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

Дисциплина: Анализ алгоритмов

Тема: Алгоритмы сортировки массивов

Студент: Платонова Ольга

Группа: ИУ7-55Б

Преподаватели: Волкова Л. Л.
Строганов Ю. В.

Москва.
2020 г.

Оглавление

Введение

В данной лабораторной работе требуется изучить и реализовать алгоритмы сортировки массивов; осуществить расчет трудоемкости алгоритмов; выполнить сравнительный анализ по времени и трудоемкости для каждой из реализаций.

В качестве рассматриваемых алгоритмов были выбраны:

- сортировка «пузырьком»;
- сортировка вставками;
- сортировка Шелла.

1. Аналитическая часть

В данном разделе рассматриваются понятия, связанные с сортировкой массивов, а также представлено описание алгоритмов сортировки.

1.1. Сортировка массива

Сортировка — это процесс упорядочивания наборов данных одного типа по возрастанию или убыванию значения какого-либо признака.

Сортировка массивов представляет из себя процесс упорядочивания N элементов: a_1, a_2, …, a_N. Каждый элемент представляет из себя запись a_i, содержащую некоторую информацию и ключ k_i, управляющий процессом сортировки [1].

Сортировка называется *устойчивой*, если она не меняет взаимного расположения элементов с одинаковыми ключами. Так, не будет производиться операция перестановки над элементами a_i, a_j при равенстве их значений ($a_i = a_j$) [2].

При конструировании эффективных алгоритмов, осуществляющих сортировку массивов, требуется использование быстродействующих алгоритмов и минимальное использование дополнительной памяти. В качестве показателя быстродействия алгоритма используют оценку:

1) количества операций присваивания;
2) количества операций сравнения.

1.2. Сортировка «пузырьком»

Сортировка пузырьком — это самый простой алгоритм сортировки, эффективный для массивов небольших размеров. Алгоритм состоит в повторяющихся проходах по сортируемому массиву. На каждой итерации последовательно сравниваются соседние элементы, и, если порядок в паре неверный, то элементы меняют местами. За каждый проход по массиву как минимум один элемент встает на свое место, поэтому необходимо совершить не более n−1 проходов, где n размер массива, чтобы отсортировать массив.

Существует множество более эффективных реализаций, однако в данной работе будет рассмотрен классический алгоритм.

1.3. Сортировка вставками

Сортировка вставками — это простой алгоритм сортировки, имеющий простейшую реализацию. Суть его заключается в том, что 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.

Данный алгоритм может быть оптимизирован при помощи использования бинарного поиска для нахождения места текущему элементу, а также при помощи замены сдвига части массива на смену указателей. В данной работе будет рассмотрена классическая реализация.

1.4. Сортировка Шелла

Сортировка Шелла — алгоритм сортировки, являющийся усовершенствованным вариантом сортировки вставками. Иными словами — это сортировка вставками с предварительными «грубыми» проходами.

Идея сортировки Шелла состоит в сравнении и сортировки сначала значений, стоящих один от другого на некотором расстоянии d, после чего процедура повторяется для некоторых меньших значений d. Завершается сортировка Шелла упорядочиванием элементов при d = 1. Значения d в данной работе рассчитывается по формуле Седжвика — формула 1.

$$d_i = \begin{cases} 9 * 2^i - 9 * 2^{i/2} + 1, & i — \text{ четное} \\ 8 * 2^i - 6 * 2^{(i+1)/2} + 1, & i — \text{ нечетное} \end{cases} \quad (1)$$

Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места. Невзирая на то, что сортировка Шелла во многих случаях медленнее, чем быстрая сортировка, она имеет ряд преимуществ:

- отсутствие потребности в памяти под стек;
- отсутствие деградации при неудачных наборах данных.

1.5. Вывод

В данном разделе были рассмотрены следующие алгоритмы сортировок массива:

1. сортировка «пузырьком» — самый простой алгоритм, не применяющийся на практике, но лежащий в основе более сложных алгоритмов;
2. сортировка вставками — самый простой в реализации алгоритм;
3. сортировка Шелла — модифицированный алгоритм сортировки вставками, имеющий преимущество по памяти.

2. Конструкторская часть

В данном разделе будут рассмотрены схемы для каждого из алгоритмов.

2.1. Разработка алгоритмов

На рисунке 1 изображена схема алгоритма *сортировки «пузырьком»*.

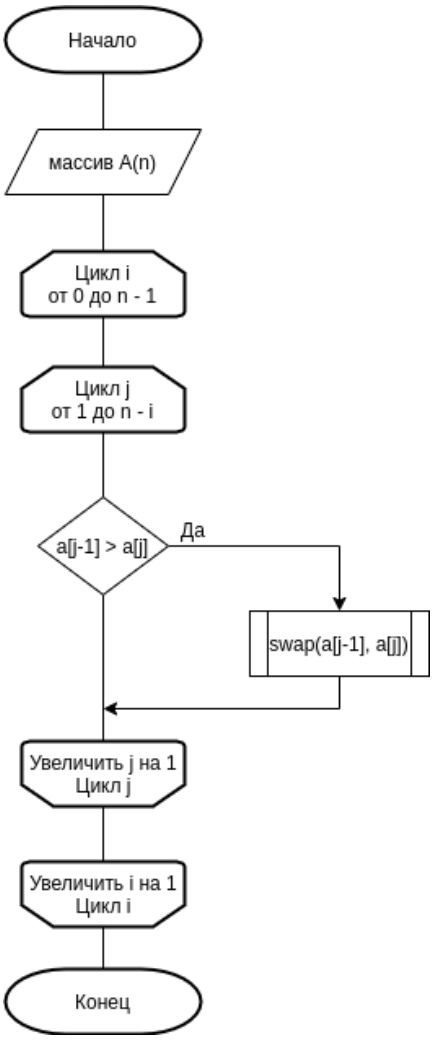


Рисунок 1. Схема алгоритма сортировки «пузырьком».

На рисунке 2 изображена схема алгоритма *сортировки вставками*.

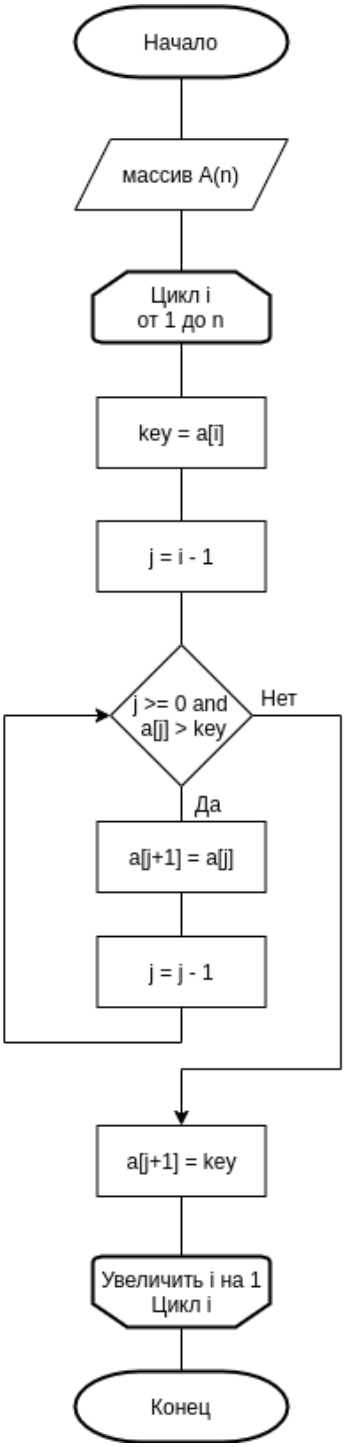


Рисунок 2. Схема алгоритма сортировки вставками.

На рисунке 3 изображена схема алгоритма *сортировки Шелла*.

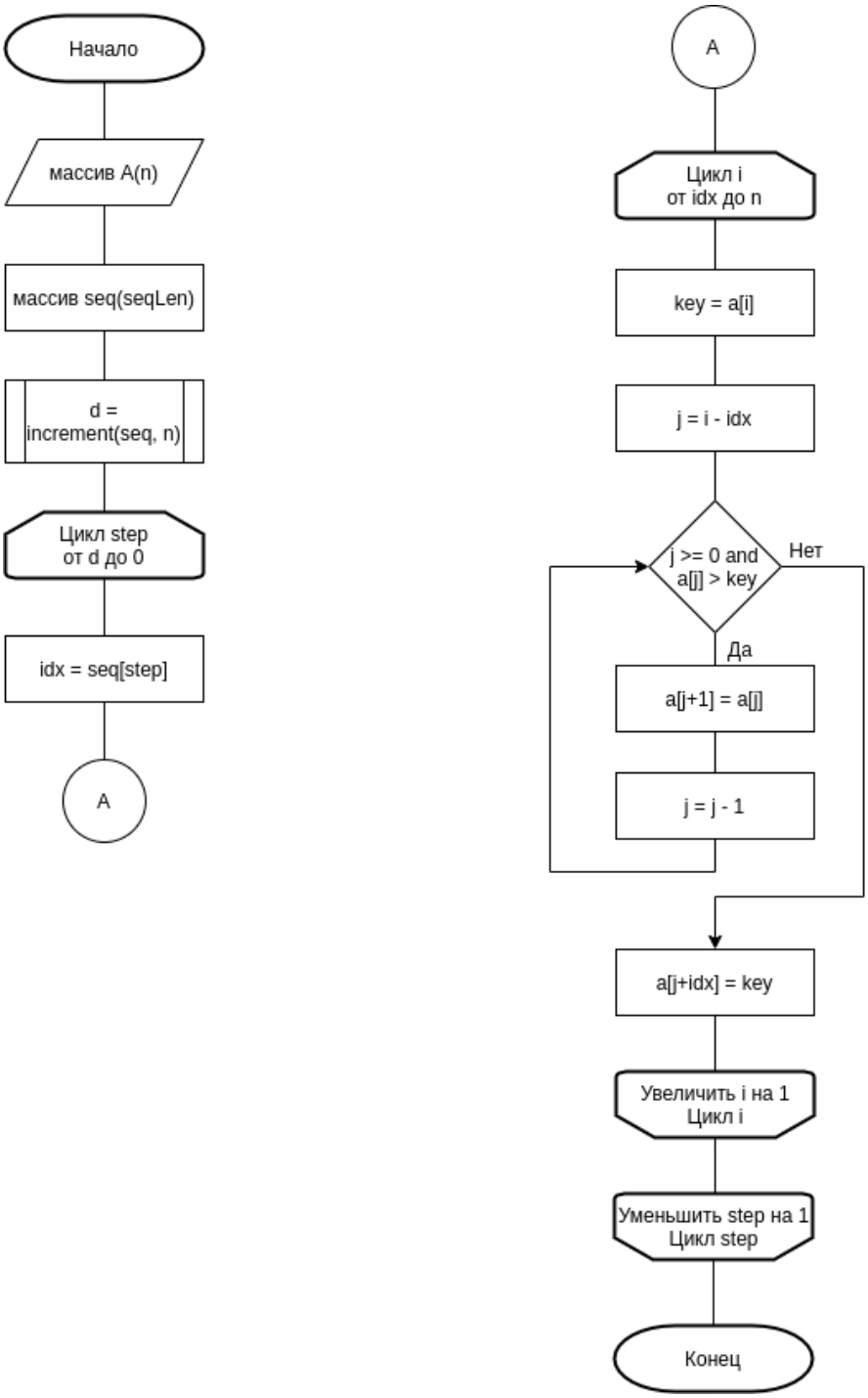


Рисунок 3. Схема алгоритма сортировки Шелла.

2.2. Анализ трудоемкости алгоритмов

Введем модель вычисления трудоемкости для анализа описанных выше алгоритмов [3].

1) Стоимость базовых операций единична
   =, +, -, *, /, %, <, >, <=, =>, !=, [], +=, -=, *=, /=, ++, --

2) Стоимость цикла *for (int i = 0; i < M; i++) {}*
   $f_{for} = f_{\text{иниц}} + f_{\text{сравн}} + M(f_{\text{тела}} + f_{\text{иниц}} + f_{\text{сравн}})$
   $f_{for} = 2 + M(f_{\text{тела}} + 2)$

3) Стоимость ветвлений
   Стоимость перехода к одной из ветвей – 0.

   $f_{if} = f_{\text{выч.усл}} + \begin{cases} min(f_a, f_b) & - \textit{ лучший случай} \\ max(f_a, f_b) & - \textit{ худший случай} \end{cases}$
   где f_a, f_b – ветви условного перехода.

4) Стоимость обмена *swap(a[j], a[j-1])*
   $f_{swap} = 3f_{\text{иниц}} + 4f_{\text{индекс}} + 2f_{\text{опер}} = 9$

Выполним вычисление трудоемкости для каждого из алгоритмов.

2.2.1. Трудоемкость алгоритма сортировки «пузырьком»

Поэтапное вычисление общей трудоемкости алгоритма представлено в таблице 1.

Таблица 1. Трудоемкость алгоритма сортировки «пузырьком»

Операция	**Стоимость**
Цикл (i от 0 до N – 1)	$3 + (N - 1)(3 + f_{\text{тела1}})$
Цикл (j от 1 до N – i)	$3 + (N - i - 1)(3 + f_{\text{тела2}})$
Условие	$4 + \begin{cases} 0 & - \textit{ лучший случай} \\ f_{swap} & - \textit{ худший случай} \end{cases}$

Внутренний цикл выполняется $\frac{n(n-1)}{2}$ раз.

Лучший случай: $\frac{7}{2}n^2 + \frac{5}{2}n - 3$

Худший случай: $8n^2 - n - 4$

2.2.2. Трудоемкость алгоритма сортировки вставками

Поэтапное вычисление общей трудоемкости алгоритма представлено в таблице 2.

Таблица 2. Трудоемкость алгоритма сортировки вставками

Операция	**Стоимость**
Цикл (i от 1 до N)	$2 + (N - 1)(9 + f_{\text{тела1}})$
Цикл (j от i – 1 до 0)	$4 + (i - 1)(4 + f_{\text{тела2}})$
Условие	$\begin{cases} 0 & - \textit{ лучший случай} \\ 5 & - \textit{ худший случай} \end{cases}$

Внутренний цикл выполняется $\frac{n(n-1)}{2}$ раз.

Лучший случай: 13n – 11

Худший случай: $\frac{9}{2}n^2 - \frac{17}{2}$ n – 11

2.2.3. Трудоемкость алгоритма сортировки Шелла

Поэтапное вычисление общей трудоемкости алгоритма представлено в таблице 3.

Таблица 3. Трудоемкость алгоритма сортировки Шелла

Операция	**Стоимость**
Цикл (step от d до 0)	$2 + A(4 + f_{\text{тела1}})$
Цикл (i от idx до n)	$2 + B(9 + f_{\text{тела2}})$
Цикл (j от i - idx до 0)	$4 + C(4 + f_{\text{тела3}})$
Условие	$\begin{cases} 0 & - \textit{ лучший случай} \\ 5 & - \textit{ худший случай} \end{cases}$

Лучший случай: 4ABC + 13AB + 6A + 2 = O(n)

Худший случай: 9ABC + 13AB + 6A + 2 = O($n^{4/3}$)

2.3. Вывод

В данном разделе были представлены схемы алгоритмов и рассчитана трудоемкость для каждого.

Сортировка «пузырьком»: лучший случай – O(n^2); худший – O(n^2).

Сортировка вставками: лучший случай – O(n); худший – O(n^2).

Сортировка Шелла (Седжвик): лучший случай – O(n); худший – O($n^{4/3}$).

3. Технологическая часть

В данном разделе рассмотрим язык программирования, среду разработки, требуемые инструменты для реализации. Также представим непосредственно реализацию.

3.1. Требования к программному обеспечению

I. Программа должна предусматривать ввод массива заданной длины. Заполнение массива должно быть доступно как с клавиатуры, так и автоматически.
II. Выбор применяемого алгоритма осуществляется пользователем из списка алгоритмов, предложенных в меню.
III. На выходе программа выводит отсортированный массив.
IV. Также необходимо предусмотреть выполнение замеров процессорного времени для каждого из алгоритмов.

3.2. Средства реализации

В данной работе используется язык программирования C++, из-за удобства представления матрицы в виде объекта класса (сокращает количество аргументов для передачи в функцию) и опыта написания на нем. Среда разработки – Qt.

Для замеров процессорного времени использовалась функция clock() [4].

3.3. Реализация алгоритмов

Листинг 1

Алгоритм сортировки массива «пузырьком».

```
#include "bubbleSort.h"

void bubbleSort(Array &arr)
{
   for (int i = 0; i < arr.numElems - 1; i++) {
      for (int j = 1; j < arr.numElems - i; j++) {
         if (arr.ptr[j] < arr.ptr[j - 1]) {
            swap(arr.ptr[j], arr.ptr[j - 1]);
         }
      }
   }
}
```

Листинг 2

Алгоритм сортировки массива вставками.

```
#include "insertionSort.h"

void insertionSort(Array &arr)
{
   for (int i = 1; i < arr.numElems; i++) {

      int key = arr.ptr[i];
      int j = i - 1;

      while (j >= 0 && arr.ptr[j] > key) {
         arr.ptr[j + 1] = arr.ptr[j];
         j--;
      }

      arr.ptr[j + 1] = key;
   }
}
```

Листинг 3

Алгоритм сортировки массива Шелла.

```
#include "shellSort.h"

const int seqLen = 40;

int increment(Array inc, int size)
{
   int p1 = 1, p2 = 1, p3 = 1;
   int s = -1;

   do {
      s++;
      if (s % 2) {
         inc.ptr[s] = 8 * p1 - 6 * p2 + 1;
      }
      else {
         inc.ptr[s] = 9 * p1 - 9 * p3 + 1;
         p2 *= 2;
         p3 *= 2;
      }
      p1 *= 2;
   } while (3 * inc.ptr[s] < size);

   return s > 0 ? --s : 0;
}

void shellSort(Array &arr)
{
   Array seq(seqLen);
```

```
  // вычисление последовательности приращений
  int d = increment(seq, arr.numElems);
  for (int step = d; step >= 0; step--) {

    int idx = seq.ptr[step];

    for (int i = idx; i < arr.numElems; i++) {

      int key = arr.ptr[i];
      int j = i - idx;

      while (j >= 0 && arr.ptr[j] > key) {
        arr.ptr[j + 1] = arr.ptr[j];
        j--;
      }

      arr.ptr[j + idx] = key;
    }
  }
}
```

3.4. Описание тестирования

Тестирование осуществляется по принципу «черного ящика».

Для проверки корректности программы необходимо предусмотреть наборы различных тестов, включающих в себя случаи отсортированных массивов, массивов, отсортированных в обратном порядке и произвольных массивов.

3.5. Вывод

В данном разделе была представлена реализация алгоритмов, а также были описаны инструменты, необходимые для разработки программы.

4. Экспериментальная часть

В данном разделе будут рассмотрены примеры работы программы, произведено тестирование, выполнены эксперименты по замеру времени, а также выполнен сравнительный анализ полученных данных.

4.1. Примеры работы программы

Взаимодействие с программой (меню, выбор способа создания массива, выбор алгоритма, вывод результата) представлено на рисунке 4.

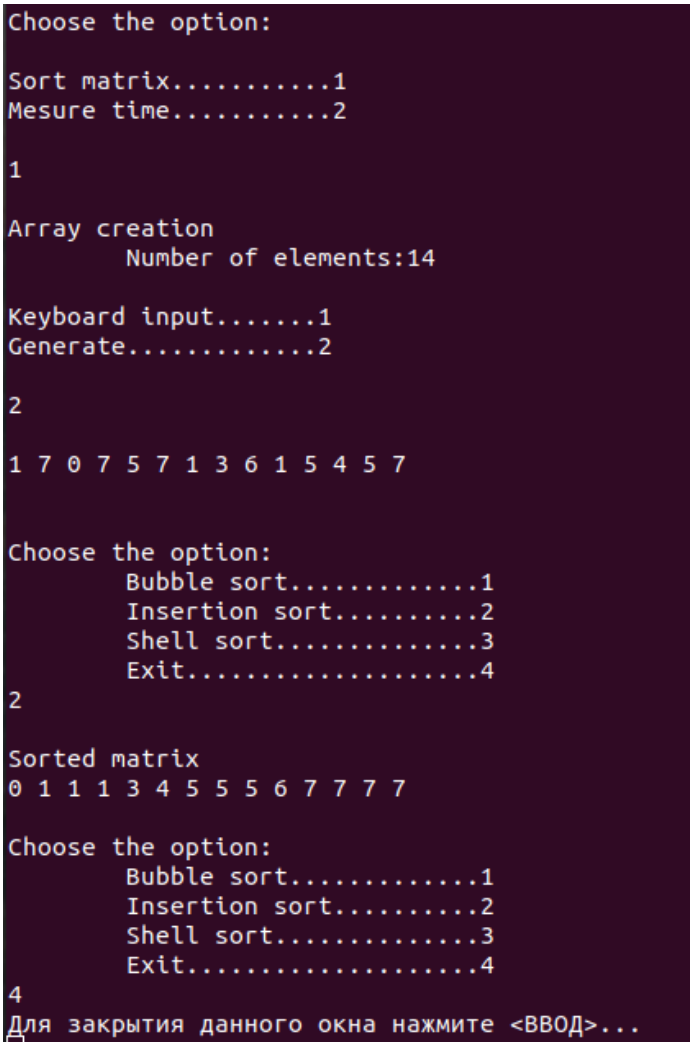


Рисунок 4. Пример взаимодействия с программой.

4.2. Результаты тестирования

Результаты тестирования приведены в таблице 4.

Таблица 4. Результат тестирования

Входной массив	Результат сортировки
1 2 3 4 5 6 7 8	1 2 3 4 5 6 7 8
8 7 6 5 4 3 2 1	1 2 3 4 5 6 7 8
1 7 0 7 5 7 1 3	0 1 1 3 5 7 7 7

Поскольку результаты работы алгоритмов совпадают, в столбце «Результат сортировки» приведен массив, одинаковый для всех алгоритмов.

4.3. Сравнение времени работы алгоритмов

Выполним эксперименты по замеру времени. Эксперименты проводятся на массивах размерами от 10 до 100 с шагом 10 и от 100 до 600 с шагом 100. Поскольку работа алгоритмов зависит от предварительной сортировки массива, разобьем тестирование на лучший (отсортированный), худший (отсортированный в обратном порядке) и произвольный случаи. Результаты замеров приведены на рисунке 4.1 – 4.3.

Рисунок 4.1. График зависимости времени работы алгоритмов от размера массива. Лучший случай.

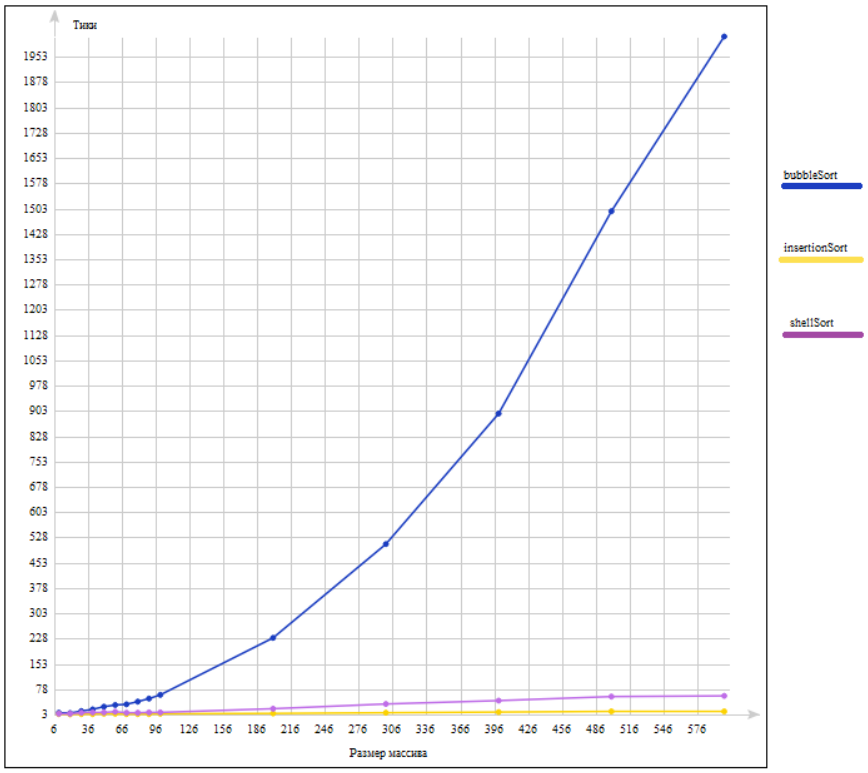


Рисунок 4.2. График зависимости времени работы алгоритмов от размера массива. Худший случай.

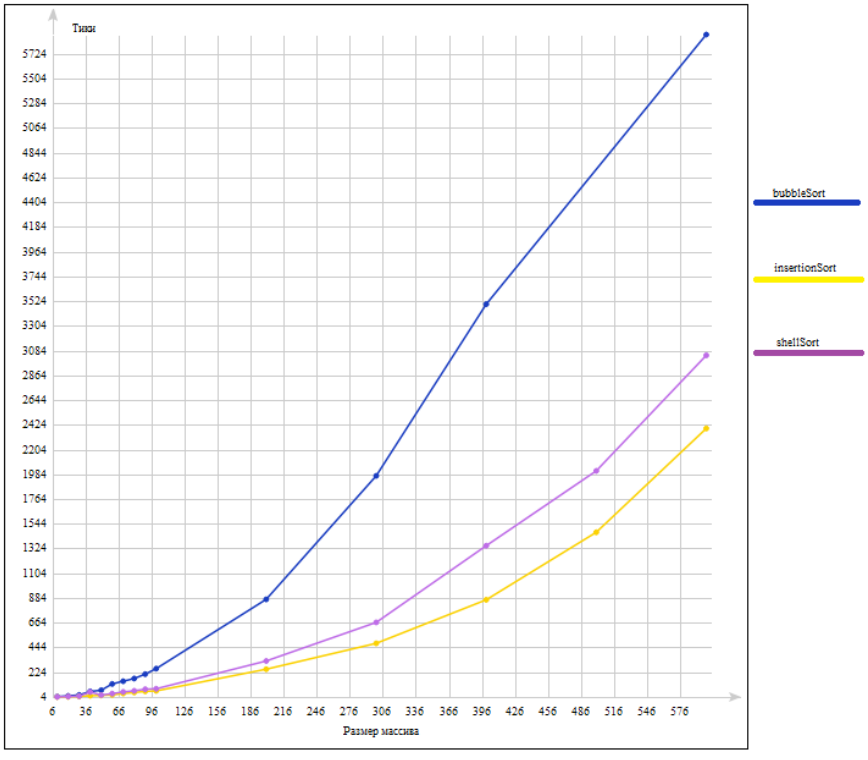
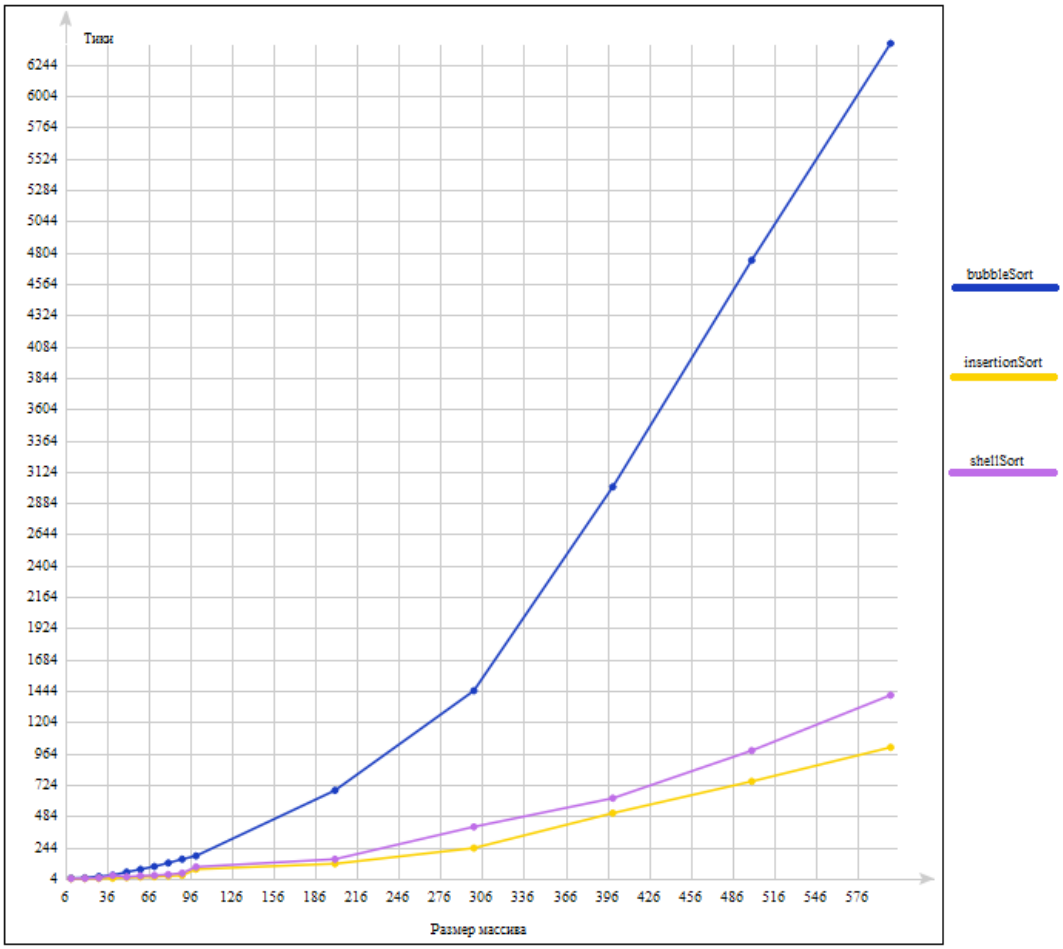


Рисунок 4.3. График зависимости времени работы алгоритмов от размера массива. Произвольный случай.



Стоит отметить, что самым долгим алгоритмом сортировки на массивах большой длины является сортировка «пузырьком». Самым быстрым – сортировка вставками. Сортировка Шелла, которая является модификацией сортировки вставками, проигрывает ей по времени, но не значительно, занимая второе места по быстродействию. Однако сортировка Шелла выигрывает по памяти, так как у нее отсутствует потребность в памяти под стек. Также, при худшем случае, у сортировки Шелла отсутствует деградация времени.

Заключение

В ходе лабораторной работы мы разработали и реализовали алгоритмы сортировки массивов: «пузырьком», вставками, Шелла, а также провели анализ трудоемкости и скорости работы каждого из метода.

В результате анализа было установлено, что самую низкую трудоемкость имеют алгоритмы сортировки вставками и Шелла для лучших случаев – O(n). Трудоемкость сортировки «пузырьком» не зависит от предварительной сортировки массива и составляет O(n^2). Трудоемкость сортировки Шелла незначительно меньше трудоемкости сортировки вставками на худших наборах – O($n^{4/3}$) и O(n^2) соответственно.

Что касается времени работы алгоритмов, самым долгим является «пузырьковая» сортировка. Затем идет сортировка Шелла. А самой быстрой сортировкой является сортировка вставками.

Так, на массивах небольшой длины (до 10 элементов), сортировка вставками работает в 2 раза быстрее «пузырьковой», а Шелла – в 1.6 раз. На массивах большой длины (100 элементов) сортировка вставками опережает пузырьковую по времени в 5 раз, а сортировка Шелла в 3 раза.

Список литературы

1. Сортировка.-URL: http://mech.math.msu.su/~shvetz/54/inf/perlproblems/chSorting_sIdeas.xhtml (дата обращения: 12.10.2020). Текст: электронный.
2. Основы сортировки.-URL: https://it.wikireading.ru/34919 (дата обращения: 12.10.2020). Текст: электронный.
3. Трудоемкость алгоритмов и временные оценки.-URL: http://techn.sstu.ru/kafedri/MetMat/shaturn/theoralg/5.htm (дата обращения: 14.10.2020). Текст: электронный.
4. Техническая документация.-URL: https://docs.microsoft.com/ru-ru/cpp/c-runtime-library/reference/clock?view=vs-2019 (дата обращения: 20.09.2020). Текст: электронный.