МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет «Информационные системы и технологии»

Кафедра «Вычислительная техника»

Дисциплина « Алгоритм и структуры данных »

КУРСОВОЙ ПРОЕКТ (РАБОТА)

Тема «Сортировка вставками – вариант 7

Выполнил студент \_\_\_Зяббаров Артур\_Альбертович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / /

подпись фамилия, инициалы

Курс 2 Группа ИВТАСбд-22

Направление/специальность «Алгоритм и структуры данных»

Руководитель Горшков Д.А.

Дата сдачи:

« » 20\_\_г.

Дата защиты:

« » 20\_\_г.

Оценка:

Ульяновск

2023 г.

Оглавление

[Сортировка вставками 2](#_Toc83957337)

[Сложность алгоритма сортировки вставками 3](#_Toc2035010363)

[Исследование алгоритма 4](#_Toc1603290218)

[Вариации реализации 4](#_Toc1691950681)

[Рабочая нагрузка при исследовании 5](#_Toc2130639939)

[Средства оценивания временных затрат 5](#_Toc1412319107)

[Характер времени в зависимости от рабочей нагрузки 6](#_Toc1260292917)

[Рекомендации по реализациям 8](#_Toc348696269)

[Вывод 8](#_Toc1445386858)

[Приложение: 9](#_Toc712582393)

# Сортировка вставками

Сортировка вставками (Insertion Sort) - это простой алгоритм сортировки, который сравнивает каждый элемент со всеми предыдущими элементами в массиве и вставляет его в нужное место. Этот алгоритм имеет сложность O(n^2) и является устойчивым и может быть эффективен для небольших массивов.

Алгоритм сортировки вставками работает следующим образом:

1. Начните со второго элемента в массиве, сравнивая его со всеми предыдущими элементами, начиная с первого.
2. Если текущий элемент меньше, чем предыдущий, то переместите предыдущий элемент на одну позицию вперед и продолжайте сравнивать текущий элемент со следующим предыдущим элементом, пока не найдете место для вставки текущего элемента.
3. После того, как найдено место для вставки текущего элемента, вставьте его в массив на соответствующую позицию.
4. Повторяйте этот процесс для каждого следующего элемента в массиве, начиная с третьего, пока все элементы не будут отсортированы.

Этот алгоритм находит широкое применение в автоматизированных системах, где требуется сортировка небольших массивов или когда массив уже частично отсортирован. Он также может быть полезен для сортировки данных в реальном времени, когда недопустимо долгое ожидание результата.

# Сложность алгоритма сортировки вставками

Сложность алгоритма сортировки вставками в худшем случае составляет O(n^2), где n - количество элементов в массиве, который нужно отсортировать. Это означает, что время выполнения алгоритма увеличивается квадратично по мере увеличения количества элементов в массиве.

Однако, сортировка вставками имеет лучшую производительность на небольших массивах, чем другие алгоритмы сортировки со сложностью O(n log n), такие как быстрая сортировка или сортировка слиянием. Поэтому, если размер входных данных невелик, то сортировка вставками может быть более эффективной, чем другие алгоритмы.

Также следует отметить, что сортировка вставками является устойчивой сортировкой, то есть она сохраняет порядок элементов с одинаковым значением, что может быть важным при сортировке объектов с несколькими ключами.

# Исследование алгоритма

## Вариации реализации

В ходе исследования алгоритма были использованы различные вариации его реализации. В частности, были рассмотрены следующие вариации:

* Язык программирования. Алгоритм был реализован на двух языках - С++ и Python.
* Типы сортируемых данных. Алгоритм был протестирован на данных различных типов: целочисленных (int) и числах с плавающей точкой (double), строках(string) и структуры.

Экспериментальное исследование с использованием разных языков программирования и типов данных помогло оценить работу алгоритма в различных условиях и на разных типах данных. Это важно для определения наиболее эффективного подхода в различных сценариях использования.

## Рабочая нагрузка при исследовании

Для исследования работы алгоритма сортировки слиянием были использованы следующие варианты рабочей нагрузки:

* Неотсортированный массив.
* Отсортированный массив.
* Обратно сортированный массив.

Эти варианты были выбраны потому, что алгоритм сортировки вставками изменяет время выполнения от O(n) до O(n^2). Таким образом, можно проверить производительность алгоритма в разных сценариях.

Неотсортированный массив был использован в качестве базового тестового примера для сравнения времени выполнения алгоритма в начальном состоянии. Отсортированный массив и обратно сортированный массив были использованы для оценки производительности алгоритма в крайних сценариях.

Таким образом, выбор этих вариантов рабочей нагрузки обусловлен необходимостью проверки производительности алгоритма в различных условиях, чтобы определить, как он будет работать в реальных приложениях.

## Средства оценивания временных затрат

Для измерения времени выполнения алгоритма на языке C++, была использована библиотека <chrono>, позволяющая измерять время с точностью до наносекунд. Для этого используется засекание времени начала сортировки командой auto start = std::chrono::steady\_clock::now();, и засекание окончании времени работы командой auto end = std::chrono::steady\_clock::now(); , и в конце кода вычимлением получаем время работы кода командой std::chrono::duration<double> diff = end - start; .

Для измерения времени выполнения алгоритма на языке Python, была использована такая же схема как на с++, которая также позволяет измерять время. В функции insertionSort до и после сортировки было замерено время командами time.monotonic(). Затем была вычислена разница между двумя моментами времени.

## Характер времени в зависимости от рабочей нагрузки

Вот как выглядит результат исследования в зависимости от рабочей нагрузки, а также в зависимости от вариации реализации:

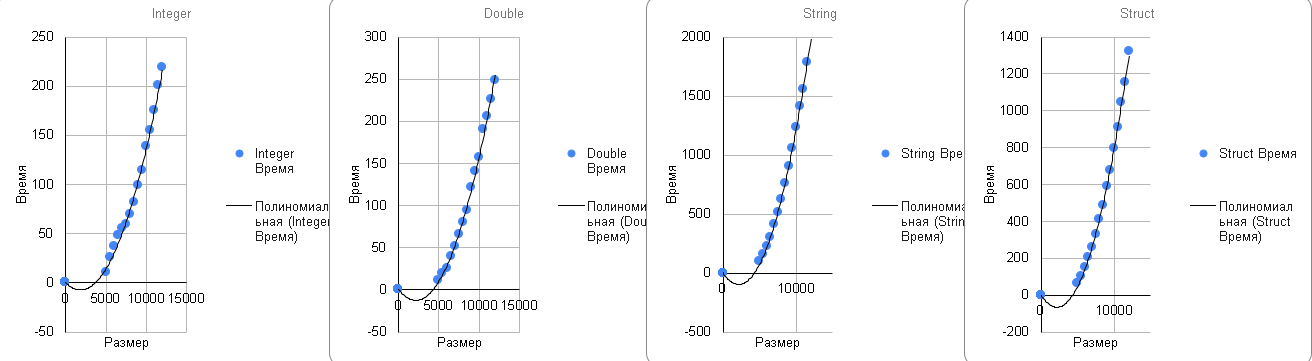


Рисунок 1. Зависимость размера от времени на C++. Случайные входные данные.

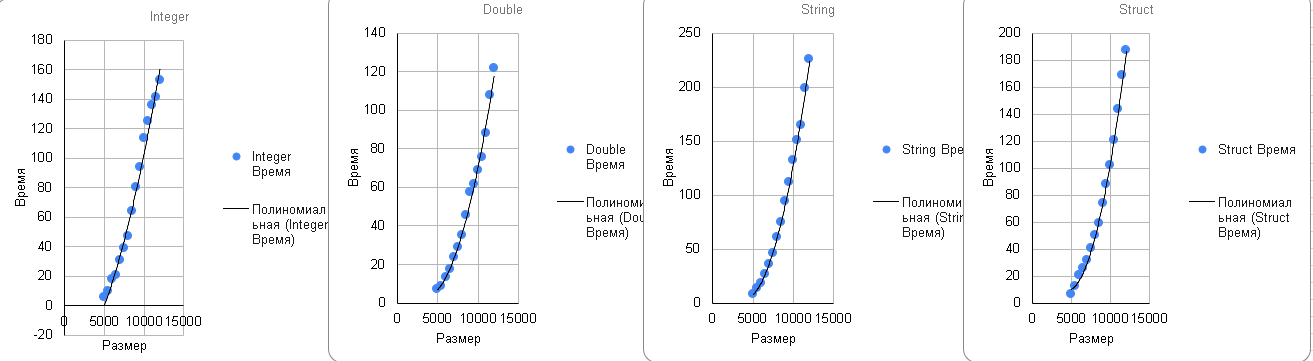


Рисунок 2. Зависимость размера от времени на Python. Случайные входные данные.

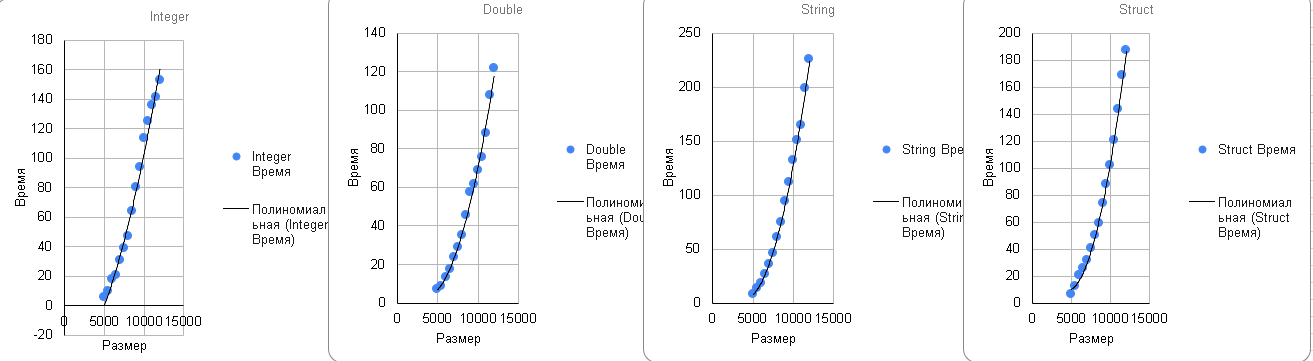


Рисунок 3. Зависимость размера от времени на C++. Входные данные отсортированные в обратном порядке.

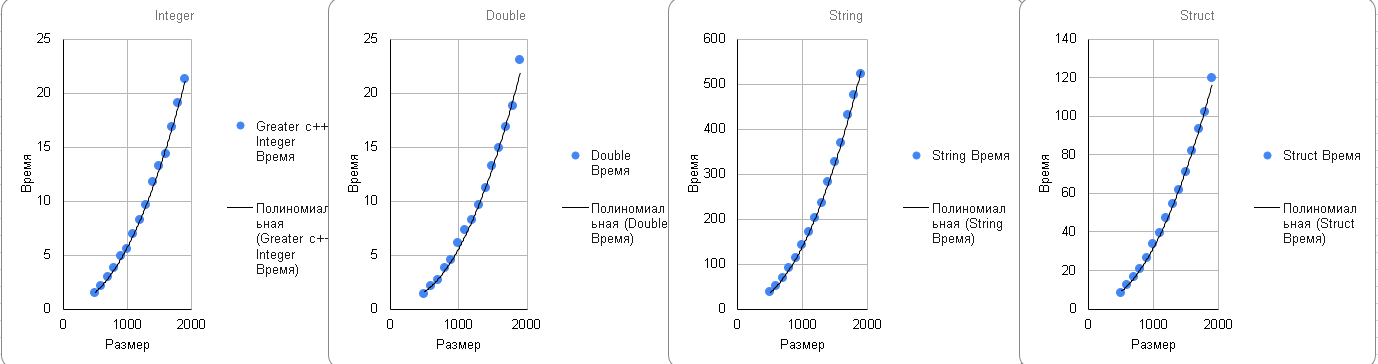


Рисунок 4. Зависимость размера от времени на Python. Входные данные отсортированные в обратном порядке.

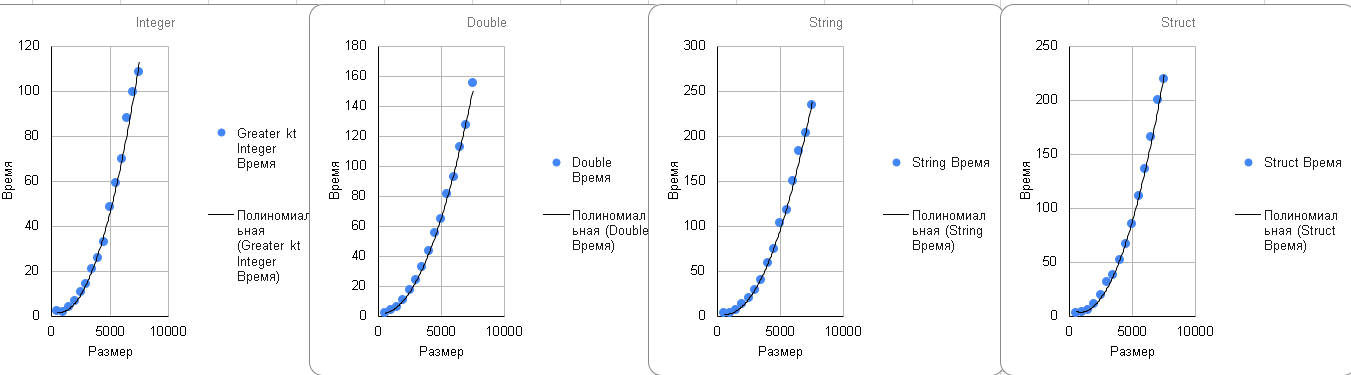


Рисунок 5. Зависимость размера от времени на C++. Отсортированные входные данные.

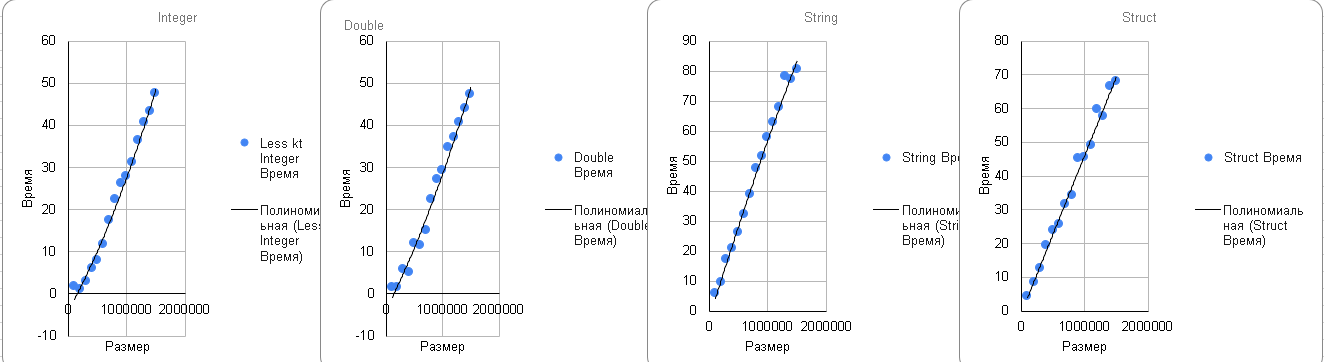


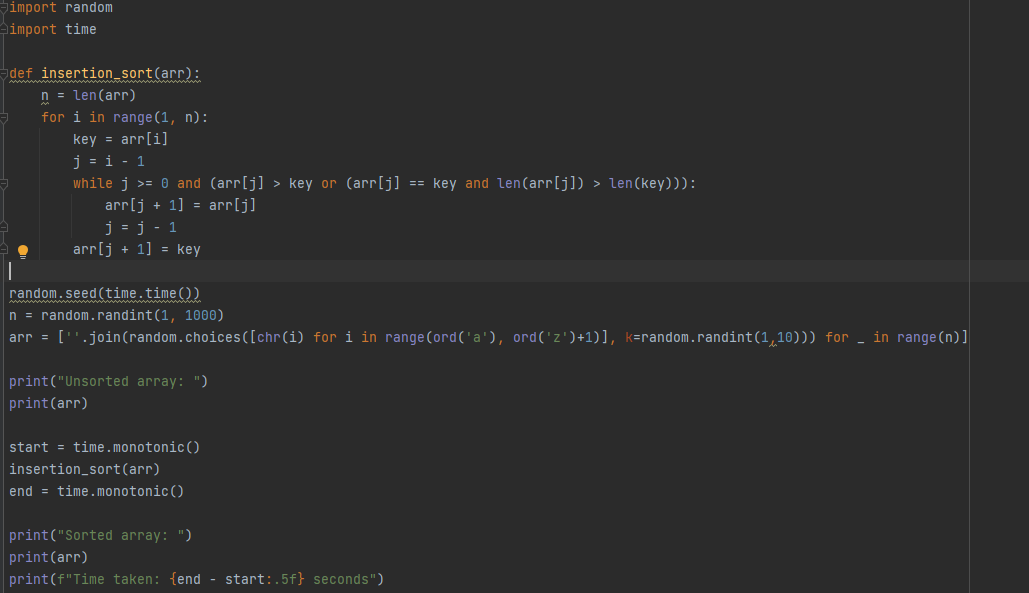
Рисунок 6. Зависимость размера от времени на Python. Отсортированные входные данные.

Время напрямую зависит от входных параметров. Например, в обоих языках программирования как в C++, так и в Python время сортировки в зависимости от типа данных было следующим:

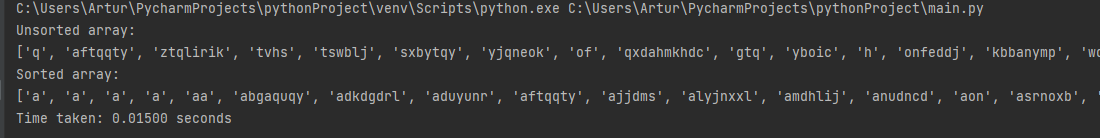
1. Integer
2. Double
3. Struct
4. String

Также при входных данных, которые были отсортированы в обратном порядке наблюдалась асимптотическая сложность O(n^2). При случайных входных данных график прогибался ближе к прямой линии, а при отсортированном списке график равняется функции y = x.

Экспериментальные оценки затрат времени исполнения алгоритма близки к теоретическим при использовании обратно отсортированного массива, то есть в этом случае график практически равен функции y = f(x^2). При использовании случайных входных данных функция принимает значение между y = x^2 и y = x

Анализ кода для string на python:  
Было дано задание сделать сортировку вставками для string значения. Был написан код, который записывает в массив от 1 до 10 алфавит, после чего сортирует по алфавиту заданный массив

Вывод кода:



## Рекомендации по реализациям

На основе результатов экспериментального исследования можно порекомендовать использовать сортировку вставками на языке C++, так как в ряде измерений он оказался быстрее. Также сортировка вставками лучше использовать при небольшом количестве элементов. Но всё же данный алгоритм может пригодиться, когда массив входных данных не нуждается в слишком большой сортировке и имеет всего несколько неправильно расположенных элементов.

# Вывод

В данной курсовой работе был проанализирован алгоритм “Сортировка вставками”. В данном алгоритме были использованы различные размеры и типы входных данных, а также разные распределения значений входных данных. Результаты исследования показали, что сортировка вставками показывает хорошие результаты на небольших размерах входных данных и при сортировке простых типов данных, такие как Int. Однако, при использовании обратно отсортированных данных, время выполнения алгоритма существенно возрастает, что делает его менее эффективным.

Из результатов исследования можно рекомендовать использовать сортировку вставками при небольшом количестве элементов и на языке программирования C++, который в ряде измерений показал более быстрые результаты, но на языке Python размер кода меньше и в целом различия во времени видны только на очень больших числах. Однако, при большом количестве данных лучше использовать более эффективные алгоритмы сортировки, такие как сортировка слиянием или быстрая сортировка.

# Приложение:

С++:

#include <iostream>

#include <random>

#include <chrono>

void insertionSort(double arr[], double n) {

double key;

int i, j;

for (i = 1; i < n; i++) {

key = arr[i];

j = i - 1;

/\* Перемещаем элементы arr[0..i-1], которые больше, чем key,

на одну позицию вперед \*/

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

int main() {

auto seed = std::chrono::system\_clock::now().time\_since\_epoch().count();

std::mt19937 generator(seed);

std::uniform\_int\_distribution<int> distribution(1, 10000);std::uniform\_real\_distribution<double> distribution2(1, 10000);

int n = distribution(generator);

double\* arr = new double[n];

// Инициализация генератора случайных чисел

// Заполнение массива случайными числами от 1 до 100

for (int i = 0; i < n; i++) {

arr[i] = distribution2(generator);

}

std::cout << "Unsorted array: \n";

for (int i = 0; i < n; i++)

std::cout << arr[i] << " ";

std::cout << std::endl;

// Засекаем время перед сортировкой

auto start = std::chrono::steady\_clock::now();

insertionSort(arr, n);

// Засекаем время после сортировки

auto end = std::chrono::steady\_clock::now();

std::cout << "Sorted array: \n";

for (int i = 0; i < n; i++)

std::cout << arr[i] << " ";

// Вычисляем время выполнения сортировки

std::chrono::duration<double> diff = end - start;

std::cout << "\nTime taken: " << diff.count() << " seconds\n";

delete[] arr;

return 0;

}

Python

import random  
import time  
  
def insertion\_sort(arr):  
 n = len(arr)  
 for i in range(1, n):  
 key = arr[i]  
 j = i - 1  
 while j >= 0 and arr[j] > key:  
 arr[j + 1] = arr[j]  
 j = j - 1  
 arr[j + 1] = key  
  
random.seed(time.time())  
n = random.randint(1, 7000)  
arr = [random.uniform(1, 1000) for \_ in range(n)]  
  
print("Unsorted array: ")  
print(arr)  
  
start = time.monotonic()  
insertion\_sort(arr)  
end = time.monotonic()  
  
print("Sorted array: ")  
print(arr)  
print(f"Time taken: {end - start:.5f} seconds")