Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе № 3

по курсу «Анализ алгоритмов»

на тему: «Алгоритмы сортировок»

Студент ИУ7-54Б (Группа) _____ (Подпись, дата) Разин А. (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) Волкова Л. Л. (И. О. Фамилия)

2023 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Упорядочение данных чрезвычайно важно в программировании. Практически сортировка и поиск в той или иной мере присутствуют во всех приложениях; в частности, при обработке больших объемов данных эффективность именно этих операций определяет эффективность, а иногда и работоспособность всей системы. Под сортировкой понимается упорядочивание элементов последовательности по какому-либо признаку [1]. Можно сказать, что достаточно четкие представления об этой области нужны при решении любой задачи на ЭВМ как обязательные элементы искусства программирования [2].

Целью данной лабораторной работы является описание и исследование трудоемкости алгоритмов сортировки.

Для поставленной цели необходимо выполнить следующие задачи.

1) Описать алгоритмы сортировки;
2) создать программное обеспечение, реализующее следующие алгоритмы сортировки;
   - перемешиванием;
   - блочная;
   - поразрядная;
3) оценить трудоемкости сортировок;
4) замерить время реализации;
5) провести анализ затрат работы программы по времени, выяснить влияющие на них характеристики;

1 Аналитическая часть

В данной части работы будут рассмотрены алгоритмы сортировок: перемешиванием, блочная и поразрядная. Также будет определена решаемая задача.

1.1 Сортировка перемешиванием

Алгоритм состоит из последовательных про проходов по массиву в 2 направлениях (от начала массива к его концу и наоборот). При каждом проходе происходит попарное сравнение ближайших значений, в случае если их порядок противоречит возрастающему, значения меняются местами и позиция, после которой произошло изменение порядка запоминается в переменной $left$ в случае прохода от начала до конца массива и в переменной $right$ иначе. Все следующие проходы по массиву происходят от позиции $left$ до $right$ пока данные позиции не равны. Таким образом при каждом проходе устанавливается наибольшее и наименьшее значения в индексах массива от $left$ до $right$ [3; 4].

1.2 Поразрядная сортировка

Смысл данной сортировки в том, что данные делятся сначала по разрядам и сортируются внутри каждого разряда. Сам алгоритм происходит в несколько этапов.

1) Алгоритм инициализирует индекс рассматриваемого разряда в числах;
2) после чего получает значение данного разряда каждого из чисел с помощью остатка деления на основание системы счисления;
3) затем полученные цифры сортируются;
4) элементы расставляются в соответствии со своими цифрами.

Данный алгоритм повторяется пока индекс рассматриваемого разряда не будет больше числа всех разрядов в числе [2; 4].

1.3 Блочная сортировка

Идея заключается в разбиении входных данных на «блоки» одинакового размера, после чего данные в блоках сортируются и результаты сортировок объединяются. Отсортированная последовательность получается путём последовательного перечисления элементов каждого блока. Для деления данных на блоки, алгоритм предполагает, что значения распределены равномерно, и распределяет элементы по блокам равномерно. Например, предположим, что данные имеют значения в диапазоне от 1 до 100 и алгоритм использует 10 блоков. Алгоритм помещает элементы со значениями 1-10 в первый блок, со значениями 11-20 — во второй, и т.д. Если элементы распределены равномерно, в каждый блок попадает примерно одинаковое число элементов. Если в списке N элементов, и алгоритм использует N блоков, в каждый блок попадает всего один или два элемента, поэтому возможно отсортировать элементы за конечное число шагов [4].

Вывод

В данной части были рассмотрены идеи поразрядной, блочной сортировки и сортировки перемешиванием.

2 Конструкторская часть

В данной части работы будут рассмотрены схемы алгоритмов сортировок, а также приведен расчет их трудоемкости.

2.1 Схемы алгоритмов

2.2 Оценка трудоемкости реализаций алгоритмов

Для последующего вычисления трудоемкости необходимо ввести модель вычислений.

1) Трудоемкость следующих базовых операций единична: +, -, =, +=, -=, ==, !=, <, >, <=, >=, [], ++, –, «, ».

   Операции *, %, / имеют трудоемкость 2;

2) трудоемкость цикла for(k = 0; k < N; k++) {тело цикла} рассчитывается как:

$$f_{for} = f_{\text{инициал.}} + f_{\text{сравн.}} + N(f_{\text{тела}} + f_{\text{инкр.}} + f_{\text{сравн.}}); \tag{2.1}$$

3) трудоемкость передачи параметра в функции и возврат из функции равны 0;

4) трудоемкость условного оператора if (условие) then A else B рассчитывается как:

$$f_{if} = f_{\text{условия}} + \begin{cases} min(f_A, f_B), & \text{л.с.} \\ max(f_A, f_B), & \text{х.с.} \end{cases} \tag{2.2}$$

В (2.2), л.с. обозначает лучший случай, х.с. - худший случай.

В приведенных сортировках n обозначает число элементов в последовательности.

2.2.1 Трудоемкость реализации сортировки перемешиванием

Лучший случай: последовательность уже отсортирована, не одного захода в тело условного оператора приведен в формуле (2.3).

В данном случаях считается, что трудоемкость обмена элементов местами равна 5.

$$f_{best} = \sum_{i=0}^{n}(3+1+1+i\cdot(1+4+1)+1+1+1+1) = \\ = \sum_{i=0}^{n}(9+6\cdot i) = \\ = 15\cdot n + \frac{n^2}{2} - \frac{n}{2} = O(n^2) \tag{2.3}$$

Худший случай: последовательность отсортирована в обратном порядке (необходимо каждый раз заходить в тело условного оператора), приведен в формуле (2.4).

$$f_{worst} = \sum_{i=0}^{n}(3+1+1+i\cdot(1+4+1+5)+1+1+1+1 = \\ = \sum_{i=0}^{n} 9+11\cdot i = \\ = 20\cdot n + \frac{n^2}{2} - \frac{n}{2} = O(n^2) \tag{2.4}$$

2.2.2 Трудоемкость реализации блочной сортировки

В данной реализации размер блока обозначается как k, трудоемкость операции добавления и удаления элемента из вектора равна 2.

Лучший случай: массив отсортирован, элементы распределены равномерно (все блоки содержат одинаковое число элементов), расчет трудоемкости данного случая приведен в (2.5).

$$f_{best} = 1+1+\frac{n}{k}\cdot(1+2+f_{shaker}+2+1+4+ \\ k\cdot(3+1+4))+1+1+ \\ \frac{n}{k}\cdot(1+4+1+1+5+1+4+1+1+n\cdot(5)) = \\ = 4+\frac{29\cdot n + n\cdot f_{shaker} + 5\cdot n^2}{k} + 8\cdot n = \\ = 4+8\cdot n+29\cdot\frac{n}{k}+n\cdot(14.5+\frac{k}{2})+\frac{5\cdot n^2}{k} \tag{2.5}$$

Худший случай: большое количество пустых блоков, массив отсортиро-

ван в обратном порядке (худший случай сортировки перемешиванием, которая используется в блочной сортировке), расчет трудоемкости приведен в выражении 2.6.

$$\begin{gathered} f_{worst} = 1 + 1 + \frac{n}{k} \cdot (1 + 2 + f_{shaker} + 2 + 1 + 4 + \\ k \cdot (3 + 1 + 4)) + 1 + 1 + \\ \frac{n}{k} \cdot (1 + 4 + 1 + 1 + 5 + 1 + 4 + 1 + 1 + k \cdot (6)) = \\ = 4 + \frac{29 \cdot n + n \cdot f_{shaker} + 6 \cdot n^2}{k} + 8 \cdot n = \\ = 4 + 8 \cdot n + 29 \cdot \frac{n}{k} + n \cdot (19.5 + \frac{k}{2}) + \frac{6 \cdot n^2}{k} \end{gathered} \tag{2.6}$$

2.2.3 Трудоемкость реализации сортировки разрядов подсчетом

$$\begin{gathered} f_{count} = 1 + 1 + 1 + n \cdot (2 + 7) + 1 + 1 + 20 \cdot (1 + 1 + 4) + \\ +2 + 1 + n \cdot (1 + 1 + 10 + 7) + 1 + 1 + n \cdot (1 + 1 + 3) = \\ = 130 + 33 \cdot n \end{gathered} \tag{2.7}$$

Данная сортировка используется при реализации поразрядной сортировки, и не содержит условных операторов, соответственно не имеет лучшего и худшего случая

2.2.4 Трудоемкость реализации поразрядной сортировки

В данном случае за w принимается количество разрядов максимально большого по модулю числа. Лучший случай: наибольшее по модулю значение находится в начале массива, расчет трудоемкости данного случая приведен в (2.8).

$$\begin{gathered} f_{best} = 1 + 1 + 1 + n \cdot (2 + 2) + 1 + 3 + w \cdot (2 + 3 + f_{count}) = \\ = 7 + 4 \cdot n + 135 \cdot w + 33 \cdot n \cdot w = O(n \cdot w) \end{gathered} \tag{2.8}$$

Худший

растаю модуля значений, расчет трудоемкости приведен в выражении 2.9.

$$f_{best} = 1 + 1 + 1 + n \cdot (2 + 2 + 2) + 1 + 3 + w \cdot (2 + 3 + f_{count}) = \\ = 7 + 6 \cdot n + 135 \cdot w + 33 \cdot n \cdot w = O(n \cdot w) \tag{2.9}$$

Вывод

В данном разделе на основе теоретических данных были построены схемы требуемых алгоритмов, а также для каждого алгоритма сортировки были выведены трудоемкости худшего и лучшего случаев.

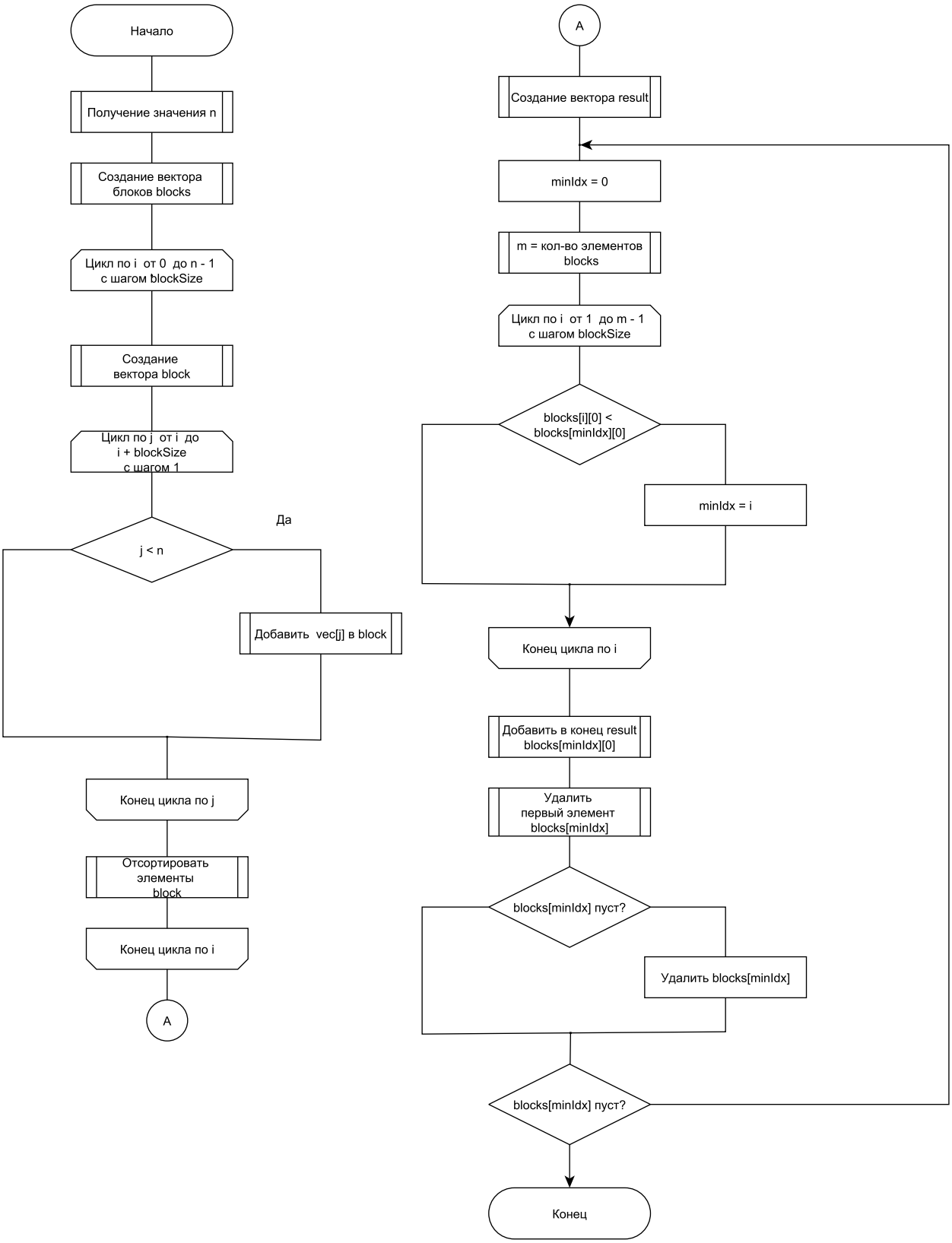

Рисунок 2.1 – Схема алгоритма сортировки перемешиванием

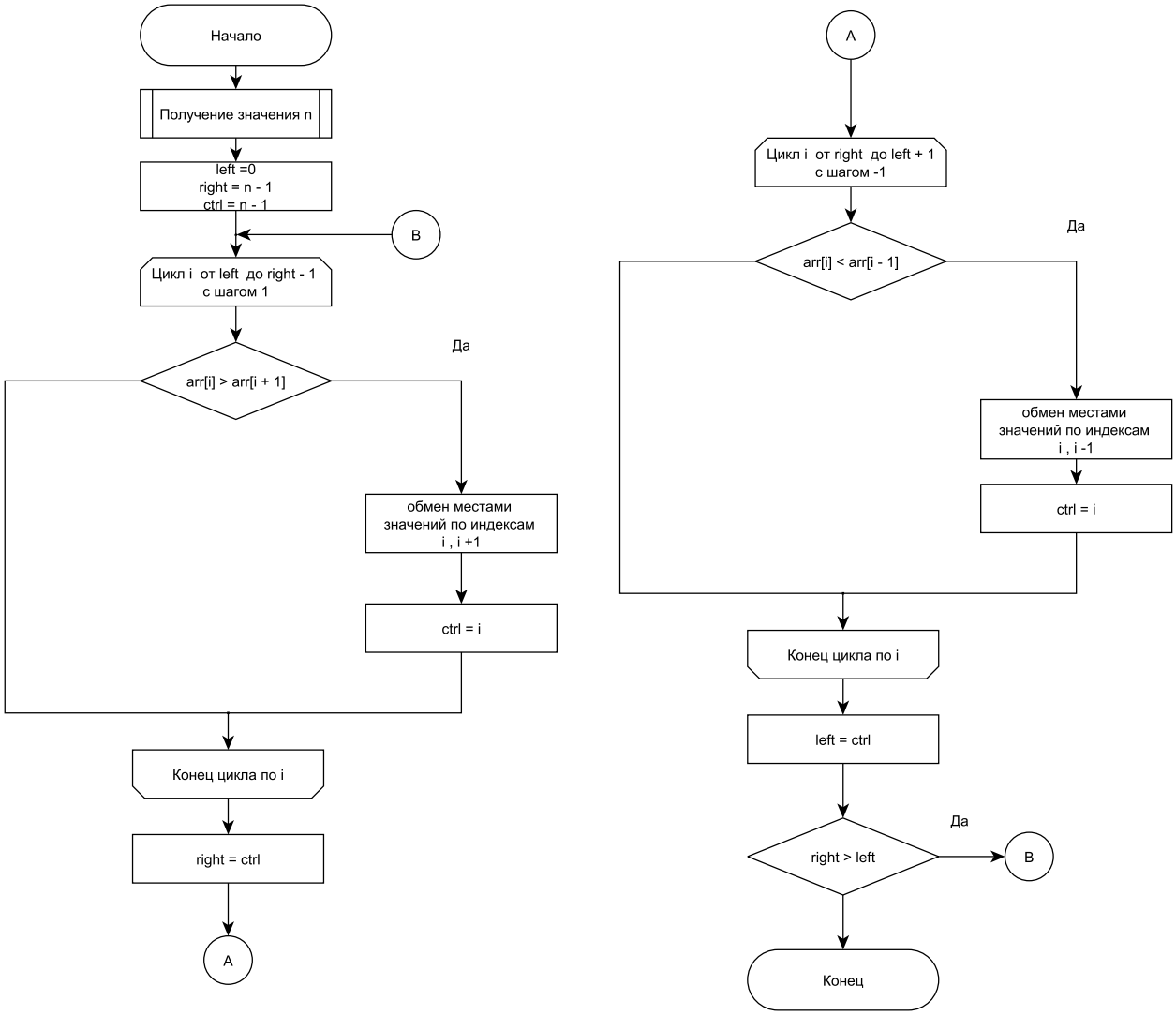

Рисунок 2.2 – Схема алгоритма блочной сортировки

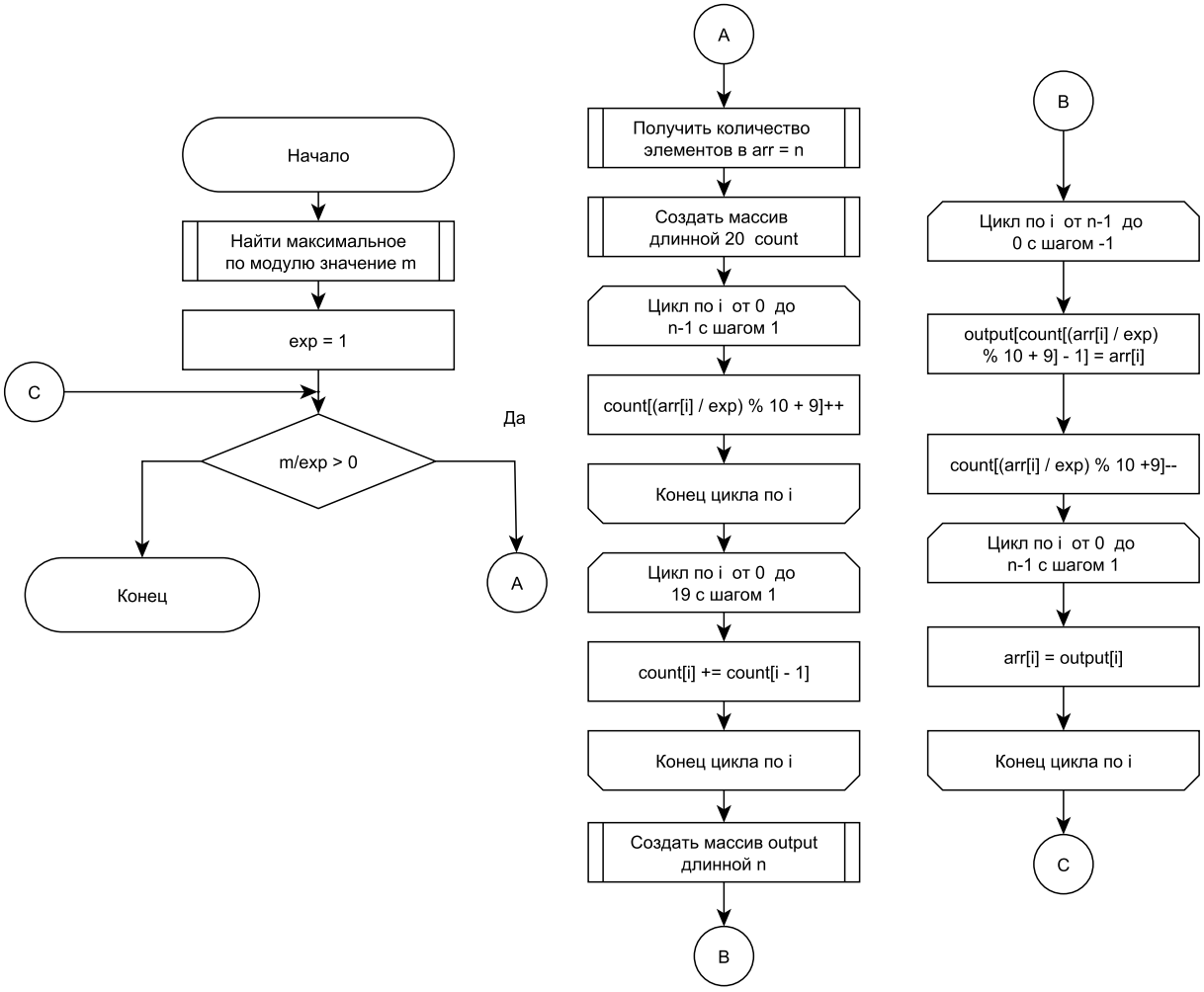

Рисунок 2.3 – Схема алгоритма поразрядной сортировки

3 Технологический раздел

В данной части работы будут описаны средства реализации программы, а также листинги, модульные и функциональные тесты.

3.1 Средства реализации

Алгоритмы для данной лабораторной работы были реализованы на языке C++, при использовании компилятора gcc версии 10.5.0, так как в стандартной библиотеке приведенного языка присутствует функция clock_gettime, которая (при использовании макропеременной CLOCK_THREAD_CPUTIME_ID) позволяет рассчитать процессорное время конкретного потока [5].

3.2 Реализация алгоритмов

Листинги исходных кодов программ А.1–А.5 приведены в приложении.

3.3 Тестирование

В таблице 3.1 приведены функциональные тесты для разработанных алгоритмов сортировки. Все тесты пройдены успешно.

Таблица 3.1 – Модульные тесты

Массив	Размер	Ожидаемый р-т	Фактический результат	
			Блочная/Перемеш.	Поразрядная
1 2 3 4	4	1 2 3 4	1 2 3 4	1 2 3 4
4 3 2 1	4	4 3 2 1	4 3 2 1	4 3 2 1
3 5 1 6	4	1 3 5 6	1 3 5 6	1 3 5 6
-5 -1 -3 -4 -2	5	-5 -4 -3 -2 -1	-5 -4 -3 -2 -1	-5 -4 -3 -2 -1
1 -3 2 9 -9	5	-9 -3 1 2 9	-9 -3 1 2 91	-9 -3 1 2 9

Вывод Были представлены листинги реализованных алгоритмов и тесты, успешно пройденные программой.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программ, постановка эксперимента и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени, представлены далее.

1) Процессор Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, 2592 МГц, ядер: 6, логических процессоров: 12;
2) Оперативная память: 16 ГБайт;
3) Операционная система: Майкрософт Windows 10 Pro [6];
4) Использованная подсистема: WSL2 [7].

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями.

4.2 Демонстрация работы программы

На рисунке 4.1 представлена демонстрация работы разработанного приложения для алгоритмов сортировок

```
Меню:
0)Выход
1)Сортировка массива с использованием сортировки перемешивания
2)Сортировка массива с использованием поразрядной сортировки
3)Сортировка массива с использованием блочной сортировки
4)Расчет времени

Введите пункт из меню: 1

Введите  размер массива для сортировки:
4
Введите массив для сортировки:
4 2 1 3
Отсортированный массив
1 2 3 4
```

Рисунок 4.1 – Пример работы программы

4.3 Временные характеристики

Результаты исследования замеров по времени приведены в таблице 4.1.

Таблица 4.1 – Полученная таблица замеров по времени различных реализаций алгоритмов сортировки

n	Поразрядная(мс)	Блочная(мс)	Перемешиванием(мс)
1000	0.15692	1.2194	2.03
2000	0.31811	4.5549	8.2541
3000	0.47956	10.05	18.767
4000	0.62436	17.109	32.384
5000	0.78017	26.401	50.514
6000	0.93926	37.91	72.854
7000	1.0944	51.295	99.256
8000	1.2494	66.616	129.05
9000	1.6587	97.898	191.14
10000	1.5698	104.19	203.08

Для таблицы 4.1 расчеты проводились с шагом 1000, сортировки производились 1000 раз, после чего результат усредняется. В качестве сортируемых значений использовались числа от 1 до 10000000. Размерность блоков определялась как $\frac{n}{2} + 2$. Данные генерировались из равномерного распределения.

По таблице 4.1 были построены графики:

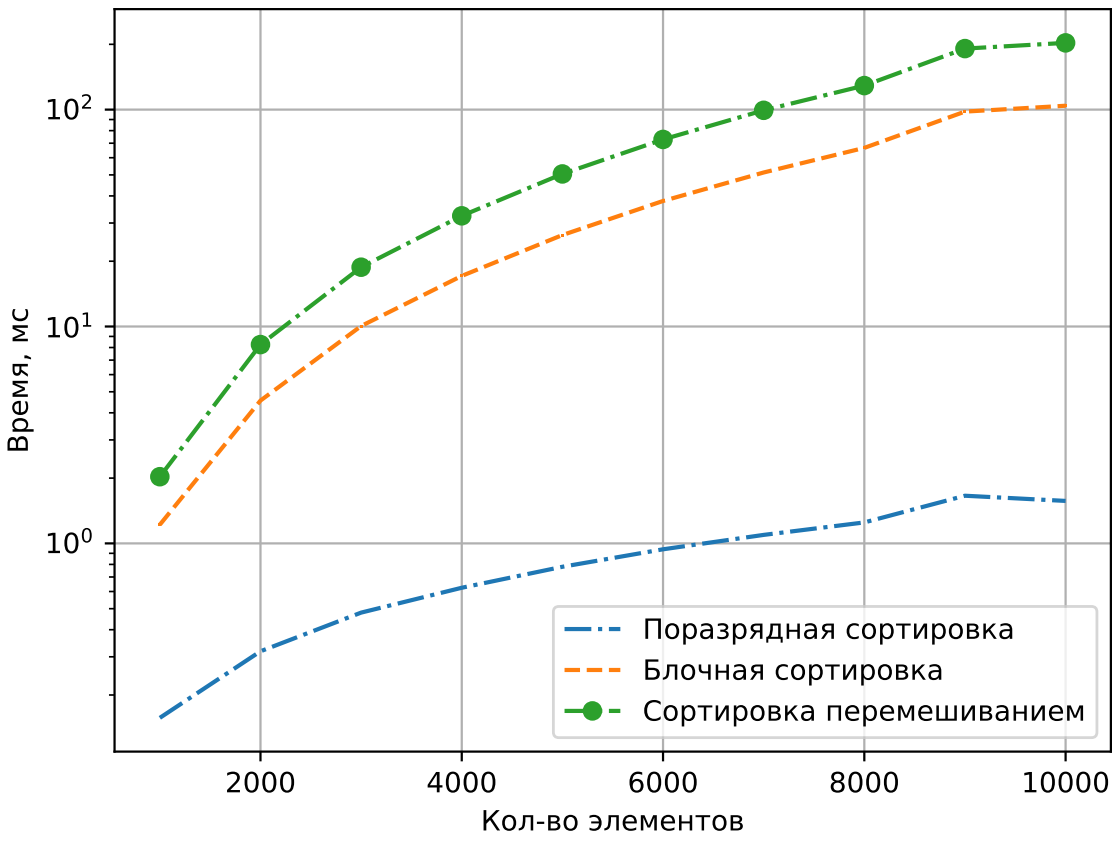

Рисунок 4.2 – Сравнение реализаций сортировок по времени с использованием логарифмической шкалы

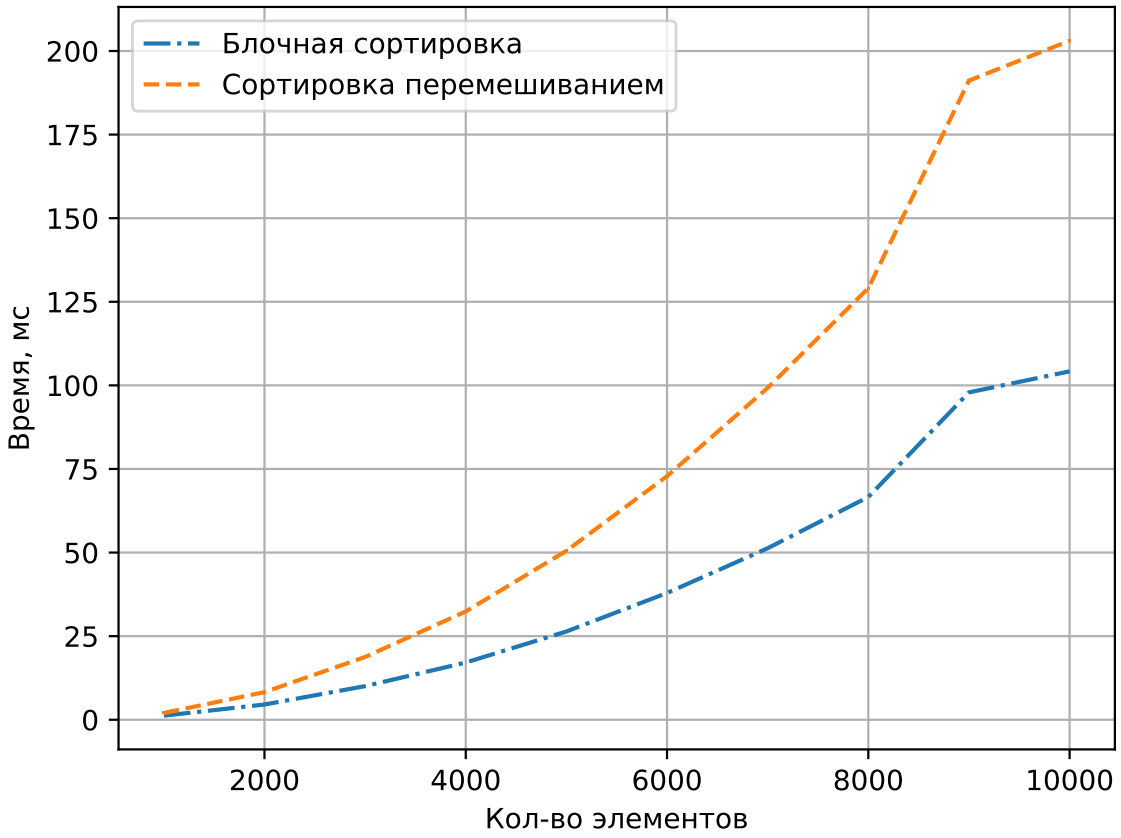

Рисунок 4.3 – Сравнение реализаций блочной сортировки и сортировки перемешиванием

Вывод

В результате анализа таблицы 4.1, было получено, что реализация алгоритма блочной сортировки требует в 1.95 раз больше времени, чем сортировка перемешиванием, поразрядная сортировка требует в 65 раз меньше времени, чем блочная сортировка. Ввиду того, что сортируемые числа неотрицательные и имеют малое число разрядов поразрядная сортировка оказалась самой эффективной по временным затратам, блочная сортировка показала лучший результат по сравнению с сортировкой перемешиваем благодаря равномерному распределению данных.

ЗАКЛЮЧЕНИЕ

В результате исследования было определено, что реализация алгоритма блочной сортировки требует в 1.95 раз больше времени, чем сортировка перемешиванием, поразрядная сортировка требует в 65 раз меньше времени, чем блочная сортировка. Ввиду того, что сортируемые числа неотрицательные и имеют малое число разрядов поразрядная сортировка оказалась самой эффективной по временным затратам, блочная сортировка показала лучший результат по сравнению с сортировкой перемешиваем благодаря равномерному распределению данных.

Цели данной лабораторной работы были достигнуты, а именно описание и исследование особенностей задач динамического программирования на алгоритмах Левенштейна и Дамерау-Левенштейна.

Целью данной лабораторной работы является изучение и исследование трудоемкости алгоритмов сортировки.

Для поставленной цели были выполнены следующие задачи.

1) Описать алгоритмы сортировки;
2) создать программное обеспечение, реализующее следующие алгоритмы сортировки;
   - перемешиванием;
   - блочная;
   - поразрядная;
3) оценить трудоемкости сортировок;
4) замерить время реализации;
5) провести анализ затрат работы программы по времени, выяснить влияющие на них характеристики;

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Шагбазян., Д. В. Алгоритмы сортировки. Анализ, реализация, применение: учебное пособие / Д.В. Шагбазян, А.А. Штанюк, Е.В. Малкина. — Нижний Новгород: Нижегородский госуниверситет, 2019. — 42 с.
2. Д. К. Искусство программирования для ЭВМ. Том 3. Сортировка ипоиск. //. — — М.: ООО «И.Д. Вильямс», 2014. — С. 824.
3. A comparative Study of Sorting Algorithms Comb, Cocktail and Counting Sorting [Электронный ресурс]. — — Режим доступа: https://www.researchgate.net/profile/Ashraf-Maghari/publication/314753240_A_comparative_Study_of_Sorting_Algorithms_Comb_Cocktail_and_Counting_Sorting/links/58c57219aca272e36dda981b/A-comparative-Study-of-Sorting-Algorithms-Comb-Cocktail-and-Counting-Sorting.pdf (дата обращения: 09.11.2023).
4. Тема 3. Компьютерный анализ данных. Лекция 10. Методы и алгоритмы обработки и анализа данных [Электронный ресурс]. — Режим доступа: http://imamod.ru/~polyakov/arc/stud/mmca/lecture_10.pdf (дата обращения: 09.11.2023).
5. C library function clock() [Электронный ресурс]. — — Режим доступа: https://linux.die.net/man/3/clock_gettime (дата обращения: 28.09.2023).
6. Windows 10 Pro 2h21 64-bit [Электронный ресурс]. — — Режим доступа: https://www.microsoft.com/ru-ru/software-download/windows10 (дата обращения: 28.09.2023).
7. Что такое WSL [Электронный ресурс]. — — Режим доступа: https://learn.microsoft.com/ru-ru/windows/wsl/about (дата обращения: 28.09.2023).

ПРИЛОЖЕНИЕ А

Листинг А.1 – Реализация алгоритма поразрядной сортировки

```
void radixSort(std::vector<int>& arr)
{

    int m = getMaxAbs(arr);
    for (int exp = 1; m / exp > 0; exp *= 10)
    countSort(arr, exp);
}
```

Листинг А.2 – Сортировка подсчетом разрядов чисел

```
void countSort(std::vector<int>& arr, int exp)
{

    int n = arr.size();
    int output[n];
    int count[20] = { 0 };
    int i;

    for (i = 0; i < n; i++)
    count[(arr[i] / exp) \% 10 + 9]++;

    for (i = 1; i < 20; i++)
    count[i] += count[i - 1];

    for (i = n - 1; i >= 0; i--)
    {
        output[count[(arr[i] / exp) \% 10 + 9] - 1] = arr[i];
        count[(arr[i] / exp) \% 10 + 9]--;
    }

    for (i = 0; i < n; i++)
    arr[i] = output[i];
}
```

Листинг А.3 – Функция поиска максимального значения по модулю в векторе

```
int getMaxAbs(const std::vector<int>& arr)
{
```

```
    int mx = abs(arr[0]);
    for (size_t i = 1; i < arr.size(); i++)
    if (arr[i] > abs(mx))
    mx = arr[i];
    return mx;
}
```

Листинг А.4 – Реализация алгоритма сортировки перемешиванием

```
void shakerSort(std::vector<int>& arr)
{
    int control = arr.size() - 1;
    int left = 0, right = control;
    do
    {
        for (int i = left; i < right; i++)
        {
            if (arr[i] > arr[i + 1])
            {
                std::swap(arr[i], arr[i + 1]);
                control = i;
            }
        }
        right = control;
        for (int i = right; i > left; i--)
        {
            if (arr[i] < arr[i - 1])
            {
                std::swap(arr[i], arr[i - 1]);
                control = i;
            }
        }
        left = control;
    } while (left < right);
}
```

Листинг А.5 – Реализация алгоритма блочной сортировки

```
    void blockSort(std::vector<int>& arr, int blockSize)
    {
        std::vector<std::vector<int>> blocks;

        for (size_t i = 0; i < arr.size(); i += blockSize)
        {
```

```
            std::vector<int> block;

            for (size_t j = i; j < i + blockSize && j <
                arr.size();
            j++)
            {
                block.push_back(arr[j]);
            }

            shakerSort(block);
            blocks.push_back(block);
        }

        int arrIndex = 0;
        while (!blocks.empty())
        {

            int minIdx = 0;
            for (size_t i = 1; i < blocks.size(); i++)
            {
                if (blocks[i][0] < blocks[minIdx][0])
                {
                    minIdx = i;
                }
            }

            arr[arrIndex] = blocks[minIdx][0];
            arrIndex++;
            blocks[minIdx].erase(blocks[minIdx].begin());

            if (blocks[minIdx].empty())
            {
                blocks.erase(blocks.begin() + minIdx);
            }
        }

    }
```