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 «Анализы Алгоритмов»

на тему: «Трудоемкость сортировок»

Студент группы ИУ7-56Б _____ (Подпись, дата) Мансуров В. М. (Фамилия И.О.)

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

Преподаватель _____ (Подпись, дата) Строганов Ю. В.. (Фамилия И.О.)

Москва — 2023 г.

Содержание

Введение

В данной лабораторной работе будет рассмотрены сортировки.

Сортировка - перегруппировка некой последовательности, или кортежа, в определенном порядке. Это одна из главных процедур обработки структурированных данных. Расположение элементов в определенном порядке позволяет более эффективно проводить работу с последовательностью данных, в частности при поиске некоторых данных.

Различают два вида сортировок: сортировку массивов и сортировку файлов. Сортировку массивов также называют внутренней, т.к. все элементы массивов хранятся в быстрой внутренней памяти машины с прямым доступом, а сортировку файлов – внешней, т.к. их элементы хранятся в медленной, но более емкой внешней памяти. При внутренней сортировке доступ к элементам может осуществляться в произвольном порядке. Напротив, при внешней сортировке доступ к элементам производится в строго определенной последовательности

Существует множество алгоритмов сортировки, но любой алгоритм сортировки имеет:

— сравнение, которое определяет, как упорядочена пара элементов;

— перестановка для смены элементов местами;

— алгоритм сортировки, использующий сравнение и перестановки.

Каждый алгоритм имеет свои достоинства, но в целом его оценка зависит от ответов, которые будут получены на следующие вопросы:

— с какой средней скоростью этот алгоритм сортирует информацию;

— какова скорость для лучшего и для худшего случаев;

— является ли естественным “поведение” алгоритма (т.е. возрастает ли скорость сортировки с увеличением упорядоченности массива);

— является ли алгоритм стабильным (т.е. выполняется ли перестановка элементов с одинаковыми значениями).

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

Для поставленной цели необходимо выполнить следующие задачи.

1) Описать расстояния Левенштейна и Дамерау-Левенштейна.
2) Создать программное обеспечение, реализующее следующие алгоритмы сортировки:
   - Шелла;
   - Пирамидальная;
   - Бусинами.
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 данном разделе будут рассмотрены три алгоритма сортировок: сортировка вставками и ее улучшение сортировка Шелла, сортировка выбором и ее улучшение пирамидальная сортировка и сортировка бусинами.

1.1 Сортировка вставками

Суть алгоритма заключается в следующем: на каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. Алгоритму необходимо для каждого нового элемента выбрать нужное место для вставки в уже упорядоченный массив данных, так что элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов

1.1.1 Сортировка Шелла

Метод предложен в 1959 году и назван по имени автора метода Дональда Шелла (Donald Shell).

Сортировка Шелла [1, 2, 3] (англ. Shell Sort) является улучшением сортировки вставками. Часто еще называемый "сортировка вставками с уменьшением расстояния". Основная идея этого метода заключается в том, чтобы в начале устранить массовый беспорядок в массиве, сравнивая далеко отстоящие друг от друга элементы. Постепенно интервал между сравниваемыми элементами уменьшается до единицы. Это означает, что на поздних стадиях сортировка сводится просто к перестановкам соседних элементов (если, конечно, такие перестановки являются необходимыми).

Пусть d - интервал между сравниваемыми элементами. Первоначально используемая Шеллом последовательность длин промежутков: $d_1 = \frac{N}{2}, d_i = \frac{d_{i-1}}{2}, \dots d_k = 1$. Процесс завершается обычной сортировкой вставками получившегося списка.

1.2 Сортировка выбором

Алгоритм сортировки выбором заключается в поиске на необработанном срезе массива или списка минимального значения и в дальнейшем обмене этого значения с первым элементом необработанного среза. На следующем шаге необработанный срез уменьшается на один элемент.

Шаги выполнения алгоритма:

1) Находим минимальный (максимальный) элемент в текущем массиве.

2) Производим обмен этого элемента со значением первой неотсортированной позиции. Обмен не нужен, если минимальный (максимальный) элемент уже находится на данной позиции.

3) Сортируем оставшуюся часть массива, исключив из рассмотрения уже отсортированные элементы.

1.2.1 Пирамидальная сортировка

Пирамидальная сортировка [1, 2, 3, 4] (англ. Heap Sort) предложена в 1964 году Дж. Уильямсом. Основана на использовании бинарного сортирующего дерева (пирамиды) и базируется на сортировке выбором, по сути является его усовершенствованием.

Пирамида (англ. binary heap) определяется как структура данных, представляющая собой объект-массив, который можно рассматривать как почти полное бинарное дерево. Каждый узел этого дерева соответствует определенному элементу массива. На всех уровнях, кроме, может быть, последнего, дерево полностью заполнено (заполненным считается уровень, который содержит максимально возможное количество узлов). Последний уровень заполняется слева направо до тех пор, пока в массиве не закончатся элементы.

В пирамиде, представленной на рисунке 1.1, число в окружности, представляющей каждый узел дерева, является значением, сортируемым в данном узле. Число над узлом — это соответствующий индекс массива. Линии, попарно соединяющие элементы массива, обозначают взаимосвязь

вида “родитель-потомок”. Родительские элементы всегда расположены слева от дочерних. Данное дерево имеет высоту, равную 3; узел с индексом 4 (и значением 8) расположен на первом уровне.

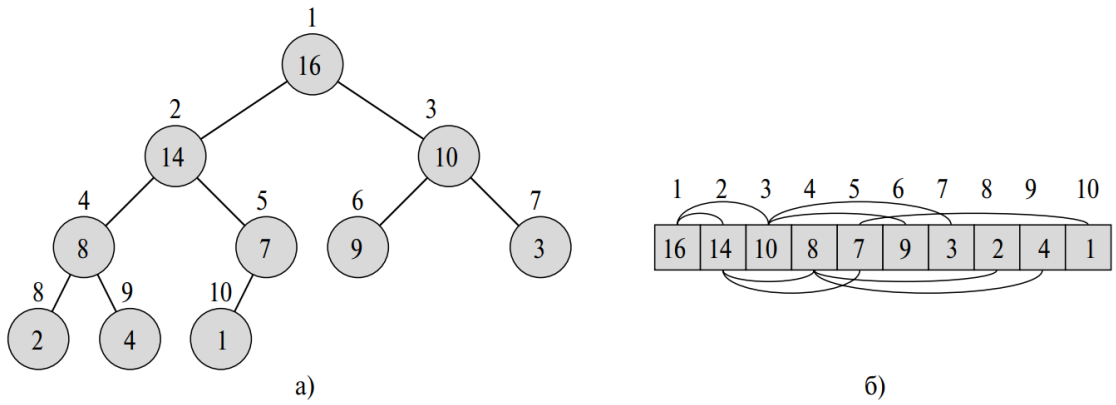

Рисунок 1.1 – Пирамида, представленная в виде а) бинарного дерева и б) массива

Обозначим, что:

1) Родительский элемент в массиве будет находится под индексом - $\frac{i}{2}$;

2) Правый потомок в массиве будет находится под индексом - $2 \cdot i$;

3) Левый потомок в массиве будет находится под индексом - $2 \cdot i + 1$.

Различают два вида бинарных пирамид: неубывающие и невозрастающие. В пирамидах обоих видов значения, расположенные в узлах, удовлетворяют свойству пирамиды (heap property), являющемуся отличительной чертой пирамиды того или иного вида. Свойство невозрастающих пирамид (max-heap property) заключается в том, что для каждого отличного от корневого узла с индексом i выполняется следующее неравенство

$$A[i/2] \geq A[i] \tag{1.1}$$

Другими словами, значение узла не превышает значение родительского по отношению к нему узла. Таким образом, в невозрастающей пирамиде самый большой элемент находится в корне дерева, а значения узлов поддерева, берущего начало в каком-то элементе, не превышают значения самого этого элемента. Принцип организации неубывающей пирамиды (min-heap) прямо противоположный. Свойство неубывающих пирамид

(min-heap property) заключается в том, что для всех отличных от корневого узлов с индексом i выполняется такое неравенство:

$$A[i/2] \leq A[i] \tag{1.2}$$

Таким образом, наименьший элемент такой пирамиды находится в ее корне.

1.3 Сортировка бусинами

Гравитационная сортировка, также известная как сортировка бусинами (англ. Bead Sort) — алгоритм сортировки, разработанный Джошуа Аруланандхамом, Кристияном Калюдом и Майклом Диннином в 2002 году.

Прародителем счётов в Европе является абак, который из Вавилона перекочевал в Египет, оттуда – в Грецию, из неё – в Рим, из которого – по всей Европе. Внешний вид и принцип действия античного «калькулятора» настолько напоминает поведение этой «простой» сортировки, что её иногда так и называют — «Абаковая сортировка» (Abacus sort).

Предположим, нужно отсортировать набор натуральных чисел. Друг под другом каждое число выложим в виде горизонтального ряда из соответствующего количества шариков. А теперь поглядим на все эти группировки шариков не по горизонталям, а по вертикалям. Сдвинем шарики вниз до упора. Снова переключимся на горизонтали и пересчитаем бусинки в каждом ряду. Получили первоначальный набор чисел, только упорядоченный.

Метод ограничен, прежде всего применим к натуральным числам, т.е. можно сортировать только положительные числа. Можно сортировать и целые, но это запутаннее - отрицательные числа придется обрабатывать отдельно отрицательные от положительных.

Вывод

В данном разделе были рассмотрены алгоритмы сортировки - бусинами, пирамидальная и Шелла и рассмотрены некоторые их неусовершенствованные сортировки, а именно сортировка вставками и выбором. Рассмотренные сортировки следует реализовать, изучить трудоемкость и проанализировать время и память для выполнения.

2 Конструкторская часть

В данном разделе приведены схемы алгоритмов сортировок - бусинами, пирамидальная, Шелла, приведено описанию используемых типов данных.

2.1 Требования к программному обеспечению

К программе предъявлены ряд функциональных требований: входные данные — массив и размер массива, выходные данные — отсортированный массив.

К программе предъявлены ряд требований:

— должна иметь интерфейс для выбора действий;

— должна динамически выделять память под массив данных;

— должна замерять процессорное время работы реализации алгоритмов.

2.2 Разработка алгоритмов

На вход алгоритмов подаются указатель на массив $array$ и размер массива n - целое положительное число.

На рисунках 2.1 – 2.4 представлены схемы алгоритмов сортировок, а именно сортировка Шелла, пирамидальная сортировка и сортировка бусинами.

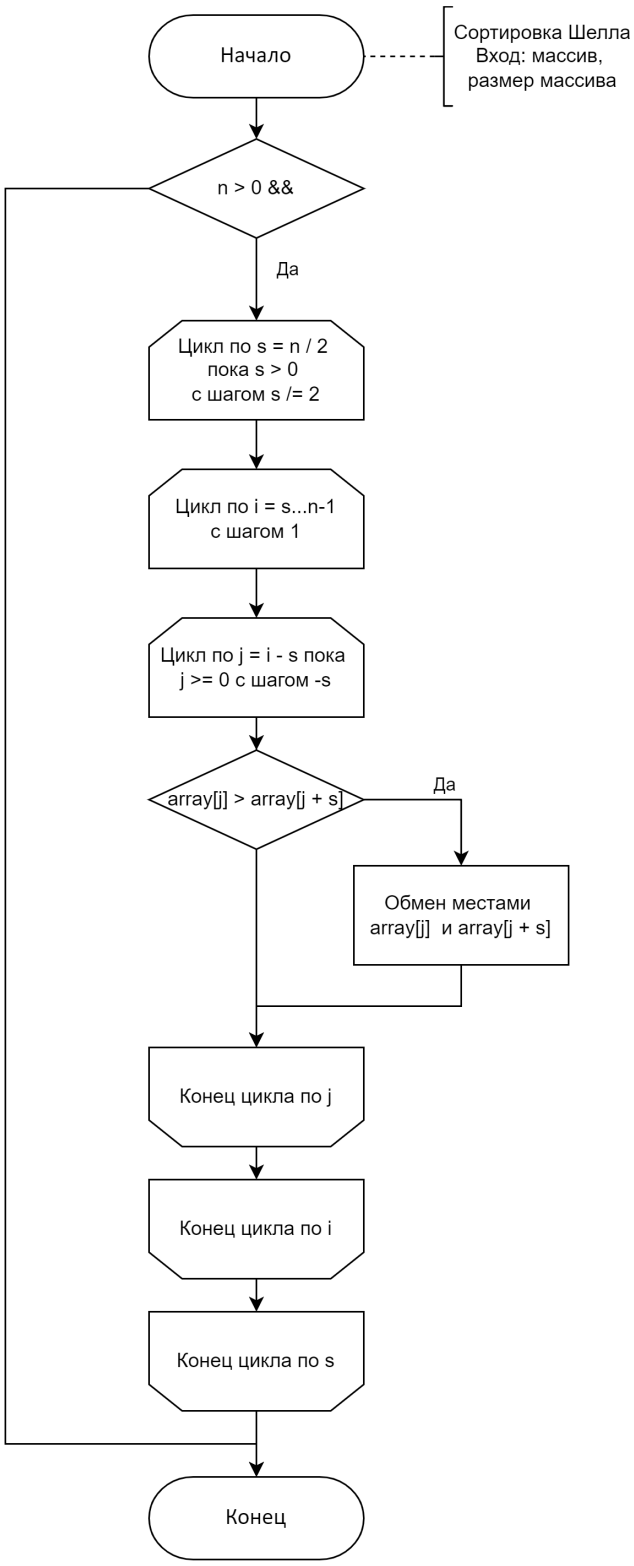

Рисунок 2.1 – Схема алгоритма сортировки методом Шелла

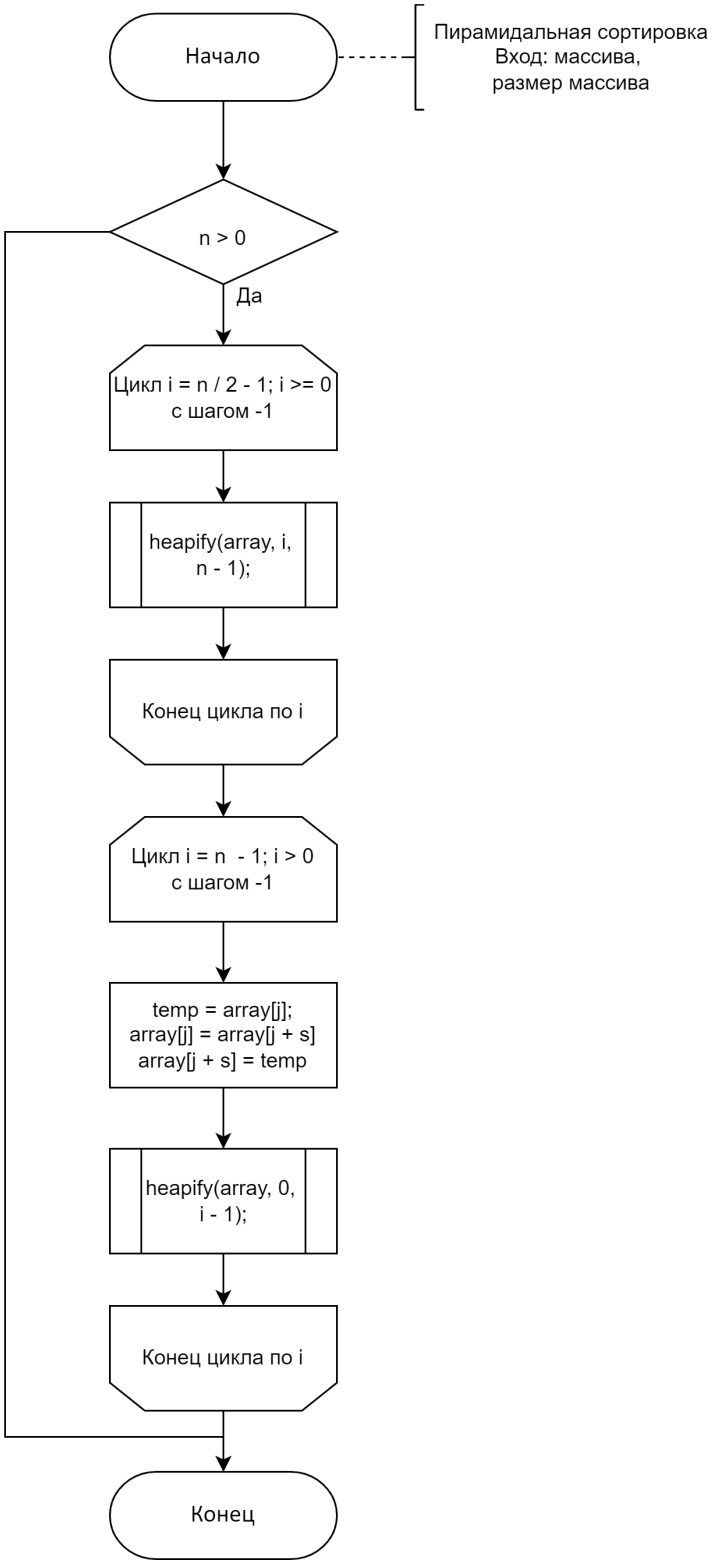

Рисунок 2.2 – Схема алгоритма пирамидальной сортировки

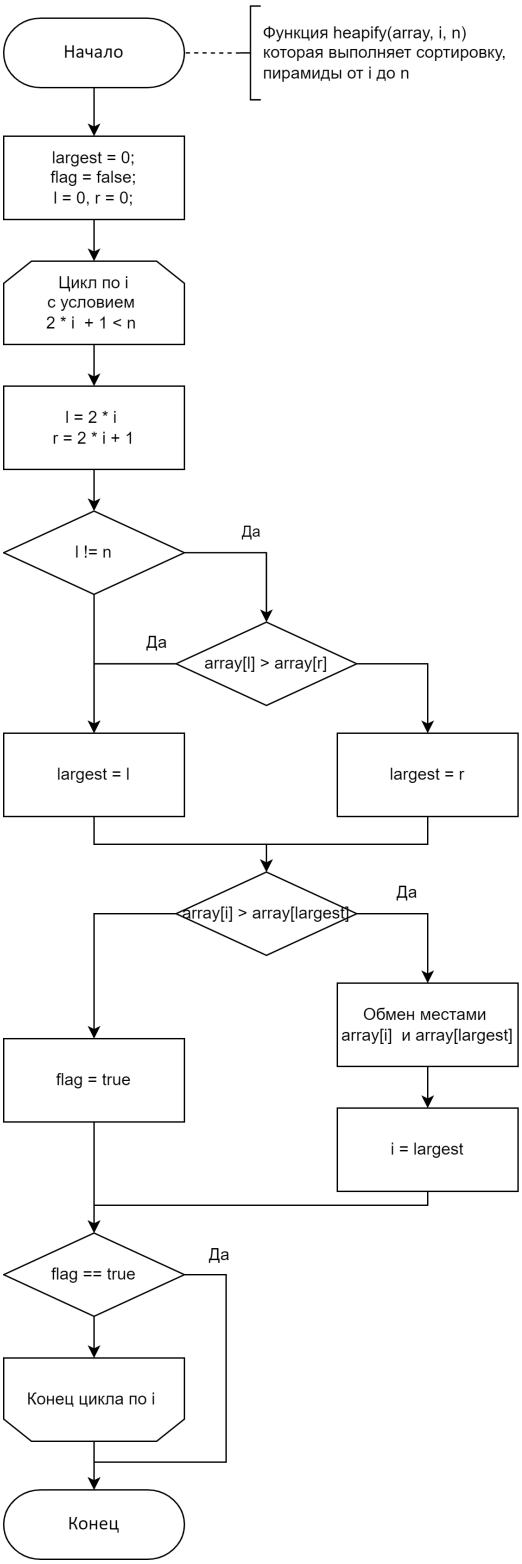

Рисунок 2.3 – Схема алгоритма функции heapify() для пирамидальной сортировки

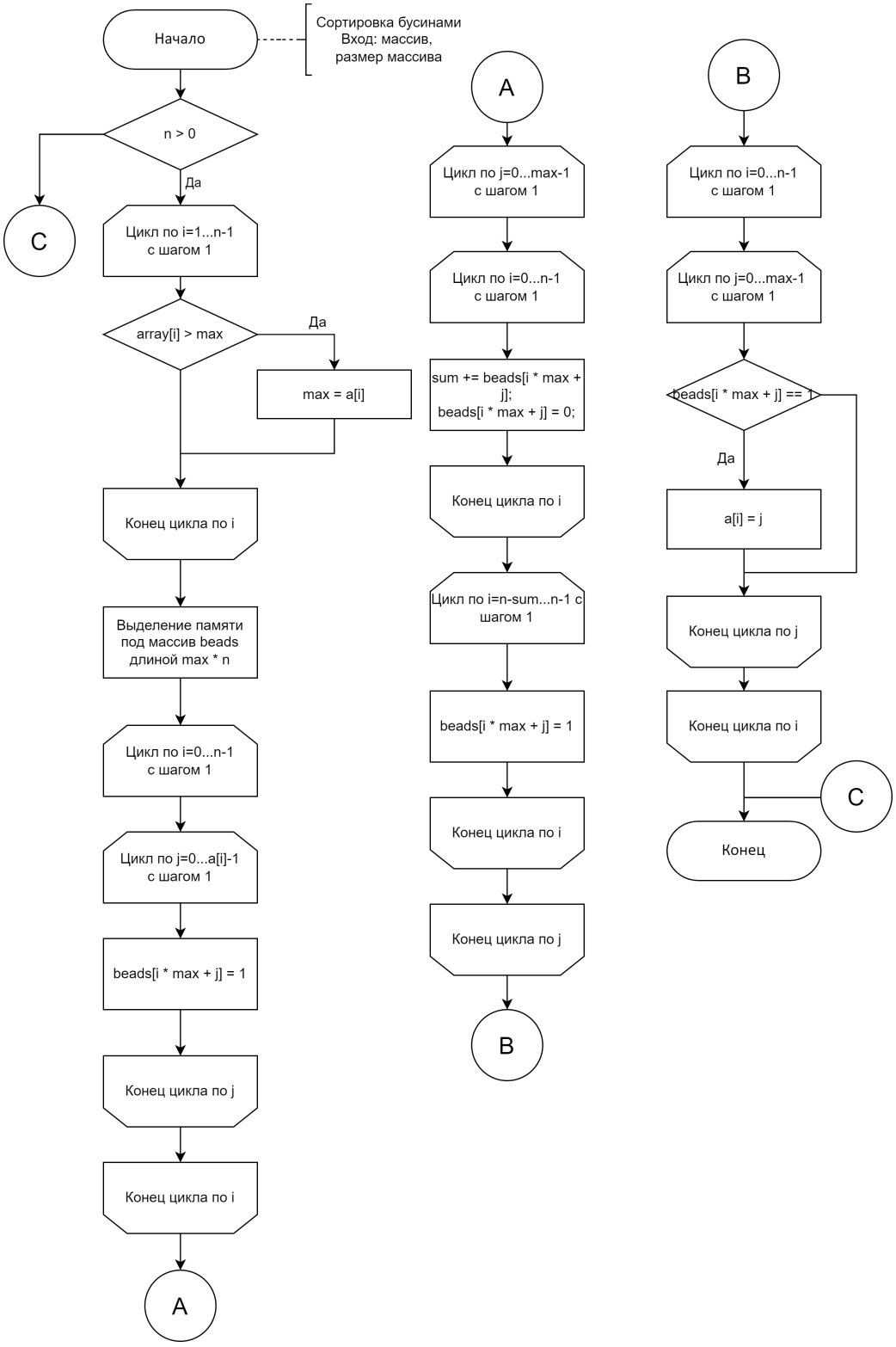

Рисунок 2.4 – Схема алгоритма сортировки бусинами

2.3 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие структуры данных:

— указатель на массив типа $int*$;

— массив типа int;

— длина массива - целое число типа $size\_t$

2.4 Модель вычислений для проведения оценки трудоемкости

Введем модель вычислений, которая потребуется для определения трудоемкости каждого отдельного взятого алгоритма сортировки.

1) Трудоемкость базовых операций имеет:

— равную 1:

$$+, -, =, + =, - =, ==, ! =, <, >, <=, >=, [], ++, --, \\ \&\&, >>, <<, ||, \&, | \tag{2.1}$$

— равную 2:

$$*, /, \%, * =, / =, \% = \tag{2.2}$$

2) Трудоемкость условного оператора:

$$f_{if} = f_{\text{условия}} + \begin{cases} min(f_1, f_2), & \text{лучший случай} \\ max(f_1, f_2), & \text{худший случай} \end{cases} \tag{2.3}$$

3) Трудоемкость цикла:

$$f_{for} = f_{\text{инициализация}} + f_{\text{сравнения}} + M_{\text{итераций}} \cdot (f_{\text{тело}} + \\ + f_{\text{инкремент}} + f_{\text{сравнения}}) \tag{2.4}$$

4) Трудоемкость передачи параметра в функции и возврат из функции равен 0.

2.5 Трудоемкость алгоритмов сортировки

2.5.1 Алгоритм сортировки Шелла

Трудоемкость в лучшем случае при отсортированном массиве, когда ничего не обменивается, но все же данные рассматриваются. Выведена формуле (2.5).

$$\begin{gathered} f_{best} = 3 + 4 + \frac{N}{4} \cdot (3 + 2 + log(N) \cdot (2 + 4 + 4)) = \\ = 7 + \frac{5N}{4} + \frac{5N \cdot log(N)}{2} = O(N \cdot log(N)) \end{gathered} \tag{2.5}$$

Трудоемкость в худшем случае при отсортированном массиве в обратном порядке. Выведена формуле (2.6).

$$\begin{gathered} f_{worst} = 7 + \frac{N}{4} \cdot (5 + log(N) \cdot (10 + log(N) \cdot (6 + 4))) = \\ = 7 + \frac{5N}{4} + \frac{5N \cdot log(N)}{2} + \frac{5N \cdot log^2(N)}{2} = O(N \cdot log^2(N)) \end{gathered} \tag{2.6}$$

2.5.2 Алгоритм пирамидальной сортировки

Трудоемкость в лучшем случае при отсортированном массиве. Выведена формуле (2.7).

$$\begin{gathered} f_{best} = 3 + 4 + \frac{N}{2} \cdot (2 + 1 + f_{heapify\_best}) + \\ +3 + (N-1)(2 + 2 + f_{swap} + 1 + f_{heapify\_best}) \end{gathered} \tag{2.7}$$

Трудоемкость в худшем случае при отсортированном массиве в обратном порядке. Выведена в формуле (2.8).

$$\begin{gathered} f_{worst} = 3 + 4 + \frac{N}{2} \cdot (2 + 1 + f_{heapify\_worst}) + \\ +3 + (N-1)(2 + 2 + f_{swap} + 1 + f_{heapify\_worst}) \end{gathered} \tag{2.8}$$

Трудоемкость перестановки элементов будет равна, формула (2.9)

$$f_{swap} = 1 + 1 + 1 = 3 \tag{2.9}$$

Трудоемкость части программы сортировки пирамиды является функция heapify, формула (2.10).

$$f_{heapify} = 5 + logN \cdot (5 + 3 + 4 + 1 + \begin{cases} 1, \\ 3 + \begin{cases} 1, \\ 1, \end{cases} \end{cases} , \quad + 3 + \begin{cases} 1, \\ 4, \end{cases}) \tag{2.10}$$

Исходя из формулы (2.10) были выведены лучший (2.11) и худший (2.12) случаи функции heapify:

$$f_{heapify\_best} = 5 + logN \cdot (5 + 3 + 4 + 2 + 4) = 5 + 17 \cdot logN = O(logN) \tag{2.11}$$

$$f_{heapify\_worst} = 5 + logN \cdot (5 + 3 + 4 + 5 + 7) = 5 + 24 \cdot logN = O(logN) \tag{2.12}$$

Исходя из выведенных выше, то лучший и худший случаи будут равны:

$$\begin{gathered} f_{best} = 7 + \frac{N}{2} \cdot (3 + 5 + 17 \cdot logN) + 3 + (N-1) \cdot (5 + 3 + \\ +5 + 17 \cdot logN) = 23N + 27N \cdot logN - 13 \cdot logN - 3 = \\ = O(N \cdot logN) \end{gathered} \tag{2.13}$$

$$\begin{gathered} f_{worst} = 7 + \frac{N}{2} \cdot (3 + 5 + 24 \cdot logN) + 3 + (N-1) \cdot (5 + 3 + \\ +5 + 24 \cdot logN) = 23N + 36N \cdot logN - 24 \cdot logN - 3 = \\ = O(N \cdot logN) \end{gathered} \tag{2.14}$$

Таким образов из выведенных результатов в формулах (2.13) и (2.14) можно понять, что лучший и худший случай имеют трудоемкость $O(N \cdot logN)$.

2.5.3 Алгоритм сортировки бусинами

Трудоемкость в лучшем случае при отсортированном массиве и худшем случае при неотсортированном массиве в обратном порядке.Выведена формуле 2.15.

$$\begin{gathered} f_{best} = 3 + 4 + (N-1) \cdot (2 + 2 + \begin{cases} 0, & \\ 2, & \end{cases}) + 1 + 2 + \\ +N \cdot (2 + 3 + L \cdot (3 + 5)) + 2 + M \cdot (2 + 3 + N \cdot (2 + 5 + 5) + \\ +3 + (N - L) \cdot (2 + 5)) + 2 + N \cdot (2 + 8 + 8M) = \\ = S + 19NM + 11N + 8M + 18 = O(S) \end{gathered} \tag{2.15}$$

В формуле (2.15) значения S или NL — сумма всех элементов в изначальном массиве, L — значение каждого элемента в массиве, M — максимальное значение из элементов в массиве.

Исходя из результата формулы (2.15) можно понять, что лучшим слу-

чаем для сортировки будет случай если все значения массива будут соответствовать минимальному значению, а худшим случаем если значение массива будут большие значения.

Вывод

В данном разделе на основе теоретических данных были построены схемы требуемых алгоритмов, выбраны используемые типы данных. Для каждого алгоритма сортировки были выведены трудоемкости худшего и лучшего случаев.

3 Технологическая часть

В данном разделе приведены требования к программному обеспечению, средства реализации, листинг кода и функциональные тесты.

3.1 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык $C++$ [5]. Данный выбор обусловлен наличием у языка встроенной библиотекой измерения процессорного времени и соответствием с выдвинутыми требованиям.

В качестве измерения время использовалось функция *clock()* из библиотеки *ctime* [6].

3.2 Сведения о модулях программы

Данная программа разбита на следующие модули:

— `main.cpp` – файл, содержащий точку входа в программу. В нем происходи общение с пользователем и вызов алгоритмов;

— `sorts.cpp` — файл содержит алгоритмы сортировка.

— `allocate.cpp` — файл содержит функции динамического выделения и очищения памяти для матрицы.

— `generate_array.cpp` – файл содержит функции генерации данных для массива.

— `read_array.cpp` – файл содержит функции чтение массива потока данных.

— `print_array.cpp` – файл содержит функцию вывода массива данных.

— `cpu_time.cpp` — файл содержит функции, замеряющее процессорное время методов сортировки.

В листингах 3.1 – 3.3 реализованы алгоритмы сортировка, а именно сортировка Шелла, пирамидальная сортировка и сортировка бусинами. В листингах 3.4 – 3.6 реализованы алгоритмы выделение, вывода матриц и алгоритм перестановки элементов местами.

Листинг 3.1 – Функция сортировки методом Шелла

```
1  void shell_sorting(int *array, size_t n)
2  {
3      if (array != nullptr && n > 0)
4          for (int s = n / 2; s > 0; s /= 2)
5              for (int i = s, j; i < n; i++)
6              {
7                  int temp = array[i];
8                  for (j = i; array[j - s] > temp && j >= s; j -=
                       s)
9                  array[j] = array[j - s];
10                 array[j] = temp;
11             }
12 }
```

Листинг 3.2 – Функция пирамидальной сортировки

```
1  void heapify(int *array, size_t i, size_t n) {
2      size_t largest;
3      int done = 0;
4      size_t l, r;
5
6      while((2 * i <= n) && !done) {
7          l = 2 * i;
8          r = 2 * i + 1;
9
10         if (l == n) {
11             largest = l;
12         } else if (array[l] > array[r])
13             largest = l;
14         else
15             largest = r;
16
17         if (array[i] < array[largest]) {
18             swap(array[i], array[largest]);
19             i = largest;
20         }
21         else
22             done = 1;
23     }
24 }
25
26 void heap_sort(int *array, size_t n) {
27     if (array != nullptr && n > 0) {
28         for (int i = n / 2; i >= 0; i--) {
29             heapify(array, i, n - 1);
30         }
31
32         for (int i = n - 1; i >= 1; i--) {
33             swap(array[0], array[i]);
34             heapify(array, 0, i - 1);
35         }
36     }
37 }
```

Листинг 3.3 – Функция сортировки бусинами

```
1  void bead_sort(int *array, size_t n) {
2      if (array != nullptr && n > 0) {
3          int i, j, max, sum;
4          unsigned char *beads;
5
6          for (i = 1, max = array[0]; i < n; i++)
7              if (array[i] > max)
8                  max = array[i];
9
10         beads = static_cast<unsigned char *>(calloc(1, max *
               n));
11
12         for (i = 0; i < n; i++)
13             for (j = 0; j < array[i]; j++)
14             beads[i * max + j] = 1;
15
16         for (j = 0; j < max; j++) {
17             for (sum = i = 0; i < n; i++) {
18                 sum += beads[i * max + j];
19                 beads[i * max + j] = 0;
20             }
21             for (i = n - sum; i < n; i++)
22                 beads[i * max + j] = 1;
23         }
24
25         for (i = 0; i < n; i++) {
26             for (j = 0; j < max && beads[i * max + j]; j++);
27             array[i] = j;
28         }
29         free(beads);
30     }
31 }
```

Листинг 3.4 – Функции выделение и освобождение памяти под массив

```
1  int *allocate_arr(size_t n)
2  {
3      int *arr = nullptr;
4      if (n > 0)
5          arr = static_cast<int *>(calloc(n, sizeof(int)));
6
7      return arr;
8  }
9
10 void free_arr(int *arr, size_t n)
11 {
12     if (n > 0)
13     free(arr);
14 }
```

Листинг 3.5 – Функция вывода print_arr

```
1  void print_array(int *array, size_t n)
2  {
3      for(int i = 0; i < n; i++)
4          std::cout << array[i] << " ";
5      std::cout << std::endl;
6  }
```

Листинг 3.6 – Функция перестановки элементов места swap

```
1  void swap(int &el1, int &el2)
2  {
3      int temp = el1;
4      el1 = el2;
5      el2 = temp;
6  }
```

3.3 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для разработанных алгоритмов сортировки. Все тесты пройдены успешно. Под обозначением "–"

в строках таблицы значит, что данная алгоритм сортировки не может работать с отрицательными числами, а именно сортировка Бусинами работающая только с натуральными числами.

Таблица 3.1 – Функциональные тесты

Массив	**Размер**	**Ожидаемый р-т**	**Фактический результат**	
			Шелла/Пирам.	**Бусинами**
41 67 34 0 69	5	0 34 41 67 69	0 34 41 67 69	0 34 41 67 69
31 57 24 -10 59	5	-10 24 31 57 59	-10 24 31 57 59	-
1 2 3 4 5	5	1 2 3 4 5	1 2 3 4 5	1 2 3 4 5
100 88 76 65 43	5	43 65 76 88 100	43 65 76 88 100	43 65 76 88 100
-59 -33 -66 -100 -31	5	-100 -66 -59 -33 -31	-100 -66 -59 -33 -31	-

Вывод

Были реализованы алгоритмы: сортировка Шелла, пирамидальная сортировка и сортировка бусинами. Проведено тестирование разработанных алгоритмов.

4 Исследовательская часть

4.1 Технические характеристики

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

— Процессор: Intel(R) Core(TM) i5-10300H CPU 2.50 ГГц [7].

— Оперативная память: 16 ГБайт.

— Операционная система: Windows 10 Pro 64-разрядная система версией 21H2 [8].

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, у которого имеется консольное меню для выбора запуска одного или всех алгоритмов или процесс замера по времени работы алгоритмов. На данном примере представляется, что при выборе алгоритма пользователю позволяется выбрать самостоятельный ввод данных или генерацию случайных данных по указанному размеру матрицы после чего выполняется выбранный алгоритм.

```
   Выберите алгоритм:
1 - Сортировка Шелла;
2 - Пирамидальная сортировка;
3 - Сортировка бусинами;
4 - Замерить время сортировок.

Выбор: 2
   Выберите действие:
1 - Сгенирировать массив;
2 - Ввести массив;

Выбор: 1
Введите размер массива: 10
Сгенерированный массив: 41 67 34 0 69 24 78 58 62 64
Результат сортировки: 0 24 34 41 58 62 64 67 69 78
Выбор: 0
```

Рисунок 4.1 – Демонстрация работы программы

4.3 Временные характеристики

Результаты эксперимента замеров по времени приведены в таблицах 4.1 – 4.3.

В таблице 4.1 приведены результаты замеров по времени лучшего случая для сортировки Шелла и пирамидальной сортировки, то есть это отсортированный массив данных по возрастанию. Лучшим случаем для сортировки Бусинами является массив с элементами минимального значе-

ния, размером от 50 до 1000 с различным шагом, то есть [50; 100] с шагом 50 и [100; 1000] с шагом 100.

Таблица 4.1 – Замеры по времени лучшего случая для сортировок, размер которых от 50 до 1000.

Количество элементов	**Время, мс**		
	Шелла	**Пирамидальный**	**Бусинами**
50	0	0	0.04008
100	0	0	0.08016
200	0	0.01002	0.3006
300	0	0.01102	0.60521
400	0	0.03006	0.95691
500	0.01002	0.04008	1.4579
600	0.02004	0.06012	2.0631
700	0.03006	0.07014	2.8858
800	0.04008	0.08016	3.5661
900	0.051	0.1012	4.5541
1000	0.051	0.16132	5.7545

По таблице 4.1 был построен графики 4.2 и 4.3, в котором показан лучший обзор сравнения сортировки Шелла и пирамидальной сортировки. Исходя из этих данных можно понять, что лучшего всего в этом случае работает сортировка Шелла. При этом разница во времени между пирамидальной сортировкой и сортировкой Шелла в 2 раза, а хуже всего работает сортировка бусинами в 100-200 раз, чем остальные сортировки.

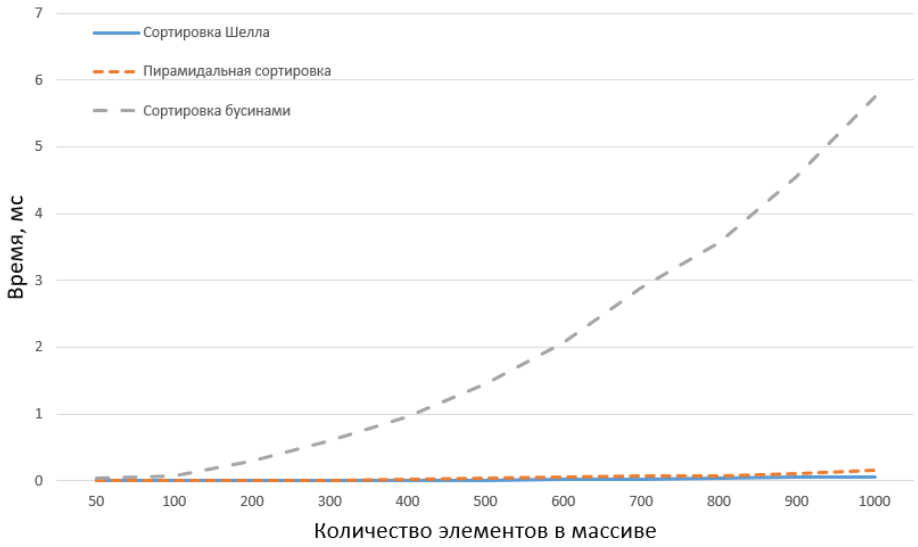

Рисунок 4.2 – Сравнение по времени сортировок Шелла, пирамидальную и бусинами с входным отсортированным массиве по возрастанию

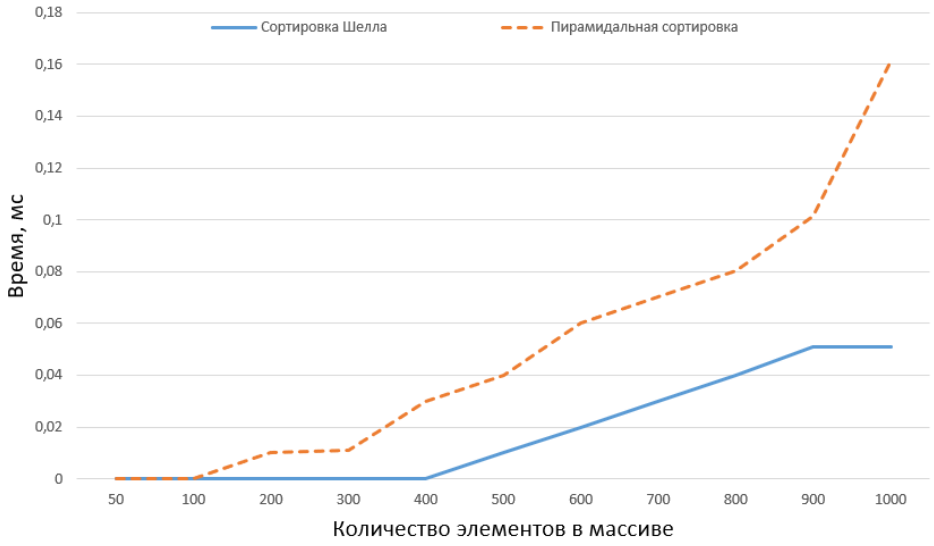

Рисунок 4.3 – Сравнение по времени сортировок Шелла и пирамидальную с входным отсортированным массиве по возрастанию

В таблице 4.2 приведены результаты замеров по времени для сортировки Шелла, пирамидальной сортировки и сортировки бусинами, где на вход поступает отсортированный по возрастанию массив данных размером от 50 до 1000 с различным шагом, то есть [50; 100] с шагом 50 и [100; 1000] с шагом 100.

Таблица 4.2 – Замеры по времени, размер которых от 50 до 1000, с входным случайным массивом

Количество элементов	**Время, мс**		
	Шелла	**Пирамидальный**	**Бусинами**
50	0	0	0.04008
100	0	0	0.08016
200	0	0.01002	0.3006
300	0	0.01102	0.60521
400	0	0.03006	0.95691
500	0.01002	0.04008	1.4579
600	0.02004	0.06012	2.0631
700	0.03006	0.07014	2.8858
800	0.04008	0.08016	3.5661
900	0.051	0.1012	4.5541
1000	0.051	0.16132	5.7545

По таблице 4.2 был построен график 4.4. Исходя из этих данных можно понять, что лучшего всего в этом случае работает сортировка Шелла. При этом разница во времени между пирамидальной сортировкой и сортировкой Шелла незначительна в 1,5 раза, а хуже всего работает сортировка бусинами в 50-100 раз, чем остальные сортировки.

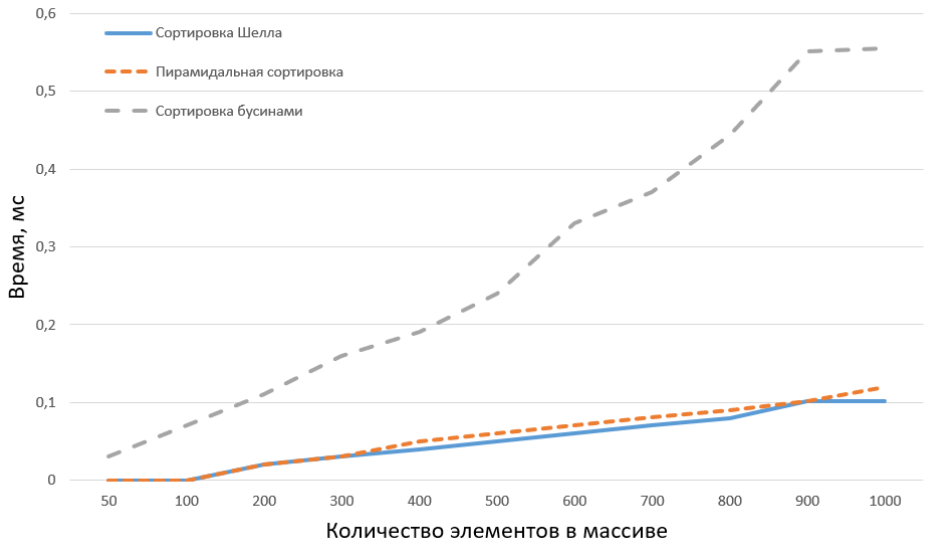

Рисунок 4.4 – Сравнение по времени сортировки Шелла, пирамидальной и бусинами с входным случайным массивом

В таблице 4.3 приведены результаты замеров худшего случая по времени для сортировки Шелла, пирамидальной сортировки и сортировки бусинами, то есть для сортировок Шелла и пирамидальной это случаи, когда

массив отсортирован по убыванию (в обратном порядке сортировки), а для сортировки бусинами учтены большие данные, так как на вход поступает массив целых чисел, то это числа больше 1000. При замерах по времени размер входного массива поступал от 50 до 1000 с различным шагом, то есть [50; 100] с шагом 50 и [100; 1000] с шагом 100.

Таблица 4.3 – Замеры по времени, размер которых от 50 до 1000, с входным отсортированным массивом по убыванию

Количество элементов	**Время, мс**		
	Шелла	**Пирамидальный**	**Бусинами**
50	0.01	0.01002	3.7405
100	0.01002	0.01002	7.4709
200	0.01002	0.02004	15.511
300	0.02004	0.03006	23.198
400	0.0501	0.04008	31.042
500	0.06012	0.0501	38.507
600	0.07014	0.06012	46.405
700	0.08016	0.07014	54.343
800	0.1012	0.08016	64.995
900	0.1206	0.1012	70.902
1000	0.13026	0.11122	80.218

По таблице 4.2 был построен графики 4.5 и 4.6, в котором показан лучший обзор сравнения сортировки Шелла и пирамидальной сортировки. Исходя из этих данных можно понять, что лучшего всего в этом случае работает пирамидальная сортировка. При этом разница во времени между пирамидальной сортировкой и сортировкой Шелла незначительна в 1,5 раза, а хуже всего работает сортировка бусинами в 500 раз, чем остальные сортировки.

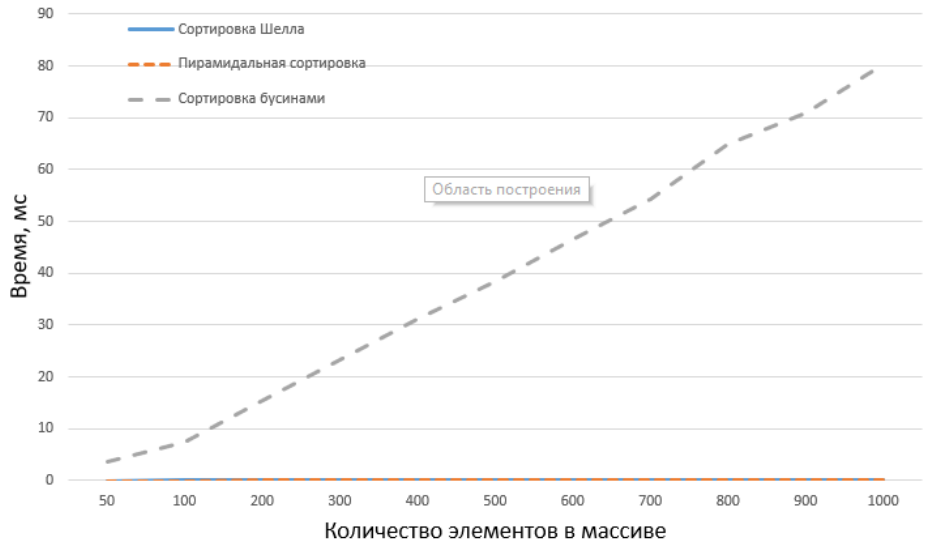

Рисунок 4.5 – Сравнение по времени сортировок Шелла, пирамидальной и бусинами с входным отсортированным массивом по убыванию

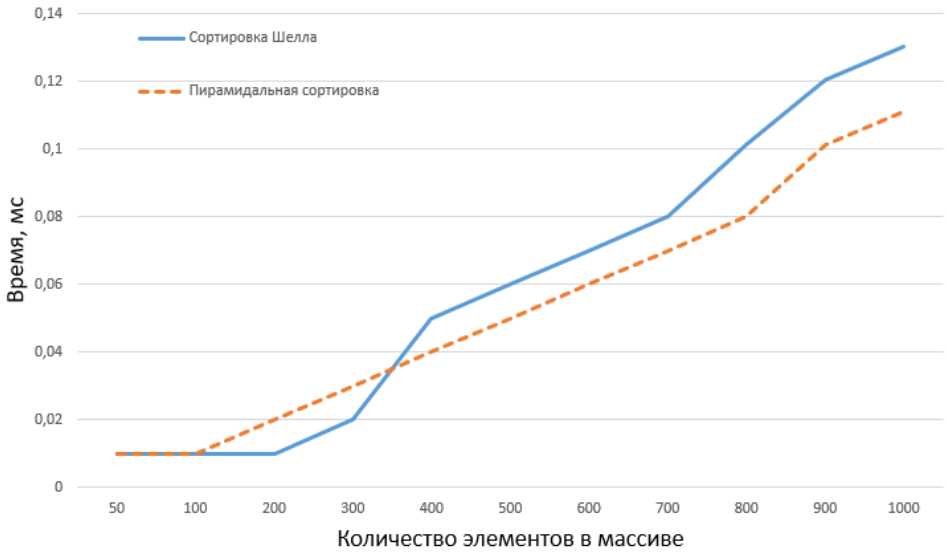

Рисунок 4.6 – Сравнение по времени сортировок Шелла и пирамидальной с входным отсортированным массивом по убыванию

4.4 Вывод

В данном разделе было произведено сравнение количества затраченного времени вышеизложенных алгоритмов. Наименее затратным по времени оказался сортировка Шелла для отсортированного массива по возрастанию и пирамидальная сортировка для отсортированного массива по убыванию, а худшей сортировкой оказалась сортировка бусинами в сравнении с остальными сортировками.

Приведенные временные характеристики показывают, что лучшей сортировкой при работе с отсортированным массивом по возрастанию является сортировка Шелла, а при работе с отсортированном массиве по убыванию (в обратном порядке для сортировки) является пирамидальная сортировка. Сортировка бусинами же показала худшие результаты по сравнению с сортировкой Шелла и пирамидальной в 50-500 раз взависимости от входных данных.

Заключение

Исходя из полученных результатов замеров по времени, сортировка бусинами в любом случае работает дольше, чем сортировка Шелла и пирамидальная сортировка, а именно 100-200 раз хуже. Сравнение результатов замеров по времени сортировки Шелла и пирамидальной сортировки показали, что сортировка Шелла работает 2 раза быстрее, чем пирамидальная сортировка на отсортированных данных, на случайных данных сортировка Шелла работает в 1,5 раза лучше, чем пирамидальная сортировка, но на отсортированных данных по убывания пирамидальная сортировка работает в 1,5 раза лучше, чем сортировка Шелла.

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

Для поставленной цели были выполнены следующие задачи.

1) Описаны расстояния Левенштейна и Дамерау-Левенштейна;
2) Создано программное обеспечение, реализующее следующие алгоритмы сортировки:
   - Шелла;
   - Пирамидальная;
   - Бусинами;
3) Оценены трудоемкости сортировок;
4) Замерено время реализации;
5) Проведен анализ затрат работы программы по времени, выяснить влияющие на них характеристики.

Список использованных источников

1 Липачѐв. Е.К. Технология программирования. Методы сортировки данных: учебное пособие. — Казань: Казанский университет, 2017. — С. 59.

2 Д.В. Шагбазян А.А. Штанюк Е.В. Малкина. Алгоритмы сортировки. Анализ, реализация, применение: учебное пособие. — Нижний Новгород: Нижегородский государственный университет, 2019. — С. 22–25.

3 Д. Кнут. Искусство программирования для ЭВМ. Том 3. Сортировка ипоиск. — М.: ООО «И.Д. Вильямс», 2014. — С. 824.

4 Кормен Т. Лейзерсон Ч. Ривест Р. Алгоритмы: построение и анализ. — М.: МЦНМО, 2000. — С. 1328.

5 Документация по Microsoft C++ [Электронный ресурс]. — Режим доступа: `https://learn.microsoft.com/ru-ru/cpp/?view=msvc-170&viewFallbackFrom=vs-2017` (дата обращения: 25.09.2022).

6 C library function clock() [Электронный ресурс]. — Режим доступа: `https://www.tutorialspoint.com/c_standard_library/c_function_clock.htm` (дата обращения: 25.09.2022).

7 Intel [Электронный ресурс]. — Режим доступа: `https://ark.intel.com/content/www/ru/ru/ark/products/201839/intel-core-i510300h-processor-8m-cache-up-to-4-50-ghz.html` (дата обращения: 25.09.2022).

8 Windows 10 Pro 2h21 64-bit [Электронный ресурс]. — Режим доступа: `https://www.microsoft.com/ru-ru/software-download/windows10` (дата обращения: 25.09.2022).