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»

Лабораторная работа №3

По предмету: «Анализ алгоритмов»

Алгоритмы сортировки

Студент: Гасанзаде М.А.,
Группа: ИУ7-56Б

Москва, 2019 г.

Оглавление

Введение

Пусть требуется упорядочить N элементов: $R_1, R_2, \dots, R_N$. На множестве элементов определено отношение порядка «<». Задачей сортировки является нахождение такой перестановки записей R_j с индексами $1, \dots, N$, после которой элементы расположились бы в порядке неубывания.

Цель работы: изучение метода динамического программирования путем реализации алгоритмов сортировки.

Задачи работы:

1) изучение алгоритмов сортировки пузырьком, сортировки вставками и быстрой сортировки;

2) применение метода динамического программирования для реализации алгоритмов сортировки пузырьком, сортировки вставками и быстрой сортировки;

3) получение практических навыков реализации указанных алгоритмов;

4) сравнительный анализ реализаций алгоритмов сортировки по затрачиваемым ресурсам;

5) экспериментальное подтверждение различий во временной эффективности реализаций алгоритмов сортировки при помощи разработанного программного обеспечения на материале замеров процессорного времени выполнения реализаций на варьирующихся размерах массивов;

6) описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1. Аналитическая часть

В данной части дано теоретическое описание алгоритмов и указание области их применения.

1.1. Описание алгоритмов

Каждый шаг сортировки пузырьком[1] состоит в проходе в обратном порядке по массиву. По пути просматриваются пары соседних элементов. Если элементы некоторой пары находятся в неправильном порядке, то меняем их местами.

После нулевого прохода по массиву в начале массива оказывается самый маленький элемент. Следующий проход делается до второго элемента, таким образом второй по величине элемент занимает правильную позицию. Делаем проходы по все уменьшающейся части массива до тех пор, пока в ней не останется только один элемент. На этом сортировка заканчивается, так как последовательность упорядочена по возрастанию.

Сортировка простыми вставками[2] в чем-то похожа на вышеизложенный метод. Аналогичным образом делаются проходы по части массива, и аналогичным же образом в его начале формируется отсортированная последовательность.

Однако в сортировке пузырьком на i-м шаге элементы a[0]...a[i] стоят на правильных местах и никуда более не переместятся. Здесь же подобное утверждение неверно. Последовательность a[0]...a[i] упорядочена. При этом по ходу алгоритма в нее будут вставляться все новые элементы. На следующем, (i+1)-м шаге алгоритма берем a[i+1] и вставляем на нужное место в готовую часть массива. Поиск подходящего места для очередного элемента входной последовательности осуществляется путем последовательных сравнений с элементом, стоящим перед ним.

При быстрой сортировке[3] из массива выбирается некоторый опорный элемент $a[i]$. Запускается процедура разделения массива, которая перемещает все элементы, меньшие, либо равные $a[i]$, влево от него, а все ключи, большие, либо равные $a[i]$ – вправо. Теперь массив состоит из двух подмножеств. Если в подмассиве более двух элементов, рекурсивно запускаем для него ту же процедуру. В конце получится полностью отсортированная последовательность [4].

1.2. Применение алгоритмов

Алгоритмы сортировки [5] имеют большое практическое применение. Их можно встретить там, где речь идет об обработке и хранении больших объемов информации. Некоторые задачи обработки данных решаются проще, если данные заранее упорядочить. В компьютерной графике свои версии сортировок для растеризации, операций над полигонами и др.

2. Конструкторская часть

В данной части приведены схемы алгоритмов, а также их сравнительный анализ.

2.1 Разработка алгоритмов

Схемы алгоритмов представлены на *рис. 1, 2, 3, 4, 5*.

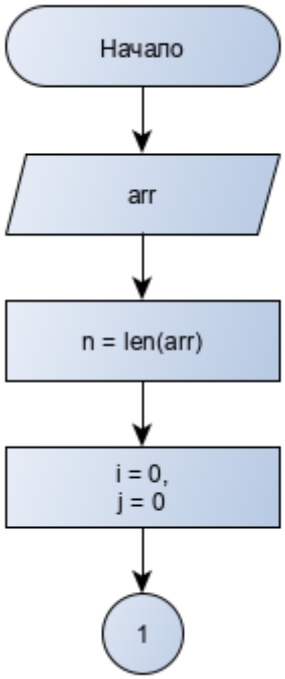

Рисунок 1– Схема алгоритма сортировки пузырьком (начало)

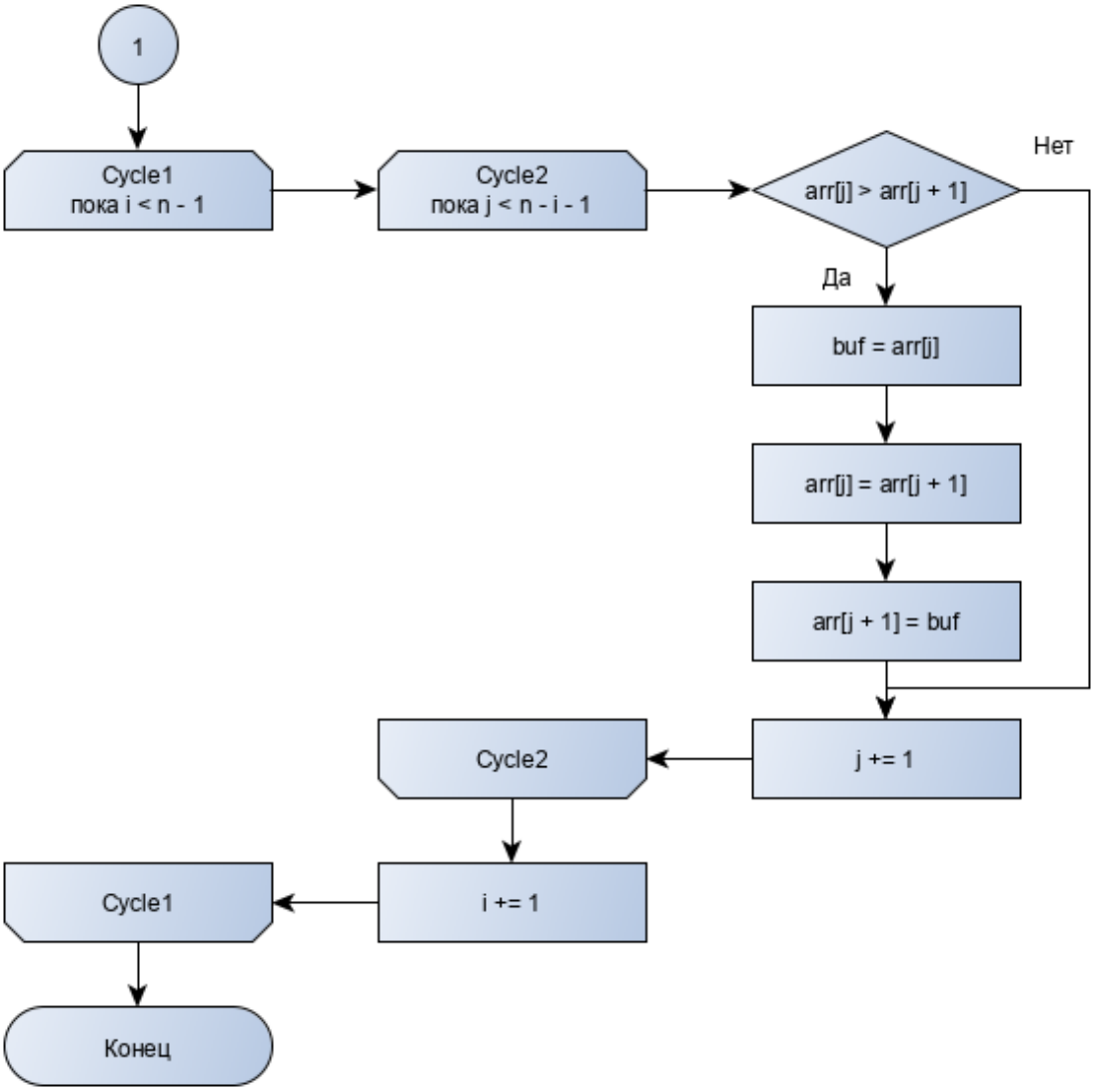

Рисунок 2 – Схема алгоритма сортировки пузырьком (конец)

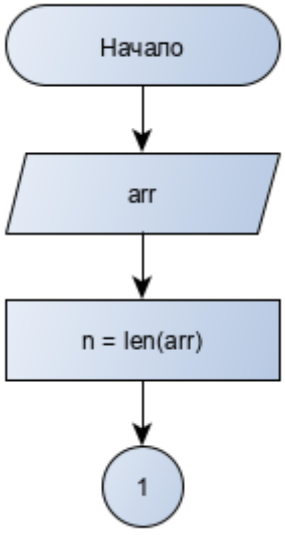

Рисунок 3 – Схема алгоритма сортировки вставками (начало)

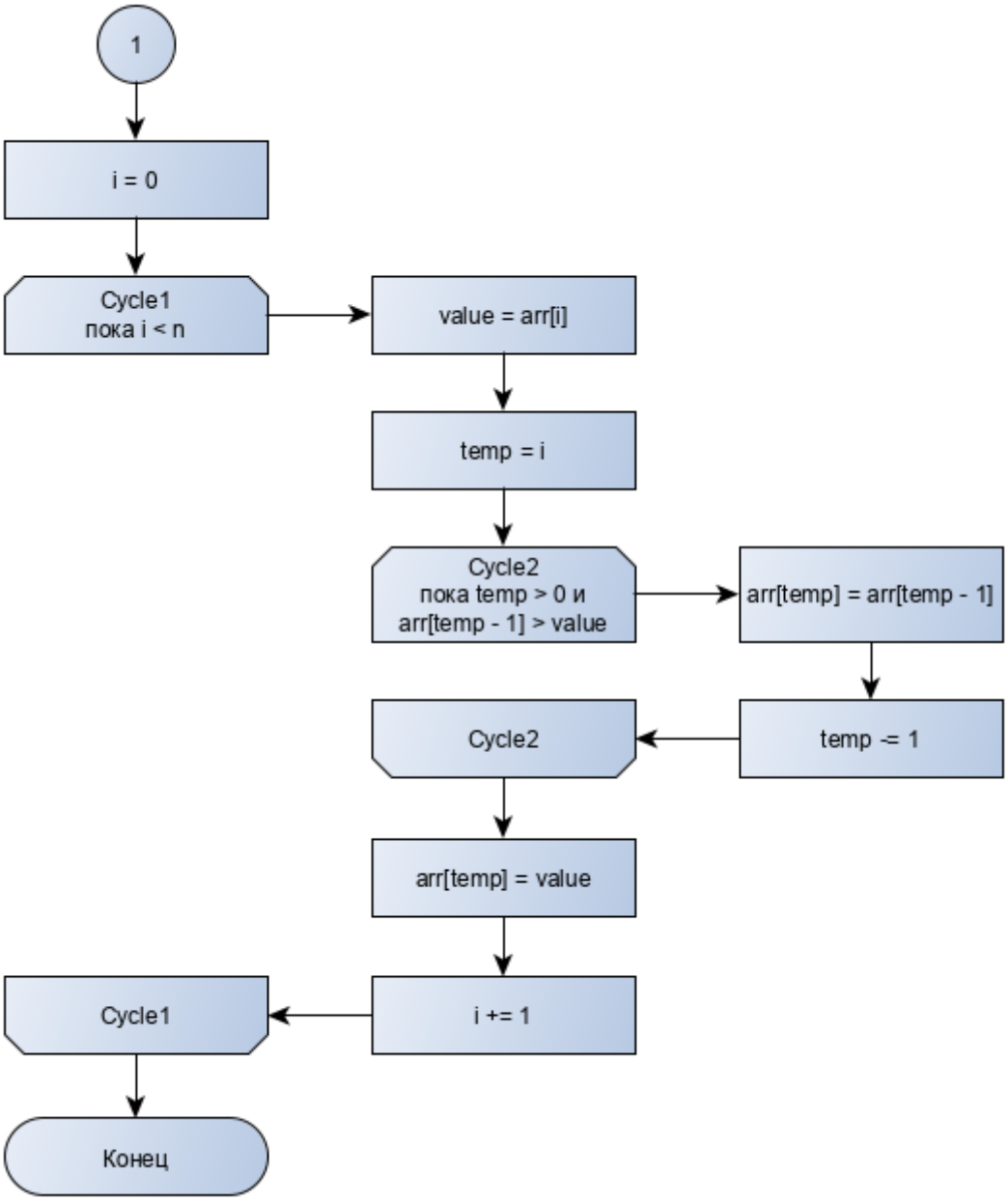

Рисунок 4 – Схема алгоритма сортировки вставками (конец)

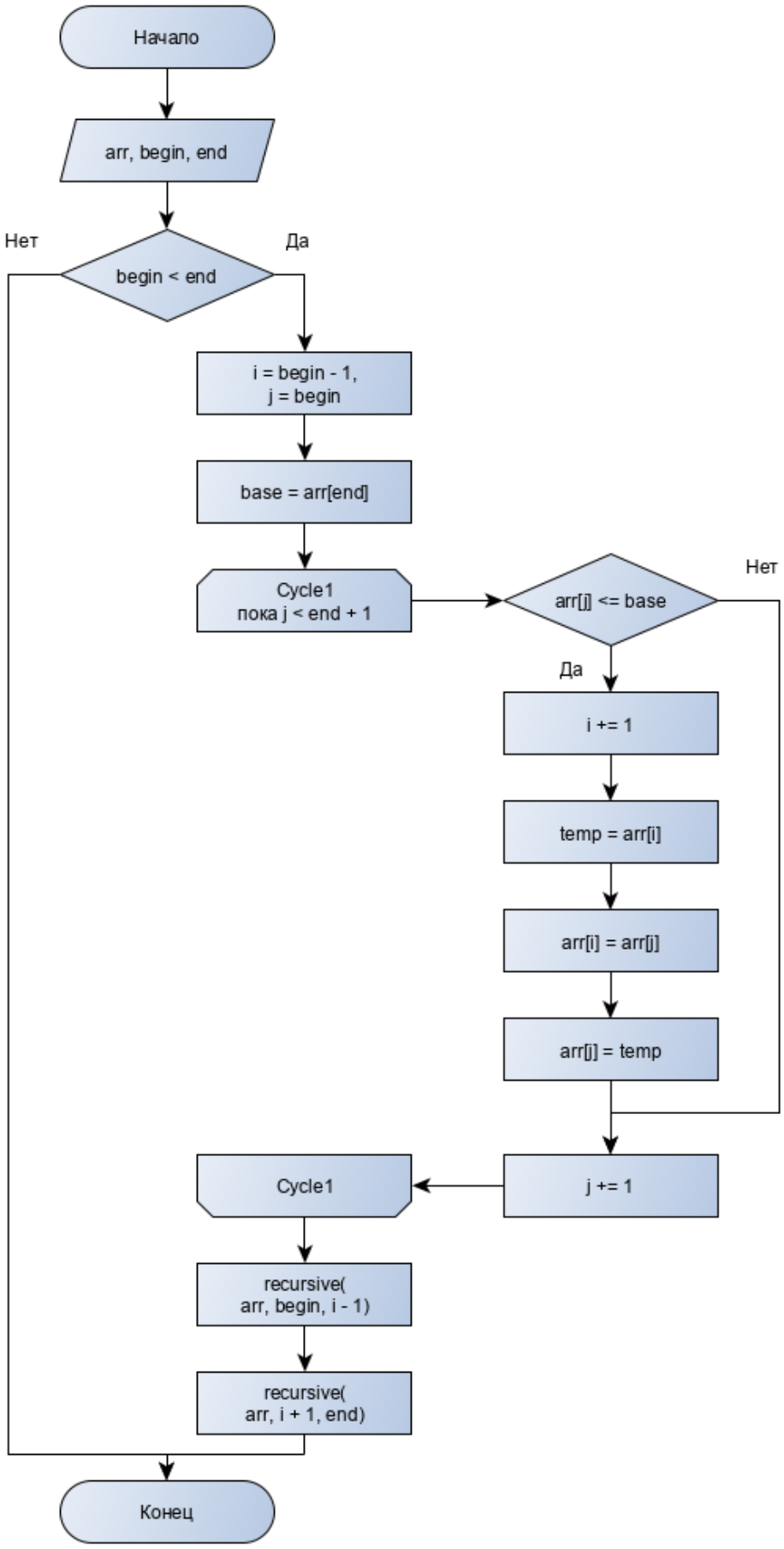

Рисунок 5 – Схема алгоритма быстрой сортировки

2.2 Сравнительный анализ алгоритмов

Введем модель вычислений трудоемкости алгоритма. Пусть трудоемкость 1 у следующих операций: +, -, *, /, %, =, ==, !=, <, <=, >, >=, []. Трудоемкость условного перехода примем за 1 (а также трудоемкость самого условия). Трудоемкость цикла: $f_{\text{цикла}} = f_{\text{иниц}} + f_{\text{сравн}} + N * (f_{\text{тела}} + f_{\text{инкрем}} + f_{\text{сравн}})$.

Сортировка пузырьком в лучшем случае (уже отсортированный массив) обладает трудоемкостью $4N^2 + 3N - 4$. В худшем случает (обратно отсортированный массив) трудоемкость увеличивается до $\frac{17}{2}N^2 - \frac{3}{2}N - 4$.

Сортировка вставками в лучшем случае (уже отсортированный массив): $O(N)$. В худшем случае (обратно отсортированный массив): $O(N^2)$.

Быстрая сортировка в лучшем случае (опорный элемент всегда делит массив на две равные части): $O(N * \log(N))$. В худшем случае (базовый элемент всегда является максимальным/минимальным в массиве): $O(N^2)$.

2.3 Вывод

Все алгоритмы в худшем случае обладают квадратичной сложностью. А в лучшем случае меньше всего сложность у алгоритма сортировки вставками. Быстрая сортировка со сложностью $O(N * \log(N))$ обладает стабильностью и может применяться на произвольных данных. Сортировка пузырьком тоже устойчива, но из-за своей сложности не рекомендуется к применению на больших объемах данных.

3. Технологическая часть

В данной части приведены используемые технические средства, а также примеры тестирования и листинг программы.

3.1 Требования к программному обеспечению

Программа должна корректно считывать массив, верно его упорядочивать его элементы по возрастанию. Требуется также обеспечить возможность замера времени работы каждого алгоритма на различных размерах массивов.

3.2 Средства реализации

Выбран язык программирования Python за кроссплатформенность, автоматическое освобождение памяти, высокую скорость разработки. Замер процессорного времени проводился функцией time[6] функция приведена на *листинге 1*.

Листинг 1. Функция замера времени.

```
def get_calc_time(func, arr):
    t2 = time.process_time()
    func(arr)
    t1 = time.process_time() - t2

    return t1
```

3.3 Листинг кода

Исходный код программы приведен *на листингах 2-4*.

***Листинг 2* – Алгоритм сортировки пузырьком**

```
def mysort_bubble(arr):
    n = len(arr)

    for i in range(n - 1):
        for j in range(n - i - 1):
            if arr[j] > arr[j + 1]:
                buf = arr[j]
                arr[j] = arr[j + 1]
                arr[j + 1] = buf

    return arr
```

***Листинг 3* - Алгоритм сортировки вставками**

```
def mysort_insert(arr):
    n = len(arr)
    for i in range(1, n, 1):
        value = arr[i]
        temp = i
        while temp > 0 and arr[temp - 1] > value:
                arr[temp] = arr[temp - 1]
                temp -= 1

        arr[temp] = value

    return arr
```

***Листинг 4* - Алгоритм быстрой сортировки**

```
def mysort_quick_rec_end(arr, begin, end):
    if begin < end:
        i = begin - 1
        base = arr[end]
        for j in range(begin, end + 1):
            if arr[j] <= base:
                i += 1
                temp = arr[i]
                arr[i] = arr[j]
                arr[j] = temp

        mysort_quick_rec_end(arr, begin, i - 1)
        mysort_quick_rec_end(arr, i + 1, end)

    return arr
```

3.4 Описание тестирования

Тестирование проводится по методу чёрного ящика. Требуется проверить корректность работы на массивах четной и нечетной длины, на единичном массиве, а также на массиве с одинаковыми элементами.

3.5 Вывод

Текущая реализация на языке Python позволяет корректно считывать массив, упорядочивать его элементы, а также производить замеры времени для определенного диапазона размеров входных матриц.

4. Экспериментальная часть

В этой части приведены пример интерфейса, входные данные тестирования, результаты замера времени и их анализ.

4.1. Примеры работы

На *рис. 1, 2* приведены изображения внешнего вида интерфейса программы во время его работы.

```
1 2 6 4 2 8 6 9 5

Сортировка пузырьком:
[1, 2, 2, 4, 5, 6, 6, 8, 9]

Сортировка Шелла:
[1, 2, 2, 4, 5, 6, 6, 8, 9]

Быстрая сортировка:
[1, 2, 2, 4, 5, 6, 6, 8, 9]
```

Рисунок 1 – Пример работы программы на произвольном массиве

```
9 8 7 6 5 4 3 2 1

Сортировка пузырьком:
[1, 2, 3, 4, 5, 6, 7, 8, 9]

Сортировка Шелла:
[1, 2, 3, 4, 5, 6, 7, 8, 9]

Быстрая сортировка:
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Рисунок 2 – Пример работы программы на обратно упорядоченном массиве

4.2. Результаты тестирования

В *рисунке 3* представлены результаты тестирования в следующем порядке: сортировка пузырьком, сортировка вставками, быстрая сортировка.

Рисунок 3

Результаты тестирования по методу черного ящика[7]

Исходный массив	Упорядоченный массив	Результат
1, 2, 3, 4, 5, 6, 7	1, 2, 3, 4, 5, 6, 7	Ответ верный
6, 5, 4, 3, 2, 1	1, 2, 3, 4, 5, 6	Ответ верный
4	4	Ответ верный
0, 0, 0, 0, 0	0, 0, 0, 0, 0	Ответ верный
7, 4, 2, 6, 7, 12, 435, 1	1, 2, 4, 6, 7, 7, 12, 435	Ответ верный

Все тесты пройдены успешно.

4.3. Постановка эксперимента по замеру времени

*З*амер времени проводился для массивов длины от 100 до 1000 с шагом 100. Каждый алгоритм замерялся для лучшего случая, худшего случая и произвольных данных. Каждое испытание повторялось 100 раз, результат одного эксперимента рассчитывался как среднее значение результатов проведенных испытаний с одинаковыми входными данными.

4.4. Сравнительный анализ на материале экспериментальных данных

На *рис 5,6,7* представлены графики результатов замеров времени.

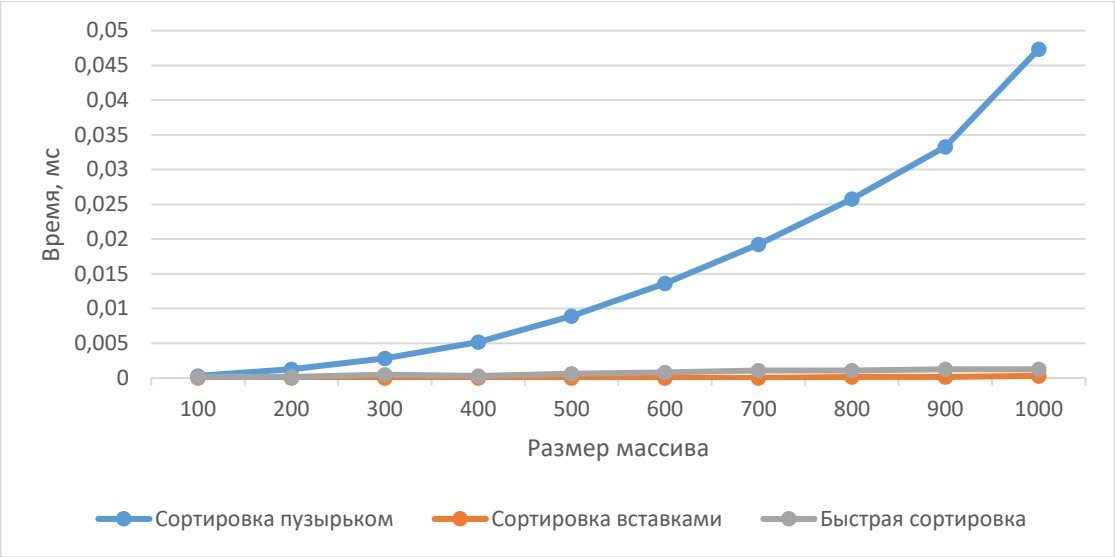

Рисунок 5 – Время работы алгоритмов на лучших данных для каждого из алгоритмов

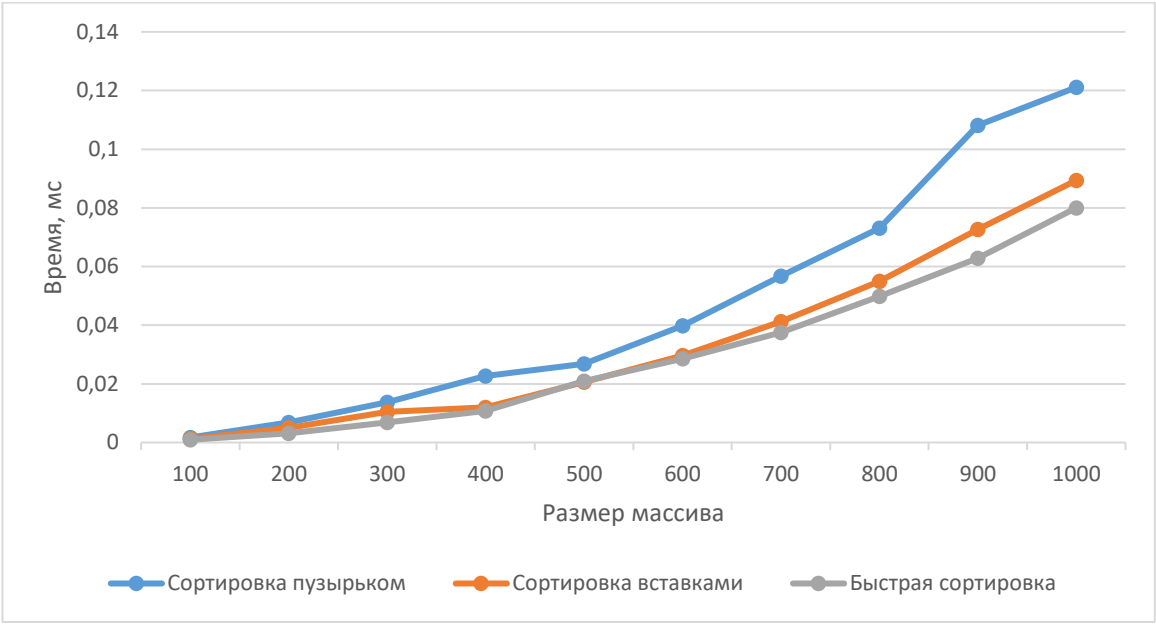

Рисунок 6 – Время работы алгоритмов на худших данных для каждого из алгоритмов

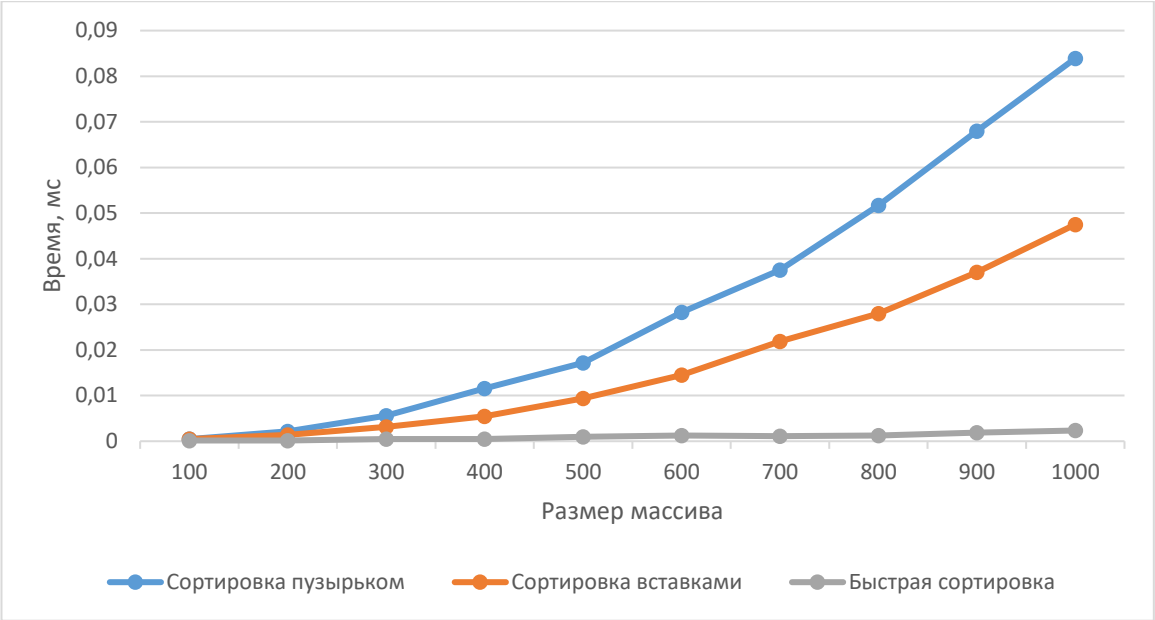

Рисунок 7 – Время работы алгоритмов на произвольных данных(random)

4.5 Вывод

По результатам эксперимента подтверждена трудоемкость алгоритмов, указанная в конструкторской части. Из результатов следует, что алгоритм быстрой сортировки работает быстрее на худшем (сложность $O(N^2)$) и произвольном (сложность $O(n * logN)$ так как сложность любого бинарного дерева составляет *log*n* где *n-это кол-во итераций*) случаях, лишь немного уступая алгоритму вставок со сложностью $O(N)$ на лучшем случае. Эксперимент показал квадратичную сложность алгоритма сортировки пузырьком.

Заключение

В рамках данной работы изучены алгоритмы сортировки пузырьком, сортировки вставками и быстрой сортировки. Применен метод динамического программирования для их реализации. Получены практические навыки реализации указанных алгоритмов. Проведен сравнительный анализ реализаций алгоритмов сортировки по времени. Подтверждены экспериментально различия во временной эффективности реализаций алгоритмов сортировки при помощи разработанного программного обеспечения на материале замеров процессорного времени выполнения на варьирующихся длинах массивов. Дано описание и обоснование полученных результатов.

Экспериментально подтверждена сложность алгоритмов. На лучшем случае для массива длины 1000 элементов сортировка вставками быстрее сортировки пузырьком на 99.3% и быстрой сортировки на 75%. На худшем случае быстрая сортировка быстрее сортировки пузырьком на 34% и сортировки вставками на 10%. На произвольном случае быстрая сортировка быстрее сортировки пузырьком на 97% и сортировки вставками на 95%.

Список литературы

1. Сортировка пузырьком // AlgoList(1) URL: http://algolist.ru/sort/bubble_sort.php (дата обращения: 9.11.2019).
2. Сортировка вставками // AlgoList(2) URL: http://algolist.ru/sort/insert_sort.php (дата обращения: 9.11.2019).
3. Быстрая сортировка // AlgoList(3) URL: http://algolist.ru/sort/quick_sort.php (дата обращения: 9.11.2019).
4. Сортировка Шелла(модиф. Быстрая сортировка)): URL:https://www.geeksforgeeks.org/shellsort/
5. Алгоритмы сортировки массивов. Внутренняя сортировка // ИНТУИТ URL: https://www.intuit.ru/studies/professional_skill_improvements/2056/courses/504/lecture/11472 (дата обращения: 5.11.2019).
6. time — Time access and conversions // Python URL: https://docs.python.org/3/library/time.html (дата обращения: 1.11.2019).
7. Kara kutu test technologisi URL:https://www.mobilhanem.com/kara-kutu-test-teknigi-ve-uygulanmasi/ (дата обращения 1.11.2019)
8. Bilgisayar tabanli sistemlerde test otomatizasyonunun tasarlanmasi ve gerçeklenmesi. (Hacettepe Üniversitesi: 2015. 70 c.) URL:shorturl.wizarmh/2lab