МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«ТЮМЕНСКИЙ ИНДУСТРИАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт геологии и нефтегазодобычи

Кафедра «Кибернетических систем»

АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

Методические указания по выполнению курсовой работы
для обучающихся направлений подготовки
09.03.01 «Информатика и вычислительная техника»,
09.03.02 «Информационные системы и технологии»
всех форм обучения

Составитель ***И.О. Лозикова,***
старший преподаватель

Тюмень
ТИУ
2021

Алгоритмы и структуры данных: Методические указания по выполнению курсовой работы для обучающихся направлений подготовки 09.03.01 «Информатика и вычислительная техника», 09.03.02 «Информационные системы и технологии» всех форм обучения / сост. И.О. Лозикова; Тюменский индустриальный университет. – Тюмень: Издательский центр БИК, ТИУ, 2021. – 27с.– Текст: непосредственный.

Методические указания рассмотрены и рекомендованы к изданию на заседании кафедры «Кибернетических систем»

«12» ноября 2021 года, протокол № 4

Аннотация

Методические указания по выполнению курсовой работы дисциплины «Алгоритмы и структуры данных» для обучающихся направления подготовки 09.03.02 «Информационные системы и технологии» всех форм обучения.

Приведено содержание основных этапов выполнения курсовой работы, требования к решению, оформлению и защите курсовой работы, варианты курсовых заданий. Даны методические указания и рекомендации по выполнению работ.

Методические указания могут быть использованы для организации выполнения курсовых и самостоятельных работ дисциплины «Структуры и алгоритмы обработки данных», «Программирование» других направлений обучения.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Методические указания представляют требования к выполнению курсовой работы по дисциплине «Алгоритмы и структуры данных» для обучающихся направлений бакалавриата 09.03.02 «Информационные системы и технологии» и 09.03.01 «Информатика и вычислительная техника».

Курсовая работа является частью программы изучения дисциплины, проводится согласно графику учебного процесса и имеет целью оценить:

– теоретические знания, практические навыки и умения студента в области алгоритмизации, программирования и структур данных;

– умение студента решать поставленную задачу;

– умение студента выполнить самостоятельно и в срок заданную работу;

– умение студента самостоятельно работать со специальной литературой.

В рамках выполнения работы необходимо разработать программу в соответствии с предложенным вариантом, подготовить пояснительную записку о проделанной работе, и защитить работу преподавателю.

График выполнения работы соответствует графику аттестации семестра. Оценка за неё выставляется в зачетную книжку.

1. ТЕМА И ВАРИАНТЫ ЗАДАНИЯ

Тема работы: исследование эффективности методов сортировок массива.

Цели работы:

✓ Изучение и реализация заданных методов сортировки.

✓ Экспериментальное и теоретическое исследование эффективности методов сортировки.

✓ На основе исследований определить эффективный метод.

Задание:

1. Реализовать в виде функций методы сортировки одномерного массива по заданному варианту Таблица 1.

2. Разработать консольную программу с заданным интерфейсом в виде меню операций и набором дополнительных операций для вывода показателей эффективности. Архитектура программы - функциональная декомпозиция всех операций.

Интерфейс программы включает следующие операции:

- изменение размера массива с шагом в заданном диапазоне,
- формирование в массиве случайной выборки значений,
- формирование в массиве упорядоченной выборки значений,
- чтение/запись заданного набора данных из файла/консоль,
- выполнения методов сортировки (по варианту задания).

Для тестирования эффективности алгоритмов сортировки программа реализует вычисления и вывод в виде таблицы показателей эффективности:

- число выполненных сравнений в процессе сортировки,
- число выполненных обменов в процессе сортировки,
- вычисление системного времени работы каждого алгоритма сортировки на одинаковом наборе данных, Приложение 2.

3. Выполнить отладку и тестирование программы с помощью меню операций. Провести тестовые испытания методов сортировки на массиве размером N=10 и проверить правильность результатов работы.

4. Выполнить сравнительное тестирование эффективности алгоритмов сортировки для лучшего, худшего и среднего случаев для массива, где размер N изменяется на отрезке [1000; 100000] с шагом dN=500. Для каждого случая фиксировать показатели: число выполненных сравнений, обменов и времени. Вывести таблицу показателей.

5. Провести анализ экспериментальных показателей эффективности алгоритмов сортировки, сравнение с теоретическими оценками эффективности методов сортировки и сделать вывод об эффективности заданных методов.

6. Составить отчёт по работе в форме пояснительной записки.

Таблица 1. Варианты задания

Вариант	**Метод сортировки 1**	**Метод сортировки 2**
1.	Алгоритм сортировки выбором	Алгоритм быстрой сортировки
2.	Алгоритм сортировки обменом (пузырьком)	Алгоритм пирамидальной сортировки
3.	Алгоритм сортировки вставками	Алгоритм сортировки Шелла.
4.	Алгоритм обменной сортировки	Алгоритм сортировки разделением
5.	Алгоритм LSD-поразрядной сортировки (величина разряда - 1) на числовой информации.	Алгоритм LSD-поразрядной сортировки (величина разряда - 4) на числовой информации.
6.	Алгоритм обменной сортировки	Алгоритм сортировки слиянием
7.	Алгоритм интроспективной сортировки	Алгоритм быстрой сортировки
8.	Алгоритм MSD-поразрядной сортировки (величина разряда - 2) на символьной информации.	Алгоритм MSD-поразрядной сортировки (величина разряда - 3) на символьной информации.
9.	Алгоритм сортировки пузырьком	Алгоритм сортировки расчёской.
10.	Алгоритм сортировки вставками	Алгоритм сортировки Timsort
11.	Алгоритм обменной сортировки	Алгоритм сортировки перемешиванием
12.	Алгоритм сортировки вставками	Алгоритм «гномья сортировка»
13.	Алгоритм сортировки выбором	Алгоритм сортировки с помощью дерева поиска
14.	Алгоритм сортировки расчёской.	Алгоритм сортировки слиянием
15.	Алгоритм пирамидальной сортировки, Heapsort	Алгоритм быстрой сортировки
16.	Алгоритм сортировки Шелла	Алгоритм сортировки обменом
17.	Алгоритм плавной сортировки	Алгоритм сортировки выбором

18.	Придурковатая сортировка, Stooge sort	Быстрая сортировка
19.	Сортировка слиянием	Блочная сортировка, Bucket sort
20.	Сортировка Bogosort	Алгоритм сортировки обменом
21.	Пирамидальная сортировка	Интроспективная сортировка
22.	Алгоритм MSD-поразрядной сортировки (величина разряда - 1) на числовой информации.	Алгоритм MSD-поразрядной сортировки (величина разряда - 2) на числовой информации.
23.	Алгоритм сортировки с помощью дерева поиска	Быстрая сортировка
24.	Алгоритм LSD-поразрядной сортировки (величина разряда - 4) на символьной информации.	Алгоритм LSD-поразрядной сортировки (величина разряда - 2) на символьной информации.
25.	Алгоритм «гномья сортировка»	Алгоритм сортировки обменом
26.	Алгоритм сортировки слиянием	Придурковатая сортировка, Stooge sort
27.	Алгоритм сортировки Timsort	Алгоритм сортировки слиянием
28.	Алгоритм сортировки с помощью дерева поиска	Алгоритм сортировки расчёской.

2. ОСНОВЫ АНАЛИЗА АЛГОРИТМОВ

Введение в анализ

При теоретическом анализе алгоритма определяется сложность по времени и сложность по памяти.

Анализ сложности алгоритма по памяти определяет, сколько памяти нужно тому или иному алгоритму для выполнения работы. При ограниченных объемах компьютерной памяти (как внешней, так и внутренней) этот анализ носит принципиальный характер. Все алгоритмы разделяются на такие, которым достаточно ограниченной памяти, и те, которым нужно дополнительное пространство. Нередко программистам приходится выбирать более медленный алгоритм просто потому, что он обходился имеющейся памятью и не требовал внешних устройств.

Сложность по времени - примерное количество операций в алгоритме (вычислительная сложность алгоритма). Важна зависимость количества операций алгоритма от размера входных данных. Алгоритмы решения одной и той же задачи сравнивают по скорости роста количества операций при разных размерах входных данных.

Анализ алгоритма оценивает, насколько быстро решается задача на массиве входных данных длины **n**. Например, сколько сравнений потребует алгоритм сортировки при упорядочении списка из **n** величин по возрастанию, или подсчитать, сколько арифметических операций нужно для умножения двух матриц размером **n** * **n**. Одну и ту же задачу можно решить с помощью различных алгоритмов. Анализ алгоритмов позволяет выбрать эффективное решение задачи. Различные характеристики предназначены для сравнения эффективности двух разных алгоритмов, решающих одну задачу. Но нельзя сравнивать между собой алгоритм сортировки и алгоритм умножения матриц, надо сравнивать друг с другом два разных алгоритма сортировки.

Допустим, выполняется следующее утверждение.

Время выполнения алгоритма alg прямо пропорционально функции T(n).

В таких случаях говорят, что алгоритм alg имеет порядок T(n) (order T(n)). Этот факт обозначается как O(T(n)). Функция T(n) называется сложностью алгоритма (growth-rate function). Поскольку в обозначении используется прописная буква O (первая буква слова order (порядок)), оно называется обозначением O-большое (Big-O notation). Если время решения задачи прямо пропорционально размеру n входных данных, то сложность задачи равна O(n), т.е. имеет порядок n. Если время решения задачи прямо пропорционально квадрату размера входных данных , т.е. n^2, то сложность задачи равна $O(n^2)$ и т.д. При этом следует иметь в виду, что символ O - это не функция.

Определение. **T(n)** — время выполнения операции для n исходных данных, измеряемое «в шагах», которые необходимо выполнить алгоритму для достижения результата, где **n** – количество обрабатываемых значений (число элементов в массиве, количество исходных данных).

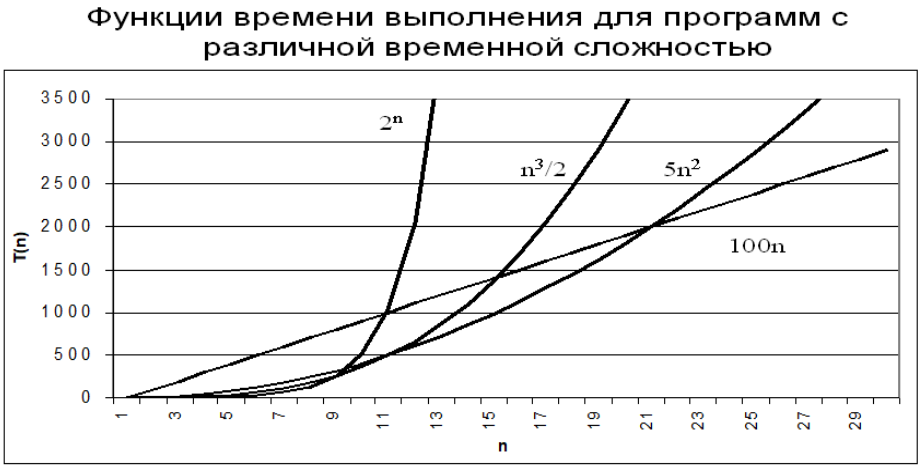


Рис. 1 Графики функций временной сложности

Точное количество операций, выполненных алгоритмом, не играет существенной роли в анализе алгоритмов. Более важной характеристикой оказывается скорость роста числа операций при возрастании объема входных данных или *порядок функции временной сложности,* рис.1. Скорость роста алгоритма называется **асимптотическим поведением алгоритма**. Асимптотический метод анализа алгоритмов – это метод описания скорости роста (предельного поведения) функций временной сложности алгоритма.

Теоретическое нахождение функции времени выполнения программ (без определения констант пропорциональности) — сложная математическая задача. Однако на практике определение времени выполнения (без значения констант) является вполне разрешимой задачей — для этого нужно знать только несколько базовых принципов.

1. *При оценке сложности алгоритма можно учитывать только старшую степень.* Например, если алгоритм имеет сложность $O(n^3 + 4n^2 + 3n)$, он имеет порядок $O(n^3)$. Из таблицы, показанной на рис. 2, видно, что слагаемое n^3 намного больше, чем слагаемые $4n^2$ и $3n$, особенно при больших значениях n, порядок функции $n^3 + 4n^2 + 3n$ совпадает с порядком функции n^3. Эти функции имеют одинаковый порядок роста. Итак, даже если сложность алгоритма равна $O(n^3 + 4n^2 + 3n)$, можно говорить, что он имеет порядок просто $O(n^3)$. Таким образом, алгоритмы имеют сложность O(T(n)), где функцией T(n) является одна из функций, представленных на рис. 2,3.

Функция	10	100	1,000	10,000	100,000	1,000,000
1	1	1	1	1	1	1
$\log_2 n$	3	6	9	13	16	19
n	10	10^2	10^3	10^4	10^5	10^6
$n * \log_2 n$	30	664	9,965	10^5	10^6	10^7
n^2	10^2	10^4	10^6	10^8	10^{10}	10^{12}
n^3	10^3	10^6	10^9	10^{12}	10^{15}	10^{18}
2^n	10^3	10^{30}	10^{301}	$10^{3,010}$	$10^{30,103}$	$10^{301,030}$

Рис. 2. Сравнение сложности алгоритмов в табличном виде

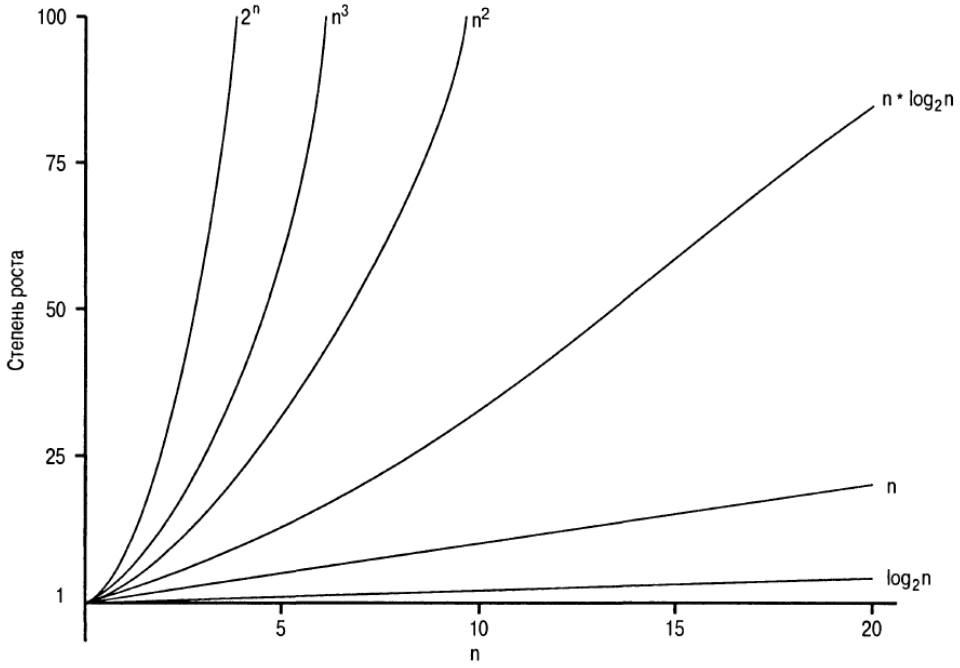


Рис. 3 Сравнение сложности алгоритмов в графическом виде

2. *При оценке сложности алгоритма можно игнорировать множитель при старшей степени.* Например, если алгоритм имеет сложность $O(5n^3)$, можно говорить, что он имеет порядок $O(n^3)$. Это утверждение следует из определения величины O(T(n)), если положить $k=5$.

3. $O(T(n))+O(g(n))=O(T(n)+g(n))$. Функции, описывающие сложность алгоритма, можно складывать. Например, если алгоритм имеет сложность $O(n^2)+ O(n)$, то говорят, что он имеет сложность $O(n^2 + n)$ или $O(n^2)$. Аналогичные правила выполняются для умножения.

Из этих свойств следует, что при оценке эффективности алгоритма нужно оценить лишь порядок его сложности.

Некоторые часто встречающиеся классы функций приведены в таблице на рис. 4. В таблице приведены значения функций каждого класса на широком диапазоне значений аргумента. Видно, что при небольших размерах входных данных значения функций отличаются незначительно,

однако при росте этих размеров разница существенно возрастает. Поэтому анализ алгоритмов проводят на больших объемах входных данных, поскольку на малых объемах принципиальная разница оказывается скрытой.

Таким образом, при анализе алгоритмов важен порядок скорости роста, к которому относится алгоритм, нежели точное количество выполняемых им операций каждого типа. Скорость роста алгоритма должна быть оценена на больших объемах входных данных.

n	$\log_2 n$	n	$n\log_2 n$	n^2	n^3	2^n
1	0.0	1.0	0.0	1.0	1.0	2.0
2	1.0	2.0	2.0	4.0	8.0	4.0
5	2.3	5.0	11.6	25.0	125.0	32.0
10	3.3	10.0	33.2	100.0	1000.0	1024.0
15	3.9	15.0	58.6	225.0	3375.0	32768.0
20	4.3	20.0	86.4	400.0	8000.0	1048576.0
30	4.9	30.0	147.2	900.0	27000.0	1073741824.0
40	5.3	40.0	212.9	1600.0	64000.0	1099511627776.0
50	5.6	50.0	282.2	2500.0	125000.0	1125899906842620.0
60	5.9	60.0	354.4	3600.0	216000.0	1152921504606850000.0
70	6.1	70.0	429.0	4900.0	343000.0	1180591620717411303424.0
80	6.3	80.0	505.8	6400.0	512000.0	1208925819614629174706176.0
90	6.5	90.0	584.3	8100.0	729000.0	1237940039285380274899124224.0
100	6.6	100.0	664.4	10000.0	1000000.0	1267650600228229401496703205376.0

Рис. 4 Классы роста функций

Классификация скоростей роста алгоритма

Асимптотический метод анализа алгоритмов позволяет их классифицировать согласно скорости роста функции временной сложности, которая определяется старшим, доминирующим членом формулы. Отбросив все младшие члены, получаем порядок функции.

Алгоритмы классифицируют по скорости роста их сложности:

- Нотация Big Ω (Omega) - алгоритмы, сложность которых растет так же быстро, как данная функция;
- Нотация Big Θ (Theta) - алгоритмы, сложность которых растет с той же скоростью;
- Нотация Big O - алгоритмы, сложность которых растет медленнее, чем функция.

Big O нотация выражает верхний лимит количества операций, наибольшее возможное количество операций. Big Ω (Omega) нотация выражает нижний лимит количества операций, наименьшее возможное количество операций. Big Θ (Theta) нотация выражает нижний и верхний лимиты количества операций, наименьшее и наибольшее количество операций.

При анализе худшего случая, среднего и лучшего случая алгоритма можно использовать все три асимптотические нотации. Нотация Big O работает с худшими случаями, а Big Ω – с лучшими.

Решение вопроса о том, какие операции считать в процессе анализа алгоритма, состоит из двух шагов. На первом шаге выбирается значимая операция или группа операций, а на втором - какие из этих операций содержатся в теле алгоритма, а какие составляют накладные расходы или уходят на регистрацию и учет данных. В качестве значимых обычно выступают операции двух типов: сравнение и арифметические операции.

Все *операторы сравнения* считаются эквивалентными, и их учитывают в алгоритмах поиска и сортировки. Важным элементом таких алгоритмов является сравнение двух величин для определения — при поиске — того, совпадает ли данная величина с искомой, а при сортировке — вышла ли она за пределы данного интервала. Операторы сравнения проверяют, равна или не равна одна величина другой, меньше она или больше, меньше или равна, больше или равна.

Арифметические операции разбиваем на две группы: аддитивные и мультипликативные. Аддитивные операторы включают сложение, вычитание, увеличение и уменьшение счетчика. Мультипликативные операторы включают умножение, деление и взятие остатка по модулю. Разбиение на эти две группы связано с тем, что умножения работают дольше, чем сложения.

Наилучший, наихудший и средний случай поведения алгоритма

Роль входных данных в анализе алгоритмов чрезвычайно велика, поскольку последовательность действий алгоритма определяется не в последнюю очередь входными данными.

Например, для того, чтобы найти наибольший элемент в списке из N элементов, можно воспользоваться следующим алгоритмом:

```
maxElem = list [0];
for (i = 1; i< N ; i++) if (list [i] > maxElem) maxElem = list[i];
```

Если список упорядочен в порядке убывания, то перед началом цикла будет сделано одно присваивание, а в теле цикла присваиваний не будет. Если список упорядочен по возрастанию, то всего будет сделано N присваиваний (одно перед началом цикла и N - 1 в цикле).

При анализе мы должны рассмотреть различные возможные множества входных значений, поскольку, если мы ограничимся одним множеством, оно может оказаться тем самым, на котором решение самое быстрое (или самое медленное). В результате мы получим ложное представление об алгоритме.

Для корректного анализа алгоритмов будем рассматривать все типы входных множеств. Мы попытаемся разбить различные входные множества на классы в зависимости от поведения алгоритма на каждом

множестве. Такое разбиение позволяет уменьшить количество рассматриваемых возможностей.

Например, число различных расстановок 10 различных чисел в списке есть 10!= 3 628 800. Применим к списку из 10 чисел алгоритм поиска наибольшего элемента. Имеется 362 880 входных множеств, у которых первое число является наибольшим; их все можно поместить в один класс. Если наибольшее по величине число стоит на втором месте, то алгоритм сделает ровно два присваивания. Множеств, в которых наибольшее по величине число стоит на втором месте, 362 880. Их можно отнести к другому классу. Видно, как будет меняться число присваиваний при изменении положения наибольшего числа от 1 до *N*. Таким образом, нужно разбить все входные множества на *N* разных классов по числу сделанных присваиваний.

Нет необходимости выписывать или описывать детально все множества входных данных, помешенных в каждый класс. Нужно знать лишь количество классов и объем работы на каждом множестве класса. Число возможных наборов входных данных может стать очень большим при увеличении *N*.

Например, 10 различных чисел можно расположить в списке 3628 800 способами. Невозможно рассмотреть все эти способы. Вместо этого мы разбиваем списки на классы в зависимости от того, что будет делать алгоритм. Для вышеуказанного алгоритма разбиение основывается на местоположении наибольшего значения. В результате получается 10 разных классов. Для другого алгоритма, например, алгоритма поиска наибольшего и наименьшего значения, наше разбиение могло бы основываться на том, где располагаются наибольшее и наименьшее значения. В таком разбиении 90 классов.

Как только выделили классы, надо посмотреть на поведение алгоритма на одном множестве из каждого класса. Если классы выбраны правильно, то на всех множествах входных данных одного класса алгоритм производит одинаковое количество операций, а на множествах из другого класса это количество операций скорее всего будет другим.

При анализе алгоритма выбор входных данных может существенно повлиять на его выполнение. Скажем, некоторые алгоритмы сортировки могут работать очень быстро, если входной список уже частично отсортирован, тогда как другие алгоритмы покажут весьма скромный результат на таком списке. А вот на случайном списке результат может оказаться противоположным.

Нельзя ограничиваться анализом поведения алгоритмов на одном входном наборе данных. Практически будем искать такие данные, которые обеспечивают как самое быстрое, так и самое медленное выполнение алгоритма. Кроме того, будем оценивать и среднюю эффективность алгоритма на всех возможных наборах данных.

Наилучший случай. Наилучшим случаем для алгоритма является такой набор данных, на котором алгоритм выполняется за минимальное время или минимальное количество операций. Такой набор данных представляет собой комбинацию данных значений, на которой алгоритм выполняет меньше всего действий. Если мы исследуем алгоритм поиска, то набор данных является наилучшим, если искомое значение (обычно называемое целевым значением или ключом) записано в первой проверяемой алгоритмом ячейке. Такому алгоритму, вне зависимости от его сложности, потребуется одно сравнение. Заметим, что при поиске в списке, каким бы длинным он ни был, наилучший случай требует постоянного времени. Вообще, время выполнения алгоритма в наилучшем случае очень часто оказывается маленьким или просто постоянным, поэтому подобный анализ мало что показывает.

Наихудший случай. Анализ наихудшего случая чрезвычайно важен, поскольку он позволяет представить максимальное время работы алгоритма. При анализе наихудшего случая необходимо найти входные данные, на которых алгоритм будет выполнять больше всего работы. Для алгоритма поиска подобные входные данные — это список, в котором искомый ключ окажется последним из рассматриваемых или вообще отсутствует. В результате может потребоваться сравнений n, где n – размер массива входных данных. Анализ наихудшего случая дает верхние оценки для времени работы частей нашей программы в зависимости от выбранных алгоритмов.

Средний случай. Анализ среднего случая является самым сложным, поскольку он требует учета множества разнообразных деталей. В основе анализа лежит определение различных групп, на которые следует разбить возможные входные наборы данных. На втором шаге определяется вероятность, с которой входной набор данных принадлежит каждой группе. На третьем шаге подсчитывается время работы алгоритма на данных из каждой группы. Время работы алгоритма на всех входных данных одной группы должно быть одинаковым, в противном случае группу следует подразбить. Среднее время работы вычисляется по формуле

$$A(n) = \sum_{i=1}^{m} p_i t_i$$

где через n обозначен размер входных данных, через m — число групп, через p_i — вероятность того, что входные данные принадлежат группе с номером i, а через t_i — время, необходимое алгоритму для обработки данных из группы с номером i.

В некоторых случаях мы будем предполагать, что вероятности попадания входных данных в каждую из групп одинаковы. Другими словами, если групп пять, то вероятность попасть в первую группу такая

же, как вероятность попасть во вторую, и т.д., то есть вероятность попасть в каждую группу равна 0.2. В этом случае среднее время работы можно либо оценить по предыдущей формуле, либо воспользоваться эквивалентной ей упрощенной формулой, справедливой при равной вероятности всех групп.

$$A(n) = \frac{1}{m}\sum_{i=1}^{m} t_i$$

Пример.

Давайте попробуем проанализировать алгоритм сортировки методом вставки. Поскольку основным действием в реализации данного алгоритма является сравнение двух значений, наш подход будет состоять в подсчете количества таких сравнений, которые потребуется выполнить при сортировке списка длиной N элементов.

При сортировке методом вставки осуществляется выбор некоторого элемента, называемого опорным; после этого он сравнивается с предшествующими ему элементами, пока для него не будет найдено надлежащее место, после чего опорный элемент вставляется в соответствующую позицию. Алгоритм начинается с выбора второго элемента списка в качестве опорного. По мере его выполнения в качестве опорных выбираются последующие элементы — пока не будет достигнут конец списка. В самом лучшем случае каждый опорный элемент уже находится на положенном ему месте. Следовательно, чтобы это было обнаружено, его потребуется сравнить только с одним элементом. Поэтому в **наилучшем случае** применение алгоритма сортировки методом вставки к списку из N элементов потребует выполнения *N - 1* сравнений. (Второй элемент сравнивается с одним элементом (первым), третий элемент — с одним элементом (вторым) и т.д.)

И наоборот, **наихудший сценарий** имеет место в том случае, когда каждый опорный элемент потребуется сравнивать со всеми впереди стоящими элементами, прежде чем удастся найти правильное место его расположения (рис. 5). Очевидно, что в этом случае исходный список упорядочен в обратном порядке. Первый опорный элемент (второй элемент списка) сравнивается с одним элементом, второй опорный элемент (третий элемент списка) — с двумя элементами и т.д. (см. рис 1). Следовательно, общее количество сравнений при сортировке списка из N элементов составит *1 + 2 + 3 + ... + (N -1)*, что эквивалентно *N(N - 1)/2* или *(1/2)(N^2 - N)*. В частности, для списка из 10 элементов алгоритму сортировки методом вставки в наихудшем случае потребуется выполнить 45 сравнений.

Рис.5 Наихудший сценарий работы алгоритма сортировки методом вставки

В среднем при сортировке методом вставки можно ожидать, что каждый опорный элемент потребуется сравнить с половиной предшествующих ему элементов. В этом случае общее количество выполненных сравнений будет вдвое меньше, чем в наихудшем случае, т.е. $(1/4)\ (N^2 - N)$ сравнений для списка длины N.

Например, если использовать сортировку методом вставки для упорядочения множества списков из 10 элементов, то среднее число производимых в каждом случае сравнений будет равно 22,5. Важность полученного выше результата состоит в том, что количество сравнений, выполненных алгоритмом сортировки методом вставки, позволяет оценить время, которое потребуется для выполнения сортировки. Эта оценка была использована для построения графика, представленного на рис. 6. Он показывает, как будет возрастать время, необходимое для выполнения сортировки методом вставки, при увеличении длины сортируемого списка. Данный график построен по оценкам работы алгоритма в наихудшем случае, когда, исходя из результатов наших исследований, для списка длиной N требуется выполнить не менее $(1/2)(N^2 - N)$ сравнений элементов. На графике отмечено несколько конкретных значений длины списка и указано время, необходимое в каждом случае. Обратите внимание, при увеличении длины списка на одно и то же количество элементов время, необходимое для сортировки списка, все больше и больше возрастает. Таким образом, с увеличением длины списка эффективность данного алгоритма уменьшается.

Выводы. Анализ алгоритмов заботится только о том, во сколько раз чаще выполняется операция по мере увеличения входных данных, что соответствует поведению **в худшем случае**. Если alg1 превосходит alg2 для очень большого входящего n, очевидно, что он будет делать это и при малом n.

Рис.6 График скорости роста сложности алгоритма сортировки методом вставки

Анализ методов сортировки массивов

Существует много методов сортировки массивов. Для того чтобы оценить насколько один метод сортировки лучше другого, необходимо анализировать эффективность метода сортировки. Естественно отличать методы сортировки *по времени*, затрачиваемому на выполнение сортировки.

Для сортировок основными считаются две операции: операция сравнения элементов и операция обмена элемента. Будем считать, что в единицу времени выполняется одна операция сравнения или обмена. Таким образом, временная сложность или трудоемкость метода имеет две основные характеристики f и g, где f – количество операций обмена, g – количество операций сравнения. f(n) и g(n) – зависят от количества элементов в массиве, то есть являются функциями от n- размера массива.

В анализе будем рассматривать асимптотическое поведение функций сложности f(n) и g(n) в зависимости от числа элементов в массиве n, при $n \to \infty$.

Поскольку для различных массивов один и тот же метод сортировки может иметь различную эффективность, то необходимо знать в каких пределах могут изменяться величины характеризующие эффективность метода, то есть определить порядок функций сложности f(n), g(n) и массивы, на которых достигаются эти значения, а также средние значения величин этих функций.

3. СТРУКТУРА ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ

Структура записки задает последовательность этапов выполнения курсовой работы и содержит следующие пункты:

1. титульный лист,

2. цель работы,

3. общее задание (пункты 1 - 5) и вариант задания,

4. описание и схема алгоритмов сортировки и ссылка на Приложение 1 с исходным текстом функций сортировки,

5. описание массива, способы формирования исходных данных для наилучшего, наихудшего и среднего случая со ссылкой на Приложение 2,

6. определение переменных программы (показателей эффективности) и описание методики тестирования эффективности алгоритмов сортировки со ссылкой на Приложение 3 и Приложение 1,

7. таблицы и графики с полученными оценками показателей эффективности алгоритмов операций для наилучшего, наихудшего и среднего случаев функционирования. Должны быть приведены следующие графики:

a. число обменов и число сравнений для лучшего, худшего и среднего случаев для *первого* алгоритма сортировки (графики совмещены в одной системе координат) при разных размерах исходных данных,

b. число обменов и число сравнений для лучшего, худшего и среднего случаев для *второго* алгоритма сортировки (графики совмещены) при разных размерах исходных данных,

c. временной график, где показана зависимость времени выполнения от размера массива для обоих методов сортировки,

d. число сравнений алгоритмов методов сортировки *для среднего случая* при разных размерах исходных данных,

e. число обменов алгоритмов методов сортировки *для среднего случая* при разных размерах исходных данных.

8. сравнительный анализ теоретических и экспериментальных оценок эффективности алгоритмов сортировок, который отвечает на вопрос: соответствует ли функция временной сложности алгоритма сортировки экспериментально полученным значениям числа операций при заданном n (количестве входных данных) соответственно для лучшего, худшего и среднего случая;

9. выводы;

10. список использованной литературы;

11. приложение с текстами программ:

- Приложение 1. Исходный текст алгоритмов сортировок программы.
- Приложение 2. Исходный текст функций формирования исходных наборов данных.
- Приложение 3. Текст программы тестирования эффективности сортировок.

4. **ТРЕБОВАНИЯ К ПРОГРАММЕ**

Программа должна решать поставленную задачу согласно заданию по варианту. Программу необходимо составить в общем виде так, чтобы

она могла быть использована для обработки массивов различных размеров в пределах заданных ограничений на размеры массивов или использовать динамические массивы. Набор дополнительных функций программы необходимо определить самостоятельно.

В программе предусмотреть ввод и вывод исходных данных и полученных результатов с необходимыми поясняющими текстами. Предусмотреть контроль ввода. Подготовить тесты и контрольные примеры. В программе рекомендуется предусмотреть возможность сохранения исходных данных и результатов в файл и чтение из файла. Это может быть как запись в файл по команде пользователя, так и автоматическое сохранение/чтение данных при завершении/запуске программы.

Во время защиты курсовой работы выполнение программы должно быть продемонстрировано на тестовых данных, подготовленных в файлах.

Программа должна быть разработана в языке программирования C++.

5. РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ РАЗДЕЛОВ КУРСОВОЙ РАБОТЫ

Для решения необходимо изучить по варианту задания методы сортировок. Это этап работы с источниками, он должен закончиться пониманием и описанием методов сортировки на примере контрольного набора данных.

Разрабатывается графическое представление (блок-схема) алгоритмов сортировки и подпрограммы реализации алгоритмов в виде функций. Тестируются на контрольном примере.

При разработке общей программы тестирования для больших объемов данных, необходимо предусмотреть функциональную декомпозицию и модульный состав программы, включая функции загрузки тестовых значений из файла, функции интерфейса пользователя, другие вспомогательные функции и модули.

При написании программы не следует забывать о хорошем стиле программирования, о таких понятиях, как читабельность, эффективность, надежность. Необходимо искать наиболее простые и естественные приемы программирования решения.

6. ТРЕБОВАНИЯ К ОФОРМЛЕНИЮ ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ

Текст записки должен быть напечатан на одной стороне стандартного листа белой бумаги формата 210 х 297 мм (А4) в редакторе MS Word. Шрифт – Times New Roman, размер – 14 pt, интервал одинарный, выравнивание по ширине. Размеры полей: слева и справа – 2,5

см, сверху – 2 см, снизу – 3 см. Абзацный отступ – 1,25 см. Нумерация страниц – внизу посередине листа. На титульном листе номер не ставится.

Заголовки оформляются заглавными буквами шрифтом размера 14 pt, начертание – полужирное. Точка в конце заголовка не ставится. Тексты программ необходимо набирать шрифтом Courier New 14 pt прямого начертания.

Таблицы описывают как данном документе методических указаний, либо как представлено в Приложениях. У таблиц обязательно должно быть название и номер – сквозной в пределах пояснительной записки.

Например, «Таблица 1» или «Таблица 2 – Модульный состав программы»

Иллюстрации должны сопровождаться подписями. Подпись к иллюстрациям оформляют обычным шрифтом, размер – 13 pt. Нумерация рисунков – сквозная в пределах всего документа. В конце подписи точка не ставится. Например: «Рисунок 1. Главное окно программы». В тексте должны быть ссылки на все рисунки. Например: «Программа имеет графический интерфейс (рис. 1)».

Образец титульного листа пояснительной записки представлен в Приложении 1.

7. ПОРЯДОК ЗАЩИТЫ РАБОТЫ

Для защиты курсовой работы студентом должны быть представлены преподавателю:

1) программа в виде исходного кода и исполняемого файла, архив которых загружается в соответствующий раздел среды EDUCON;

2) Пояснительная записка электронном и печатном виде. Электронный вариант должен быть загружен в систему EDUCON предварительно.

Защита включает в себя:

1) демонстрацию программы на тестах/контрольном примере и исходного кода;

2) ознакомление преподавателя с Пояснительной запиской;

3) ответы на вопросы преподавателя (например, «почему было реализовано именно таким образом», «имело ли смысл предусмотреть в программе такие-то функции» и т.п.).

При выставлении баллов за курсовую работу оценивается программа (до 50 баллов), Пояснительная записка (до 30 баллов) и защита студентом работы (до 20 баллов).

Библиографический список литературы

1. Ахо Альфред В. Структуры данных и алгоритмы: классическое издание. / Ахо Альфред В., Хопкрофт Джон Э., Ульман Джеффри Д. – Москва: Вильямс, 2018 – 400с. – Текст: непосредственный.
2. Вирт Н. Алгоритмы и структуры данных / Вирт Н. – Саратов: Профобразование, 2019. – 272 с. // ЭБС «IPRbooks» [сайт]. – URL: http://www.iprbookshop.ru/88753.html (дата обращения: 22.11.2021) – Текст: электронный.
3. Кормен Томас Х. Алгоритмы. Вводный курс. / Кормен Томас Х. – Москва: Вильямс, 2020 – 208с. – Текст : непосредственный.
4. Алгоритмы. Построение и анализ. / Кормен Томас Х., Лейзерсон Чарльз И., Ривест Рональд Л., Штайн Клиффорд. – Москва: Вильямс, 2019 – 1328с.
5. Лэнгсам Й. Структуры данных для персональных ЭВМ / Й. Лэнгсам, М. Огенстайн, А. Тененбаум – Москва: Мир, 1989.– 568 с. – Текст: непосредственный.
6. Круз Л., Структуры данных и проектирование программ / Роберт Круз Л. ; перевод К. Г. Финогенова. – 3-е изд. – Москва : Лаборатория знаний, 2017. – 766 с. // ЭБС «IPRbooks» [сайт]. – URL: http://www.iprbookshop.ru/89031.html (дата обращения: 22.11.2021) – Текст: электронный
7. Селиванова И.А. Построение и анализ алгоритмов обработки данных [Электронный ресурс]: учебно-методическое пособие/ Селиванова И.А., Блинов В.А.– Екатеринбург: Уральский федеральный университет, ЭБС АСВ, 2015.– 108 с. // ЭБС «IPRbooks» [сайт]. – URL: http://www.iprbookshop.ru/68277.html (дата обращения: 22.11.2021) – Текст: электронный.
8. Чурина, Т. Г. Методы программирования: алгоритмы и структуры данных. Ч.3. Динамические структуры данных, алгоритмы на графах: учебное пособие / Т. Г. Чурина, Т. В. Нестеренко. – Новосибирск: Новосибирский государственный университет, 2014. – 215 с. // ЭБС «IPRbooks» [сайт]. – URL: http://www.iprbookshop.ru/93563.html (дата обращения: 22.11.2021) – Текст: электронный.
9. Каррано Ф.М. Абстракция данных и решение задач на C++. Стены и зеркала / Ф.М. Каррано, Дж.Дж. Причард – 3– е издание, пер. с англ. – Москва: Издательский дом "Вильямс", 2003. – 848 с. – Текст: непосредственный.
10. Штерн В. Основы C++: Методы программной инженерии / В. Штерн – Москва: Издательство "Лори", 2003. – 860 с. – Текст: непосредственный.

Приложение 1 Образец оформления титульного листа

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«ТЮМЕНСКИЙ ИНДУСТРИАЛЬНЫЙ УНИВЕРСИТЕТ»

<Институт>

<Кафедра>

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по дисциплине «Алгоритмы и структуры данных»

на тему: Исследование эффективности методов сортировок массива:

<названия методов>

Вариант <номер>

Выполнил:

студент группы <номер группы>
<Фамилия><Имя>

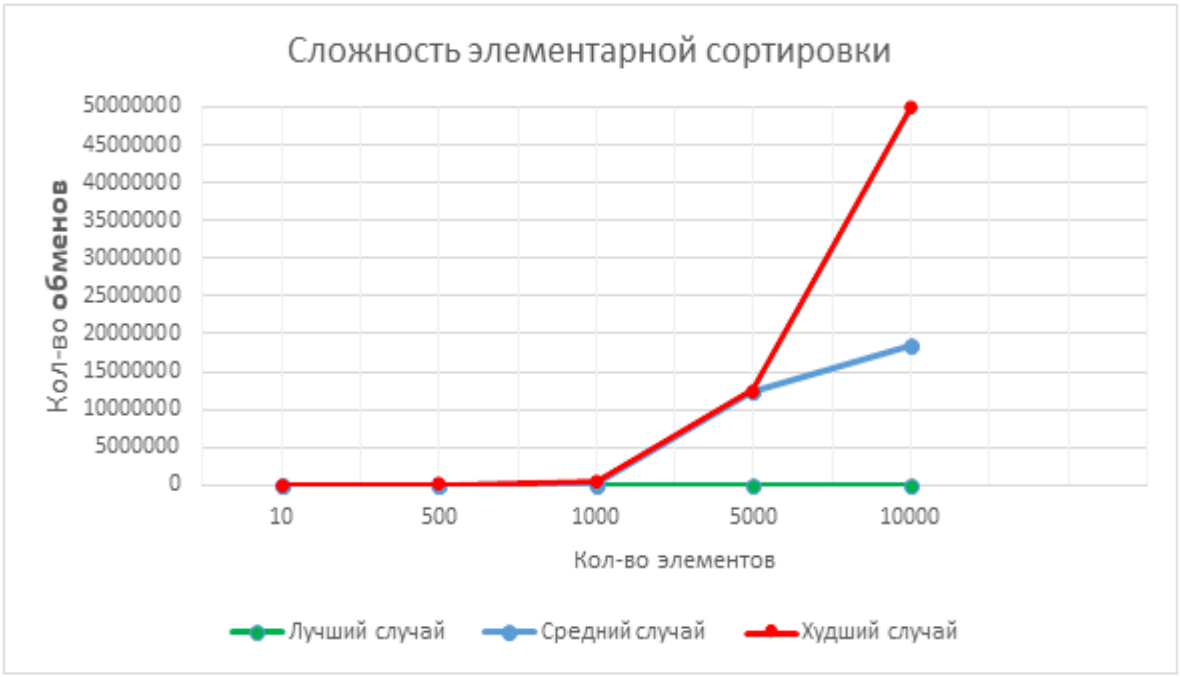
Проверил:

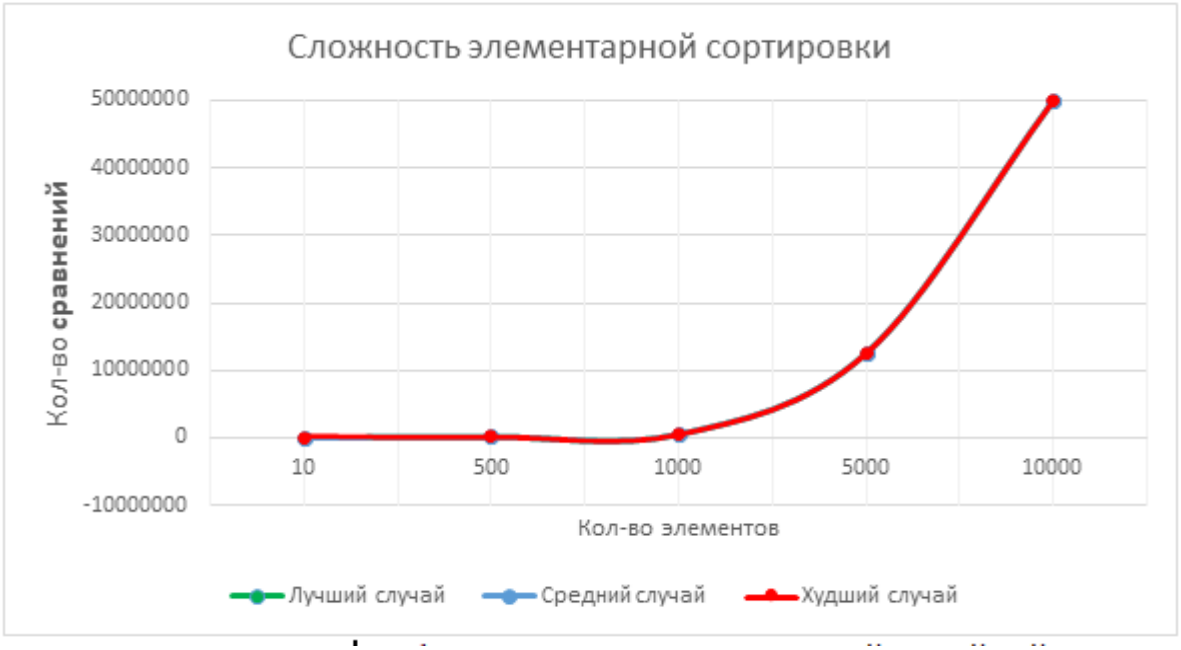
Преподаватель <ФИО преподавателя>

Тюмень
ТИУ
<год>

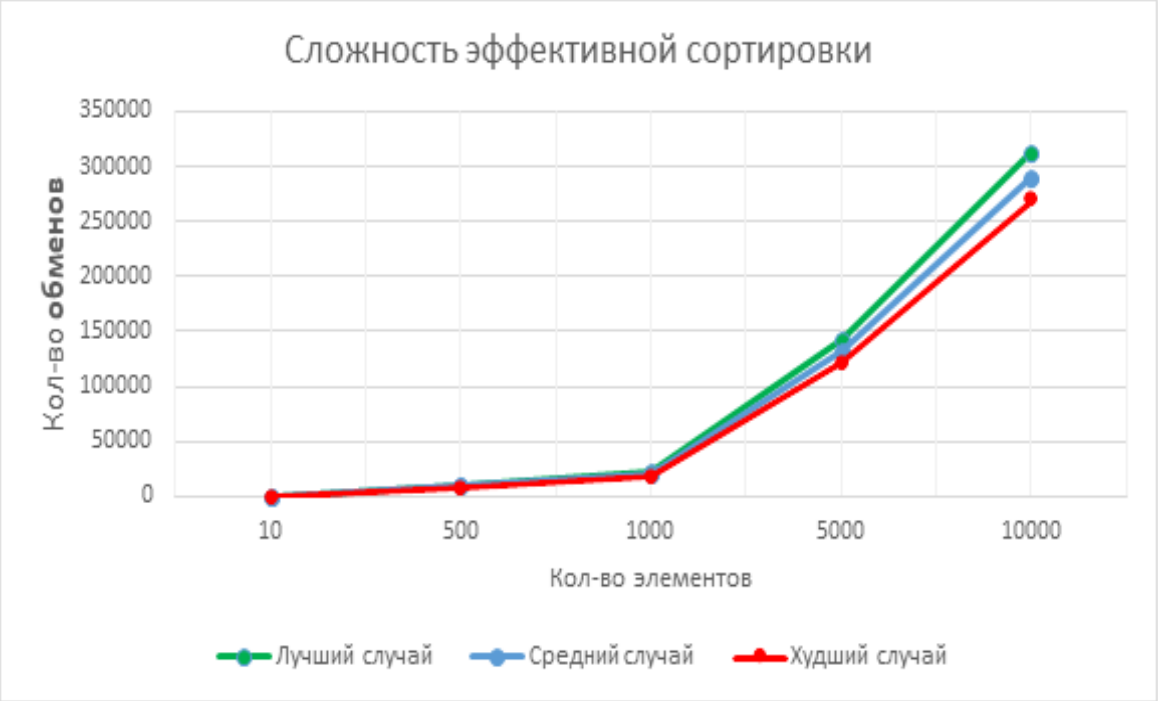
Приложение 2 Примеры графиков с полученными оценками показателей эффективности алгоритмов

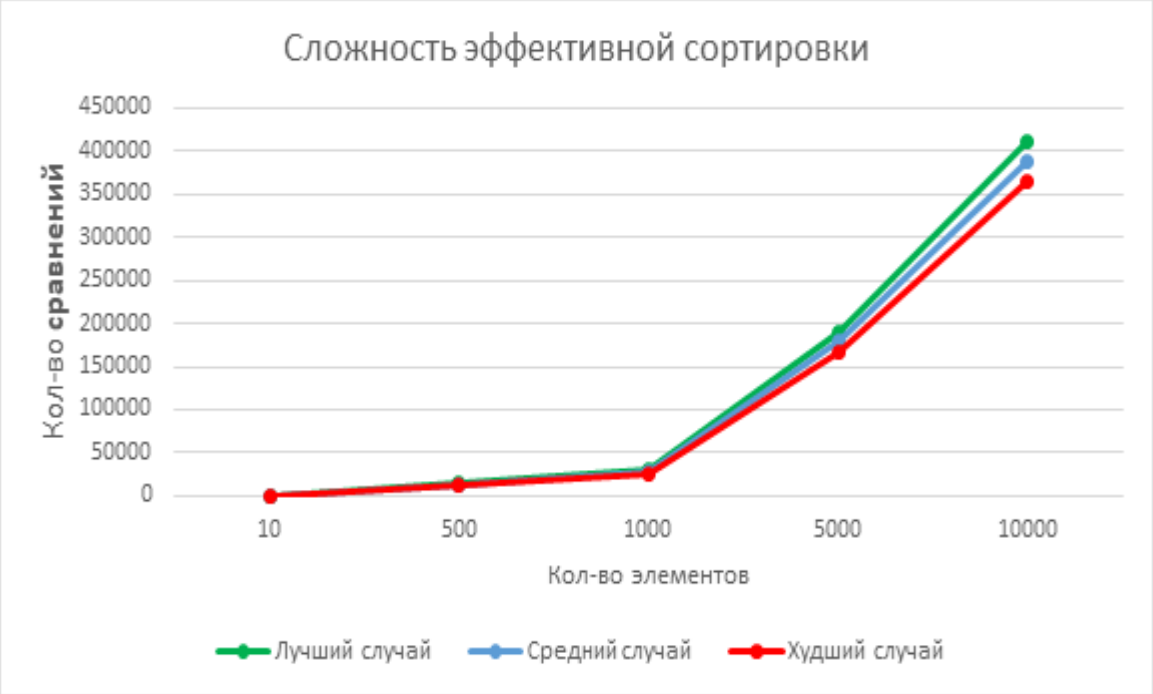
а) График числа обменов и числа сравнений для элементарного алгоритма



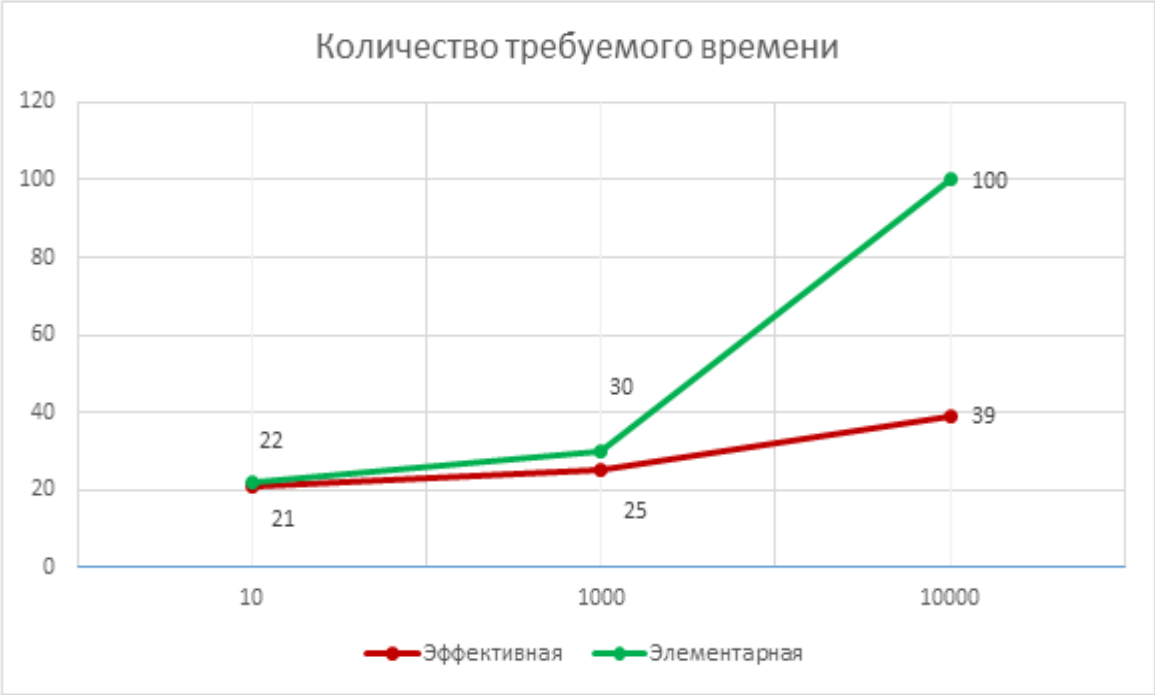


b) График числа обменов и числа сравнений для эффективного алгоритма





c) Временной график, зависимость времени работы алгоритма от количества элементов в массиве



d) График числа сравнений алгоритмов элементарной и эффективной сортировки для среднего случая

е) График числа обменов алгоритмов элементарной и эффективной сортировки для среднего случая

Приложение 3Программирование определение времени выполнения сортировки

Определить количество секунд, которое выполняется программа можно с помощью функции clock()

```
#include <stdio.h>
#include <time.h>
int main() {
clock_t   start1 = clock();

// вызов функции сортировки массива. Например

 bubbleSort(A, n, k, o);

clock_t end1 = clock();

cout<<"The time: %f seconds\n"<<(float)(end1 - start1)/ CLOCKS_PER_SEC;

}
```

Текущее системное время сохраняется в переменной с типом данных clock_t — это целое число секунд. При выводе вычисляем выражение разницы двух моментов времени (start1, end1).

Таким набором операторов C++ вы сможете замерить время работы части программы, результат будет в секундах.

Учебное издание

АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

Методические указания по выполнению курсовой работы

Составитель

Лозикова Инна Олеговна

В авторской редакции

Подписано в печать 10.11.2021. Формат 60/90 1/16. Печ. л. 2.

Тираж 30 экз. Заказ №.

Библиотечно-издательский комплекс
федерального государственного бюджетного образовательного
учреждения высшего образования
«Тюменский индустриальный университет».
625000, Тюмень, ул. Володарского, 38.

Типография библиотечно-издательского комплекса.
625039, Тюмень, ул. Киевская, 52.

28