

Создается массив, в котором после завершения алгоритма будет лежать ответ.

Требуется поочередно вставлять элементы из исходного массива так, чтобы элементы в массиве-ответе всегда были отсортированы.

Асимптотика в среднем и худшем случае – O(n2), в лучшем – O(n).

Реализовывать алгоритм удобнее по-другому (создавать новый массив и реально что-то вставлять в него относительно сложно): просто сделать так, чтобы отсортирован был некоторый префикс исходного массива, вместо вставки менять текущий элемент с предыдущим, пока они стоят в неправильном порядке.

## 3.5 Сортировка Шелла / Shellsort

Используется та же идея, что и в сортировке с расческой, и применяется к сортировке вставками.

Фиксируется некоторое расстояние. Тогда элементы массива разобьются на классы – в один класс попадают элементы, расстояние между которыми кратно зафиксированному расстоянию. Каждый класс сортируется вставками.

В отличие от сортировки расческой, неизвестен оптимальный набор расстояний. Существует довольно много последовательностей с разными оценками:

* последовательность Шелла – первый элемент равен длине массива, каждый следующий вдвое меньше предыдущего. Асимптотика в худшем случае – O(n2);
* последовательность Хиббарда – 2n — 1, асимптотика в худшем случае – O(n1,5);
* последовательность Седжвика (формула нетривиальна, ее можно увидеть в ссылке ниже) — O(n4/3);
* последовательность Пратта (все произведения степеней двойки и тройки) — O(nlog2n).

Следует отметить, что все эти последовательности нужно рассчитать только до размера массива и запускать от большего от меньшему (иначе получится просто сортировка вставками).   
  
Более подробно про сортировку можно почитать в [Википедии [4]](https://ru.wikipedia.org/wiki/Сортировка_Шелла) и в [статье [5]](https://habr.com/ru/post/204968/).

## 3.6 Сортировка деревом / Treesort

В данном алгоритме элементы вставляются в двоичное дерево поиска. После того, как все элементы вставлены достаточно обойти дерево в глубину и получить отсортированный массив. Если использовать сбалансированное дерево, например красно-черное, асимптотика будет равна O(nlogn) в худшем, среднем и лучшем случае.

Про деревья поиска можно узнать в [Википедии [6]](https://ru.wikipedia.org/wiki/Двоичное_дерево_поиска),[статье [7]](https://habr.com/ru/post/65617/) и [статье [8]](https://habr.com/ru/post/66926/).

## 3.7 Гномья сортировка / Gnomesort

Алгоритм похож на сортировку вставками. Поддерживается указатель на текущий элемент, если он больше предыдущего или он первый — указатель смещается на позицию вправо, иначе меняется текущий и предыдущий элементы местами и происходит смещение влево.

## 3.8 Сортировка выбором / Selectionsort

В данном алгоритме ищется минимум в массиве после текущего элемента и меняется местами, если надо.

Таким образом, после i-ой итерации первые i элементов будут стоять на своих местах.

Асимптотика: O(n2) в лучшем, среднем и худшем случае. Нужно отметить, что эту сортировку можно реализовать двумя способами – сохраняя минимум и его индекс или просто переставляя текущий элементс рассматриваемым, если они стоят в неправильном порядке. Первый способ является предпочтительным.

## 3.9 Пирамидальная сортировка / Heapsort

Развитие идеи сортировки выбором. Следует воспользоваться структурой данных «куча» (или «пирамида», откуда и название алгоритма). Она позволяет получать минимум за O(1), добавляя элементы и извлекая минимум за O(logn).

Таким образом, асимптотика O(nlogn) в худшем, среднем и лучшем случае.   
  
Про кучу можно прочитать [в Википедии [9]](https://ru.wikipedia.org/wiki/Куча_(структура_данных)) и [статье [10]](https://habrahabr.ru/post/112222/).

## 3.10 Быстрая сортировка / Quicksort

Выбирается некоторый опорный элемент. После этого все элементы меньше опорного перекидываются налево, а больше – направо. Алгоритм рекурсивно вызывается от каждой из частей.

В итоге получается отсортированный массив, так как каждый элемент меньше опорного стоял раньше каждого большего опорного.

Асимптотика: O(nlogn) в среднем и лучшем случае, O(n2). Наихудшая оценка достигается при неудачном выборе опорного элемента.

## 3.11 Сортировка слиянием / Mergesort

Сортировка, основанная на парадигме «разделяй и властвуй».

Массив делится пополам, рекурсивно сортируются части, после чего выполняется процедура слияния: поддерживаются два указателя, один на текущий элемент первой части, второй – на текущий элемент второй части.

Из этих двух элементов выбирается минимальный, вставляется в ответ и сдвигается указатель, соответствующий минимуму.

Слияние работает за O(n), уровней всего logn, поэтому асимптотика O(nlogn).

Эффективно заранее создать временный массив и передать его в качестве аргумента функции. Эта сортировка рекурсивна, как и быстрая, а потому возможен переход на квадратичную при небольшом числе элементов.

## 3.12 Сортировка подсчетом / Countingsort

Создается массив размера r – l, где l – минимальный, а r – максимальный элемент массива.

После этого алгоритм проходит по массиву и считает количество вхождений каждого элемента. После этого алгоритм проходит по массиву значений и выписывает каждое число столько раз, сколько нужно.

Асимптотика – O(n + r — l). Можно модифицировать этот алгоритм, чтобы он стал стабильным: для этого определить место, где должно стоять очередное число (это просто префиксные суммы в массиве значений) и идти по исходному массиву слева направо, ставя элемент на правильное место и увеличивая позицию на 1.

Эта сортировка не используется в задании, поскольку не позволяет создавать массивы требуемого размера для больших чисел. Однако она может быть полезной.

## 3.13 LSD (leastsignificantdigit)

Является одним из вариантов цифровой или поразрядной сортировки (radixsort).

Алгоритм использует двоичное представление чисел. На каждом шаге алгоритм будет сортировать числа таким образом, чтобы они были отсортированы по первым k \* i битам, где k – некоторая константа. Из данного определения следует, что на каждом шаге достаточно стабильно сортировать элементы по новым k битам. Для этого идеально подходит сортировка подсчетом (необходимо 2k памяти и времени, что немного при удачном выборе константы).

Асимптотика: O(n), если считать, что числа фиксированного размера (а в противном случае нельзя было бы считать, что сравнение двух чисел выполняется за единицу времени).

## 3.14 Битонная сортировка / Bitonicsort:

Идея данного алгоритма заключается в том, что исходный массив преобразуется в битонную последовательность – последовательность, которая сначала возрастает, а потом убывает.

Ее можно эффективно отсортировать следующим образом: разбить массив на две части, создать два массива, в первый добавить все элементы, равные минимуму из соответственных элементов каждой из двух частей, а во второй – равные максимуму.

Утверждается, что получатся две битонные последовательности, каждую из которых можно рекурсивно отсортировать тем же образом, после чего можно склеить два массива (так как любой элемент первого меньше или равен любого элемента второго).

Для того, чтобы преобразовать исходный массив в битонную последовательность, требуется сделать следующее: если массив состоит из двух элементов, можно просто завершиться, иначе разделить массив пополам, рекурсивно вызвав от половинок алгоритм, после чего отсортировать первую часть по порядку, вторую в обратном порядке и склеить. Очевидно, получится битонная последовательность.

Асимптотика: O(nlog2n), поскольку при построении битонной последовательности используется сортировка, работающая за O(nlogn), а всего уровней было logn. Также следует заметить, что размер массива должен быть равен степени двойки, так что, возможно, придется его дополнять фиктивными элементами (что не влияет на асимптотику).

# 4 Самостоятельная работа

Провести сравнение двух методов сортировки.

1. Реализовать 3 алгоритма сортировки, в соответствии с [вариантом](#_Приложение_B:_Варианты), определенным преподавателем.
2. Подготовить отчет, в котором описать реализованные алгоритмы, привести блок-схему и код программы.
3. Провести экспериментальное сравнение производительности работы алгоритмов:
   * оценить число операций сравнения и число операций обмена (перемещений) элементов, (*Например, используйте глобальные счетчики, которые увеличиваются в соответствующих методах*.)
   * время работы (*Реализовать метод time( )*).
4. Сравнение методов сортировки провести для *n* = 100, 1 000, 10 000и следующем порядке входных элементов:
   1. массив случайных чисел;
   2. массив случайных чисел с большим количеством повторений одного элемента (K%);
   3. массив из Lотсортированных подмассивов;
   4. отсортированный массив, в которомM% чисел заменены на случайные.

*Учитывайте, что сравнение методов сортировки проводится для одинаковых входных данных.*

1. Результаты экспериментов оформить на основе нескольких запусков программы в виде трех сводных таблиц по образцу (Таблица 1)
2. Составить 4 графика для каждого массива: по оси Х – метод сортировки, по оси Y– значение времени на обработку.
3. Написать выводы по лабораторной работе.

Таблица 1 – Результаты сортировки методом … (наименование метода).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **n** | **Параметр** | **Сгенерированные массивы** | | | | **Среднее значение** |
| **Массив а** | **Массив б** | **Массив в** | **Массив г** |
| 103 | Compare |  |  |  |  |  |
|  | Swap |  |  |  |  |  |
|  | Time |  |  |  |  |  |
| 104 | Compare |  |  |  |  |  |
|  | Swap |  |  |  |  |  |
|  | Time |  |  |  |  |  |
| … | … | … | … | … | … | … |

# Контрольные вопросы

1. Сформулируйте задачу сортировки.
2. Перечислите типы сортировок, охарактеризуйте их.
3. Охарактеризуйте подход к решению задач «метод разделяй и властвуй». Приведите примеры.
4. Какой метод сортировки показал лучшие результаты? Почему?
5. Какой метод сортировки показал худшие результаты? Почему?
6. Чему равно время работы быстрой сортировки в случае, когда все элементы массива А одинаковы по величине?

# Литература

1. Кормен, Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. – Москва: Вильямс, 2013. – 1328 с. [PDF (167 Мб)](https://yadi.sk/i/IMPynFi0WSOqSw)[DjVu (18 Мб)](https://yadi.sk/i/luHQRgdK2uHWhw)
2. Кнут В. Искусство программирования. Том 3. Сортировка и поиск. [PDF (63,5 Мб)](https://yadi.sk/i/TpZYY_neYi5xPQ)
3. <https://habr.com/ru/post/204600/> Пузырьковая сортировка и все-все-все
4. <https://ru.wikipedia.org/wiki/Сортировка_Шелла>
5. <https://habr.com/ru/post/204968/>Сортировка Шелла
6. <https://ru.wikipedia.org/wiki/Двоичное_дерево_поиска>
7. <https://habr.com/ru/post/65617/> Структуры данных: бинарные деревья. Часть 1
8. <https://habr.com/ru/post/66926/> Структуры данных: бинарные деревья. Часть 2: обзор сбалансированных деревьев
9. <https://ru.wikipedia.org/wiki/Куча_(структура_данных)>
10. <https://habr.com/ru/post/112222/> Структуры данных: двоичная куча (binaryheap)

# Приложение А: Пример оформления

**Отчет по лабораторной работе №3**

На тему «Сортировки»

Вариант №5

Выполнил: ст. группы 2171199

Пушкин А.С.

Проверил:

Хузятова Л.Б.

# Приложение B: Варианты

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Вариант** | **K, %** | **L** | **M, %** | **Алгоритм 1** | **Алгоритм 2** | **Алгоритм 3** |
| **1** | 10 | 4 | 1 | 3.1 | 3.3 | 3.9 |
| **2** | 20 | 4 | 1 | 3.2 | 3.3 | 3.10 |
| **3** | 25 | 4 | 1 | 3.4 | 3.3 | 3.11 |
| **4** | 30 | 4 | 1 | 3.7 | 3.3 | 3.13 |
| **5** | 35 | 4 | 2 | 3.8 | 3.3 | 3.9 |
| **6** | 45 | 5 | 2 | 3.1 | 3.5 | 3.10 |
| **7** | 50 | 5 | 2 | 3.2 | 3.5 | 3.11 |
| **8** | 10 | 5 | 2 | 3.4 | 3.5 | 3.13 |
| **9** | 20 | 5 | 3 | 3.7 | 3.5 | 3.9 |
| **10** | 25 | 5 | 3 | 3.8 | 3.5 | 3.10 |
| **11** | 30 | 10 | 3 | 3.1 | 3.6 | 3.11 |
| **12** | 35 | 10 | 3 | 3.2 | 3.6 | 3.13 |
| **13** | 45 | 10 | 4 | 3.4 | 3.6 | 3.9 |
| **14** | 50 | 10 | 4 | 3.7 | 3.6 | 3.10 |
| **15** | 10 | 10 | 4 | 3.8 | 3.6 | 3.11 |
| **16** | 20 | 12 | 4 | 3.1 | 3.14 | 3.13 |
| **17** | 25 | 12 | 5 | 3.2 | 3.14 | 3.9 |
| **18** | 30 | 12 | 5 | 3.4 | 3.14 | 3.10 |
| **19** | 35 | 12 | 5 | 3.7 | 3.14 | 3.11 |
| **20** | 45 | 12 | 5 | 3.8 | 3.14 | 3.13 |
| **21** | 50 | 20 | 6 | 3.1 | 3.6 | 3.10 |
| **22** | 20 | 20 | 6 | 3.4 | 3.14 | 3.9 |