СОДЕРЖАНИЕ

Введение 3

1 Аналитический раздел 4

2 Конструкторский раздел 6

2.1 Схемы алгоритмов 6

2.2 Трудоёмкость алгоритмов 11

2.2.1 Сортировка пузырьком с флагом 11

2.2.2 Сортировка вставками 11

2.2.3 Пирамидальная сортировка 12

2.3 Выводы 12

3 Технологический раздел 13

3.1 Средства реализации 13

3.2 Листинг кода 13

3.3 Проведение тестирования 15

4 Исследовательский раздел 16

4.1 Сравнительный анализ на основе замеров времени работы алгоритмов 16

Заключение 19

Список использованных источников 20

ВВЕДЕНИЕ

Цель работы: оценить трудоёмкость алгоритмов сортировки на основе сортировки пузырьком с флагом, сортировки выбором и пирамидальной сортировки.

При выполнении лабораторной работы поставлены такие задачи:

1) изучить алгоритмы сортировки;

2) дать теоретическую оценку трудоёмкости сортировки пузырьком с флагом, сортировки выбором и пирамидальной сортировки;

3) реализовать три алгоритма;

4) провести их сравнительный анализ.

1 Аналитический раздел

В данном разделе будет представлено понятие сортировки, рассмотрены некоторые алгоритмы.

Алгоритм сортировки - алгоритм упорядочивания элементов в списке. Исходными данными для задачи сортировки является последовательность чисел $\langle a_1,a_2,\ldots,a_n\rangle$. Результатом должна стать последовательность $\langle a'_1,a'_2,\ldots,a'_n\rangle$, состоящая из тех же чисел, идущих в неубывающем порядке [1].

Сортировка пузырьком с флагом

Принцип действий заключается в обходе массив от начала до конца, попутно меняя местами неотсортированные соседние элементы. В результате первого прохода на последнее место будет перемещён максимальный элемент. После каждой итерации длина неотсортированной части массива уменьшается на 1. Алгоритм переноса максимального элемента в конец продолжается до тех пор, пока длина неотсортированной части не станет равной 1, или пока на очередной итерации не произойдёт ни одного обмена, за контроль чего и отвечает флаг.

Сортировка выбором

На очередной итерации будем находить минимум в массиве после текущего элемента и менять его с ним, если надо. Таким образом, после i-ой итерации первые i элементов будут стоять на своих местах. Нужно отметить, что эту сортировку можно реализовать двумя способами – сохраняя минимум и его индекс или просто переставляя текущий элемент с рассматриваемым, если они стоят в неправильном порядке. Первый способ оказался немного быстрее, поэтому он и реализован.

Пирамидальная сортировка

Идея данного алгоритма сортировки заключается в построении возрастающей пирамиды – почти заполненного двоичного дерева, в котором значение каждого элемента больше или равно значения всех его потомков [2]. Для сортировки применяется алгоритм просеивания вниз, которая перемещает

выбранный элемент на такую позицию, в которой он не нарушает свойство возрастающей пирамиды. Просеивание вниз:

1) если элемент листовой или его значение $\geqslant$ значение потомков, то конец;

2) меняем местами значения элемента и его потомка, имеющего максимальное значение;

3) выполняем процедуру просеивания вниз для изменившегося потомка.

Вывод

В данном разделе были рассмотрены алгоритмы сортировки пузырьком с флагом, выбором и пирамидальная сортировка.

2 Конструкторский раздел

В данном разделе будут рассмотрены схемы алгоритмов сортировки, введена модель вычисления трудоёмкости и в соответствии с ней подсчитана суммарная трудоёмкость для каждого из 3-х алгоритмов.

2.1 Схемы алгоритмов

На рисунках 2.1-2.3 представлены схемы алгоритмов сортировки пузырьком с флагом, выбором и пирамидальной сортировки.

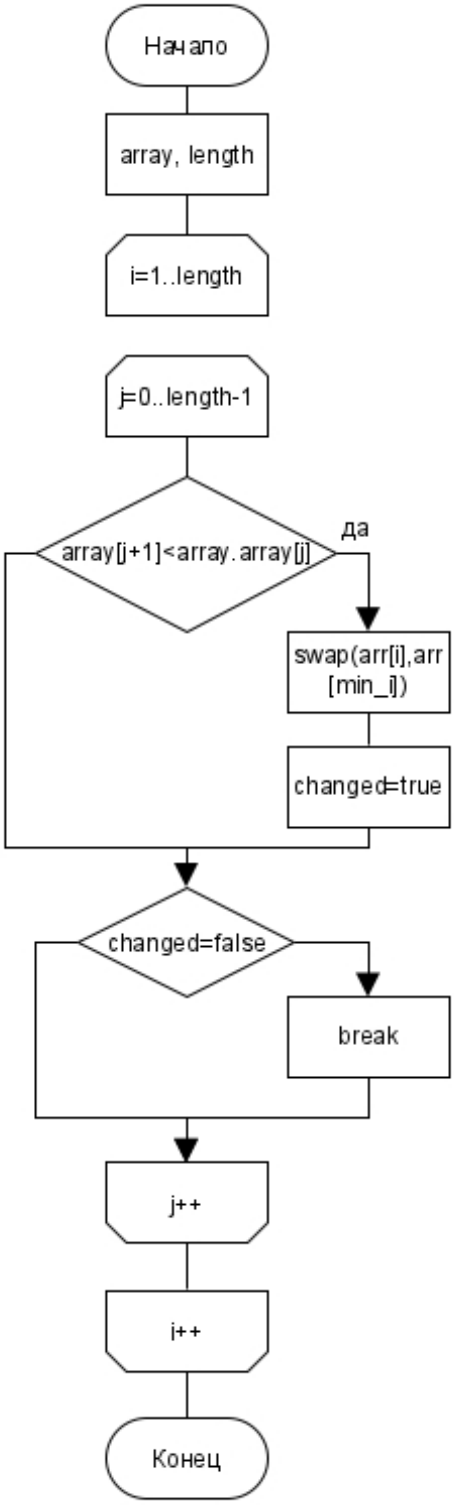

Рисунок 2.1 — Алгоритм сортировки пузырьком с флагом

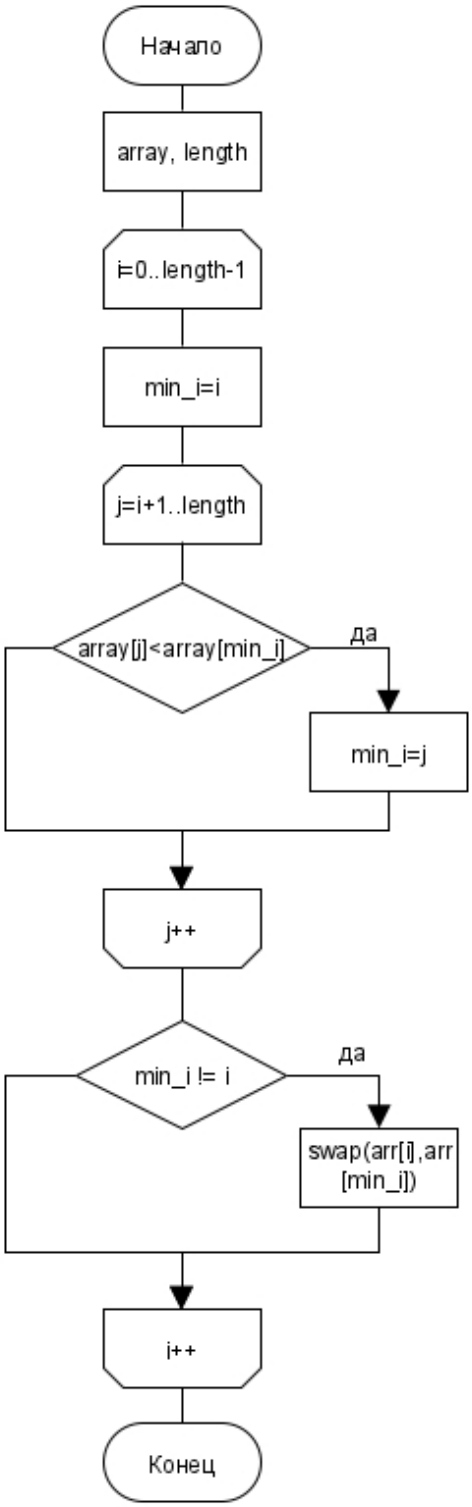

Рисунок 2.2 — Алгоритм сортировки выбором

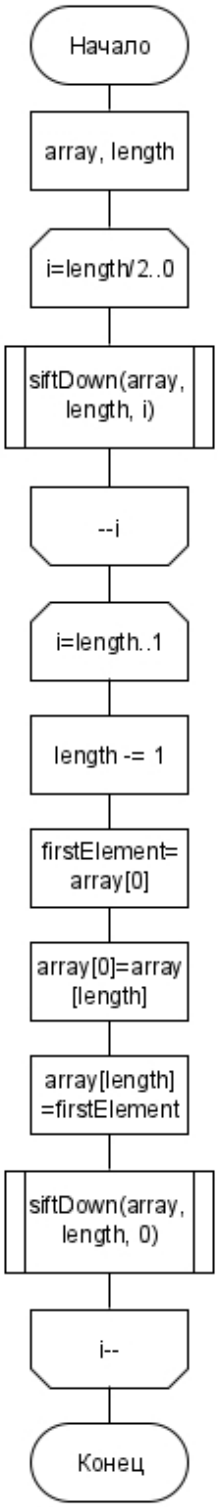

Рисунок 2.3 — Алгоритм пирамидальной сортировки

На рисунке 2.4 представлена схема алгоритма просеивания вниз.

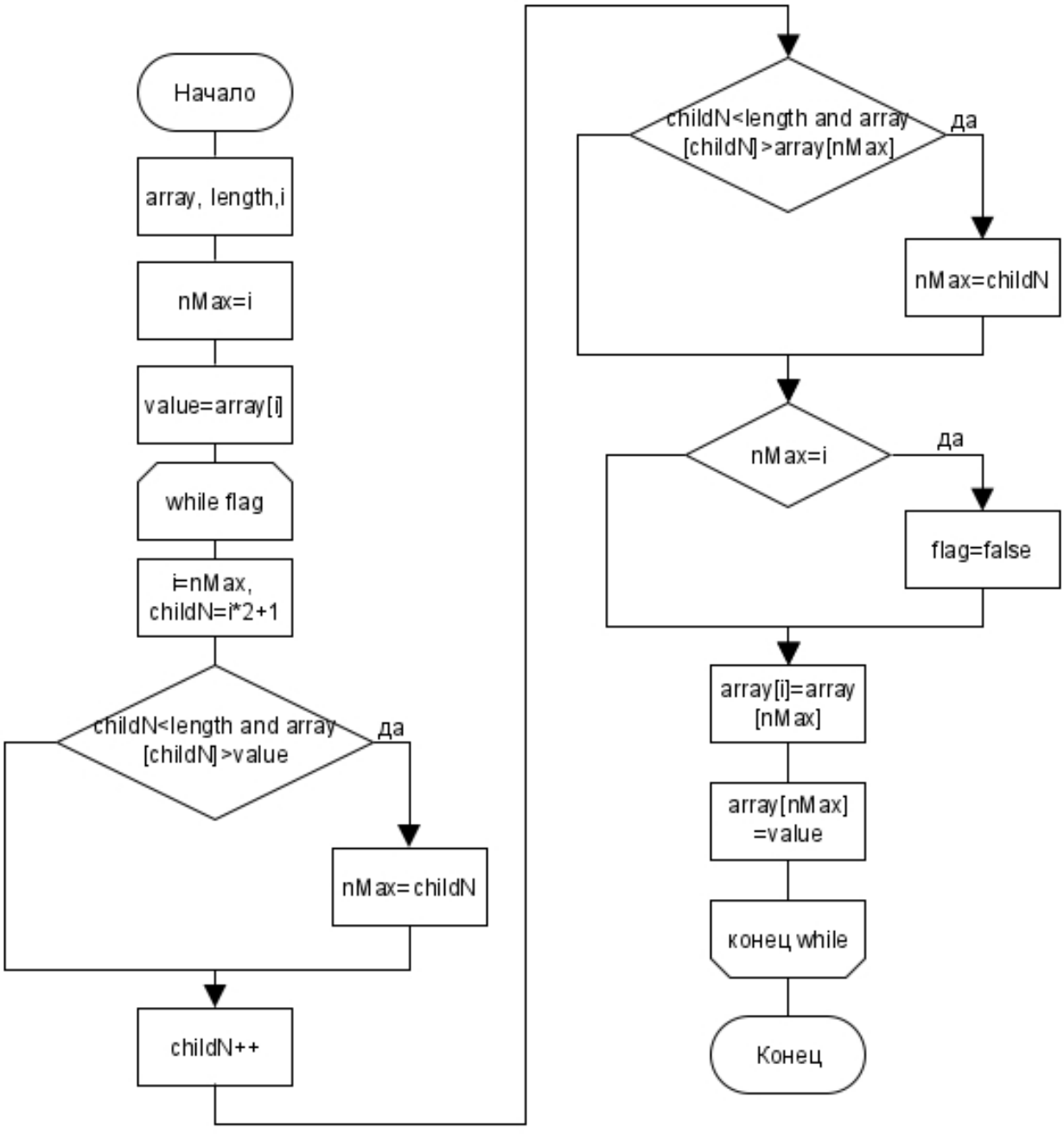

Рисунок 2.4 — Алгоритм просеивания вниз

2.2 Трудоёмкость алгоритмов

Введём модель вычислений трудоёмкости:

1) стоимость базовых операций 1: =,+,-,*,==,!=,<,>,<=,>=,»,+=,-=,*=,/=,[],«

2) оценка трудоёмкости цикла $f_{for} = f_{init} + f_{comp} + N(f_{body} + f_{inc} + f_{comp})$;

3) оценка трудоёмкости условного оператора, стоимость перехода положим 0, тогда

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

2.2.1 Сортировка пузырьком с флагом

Лучший случай: массив, отсортированный в правильном порядке, в данном случае вложенный цикл выполниться всего 1 раз, после чего функция завершится $f = 3 + 3 + (n-1)(3+3) + 1 = 6n + 1$.
Худший случай: массив, отсортированный в обратном порядке, что вызовет необходимость каждый раз производить обмен.

$$\begin{aligned} f &= 3 + (n-1)(3+3) + \frac{1+(n-1)}{2}(n-1)(3+3+5+3+1) = \\ &= 3 + 6n - 6 + \frac{n-1+n^2-2n+1}{2} * 15 = 7.5n^2 - 1.5n - 3 \end{aligned} \tag{2.2}$$

2.2.2 Сортировка вставками

Лучший случай: отсортированный массив, при этом всё равно будут проверяться все элементы массива, но не будет выполнено ни одного обмена. Трудоёмкость составит:

$$\begin{aligned} f &= 3 + (n-1) * (1+2+1+2+1+(\frac{1+n}{2})(2+3)+1) = \\ &= 3 + (n-1) * (8 + 2.5n + 2.5) = 2.5n^2 + 7.5n - 7.5 \end{aligned} \tag{2.3}$$

Худший случай: массив отсортирован в обратном порядке, в связи с чем на каждом шаге внутреннего цикла будет происходить переприсваивание, и

будет происходить обмен:

$$f = 3 + (n-1) * (1 + 2 + 1 + 2 + 1 + (\frac{1+n}{2})(2 + 3 + 1) + 1 + 7) = \\ = 3 + (n-1) * (15 + 3n + 3) = 3n^2 + 15n - 15 \quad (2.4)$$

2.2.3 Пирамидальная сортировка

Трудоёмкость алгоритма сортировки с учётом построения пирамиды будет равно $f = O(n \ln n)$ для лучшего, худшего и среднего случаев.

2.3 Выводы

В данном разделе были рассмотрены схемы алгоритмов сортировки, введена модель вычисления трудоёмкости, рассчитаны трудоёмкости алгоритмов. Из формул видно, что наиболее эффективный в общем случае – алгоритм пирамидальной сортировки, в то время как сортировка пузырьком с флагом наименее трудоёмкая при наилучшем случае.

3 Технологический раздел

В данном разделе будут представлены листинги кода реализованных алгоритмов.

3.1 Средства реализации

В данной работе используется язык программирования C++. Среда разработки Visual Studio Code. Для замера процессорного времени используется функция QueryPerformanceCounter из библиотеки windows.h.

3.2 Листинг кода

В листингах 3.1–3.3 приведены коды алгоритмов сортировки пузырьком с флагом, сортировки выбором и пирамидальной.

Листинг 3.1 — Сортировка пузырьком с флагом

```
1  void bubbleSort(array_t array)
2  {
3      bool changed = false;
4      for (int i = 1; i < array.length; i++)
5      {
6          for (int j = 0; j < array.length - i; j++)
7              if (array.array[j + 1] < array.array[j])
8              {
9                  const int temp = array.array[j];
10                 array.array[j] = array.array[j + 1];
11                 array.array[j + 1] = temp;
12                 changed = true;
13             }
14         if (!changed)
15             break;
16     }
17 }
```

Листинг 3.2 — Сортировка выбором

```
1  void selectionSort(array_t array)
2  {
3      for(int i=0; i<array.length-1; i++)
4      {
5          int min_i = i;
```

```
6           for(int j=i+1;j<array.length;j++)
7           {
8               if (array.array[j] < array.array[min_i])
9               min_i = j;
10          }
11          if (min_i != i)
12          {
13              int tmp = array.array[i];
14              array.array[i] = array.array[min_i];
15              array.array[min_i] = tmp;
16          }
17      }
18  }
```

Листинг 3.3 — Пирамидальная сортировка и необходимая для её работы функция siftDown

```
1   void siftDown(array_t array, int i)
2   {
3       int nMax = i;
4       int value = array.array[i];
5       while (true)
6       {
7           i = nMax;
8           int childN = i*2 + 1;
9           if ((childN < array.length) && (array.array[childN] > value))
10              nMax = childN;
11          childN++;
12          if ((childN < array.length) && (array.array[childN] >
                array.array[nMax]))
13              nMax = childN;
14          if (nMax == i)
15              break;
16          array.array[i] = array.array[nMax];
17          array.array[nMax] = value;
18      }
19  }
20
21  void heapSort(array_t array)
22  {
23      for (int i = array.length / 2 -1; i >= 0; --i)
24          siftDown(array, i);
25      while (array.length > 1)
26      {
27          array.length--;
28
29          int firstElement = array.array[0];
```

```
30            array.array[0] = array.array[array.length];
31            array.array[array.length] = firstElement;
32
33            siftDown(array, 0);
34        }
35    }
```

3.3 Проведение тестирования

Изначально было проведено тестирование алгоритма сортировки пузырьком на массивах: [1,2,1,1,1], [5,4,3,2,1], [1,2,3,4,5], [5,1,2,4,3], все тесты были пройдены успешно.

После чего была проведена серия тестов на массивах длиной от 1 до 10. Сначала выполнялась сортировка пузырьковым алгоритмом, результат которого принимался за контрольное значение, после чего результаты сортировки алгоритмом выбора и пирамидальной сортировки сравнивались с контрольным значением и при совпадении результат считался корректным. Массивы заполнялись случайным образом и в возрастающем порядке. Все тесты были пройдены успешно.

4 Исследовательский раздел

В данном разделе будет измерено время работы алгоритмов и сделаны выводы на основе полученных данных.

4.1 Сравнительный анализ на основе замеров времени работы алгоритмов

Для каждого из алгоритмов был проведён замер времени на массивах длиной от 5 до 50000 тысяч, с шагом 5000, для каждой длины были проведены измерения на массивах, отсортированных в убывающем, возрастающем порядках и заполненных случайными элементами. Для каждого теста было проведено 10 замеров, на графиках представлен усреднённый результат.

На рисунке 4.1 представлено сравнение времени работы алгоритмов при случайно заполненных массивах. На рисунках 4.1–4.3 время, обозначаемое t представлено в миллисекундах.

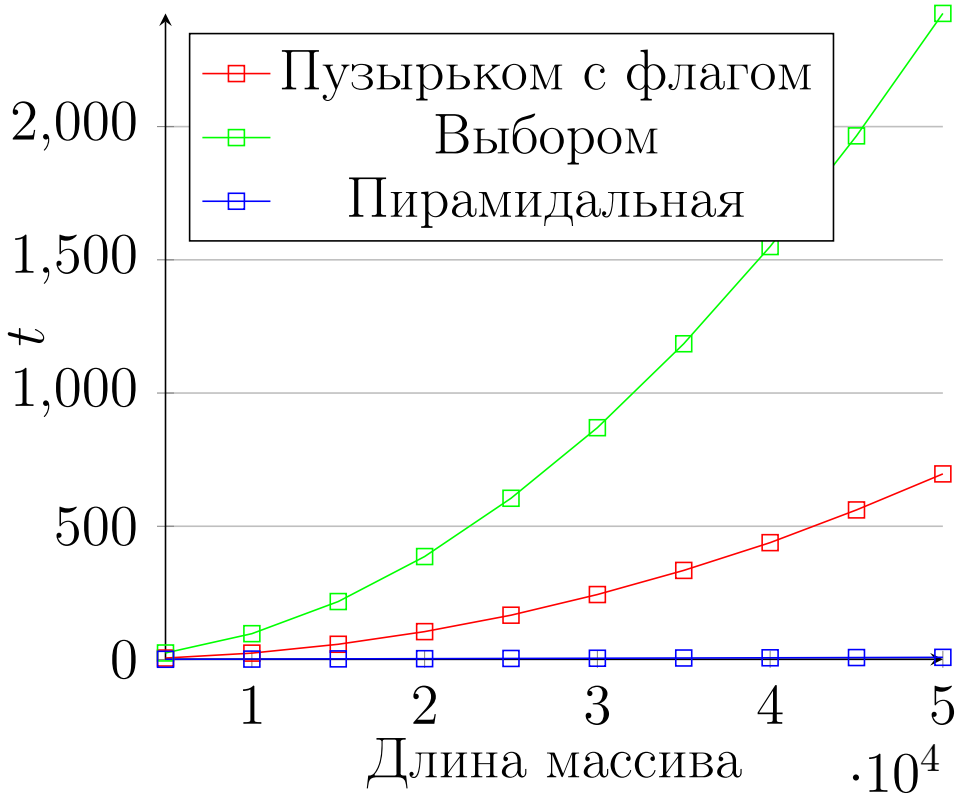

Рисунок 4.1 — Сравнение времени работы алгоритмов при случайно заполненных массивах

На рисунке 4.2 представлено сравнение времени работы алгоритмов при массивах, заполненных в возрастающем порядке.

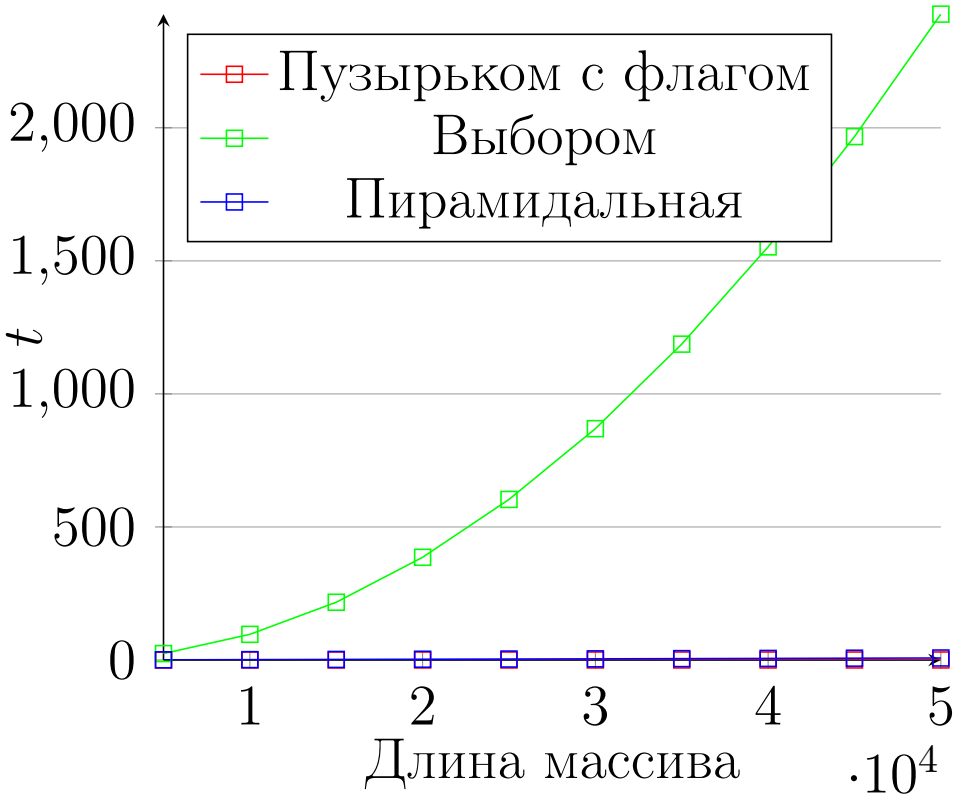

Рисунок 4.2 — Сравнение времени работы алгоритмов при массивах отсортированных в возрастающем порядке

На рисунке 4.3 представлено сравнение времени работы алгоритмов при массивах, заполненных в убывающем порядке.

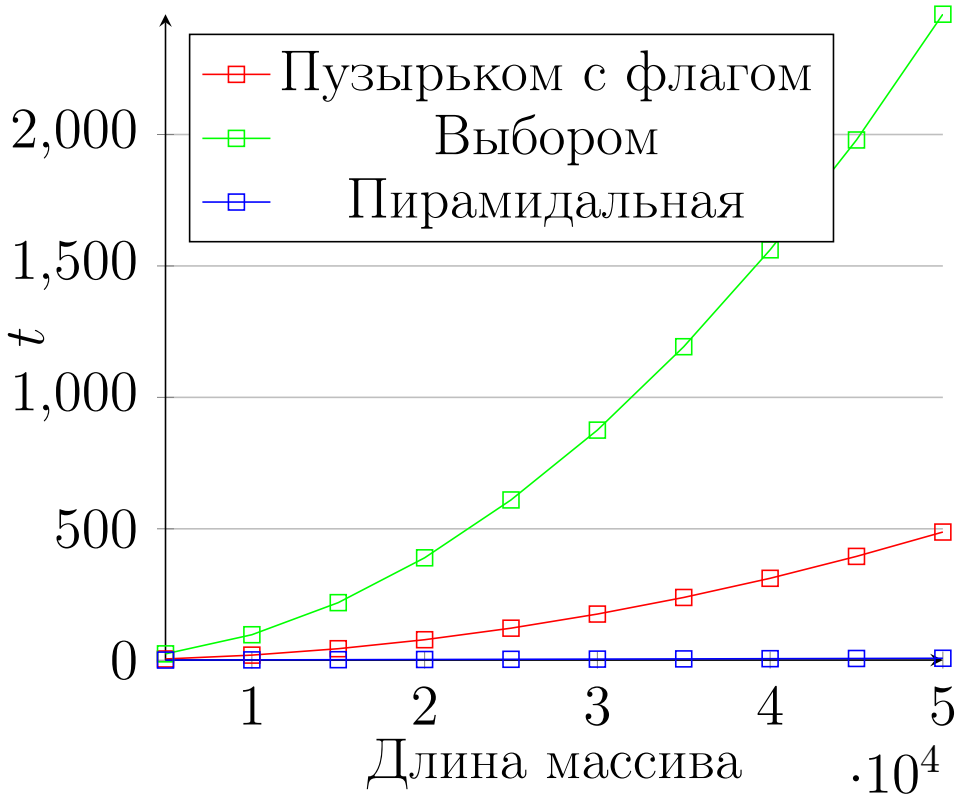

Рисунок 4.3 — Сравнение времени работы алгоритмов при массивах отсортированных в убывающем порядке

Из рисунков 4.1–4.3 видно, что наиболее эффективным по времени в случаях с случайно заполненными массивами и массивами, заполненными в убывающем порядке оказался алгоритм пирамидальной сортировки, в то время как сортировка пузырьком с флагом самая быстрая при массивах, отсортированных в возрастающем порядке.

Таким образом, данные, полученные в результате проведённых экспериментов подтверждают корректность рассчитанных ранее трудоёмкостей алгоритмов.

Вывод

В итоге, можно сказать о том, что пирамидальная сортировка является наиболее эффективной по времени среди рассмотренных алгоритмов.

ЗАКЛЮЧЕНИЕ

В результате лабораторной работы цель была достигнута – была дана оценка трудоёмкости алгоритмов сортировки. Все поставленные задачи были выполнены: изучены алгоритмы сортировки пузырьком с флагом, выбором и пирамидальная сортировка, дана их теоретическая трудоёмкость, они были реализованы и проведён их сравнительный анализ.

Было определено, что наиболее эффективный по трудоёмкости и по времени – пирамидальный алгоритм сортировки, что связано с его реализацией структуры дерева, имеющего высоту log(n).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Т. Кормен Ч. Лейзерсон Р.Ривест К. Штайн. Алгоритмы. Построение и анализ. Издание 3-е. — М., 2013. — 1328 с.

2. Лекция 5: Пирамидальная сортировка[Электронный ресурс]. — 2008. — Режим доступа: https://web.archive.org/web/20090315200203/http://iproc.ru/parallel-programming/lection-5/ (дата обращения: 13.10.2020).