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»

«Трудоемкость алгоритмов сортировки»

Студент Маслова Марина Дмитриевна

Группа ИУ7-53Б

Оценка (баллы) _____

Преподаватель Волкова Лилия Леонидовна

2021 г.

Содержание

Введение

Сортировка – процесс перестановки объектов данного множества в определенном порядке. Сортировка служит для последующего облегчения поиска элементов в отсортированном множестве. При этом соритровка является примером огромного разообразия алгоритмов, выполняющих одну и ту же задачу. Несмотря на большое количество алгоритмов сортировки, любой из них можно разбить на три основные части:

- сравнение, определяющее порядок элементов в паре;
- перестановка, меняющая элементы в паре местами;
- сам сортирующий алгоритм, который выполняет два предыдущих шага до полного упорядочивания.

Ввиду большого количества алгоритмов сортировки одни из них имеют преимущества над другими, что приводит к необходимости их сравнительного анализа [1].

Целью данной работы является получение навыков анализа алгоритмов сортировок.

Для достижения поставленной цели необходимо выполнить следующие **задачи**:

- изучить три алгоритма сортировки: вставками, перемешиванием, выбором;
- разработать каждый из алгоритмов;
- дать теоретическую оценку трудоемкости алгоритмов сортировки;
- реализовать каждый алгоритм;
- провести тестирование реализованных алгоритмов;
- провести сравнительный анализ алгоритмов по процессорному времени работы реализации;

1 Аналитическая часть

В данном разделе представлено теоретическое описание алгоритмов сортировки вставками, перемешиванием и выбором.

1.1 Алгоритм сортировки вставками

В **алгоритме сортировки вставками** сортируемая последовательность условно делится на входную неотсортированную часть и выходную отсортированную часть. На каждом шаге из неотсортированной части выбирается элемент и помещается на нужную позицию в уже отсортированной части. В начале алгоритма считается, что первый элемент последовательности является отсортированной частью, поэтому вставка элементов в отсортированную часть начинается со второго элемента [2].

1.2 Алгоритм сортировки перемешиванием

Алгоритм сортировки перемешиванем является модификацией алгоритма сортировки пузырьком. В отличие от сортировки пузырьком, где происходит обход последовательности только в одном направлении, в алгоритме сортировки перемешиванием после достижения одной из границ рабочей части последовательности (то есть той части, которая еще не отсортирована и в которой происходит смена элементов) меняет направление движения. При этом при движении в одном направлении алгоритм перемещает к границе рабочей области максимальный элемент, а в другом направлении – минимальный элемент. Границы рабочей части последовательности устанавливаются в месте последнего обмена [2].

1.3 Алгоритм сортировки выбором

Алгоритм сортировки выбором в текущей последовательности находит минимальный/максимальный элемент, производит обмен этого элемента со значением первой неотсортированной позиции и сортирует оставшуюся часть последовательности, исключив из рассмотрения уже отсортированные элементы [2].

1.4 Вывод

В данном разделе были рассмотрены алгоритмы сортировки вставками, перемешиванием и выбором. Из представленных описаний можно предъявить ряд требований к разрабатываемому программному обеспечению:

- на вход подается последовательность, которую необходимо отсортировать;
- на выходе должна выдаваться отсортированная последовательность;
- элементы последовательности должны быть представимы любым базовым типом, для которого определена операция сравнения.

2 Конструкторская часть

В данном разделе разрабатываются алгоритмы сортировки, структура программы и способы её тестироваия, также проводится оценка трудоемкости каждого из алгоритмов.

2.1 Разработка алгоритмов

На рисунке 2.1 представлена схема алгоритма сортировки вставками, на рисунке 2.2 — схема алгоритма сортировки перемешиванием, на рисунке 2.3 — схема алгоритма сортировки выбором.

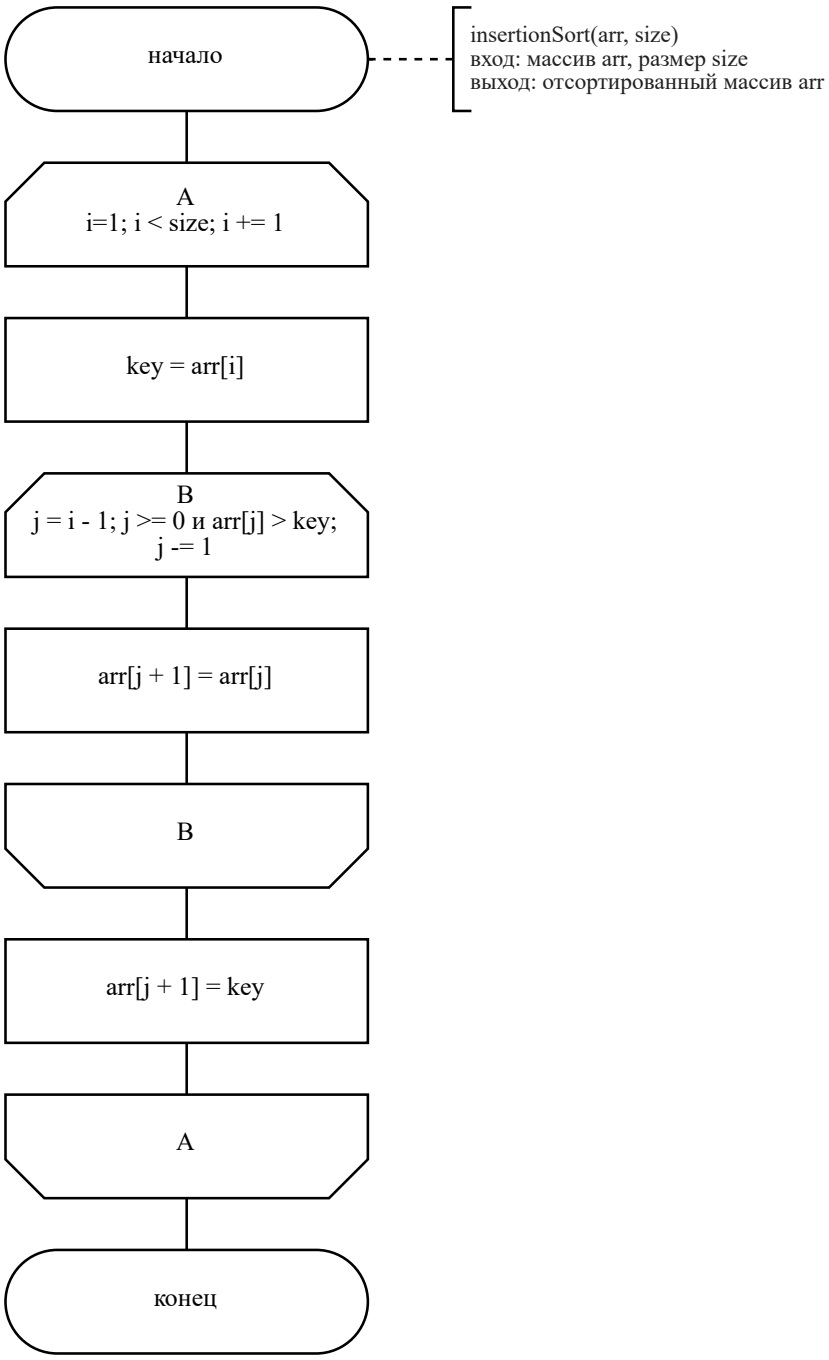

Рисунок 2.1 – Схема алгоритма сортировки вставками

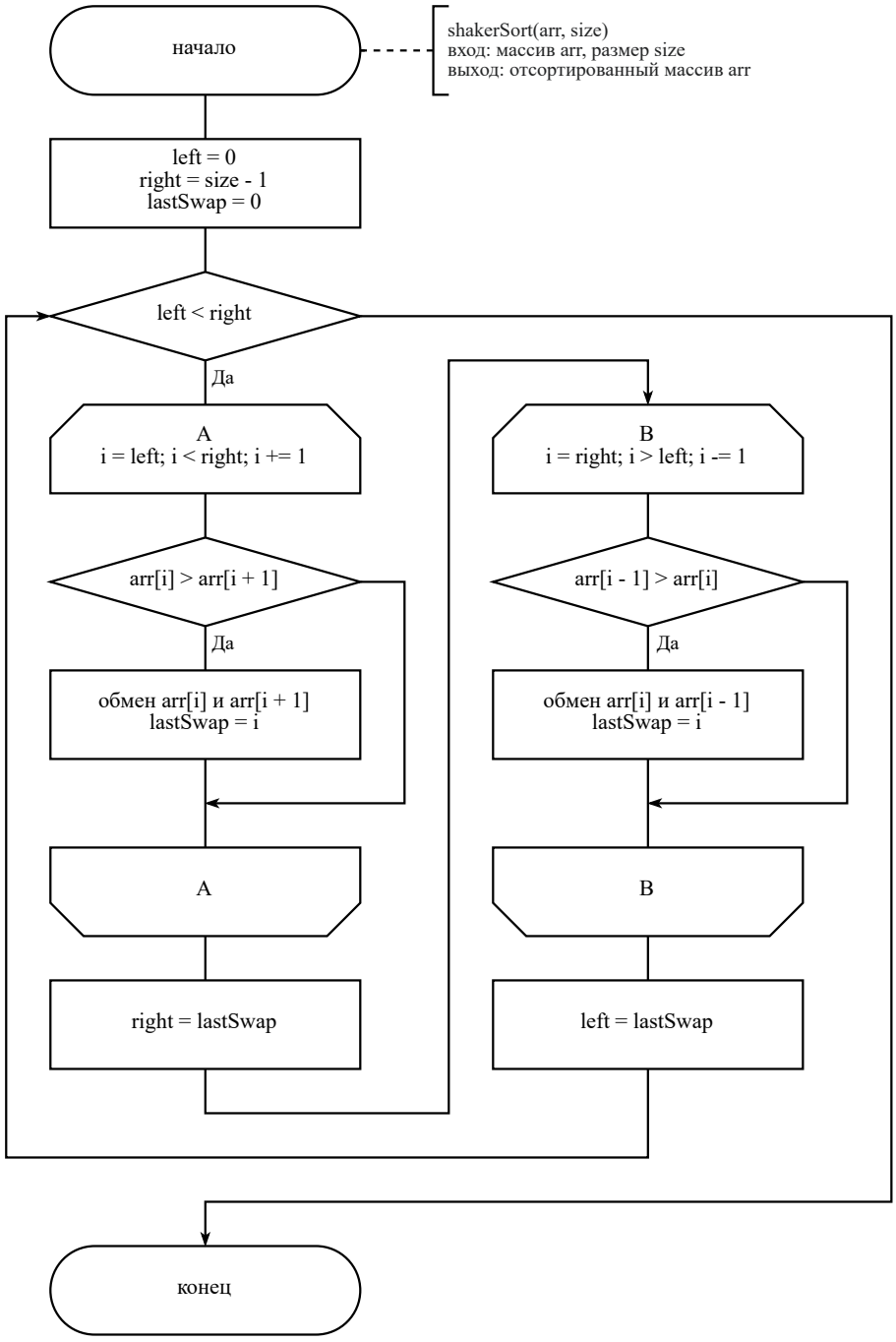

Рисунок 2.2 – Схема алгоритма сортировки перемешиванием

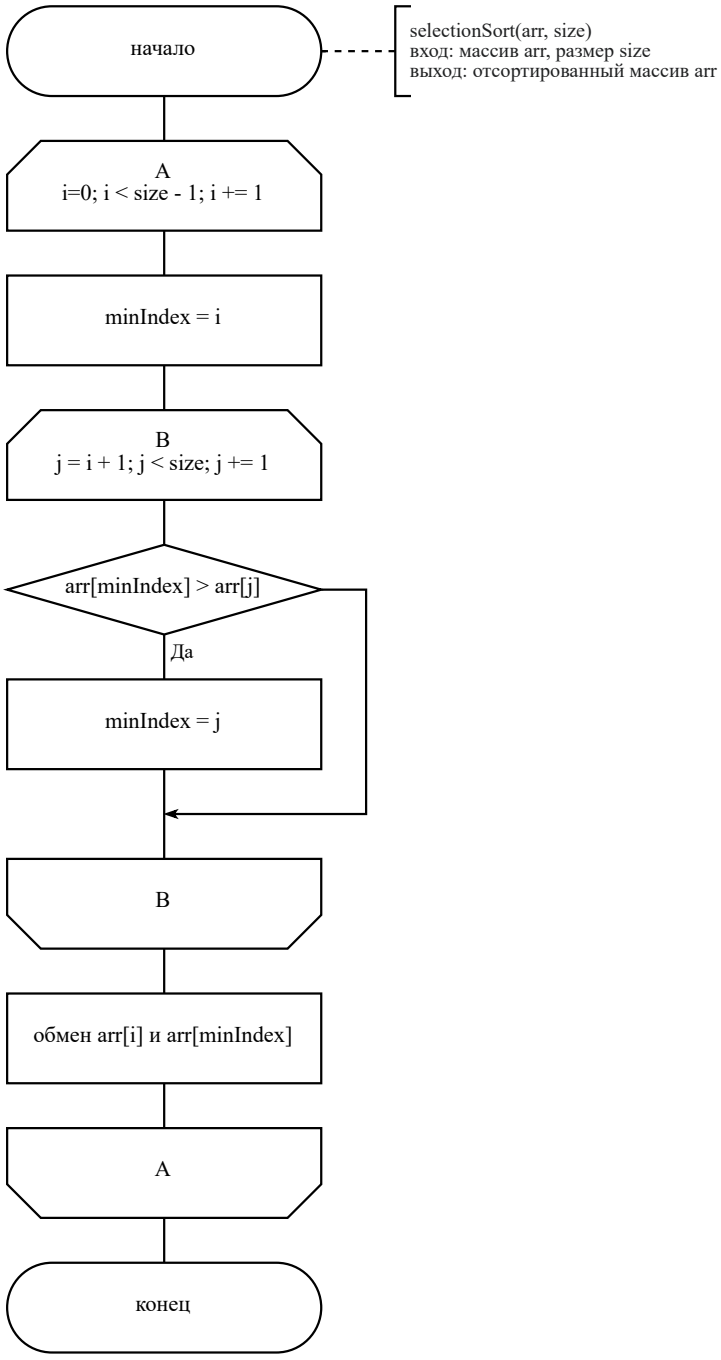

Рисунок 2.3 – Схема алгоритма сортировки выбором

2.2 Оценка трудоемкости алгоритмов

В данном подразделе производится оценка трудоемкости каждого из алгоритмов.

2.2.1 Модель вычислений

Введем модель вычислений для оценки трудоемкости алгоритмов:

– операции из списка 2.1 имеют трудоемкость 2;

$$*,\ /,\ //,\ \%,\ *=,\ /=,\ //= \tag{2.1}$$

– операции из списка 2.2 имеют трудоемкость 1;

$$\begin{gathered} =,\ +,\ -,\ +=,\ -=,\ <,\ >,\ ==,\ !=, \\ >=,\ <=,\ [],\ <<,\ >>,\ ++,\ --, and, or \end{gathered} \tag{2.2}$$

– трудоемкость оператора выбора `if` условие `then` A `else` B рассчитывается по формуле 2.3:

$$f_{if} = f + \begin{cases} f_A, & \text{если условие выполняется;} \\ f_B, & \text{иначе} \end{cases} \tag{2.3}$$

– трудоемкость оператора цикла рассчитывется по формуле 2.4:

$$f_{\text{for}} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \tag{2.4}$$

– трудоемкость вызова функции равна 0.

2.2.2 Алгоритм сортировки вставками

В лучшем случае алгоритму подается на вход упорядоченная последовательность. При таких входных данных цикл A отрабатывает $N - 1$ раз, где $N = size$, а внутренний цикл B на каждой итерации осуществляет только инициализацию и проверку условия, которое оказывается ложным, из-за чего тело не выполняется. Таким образом, в лучшем случае трудоемкость алгорит-

ма сортировки вставками равна (формула 2.5):

$$f^{\wedge} = 2 + (N-1)(2+2+4+3) = 13N - 11 = O(N) \tag{2.5}$$

В худшем случае алгритму подается на вход последовательность, упорядоченная в обратном порядке. При таких входных данных цикл B отрабатывает в среднем $\frac{N+2}{2}$. Таким образом, в худшем случае трудоемкость алгоритма сортировки вставками равна (формула 2.6):

$$f^{\vee} = 2+(N-1)(2+2+6+\frac{N+2}{2}(5+4)+3) = \frac{9}{2}N^2+\frac{35}{2}N-20 = O(N^2) \tag{2.6}$$

2.2.3 Алгоритм сортировки перемешиванием

В лучшем случае алгоритму подается на вход упорядоченная последовательность. При таких входных данных первый внутренний цикл A осуществлет полный проход по последовательность в одну сторону и не находит обменов соседний элементов, поэтому на перой же итерации внешнего цикла правая граница становится равной левой границе, из-за чего второй внутренний цикл B не отрабатывает, произведя только ининциализацию и одну проверку условия, а внешний цикл завершает свою работу после первой итерации. Таким образом, в лучшем случае трудоемкость алгоритма сортировки перемешиваним равна (формула 2.7):

$$f^{\wedge} = 4 + 1 + 2 + (N-1)(2+4) + 1 + 2 + 1 + 1 = 6N + 6 = O(N) \tag{2.7}$$

В худшем случае алгоритму подается на вход упорядоченная в обратном порядке последовательность. Внешний цикл отрабатывает $\frac{N}{2}$ раз, так как за одну итерацию на свое место ставятся 2 элемента. Количество итераций каждого цикла зависит от четности размера подаваемой последовательнсти, если N — четно, то цикл A отработает $\frac{N-1+1}{2}$ итераций, а цикл B — $\frac{N-2+1}{2}$, если N — нечетно, то цикл A отработает $\frac{N-1+2}{2}$ итераций, а цикл B — $\frac{N-2}{2}$, однако и в том, и другом случае в общем два цикла отработают $\frac{2N-1}{2}$ итераций, а так как сложность тел обоих циклов одинаковая при вычислениях можно сразу пользоваться общим числом итераций. Таким образом, в худшем случае

трудоемкость алгоритма сортировки перемешиванием равна (формула 2.8):

$$f^{\vee} = 4 + \frac{N}{2}(1+2+1+2+1+\frac{2N-1}{2}(2+4+9+1)) = \\ = 4 + \frac{N}{2}(16N-1) = 8N^2 - \frac{N}{2} + 4 = O(N^2) \quad (2.8)$$

2.2.4 Алгоритм сортировки выбором

В любом случае в алгоритме сортировки выбором внешний цикл A выполняет $N-1$ итераций, а внутренний цикл в среднем $\frac{N}{2}$ итерации.

В лучшем случае на вход алгоритму подается упорядоченная последовательность. При таких входных данных никогда не выполняется блок по ветке `then` условия. Таким образом, в лучшем случае трудоемкость алгоритма сортировки выбором равна (формула 2.9):

$$f^{\wedge} = 3 + (N-1)(3+1+3+\frac{N}{2}(2+3)+7) = \\ = 3 + (N-1)(14+\frac{5}{2}N) = \frac{5}{2}N^2 + \frac{23}{2}N - 11 = O(N^2) \quad (2.9)$$

В худшем случае на вход алгоритму подается упорядоченная в обратном порядке последовательность. При таких входных данных всегда выполняется блок по ветке `then` условия. Таким образом, в лучшем случае трудоемкость алгоритма сортировки выбором равна (формула 2.10):

$$f^{\wedge} = 3 + (N-1)(3+1+3+\frac{N}{2}(2+3+1)+7) = \\ = 3 + (N-1)(14+3N) = 3N^2 + 11N - 11 = O(N^2) \quad (2.10)$$

2.3 Структура разрабатываемого ПО

Для реализации разрабатываемого программного обеспечения будет использоваться метод структурного программирования. Каждый из алгоритмов будет представлен отдельной функцией, при необходимости будут выделены подпрограммы для каждой из них. Также будут реализованы функции для ввода-вывода и функция, вызывающая все подпрограммы для связности и полноценности программы.

2.4 Классы эквивалентности при тестировании

Для тестирования программного обеспечения во множестве тестов будут выделены следующие классы эквивалентности:

- пустой массив;
- упорядоченный массив четной длины;
- упорядоченный массив нечетной длины;
- упорядоченный в обратном порядке массив четной длины;
- упорядоченный в обратном порядке массив нечетной длины;
- случайный массив;
- массив из одного элемента.

2.5 Вывод

В данном разделе были разработаны алгоритмы сортировки вставками, перемешиванием и выбором, также была произведен оценка трудоемкостей алгоритмов. Для дальнейшей проверки правильности работы программы были выделены классы эквивалентности тестов.

3 Технологическая часть

В данном разделе описаны требования к программному обеспечению, средства реализации, приведены листинги кода и данные, на которых будет проводиться тестирование.

3.1 Требования к программному обеспечению

Программа должна предоставлять следующие возможности:

- выбор режима работы: для единичного эксперимента и для массовых экспериментов;
- в режиме единичного эксперимента выбор функции для интегрирования, ввод пределов интегрирования, точности и числа потоков для параллельной реализации;
- в режиме массовых экспериментов измерение времени работы каждого из алгоритмов в зависимости от точности и числа потоков.

3.2 Средства реализации

Для реализации данной лабораторной работы выбран компилируемый язык програмирования C++[**?**], так как он предоставляет необходимые библиотеки для работы с потоками. Интерпретируемый язык программирования высокого уровня Python[3] был выбран для визуализации данных эксперимента, так как он предоставляет большое число настроек параметров графика с использованием простого синтаксиса.

В качестве среды разработки выбран текстовый редактор Vim[4] с установленными плагинами автодополнения и поиска ошибок в процессе написания, так как он реализует быстрое перемещение по тексту программы и простое взаимодействие с командной строкой.

Замеры времени проводились при помощи функции std::chrono::system_clock::now() из библиотеки chrono[**?**].

3.3 Листинги кода

В данном подразделе представлены листинги кода ранее описанных алгоритмов:

– последовательный алгоритм численного итегрирования методом средних прямоугольников с заданной точностью (листинги 3.1-**??**);
– параллельный алгоритм численного итегрирования методом средних прямоугольников с заданной точностью (листинг 3.3-**??**).

Листинг 3.1 – Функция интегрирования методом средних прямоугольников с заданным числом участков разбиения (последовательный алгоритм)

```
1
2  long double midpoint(double begin, double end, unsigned int n, function_t func)
3  {
4      double res = 0;
5      double step = (end - begin) / n;
6      double x = begin + step / 2;
7
8      while (x < end + step / 2)
9      {
10         long double y = func(x);
11         res += y * step;
12         x += step;
13     }
14
15     return res;
```

Листинг 3.2 – Функция вычисления интеграла с заданной точностью (последовательный алгоритм)

```
1
2  long double integralByPrecision(interval_t &interval, function_t func)
3  {
4      unsigned long n = 2;
5      long double delta = 1;
6      long double res = 0, obt_res = 0;
7
8      while (delta > interval.eps)
9      {
10         res = obt_res;
11         obt_res = midpoint(interval.begin, interval.end, n, func);
12         n *= 2;
13         delta = abs(res - obt_res);
14     }
15
16     return res;
```

Листинг 3.3 – Функция интегрирования методом средних прямоугольников с заданным числом участков разбиения (параллельный алгоритм)

```
1
2  void parallelMidpoint(double begin, double end, unsigned int n, function_t func,
3                        long double &res, int threads_num, int i, mutex &mut)
4  {
5      double local_res = 0;
6      double step = (end - begin) / n;
7      double x = begin + i * step + step / 2;
8
9      while (x < end + step / 2)
10     {
11         long double y = func(x);
12         local_res += y * step;
13         x += step * threads_num;
14     }
15
16     mut.lock();
17     res += local_res;
18     mut.unlock();
```

Листинг 3.4 – Функция создания потоков для интегрирования методом средних прямоугольников

```
1
2  long double multithreading(int threads_num, interval_t &interval,
3                             unsigned int n, function_t func)
4  {
5      long double res = 0;
6      mutex res_mut;
7
8      vector<thread> threads(threads_num);
9
10     for (int i = 0; i < threads_num; i++)
11     {
12         threads[i] = thread(parallelMidpoint, interval.begin, interval.end, n,
13                             func, ref(res), threads_num, i, ref(res_mut));
14     }
15
16     for (int i = 0; i < threads_num; i++)
17     {
18         threads[i].join();
19     }
20
21     return res;
```

Листинг 3.5 – Функция вычисления интеграла с заданной точностью (параллельный алгоритм)

```
1
2  long double integralByPrecision(interval_t &interval, function_t func)
3  {
4      unsigned long n = 2;
5      long double delta = 1;
6      long double res = 0, obt_res = 0;
7
8      while (delta > interval.eps)
9      {
10         res = obt_res;
11         obt_res = midpoint(interval.begin, interval.end, n, func);
12         n *= 2;
13         delta = abs(res - obt_res);
14     }
15
16     return res;
```

3.4 Описание тестирования

В таблице 3.1 приведены функциональные тесты для алгоритмов интегрирования на функции $f(x) = x^2$.

Таблица 3.1 – Функциональные тесты

Пределы интегрирования	**Ожидаемый результат**
0 0	0
0 1	0.3333
1 0	-0.333
-1 1	0.6666

3.5 Вывод

В данном разделе были реализованы последовательный и параллельный алгоритмы численного интегрирования методом срединих прямоугольников. Также были написаны тесты для каждого класса эквивлентности, описанного в конструкторском разделе.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

– Операционная система: Manjaro [5] Linux x86_64.

– Память: 8 GiB.

– Процессор: Intel® Core™ i5-8265U, 4 физических ядра, 8 логических ядра[6].

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, окружением, а также непосредственно системой тестирования.

4.2 Примеры работы программы

На рисунке 4.1 представлены результаты работы программы.

```
РАСПАРАЛЛЕЛИВАНИЕ АЛГОРИТМА ЧИСЛЕННОГО ИНТЕГРИРОВАНИЯ

     МЕНЮ
1 -- подсчет интеграла;
2 -- сравнение последовательного и параллельного алгоритмов;
0 -- выход.

Выбор:
1
Функции:
     1 -- x^2;
     2 -- x * sin x;
     3 -- √(e^(sin x * cos x) + 6 * |x + 6|) +
          + ln|x^3 - 3x + 1|;
Выберете функцию: 3
Введите нижний предел: -10
Введите верхний предел: 10
Введите количество знаков после запятой: 6
Введите количество потоков: 4

ЗНАЧЕНИЕ ИНТЕГРАЛА
Последовательный алгоритм: 198.740798
Время: 812399544 нс
Параллельный алгоритм: 198.740798
Время: 306246758 нс
```

Рисунок 4.1 – Пример работы программы

4.3 Результаты тестирования

Программа была протестирования на входных данных, приведенных в таблице 3.1. Полученные результаты работы программы совпали с ожидаемыми результатами.

4.4 Постановка эксперимента по замеру времени

Для оценки времени работы последовательной и параллельной реализации алгоритма численного интегрирования методом средних прямоугольников был проведен эксперимент, в котором определялось влияние количества потоков и точности вычислений на время работы алгоритмов. Тестирование проводилось на количестве потоков, равном степеням 2 от 1 до 64, и на точности вычислений от 1 до 7 знаков после запятой. Так как от запуска к запуску время, затрачиваемое на выполнение алгоритма, менялось в определенном промежутке, необходимо было усреднить вычисляемые значения. Для этого каждый алгоритм в каждом случае запускался по 10 раз, и для полученных 10 значений определялось среднее арифметическое, которое заносилось в таблицу результатов.

Результаты эксперимента были представлены в виде таблиц и графиков, приведенных в следующем подразделе.

4.5 Результаты эксперимента

В таблице 4.1 представлены результаты измерения времени работы паралелльного алгоритма в зависимости от числа потоков. На рисунке представлен соответствующий график, для сравнения на графике приведено также время последовательной реализации.

В таблице 4.2 представлены результаты измерения времени работы последовательного и паралелльного алгоритмов в зависимости от числа знаков после запятой, соответствующие заданной точности вычислений интеграла. На рисунке представлен соответствующий график, для наглядости на графике представлены только точки с точностью от 4 до 7 знаков после запятой.

Таблица 4.1 – Время работы от числа потоков (* – последовательный)

Число потоков	**Время, нс**
*	880622058
1	839013147
2	446833477
4	338605815
8	300881333
16	317716474
32	622792430
64	1561156672

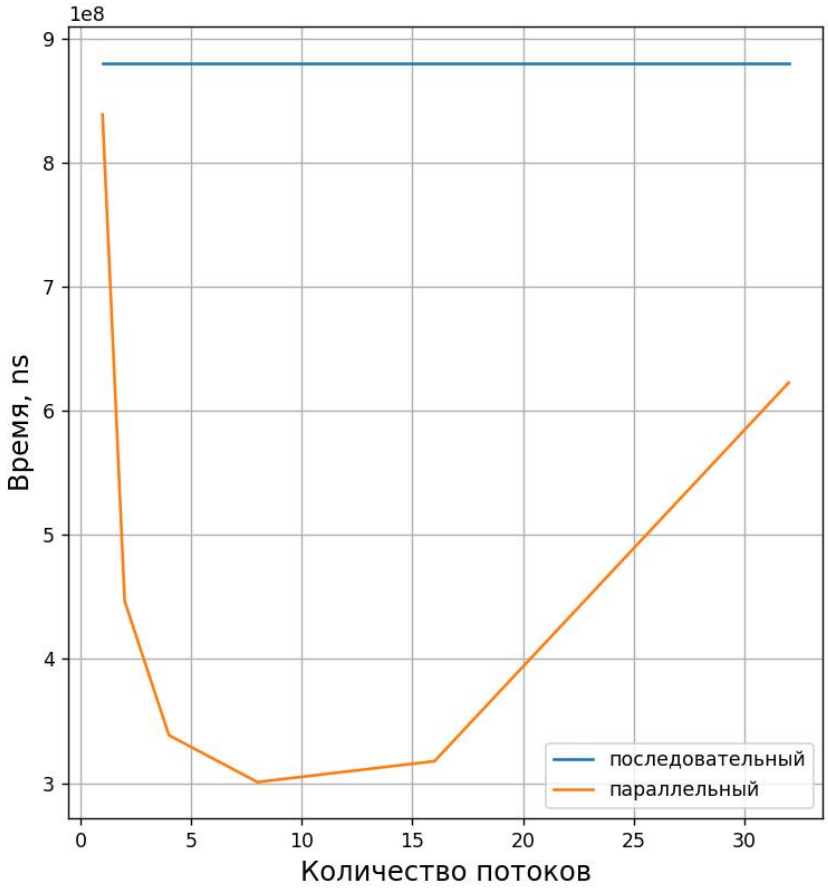

Рисунок 4.2 – График зависимости времени работы от числа потоков

Таблица 4.2 – Время работы от точности

Точность	**8 потоков, нс**	**Последовательный, нс**
1e-1	24051708	167321
1e-2	161824161	1063368
1e-3	264771774	8724506
1e-4	364664568	23780663
1e-5	666893339	849861484
1e-6	967859493	1701109328
1e-7	1272447909	2549415743

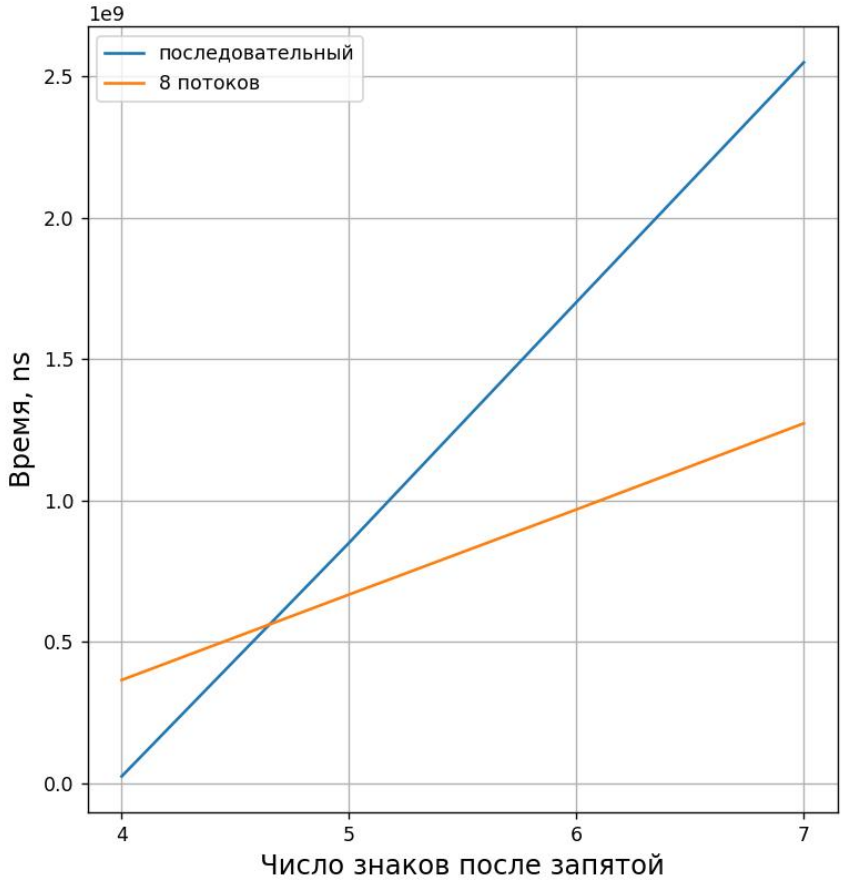

Рисунок 4.3 – График зависимости времени работы реализаций от точности

4.6 Вывод

По результатам эксперимента можно сделать следующие выводы:

- при движении от наименьшего числа потоков к числу потоков равному числу логических ядер процессора время уменьшается;
- при превышении числа логических ядер время увеличивается, так как затрачивается время на переключение ядра между потоками;
- последовательная реализация работает примерно в3 раза медленнее параллельной реализации на оптимальном числе потоков (8);
- при этом параллельная реализация только при числе потоков большем 32 работает медленнее, что говорит о том, что при таком числе реализуемых потоков выигрыш от распараллеливания перекрывается временем, затрачиваемым на переключения между потоками;
- в зависимости от точности выигрыш от распараллеливания начинает проявляться только при точности от 5 и выше знаков после запятой (выигрыш примерно в 2 раза); это значит, что меньшей точности время, затрачиваемое на создание и запуск потоков, больше чем выигрыш, получаемый от распараллеливания.

Таким образом, для достижения наибольшей скорости вычислений необходимо использовать число потоков равное числу логических потоков на машине. При этом нецелесообразно использовать параллельную реализацию при проведении вычислений интегралов с точностью ниже, чем 5 знаков после запятой. В таком случае, необходимо либо использовать параллельную реализацию, либо уменьшить число потоков, чтобы затрачивать меньше времени на их создание.

Заключение

В ходе исследования был проведен сравнительный аналализ последовательной и параллельной реализации алгорима численного интегрирования методом средних прямоугольников. В результате исследования было выяснено, что при распараллеливании можно добиться улучшения скорости вычислений примерно в 3 раза, но при условии использования оптимального числа потоков, равного числу логических ядер процессора, и вычисления интегралов в точностью от 5 знаков после запятой.

В ходе выполения лабораторной работы:

- были описаны и разработаны последовательный и параллельный алгоритм численного интегрирования методом средних прямоугольников;
- был реализован каждый из описанных алгоритмов;
- по экспериментальным данным были сделаны выводы об эффективности по времени каждого из реализованных алгоритмов;
- были получены зависимости времени работы паралелльного алгоритма от числа потоков и времени работы каждого из алгритмов от точности вычислений.

Таким образом, все поставленные задачи были выполнены, а цель достигнута.

Список литературы

[1] Вирт Н. Алгоритмы и структуры данных. М.: Мир, 1989. с. 360.

[2] Шагбазян Д. В., Штанюк А. А., Малкина Е. В. Алгоритмы сортировки. Анализ, реализация, применение. Нижний Новгород: Нижегородский госуниверситет, 2019. с. 42.

[3] Welcome to Python [Электронный ресурс]. Режим доступа: `https://www.python.org` (дата обращения: 12.10.2021).

[4] welcome home : vim online [Электронный ресурс]. Режим доступа: `https://www.vim.org/` (дата обращения: 12.10.2021).

[5] Manjaro — enjoy the simplicity [Электронный ресурс]. Режим доступа: `https://manjaro.org/` (дата обращения: 17.10.2021).

[6] Процессор Intel® Core™ i5-8265U [Электронный ресурс]. Режим доступа: `https://ark.intel.com/content/www/ru/ru/ark/products/149088/intel-core-i5-8265u-processor-6m-cache-up-to-3-90-ghz.html` (дата обращения: 17.10.2021).