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: «Трудоемкость сортировок»

Студент группы ИУ7-54Б _____ (Подпись, дата) Спирин М.П. (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 будут рассмотрены сортировки.

Сортировка — перегруппировка некой последовательности или кортежа в определенном порядке [1]. 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 выполняет следующие действия:

— 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 алгоритмов сортировки.

Для поставленной цели необходимо выполнить несколько задач.

1) Рассмотреть алгоритмы сортировок вставками, пирамидальной и битонной.

2) Создать программное обеспечение, реализующее следующие алгоритмы сортировки:

   — битонная;

   — пирамидальная;

   — вставками.

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, сортировка выбором, ее улучшение — пирамидальная сортировка и битонная сортировка.

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.

1.2 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 (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 [2, 1, 3, 4].

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 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.

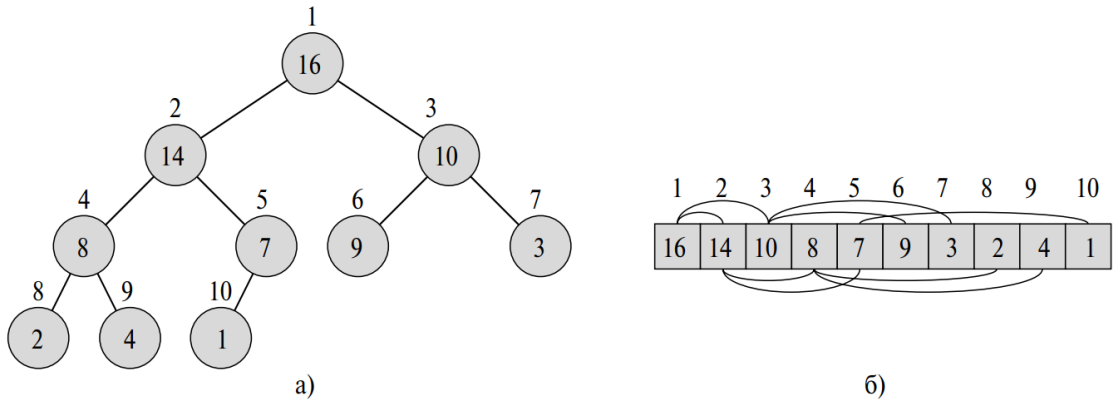

Рисунок 1.1 – Пирамида, представленная в виде а) бинарного дерева и б) массива

Обозначим следующие особенности:

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 Битонная сортировка

Битонная сортировка (англ. Bitonic sorter) — параллельный алгоритм сортировки данных, метод для создания сортировочной сети. Разработан американским информатиком Кеннетом Бэтчером в 1968 году.

Идея данного алгоритма заключается в том, что исходный массив преобразуется в битонную последовательность — последовательность, ко-

торая сначала возрастает, а потом убывает. Ее можно эффективно отсортировать следующим образом: разбить массив на две части, создадь два массива, в первый добавить все элементы, равные минимуму из соответственных элементов каждой из двух частей, а во второй — равные максимуму. В результате получатся две битонные последовательности, каждую из которых можно рекурсивно отсортировать тем же образом, после чего можно объединить два массива (так как любой элемент первого меньше или равен любого элемента второго). Для того, чтобы преобразовать исходный массив в битонную последовательность, можно сделать следующее: если массив состоит из двух элементов, можно просто завершить выполнение, иначе нужно разделить массив пополам, рекурсивно вызвать от половин массива алгоритм, после чего отсортировать первую часть по порядку, вторую в обратном порядке и объединить их. В результате получается битонная последовательность.

Также важно заметить, что размер массива должен быть равен степени двойки.

Вывод

В данном разделе были рассмотрены алгоритмы сортировки - битонная, пирамидальная и вставками. Рассмотренные сортировки следует реализовать, изучить трудоемкость и проанализировать время, необходимое для выполнения.

2 Конструкторская часть

В данном разделе приведены схемы алгоритмов сортировок — битонной, пирамидальной и вставками, приведено описание используемых типов данных.

2.1 Требования к ПО

К программе предъявлен ряд функциональных 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 – 2.6 представлены схемы алгоритмов сортировок, а именно сортировка вставками, пирамидальная сортировка и битонная сортировка.

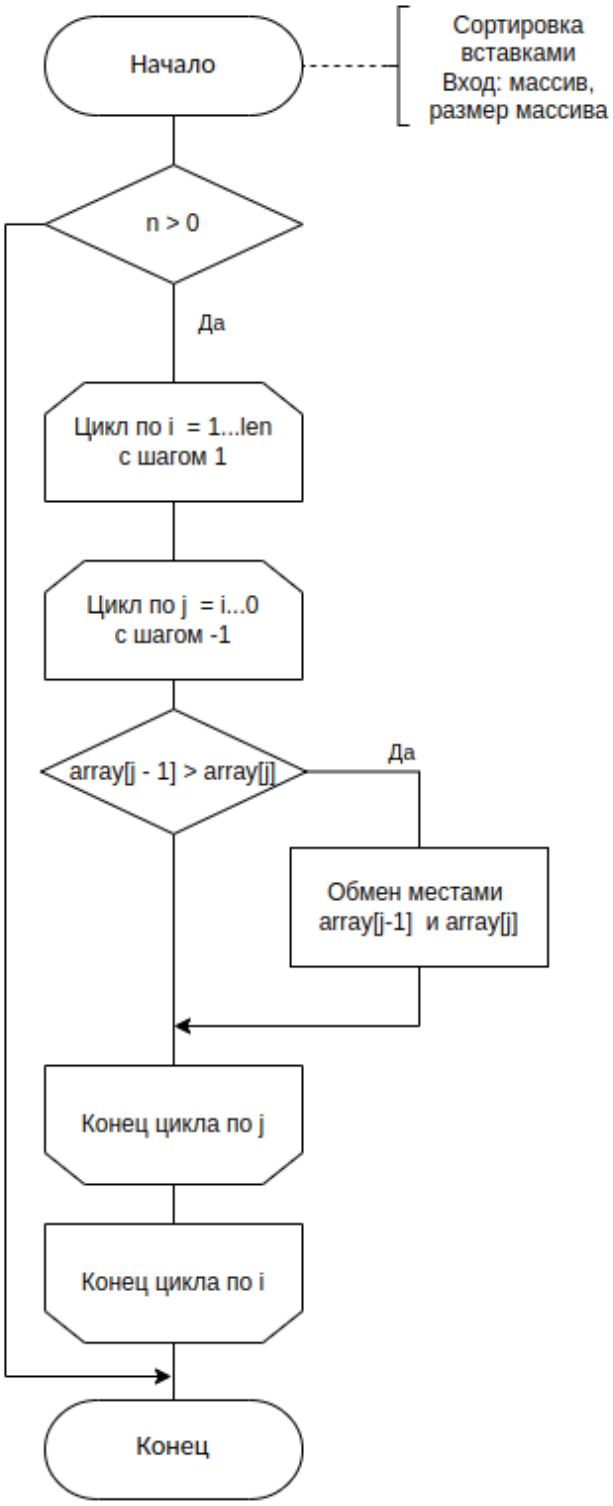

Рисунок 2.1 – Схема алгоритма сортировки вставками

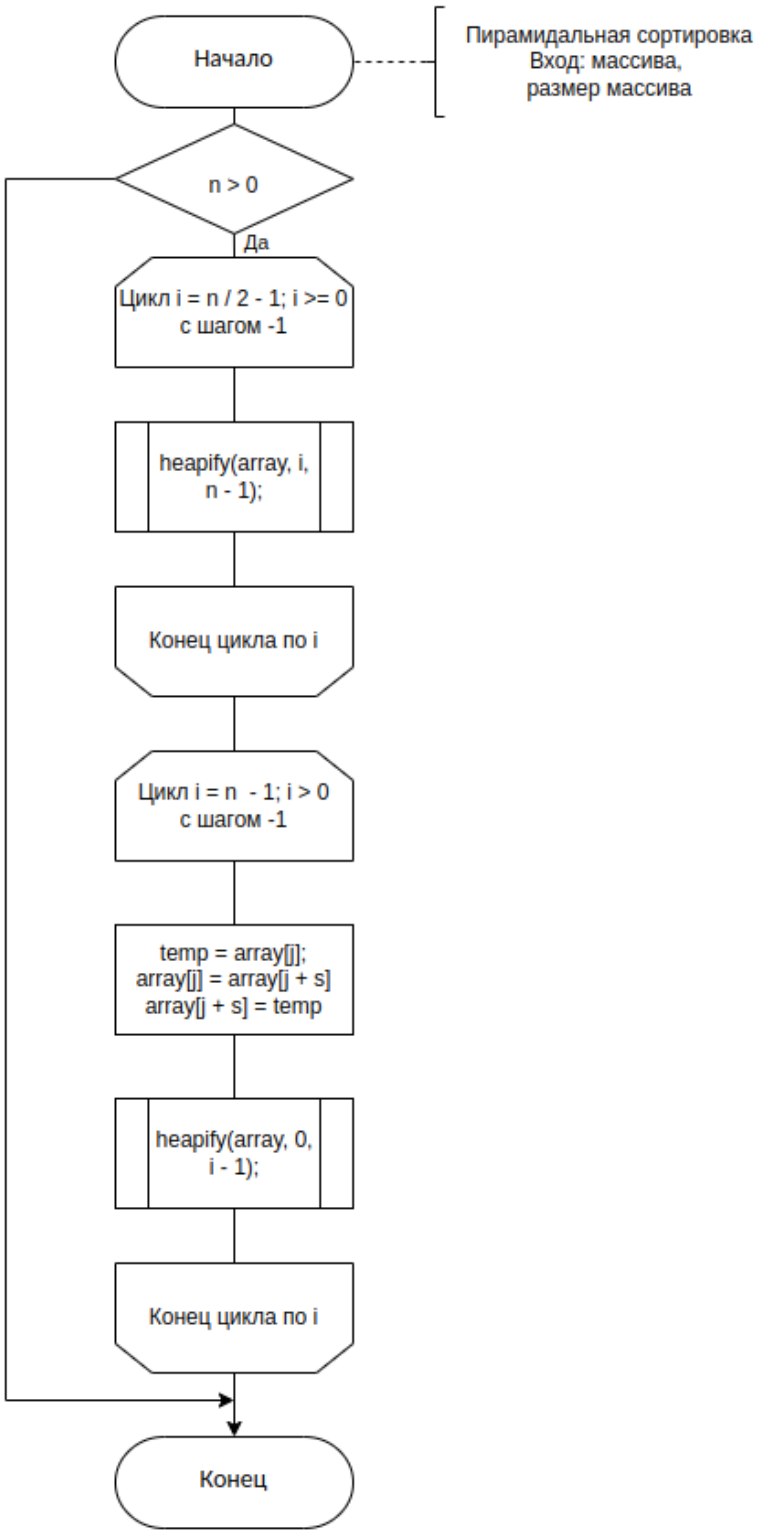

Рисунок 2.2 – Схема алгоритма пирамидальной сортировки

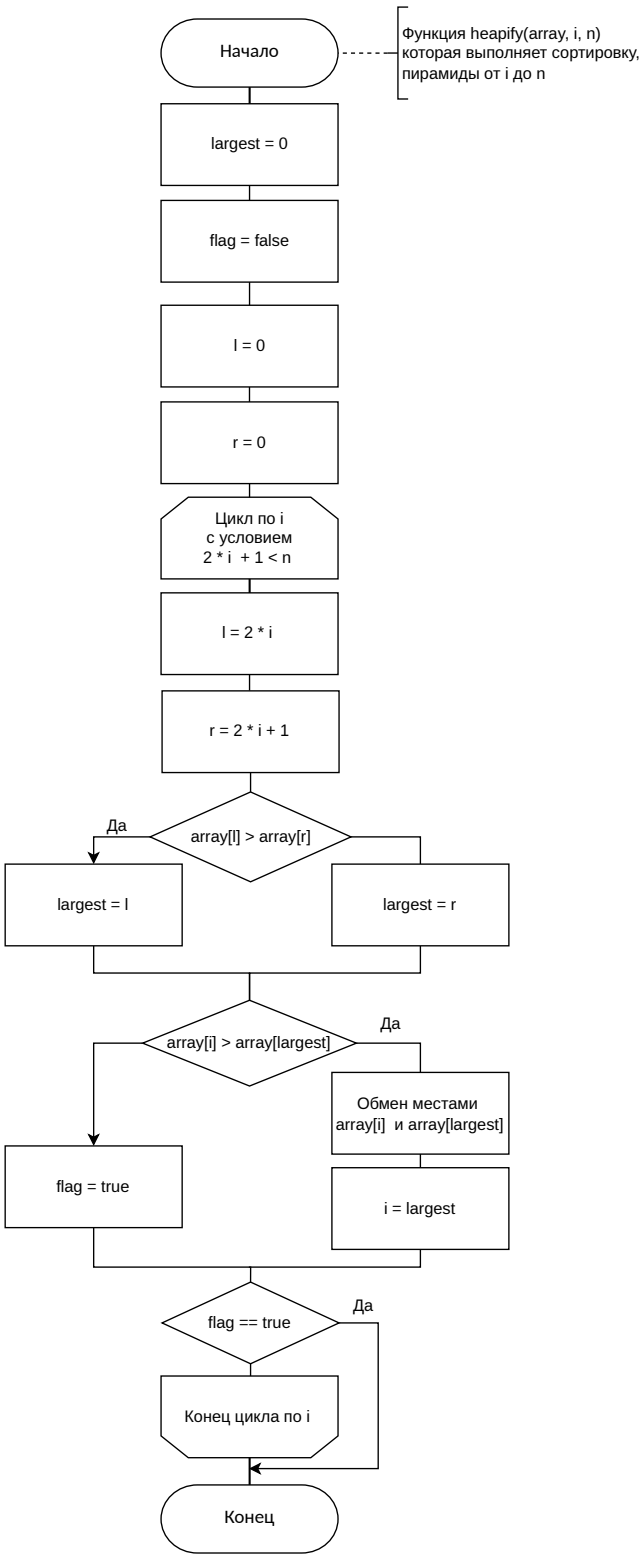

Рисунок 2.3 – Схема алгоритма функции heapify() для пирамидальной сортировки

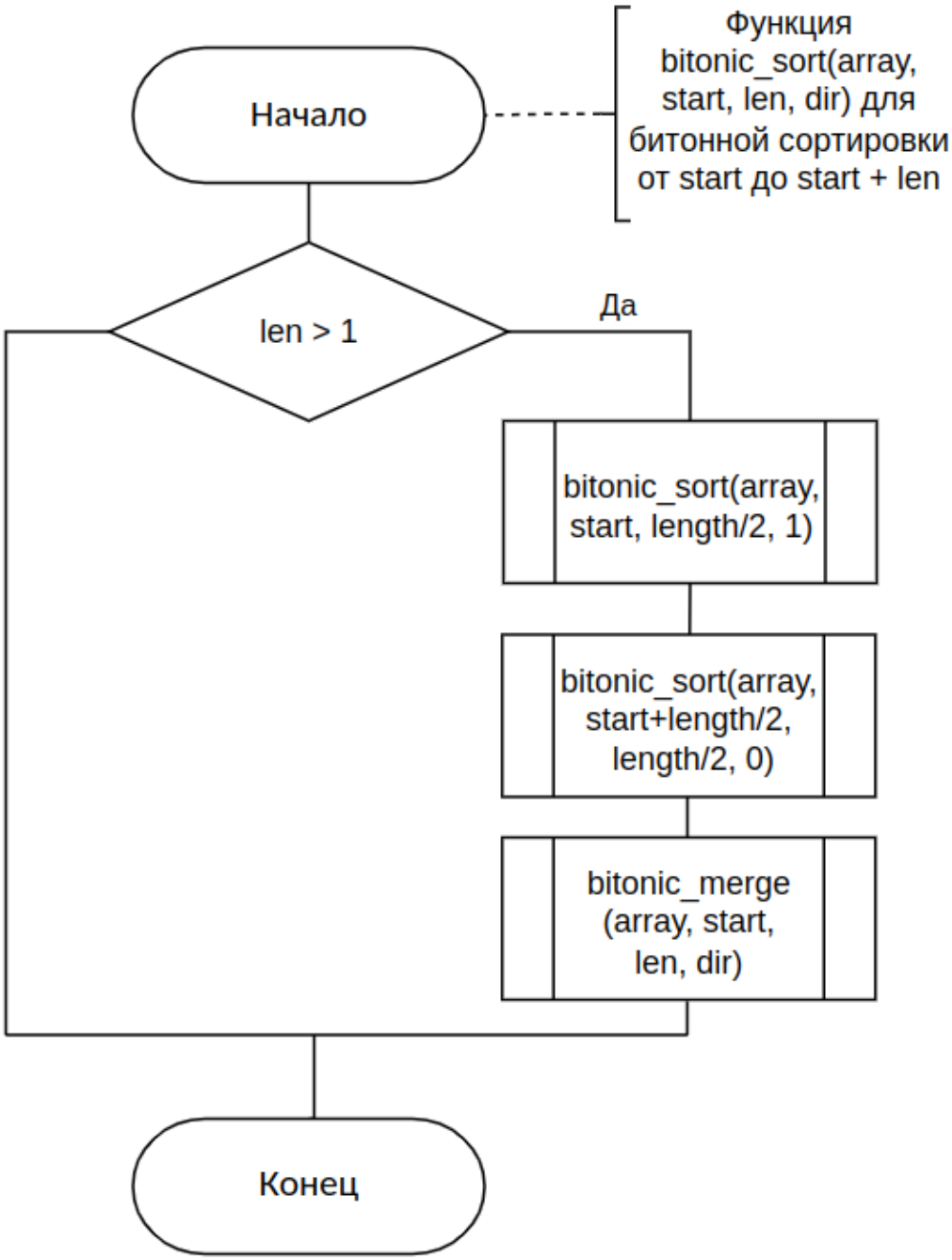

Рисунок 2.4 – Схема алгоритма функции bitonic_sort() для битонной сортировки

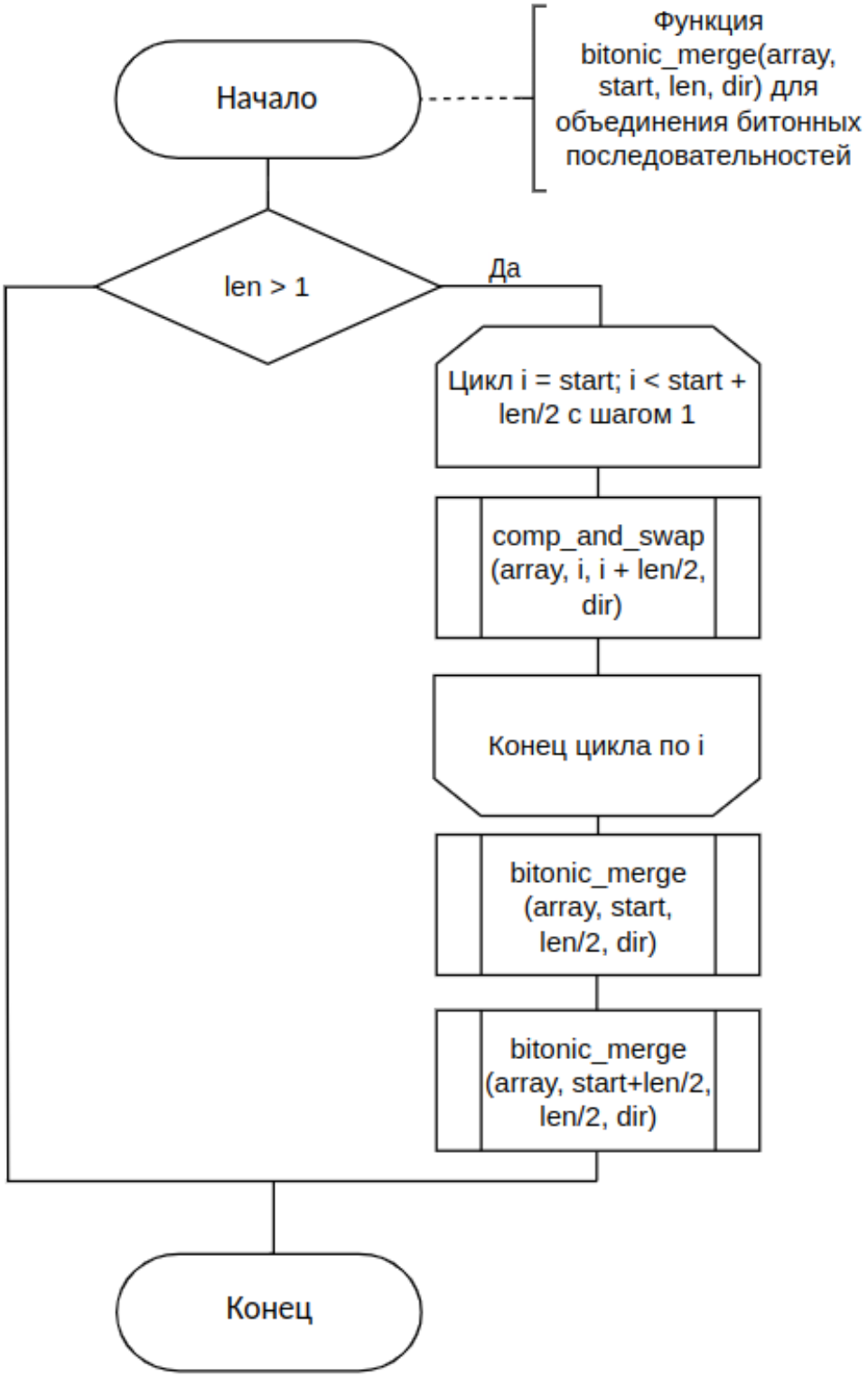

Рисунок 2.5 – Схема алгоритма функции bitonic_merge() для битонной сортировки

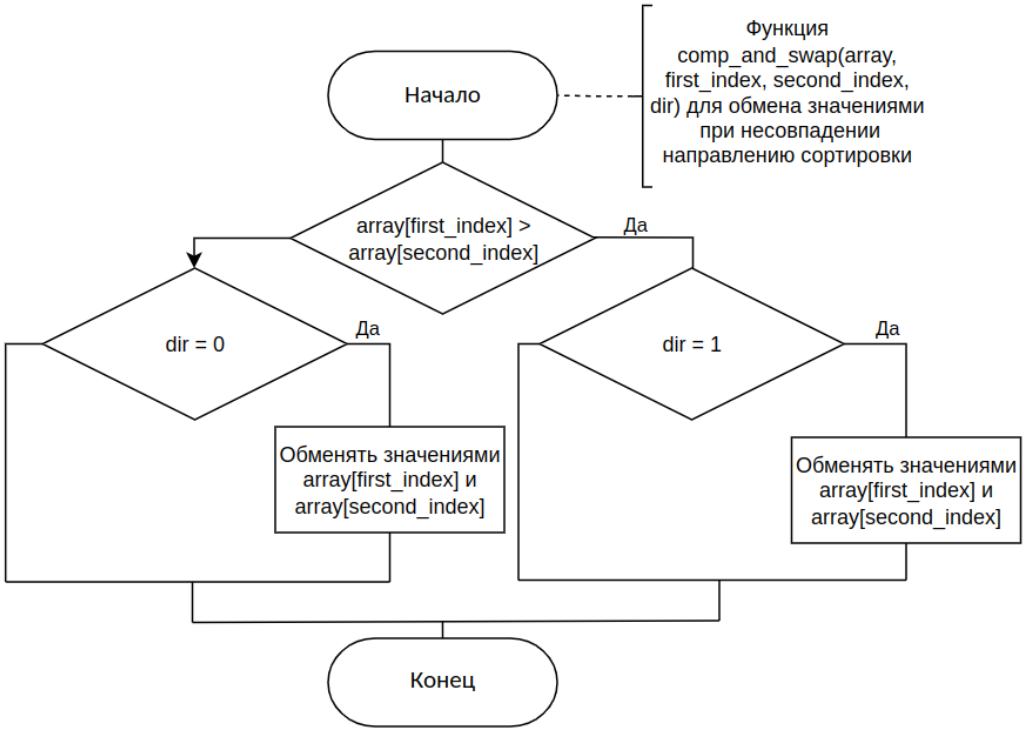

Рисунок 2.6 – Схема алгоритма функции comp_and_swap() для битонной сортировки

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 для оценки трудоемкости

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

1) Трудоемкость базовых операций:

   — для операций $+, -, =, +=, -=, ==, !=, <, >, <=, >=, [], ++, --, \&\&, >>, <<, ||, \&, |$ трудоемкость равна 1;

   — для операций $*, /, \%, *=, /=, \%=$ трудоемкость равна 2.

2) Трудоемкость условного оператора:

$$f_{if} = f_{\text{условия}} + \begin{cases} min(f_1, f_2), & \text{лучший случай,} \\ max(f_1, f_2), & \text{худший случай,} \end{cases} \tag{2.1}$$

   где $f1$ и $f2$ - трудоемкости соответствующих ветвей условного оператора.

3) Трудоемкость цикла:

$$f_{for} = f_{\text{инициализация}} + f_{\text{сравнения}} + M_{\text{итераций}} \cdot (f_{\text{тело}} + \\ + f_{\text{инкремент}} + f_{\text{сравнения}}). \tag{2.2}$$

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

2.5 Трудоемкость алгоритмов сортировки

2.5.1 Алгоритм сортировки вставками

Трудоемкость в лучшем случае при отсортированном массиве. Выведена в следующей формуле:

$$f_{best} = 1 + (N - 1) \cdot 7 = 7 \cdot N - 6 = O(N). \tag{2.3}$$

Трудоемкость в худшем случае при отсортированном массиве в обратном порядке. Выведена в следующей формуле:

$$f_{worst} = 1 + (N - 1) \cdot 8 + \frac{(N - 1) \cdot N \cdot 7}{2} = O(N^2). \tag{2.4}$$

2.5.2 Алгоритм пирамидальной сортировки

Трудоемкость в лучшем случае при отсортированном массиве. Выведена в следующей формуле:

$$\begin{gathered} f_{best} = 3 + 4 + \frac{N}{2} \cdot (2 + 1 + f_{heapify\_best}) + \\ +3 + (N-1)(2 + 2 + f_{swap} + 1 + f_{heapify\_best}). \end{gathered} \tag{2.5}$$

Трудоемкость в худшем случае при отсортированном массиве в обратном порядке. Выведена в следующей формуле:

$$\begin{gathered} f_{worst} = 3 + 4 + \frac{N}{2} \cdot (2 + 1 + f_{heapify\_worst}) + \\ +3 + (N-1)(2 + 2 + f_{swap} + 1 + f_{heapify\_worst}). \end{gathered} \tag{2.6}$$

Трудоемкость перестановки элементов будет равна результату вычисления следующей формулы:

$$f_{swap} = 1 + 1 + 1 = 3. \tag{2.7}$$

Часть трудоемкости пирамидальной сортировки содержится в функции heapify(), формула для ее расчета:

$$f_{heapify} = 5 + logN \cdot (5 + 3 + 4 + 1 + \begin{cases} 1, \\ 3 + \begin{cases} 1, \\ 1, \end{cases} \end{cases} , \quad + 3 + \begin{cases} 1, \\ 4, \end{cases}) \tag{2.8}$$

Исходя из формулы (2.8) были выведены лучший (2.9) и худший (2.10) случаи функции heapify:

$$f_{heapify\_best} = 5 + logN \cdot (5 + 3 + 4 + 2 + 4) = 5 + 17 \cdot logN = O(logN) \tag{2.9}$$

$$f_{heapify\_worst} = 5 + logN \cdot (5 + 3 + 4 + 5 + 7) = 5 + 24 \cdot logN = O(logN) \tag{2.10}$$

Исходя из выведенных выше трудоемкостей, можно вычислить трудоемкость лучшего и худшего случая по формулам:

$$f_{best} = 7 + \frac{N}{2} \cdot (3 + 5 + 17 \cdot logN) + 3 + (N - 1) \cdot (5 + 3 + \\ +5 + 17 \cdot logN) = 23N + 27N \cdot logN - 13 \cdot logN - 3 = \\ = O(N \cdot logN), \tag{2.11}$$

$$f_{worst} = 7 + \frac{N}{2} \cdot (3 + 5 + 24 \cdot logN) + 3 + (N - 1) \cdot (5 + 3 + \\ +5 + 24 \cdot logN) = 23N + 36N \cdot logN - 24 \cdot logN - 3 = \\ = O(N \cdot logN). \tag{2.12}$$

Таким образов из выведенных в формулах (2.11) и (2.12) результатов можно понять, что лучший и худший случай имеют трудоемкость $O(N \cdot logN)$.

2.5.3 Алгоритм битонной сортировки

Пусть a - среднее количество обменов на каждом шаге, причем $0 < a < \frac{N}{2}$, тогда итоговая трудоемкость алгоритма:

$$f_{best} = \frac{logN \cdot (log(N) + 1)}{2} \cdot (\frac{N}{2} \cdot 4 + a \cdot 7) = O((logN)^2 \cdot N) \tag{2.13}$$

Можно заметить, что результирующая трудоемкость не зависит от a, следовательно лучший и худший случай имеют одинаковую трудоемкость $O((logN)^2 \cdot N)$.

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 средства реализации программного обеспечения, листинги 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 библиотеки измерения процессорного времени и соответствием с выдвинутыми требованиям.

Для измерения времени использовалась функция *clock()* из библиотеки *ctime* [6].

3.2 Сведения о файлах программы

Данная программа разбита на следующие файлы:

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

— `src/algs.cpp` — файл содержит алгоритмы сортировка;

— `src/io.cpp` — файл содержит функции для ввода и вывода массива;

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

В листингах 3.1 – 3.3 реализованы алгоритмы сортировки, а именно сортировка вставками, пирамидальная сортировка и битонная сортировка.

Листинг 3.1 – Функция сортировки вставками

```
1  int insert_sort(int *array, int len)
2  {
3      for (int i = 1; i < len; i++)
4      {
5          for (int j = i; j > 0 && array[j - 1] > array[j]; j--)
6          {
7              int tmp = array[j - 1];
8              array[j - 1] = array[j];
9              array[j] = tmp;
10         }
11     }
12     return 0;
13 }
```

Листинг 3.2 – Функция пирамидальной сортировки

```
1  void heapify(int *array, int i, int n)
2  {
3      int largest;
4      int done = 0;
5      int l, r;
6
7      while((2 * i <= n) && !done)
8      {
9          l = 2 * i;
10         r = 2 * i + 1;
11
12         if (l == n)
13         {
14             largest = l;
15         } else if (array[l] > array[r])
16             largest = l;
17         else
18             largest = r;
19
20         if (array[i] < array[largest])
21         {
22             int temp = array[i];
23             array[i] = array[largest];
24             array[largest] = temp;
25             i = largest;
26         }
27         else
28             done = 1;
29     }
30 }
31
32 void heap_sort(int *array, int n)
33 {
34     if (array != nullptr && n > 0)
35     {
36         for (int i = n / 2; i >= 0; i--)
37         {
38             heapify(array, i, n - 1);
39         }
40
```

```
41          for (int i = n - 1; i >= 1; i--)
42          {
43              int temp = array[0];
44              array[0] = array[i];
45              array[i] = temp;
46              heapify(array, 0, i - 1);
47          }
48      }
49  }
```

Листинг 3.3 – Функция битонной сортировки

```
1  void compAndSwap(int *array, int first_index, int second_index,
       int dir)
2  {
3      int temp;
4      if (dir == (array[first_index] > array[second_index]))
5      {
6          temp = array[first_index];
7          array[first_index] = array[second_index];
8          array[second_index] = temp;
9      }
10 }
11
12 void bitonicMerge(int *array, int start, int len, int dir)
13 {
14     if (len > 1)
15     {
16         int half_len = len / 2;
17         for (int i = start; i < start + half_len; i++)
18             compAndSwap(array, i, i + half_len, dir);
19         bitonicMerge(array, start, half_len, dir);
20         bitonicMerge(array, start + half_len, half_len, dir);
21     }
22 }
23
24 void bitonicSort(int *array, int start, int len, int dir)
25 {
26     if (len > 1)
27     {
28         int half_len = len / 2;
29
30         bitonicSort(array, start, half_len, 1);
31
32         bitonicSort(array, start + half_len, half_len, 0);
33
34         bitonicMerge(array, start, len, dir);
35     }
36 }
37
38 void bitonic_sort(int *array, int length, int up)
39 {
```

```
40      bitonicSort(array, 0, length, up);
41  }
```

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 данный алгоритм сортировки не может работать с числом элементов, не являющимся степенью двойки, что истинно для битонной сортировки.

Таблица 3.1 – Функциональные тесты

Массив	**Размер**	**Ожидаемый р-т**	**Фактический результат**	
			Вст./Пирам.	**Бит.**
41 67 34 0	4	0 34 41 67	0 34 41 67	0 34 41 67
31 57 24 -10	4	-10 24 31 57	-10 24 31 57	-10 24 31 57
1 2 3 4	4	1 2 3 4	1 2 3 4	1 2 3 4
100 88 76 65	4	65 76 88 100	65 76 88 100	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 вставками, пирамидальная сортировка и битонная сортировка. 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) i7-1165G7 CPU 2.80 ГГц.

— Оперативная память: 15 ГБайт.

— Операционная система: Ubuntu 64-разрядная система версии 22.04.3.

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.

```
Выберите команду:
1 - Отсортировать массив с помощью сортировки вставками
2 - Отсортировать массив с помощью пирамидальной сортировки
3 - Отсортировать массив с помощью битонной сортировки
4 - Сравнить время выполнения
5 - Остановить выполнение
1
Введите длину массива: 5
Введите 5 элементов массива через пробел: -1 8 2 14 -6
Отсортированный массив: -6 -1 2 8 14
Выберите команду:
1 - Отсортировать массив с помощью сортировки вставками
2 - Отсортировать массив с помощью пирамидальной сортировки
3 - Отсортировать массив с помощью битонной сортировки
4 - Сравнить время выполнения
5 - Остановить выполнение
```

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 вставками и пирамидальной сортировки, то есть при входном массиве отсортированном по возрастанию. Чтобы сравнить с ними битонную сортировку, использовались только массивы с длиной, равной степенью двойки.

Таблица 4.1 – Замеры по времени лучшего случая для сортировок, размер которых от 32 до 1024.

Количество элементов	**Время, мс**		
	Вставками	**Пирамидальной**	**Битонный**
64	6.427	0.002	4536.000
128	24.148	0.004	10802.000
256	90.998	0.008	25345.000
512	350.311	0.019	62633.000
1024	1420.597	0.060	145597.000

По таблице 4.1 был построен рисунок 4.2, где можно сравнить в лучшем случае сортировки вставками, пирамидальную и битонную. 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 вставками значительна — в 1000-3000 раза, а хуже всего работает битонная сортировка — в 1000-2500 раз медленнее, чем сортировка вставками.

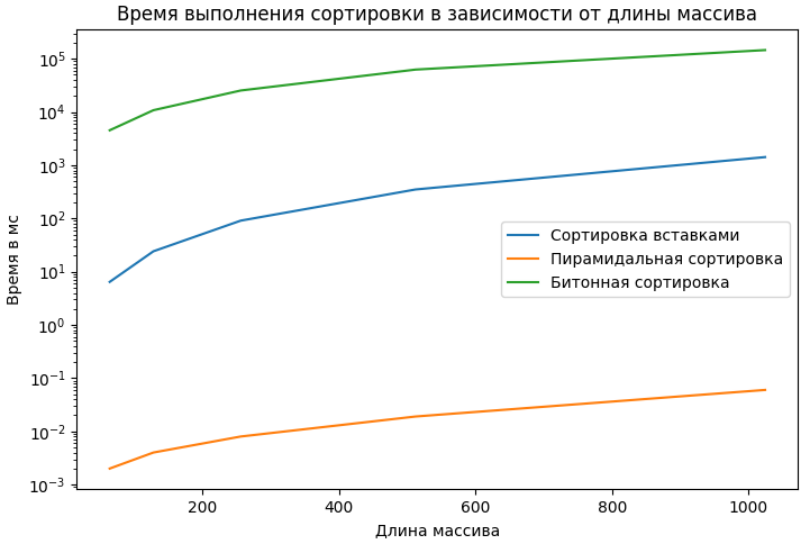

Рисунок 4.2 – Сравнение по времени сортировок вставками, пирамидальной и битонной с входным отсортированным по возрастанию массивом

В таблице 4.2 приведены результаты замеров по времени для сортировки вставками, пирамидальной сортировки и битонной сортировки, где на вход поступает отсортированный по возрастанию массив данных размером, равным степеням двойки от 32 до 1024.

Таблица 4.2 – Замеры по времени, размер которых от 32 до 1024, с входным случайным массивом

Количество элементов	**Время, мс**		
	Вставками	**Пирамидальной**	**Битонной**
64	3.812	0.004	5900.000
128	13.240	0.008	14464.000
256	47.790	0.018	34168.000
512	180.531	0.038	79308.000
1024	767.925	0.083	186057.000

По таблице 4.2 были построен рисунок 4.3. 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 вставками значительна — в 1000-3000 раза, а хуже всего работает битонная сортировка — в 1000-2500 раз медленнее, чем сортировка вставками.

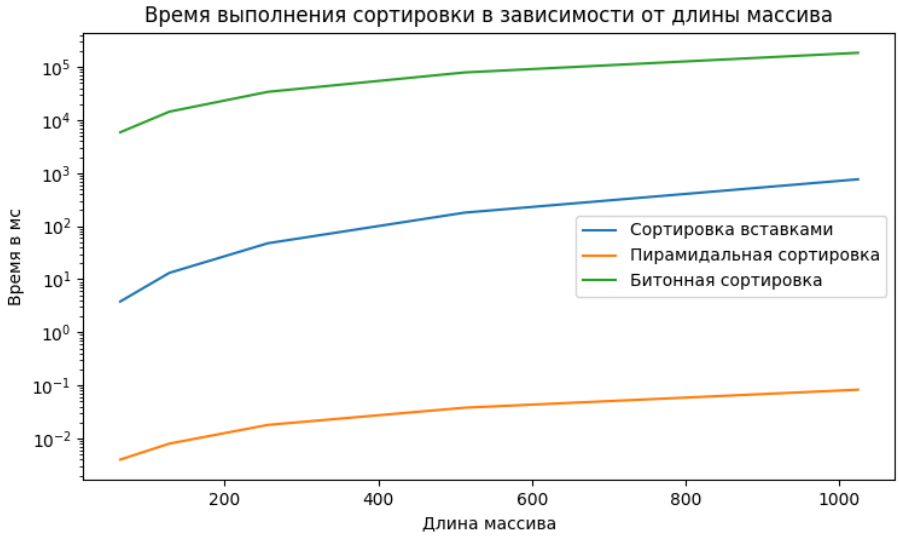

Рисунок 4.3 – Сравнение по времени сортировок вставками, пирамидальной и битонной с входным случайным массивом

В таблице 4.3 приведены результаты замеров худшего случая по времени для сортировки вставками, пирамидальной сортировки и битонной сортировки, то есть для сортировок вставками и пирамидальной это случаи, когда массив отсортирован по убыванию (в обратном порядке сортировки). При замерах по времени поступал массив размером, равным степеням двойки от 32 до 1024.

Таблица 4.3 – Замеры по времени, размер которых от 32 до 1024, с входным отсортированным массивом по убыванию

Количество элементов	**Время, мс**		
	Вставками	**Пирамидальный**	**Битонный**
64	0.434	0.003	6673.000
128	0.427	0.004	10868.000
256	0.687	0.009	25146.000
512	1.232	0.021	58100.000
1024	2.266	0.064	135432.000

По таблице 4.2 были построен рисунок 4.4, в которых сравниваются в худшем случае сортировка вставками и пирамидальная сортировка. 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 вставками значительна — в 1000-3000 раза, а хуже всего работает битонная сортировка — в 1000-2500 раз медленнее, чем сортировка вставками.

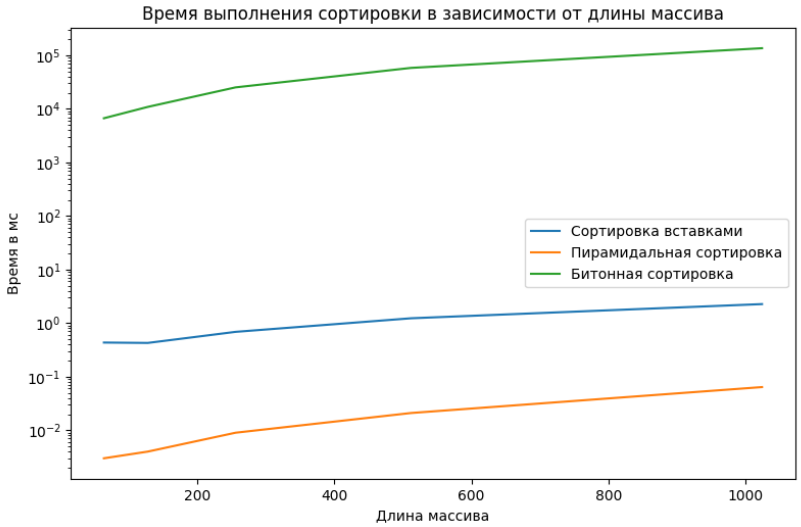

Рисунок 4.4 – Сравнение по времени сортировок вставками, пирамидальной и битонной с входным отсортированным массивом по убыванию

4.4 Вывод

В данном разделе было произведено сравнение количества затраченного времени вышеизложенных алгоритмов. Наименее затратным по времени оказалась пирамидальная сортировка для отсортированного массива по возрастанию и пирамидальная сортировка для отсортированного массива по убыванию, а худшей сортировкой оказалась битонная сортировка.

Приведенные временные характеристики показывают, что лучшей сортировкой при работе с массивом является пирамидальная сортировка. Битонная сортировка же показала худшие результаты по сравнению с сортировкой вставками и пирамидальной, требуя 1000-3000 больше времени в зависимости от входных данных.

Заключение

Исходя из полученных результатов замеров по времени, битонная сортировка в любом случае работает дольше, чем сортировка вставками и пирамидальная сортировка, а именно 1000-2500 раз медленнее. Сравнение результатов замеров времени, необходимого для работы сортировки вставками и пирамидальной сортировки показали, что сортировка вставками в любом случае работает в 1000-2500 раз медленнее, чем пирамидальная сортировка.

Цель лабораторной работы была выполнена. Для поставленной цели были выполнены все задачи.

1) Рассмотрены алгоритмы сортировок вставками, пирамидальной и битонной.

2) Создано программное обеспечение, реализующее следующие алгоритмы сортировки:

   — вставками;

   — пирамидальная;

   — битонная.

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. 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.

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

3. Д. Кнут. Искусство программирования для ЭВМ. Том 3. Сортировка и поиск. — 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).