**1. Основные структуры данных**

**2. Постановка задачи сортировки**

**3. Алгоритм сортировки методом прямого выбора**

**4. Классы сложности алгоритмов**

**5. Алгоритмы пузырьковой и шейкерной сортировок**

**6. Алгоритм сортировки методом прямого включения**

**7. Алгоритм сортировки методом Шелла**

**8. Алгоритм двоичного поиска в упорядоченном массиве**

**9. Алгоритм сортировки данных с произвольной структурой**

**10. Алгоритмы индексации данных**

**11. Алгоритм пирамидальной сортировки**

**12. Теорема о сложности сортировки**

**13. Алгоритм быстрой сортировки Хоара**

**14. Динамические структуры данных. Адреса и указатели**

**15. Динамически распределяемая память**

**16. Алгоритм индексации через массив указателей**

**17. Алгоритмы работы с линейными списками**

**18. Задача сортировки последовательностей**

**19. Алгоритм прямого слияния**

**20. Алгоритм цифровой сортировки**

**21. Хеш-функции и их приложение к задаче поиска**

**22. Хеширование методом прямого связывания**

**23. Алгоритм открытой адресации: линейные и квадратичные пробы**

1. **Основные структуры данных**

Любая программа в процессе работы обрабатывает некоторые данные. По способу представления в памяти компьютера данные можно разделить на две группы: статические и динамические. Статические данные имеют фиксированную структуру, поэтому размер выделенной для них памяти постоянен. Динамические данные изменяют свою структуру в процессе работы программы, при этом объём памяти изменяется. Основные структуры данных представлены в следующей таблице

****

[**https://eios.sibsutis.ru/pluginfile.php/44046/mod\_resource/content/1/%D0%9F%D0%BE%D1%81%D0%BE%D0%B1%D0%B8%D0%B5\_%D0%A1%D0%B8%D0%90%D0%9E%D0%94.pdf**](https://eios.sibsutis.ru/pluginfile.php/44046/mod_resource/content/1/%D0%9F%D0%BE%D1%81%D0%BE%D0%B1%D0%B8%D0%B5_%D0%A1%D0%B8%D0%90%D0%9E%D0%94.pdf)

**2. Постановка задачи сортировки**

Пусть дан массив и для всех его элементов определены операции отношения: меньше, больше, равно. Необходимо отсортировать массив, т.е. переставить элементы массива А таким образом, что выполняется одно из следующих неравенств: a1 ≤ а2 ≤ а3 ≤ … ≤ аn a1 ≥ a2 ≥ a3 ≥ ... ≥ an Если выполняется первое неравенство, то массив сортируется по возрастанию и такой порядок элементов будем называть прямым. Если выполняется второе неравенство, то массив отсортирован по убыванию и такой порядок элементов будем называть обратным.

Сортировка называется устойчивой, если после её проведения в массиве не меняется относительный порядок элементов с одинаковыми ключами.

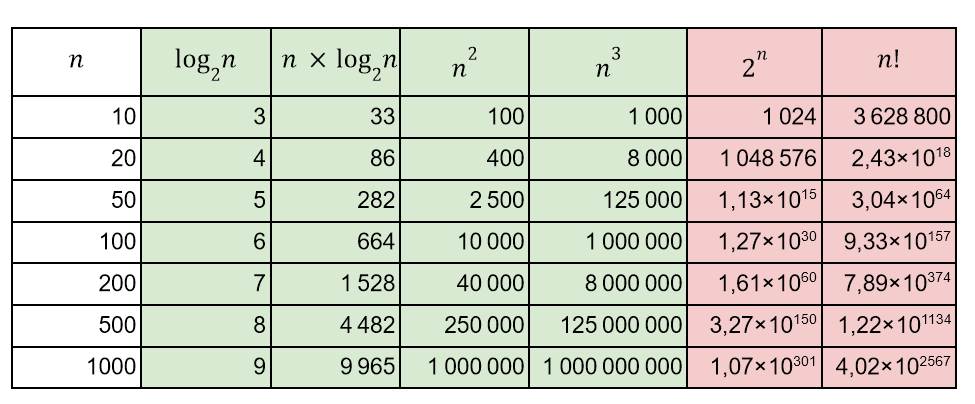
Для проверки правильности сортировки массива могут использоваться следующие приемы. Вычисление контрольной суммы элементов массива до и после сортировки дает возможность проверить потерю элементов массива во время процесса сортировки. Определение количества серий элементов массива, т.е. неубывающих последовательностей из элементов массива, позволяет проверить правильность упорядочивания массива, поскольку массив, отсортированный по возрастанию, состоит из одной серии, а в массиве, отсортированном по убыванию, количество серий равно количеству элементов в массиве

**3. Алгоритм сортировки методом прямого выбора**

метод прямого выбора, заключается в следующем. Находим наименьший элемент массива и обмениваем его с первым элементом массива. Таким образом, первый элемент можно больше не рассматривать. Среди оставшихся элементов находим наименьший элемент и обмениваем его со вторым элементом массива. Среди оставшихся элементов находим наименьший и переставляем его на третье место и т. д.

****

**4. Классы сложности алгоритмов**



отличать методы сортировки по времени, затрачиваемому на реализацию сортировки.

Для сортировок основными считаются две операции: операция сравнения элементов и операция пересылки элемента.

Будем считать, что в единицу времени выполняется одна операция сравнения или пересылки. Таким образом, время или трудоемкость метода имеет две составляющие М и С, где M – количество операций пересылки. C– количество операций сравнения.

Нетрудно видеть, что M и C – зависят от количества элементов в массиве, т.е. являются функциями от длины массива.

Часто бывает трудно определить точное выражение для трудоемкости алгоритма. В этом случае пользуются асимптотической оценкой времени работы.

Будем говорить, что функция g(x) асимптотически доминирует на f(x) или g(x)=O(f(x)), если |g(x)|≤const|f(x)| при x→ ∞. В дальнейшем будем рассматривать асимптотическое поведение величин М и С в зависимости от числа элементов в массиве n, при n→ ∞.

Для функций f, f1, f2 и константы k справедливы свойства:

1. f = O(f)

2. O(k\*f) = O(f)

3. O(f1+f2) = O(f1) +O(f2)

**5. Алгоритмы пузырьковой и шейкерной сортировок**

метод пузырьковой сортировки заключается в следующем. Двигаясь от конца массива к его началу, будем сравнивать между собой соседние элементы. При этом если правый элемент aj будет меньше чем левый aj-1, j=n, n-1, … ,2, то поменяем их местами. Таким образом, при первом проходе наименьший элемент переместится на первое место и можно не учитывать его при сортировке оставшейся части массива. При втором проходе наименьший элемент из оставшихся “всплывёт” на второе место. Через (n-1) итераций массив будет отсортирован. Метод обеспечивает устойчивую сортировку.

Mсред=О(n 2 ), при n→∞

Шейкерная сортировка — разновидность [пузырьковой сортировки](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0_%D0%BF%D1%83%D0%B7%D1%8B%D1%80%D1%8C%D0%BA%D0%BE%D0%BC)

***Во-первых***, если при движении по части массива перестановки не происходят, то эта часть массива уже отсортирована и, следовательно, её можно исключить из рассмотрения.

***Во-вторых***, при движении от конца массива к началу минимальный элемент «всплывает» на первую позицию, а максимальный элемент сдвигается только на одну позицию вправо.

Границы рабочей части массива (то есть части массива, где происходит движение) **устанавливаются в месте последнего обмена** на каждой итерации. Массив просматривается поочередно справа налево и слева направо.

Лучший случай для этой сортировки — отсортированный массив (𝑂(𝑛)O(n)), худший — отсортированный в обратном порядке (𝑂(𝑛2)O(n^2)).

**6. Алгоритм сортировки методом прямого включения**

Метод прямого включения заключается в следующем. Начиная с i = 2, i=2,… n, берём очередной i–й элемент массива и включаем его на нужное место среди первых (i-1) элементов, при этом все элементы, которые больше ai сдвигаются на одну позицию вправо.

средняя трудоемкость этого метода имеет квадратичный порядок, т.е. С = О(n 2 ) М = О(n 2 ), при n→∞.

**7. Алгоритм сортировки методом Шелла**

в методе прямого включения, чем больше упорядочен массив, тем меньше операций требуется для его сортировки. имеет смысл попытаться предварительно несколько улучшить порядок элементов в массиве, а затем отсортиро- 19 вать массив методом прямого включения.

Предварительное упорядочивание будем проводить с помощью k – сортировок. Суть k – сортировки заключается в следующем. Массив разбивается на последовательности с шагом k ai , ak+i, a2k+i, …,a[n/k]k+i, i = 1, 2,…,k и сортировка происходит только внутри этих последовательностей. Обозначим через H последовательность из m возрастающих шагов H=(h1, h2, … hm), где h1=1, h1 < h2 < h3 < … < hm Метод Шелла состоит в последовательном проведении hi-сортировки, i=m, m1,…, 1, причем h1=1 гарантирует, что массив будет полностью отсортирован, поскольку 1-сортировка является методом прямого включения.

Эффективность метода зависит от выбора значений шагов.

часто используется следующая последовательность шагов, предложенная Кнутом. h1=1, hi=2hi-1 +1, i=2,… m, m=⎣log2 n⎦ −1 При такой последовательности шагов средний порядок трудоёмкости O(n 1.2), n → ∞

Метод Шелла не устойчив

**8. Алгоритм двоичного поиска в упорядоченном массиве**

Алгоритм двоичного поиска в упорядоченном массиве сводится к следующему. Берём средний элемент отсортированного массива и сравниваем с ключом X. Возможны три варианта: 1. Выбранный элемент равен X. Поиск завершён. 2. Выбранный элемент меньше X. Продолжаем поиск в правой половине массива. 3. Выбранный элемент больше X. Продолжаем поиск в левой половине массива.

трудоёмкость двоичного поиска в обоих случаях С=O(log n), n → ∞

**L: = 1, R: =n, Найден: = нет**

**DO (L≤R)**

**m: =**

**(L  R)/ 2**

**IF (am=X) Найден: =да OD FI**

**IF (am < X) L: = m+1**

**ELSE R: = m-1**

**FI**

**OD**

**Алгоритм на псевдокоде**

**Поиск элемента с ключом X**

**L: = 1, R: =n**

**DO (L<R)**

**m: =**

**(L  R)/ 2**

**IF (am < X) L: = m+1**

**ELSE R: = m**

**FI**

**OD**

**IF (ar=X) Найден: =да**

**ELSE Найден: =нет**

**FI**

**9. Алгоритм сортировки данных с произвольной структурой**

**Сравнение данных произвольной структуры:**

Если необходимо отсортировать телефонный справочник по фамилиям абонентов, то логическая функция Less (меньше) может выглядеть следующим образом:

**function less ( x,y: <тип записи>): boolean;**

**begin**

**less:=x.Name <y.Name;**

**end;**

Такой подход позволяет путем изменения функции Less учитывать любые сложные условия упорядочивания массива элементов произвольной структуры.

**Сортировка по множеству ключей. Индексация:**

Вначале построения индексный массив В заполняется целыми числами от 1 40 до n. Затем производится сортировка, но при условии, что в операциях сравнения элементы массива А индексируются через массив В. Перестановки делаются только в массиве В. Тогда при доступе к элементам массива А через индексный массив В А[B[i]] можно работать с массивом А как с упорядоченным по возрастанию (например, производить быстрый поиск элементов), в то время как сами элементы А физически не переставляются.

**Индексация через массив указателей:**

Индексация через массив указателей отличается от обычной индексации тем, что вместо номеров элементов в индексный массив записываются адреса сортируемых элементов. К достоинствам такой индексации можно отнести то, что исходные данные могут располагаться не только в массиве, а произвольным образом в динамической памяти.

**10. Алгоритмы индексации данных**

Пусть рассмотренный выше телефонный справочник необходимо использовать для быстрого поочередного поиска абонентов или по номеру телефона, или по фамилии абонента. Пересортировка массива то по одному, то по другому ключу требует значительных затрат времени. Для эффективного решения подобной задачи используется прием, называемый индексацией, или созданием индексного массива

**Сортировка по множеству ключей. Индексация:**

Вначале построения индексный массив В заполняется целыми числами от 1 40 до n. Затем производится сортировка, но при условии, что в операциях сравнения элементы массива А индексируются через массив В. Перестановки делаются только в массиве В. Тогда при доступе к элементам массива А через индексный массив В А[B[i]] можно работать с массивом А как с упорядоченным по возрастанию (например, производить быстрый поиск элементов), в то время как сами элементы А физически не переставляются.

**Индексация через массив указателей:**

Индексация через массив указателей отличается от обычной индексации тем, что вместо номеров элементов в индексный массив записываются адреса сортируемых элементов. К достоинствам такой индексации можно отнести то, что исходные данные могут располагаться не только в массиве, а произвольным образом в динамической памяти.