МГТУ им. Баумана

Лабораторная работа №3

По курсу: "Анализ алгоритмов"

Алгоритмы сортировки

Работу выполнила: Козаченко Александр, ИУ7-54Б

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

Москва, 2019

Оглавление

Введение

На данный момент существует огромное количество вариаций сортировок. Эти алгоритмы необходимо уметь сравнивать, чтобы выбирать наилучше подходящие в конкретном случае.

Эти алгоримы оцениваются по:

- Времени быстродействия
- Затратам памяти

Целью данной лабораторной работы является изучение применений алгоритмов сортировки, обучение расчету трудоемкости алгоритмов.

1 | Аналитическая часть

1.1 Описание алгоритмов

Сортировка массива — одна из самых популярных операций над массивом. Алгоритмы реализуют упорядочивание элементов в списке. В случае, когда элемент списка имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки **Область применения:**

- физика,
- математика,
- экономика,
- итд.

1.1.1 Сортировка вставками

На каждом шаге выбирается один из элементов неотсортированной части массива (максимальный/минимальный) и помещается на нужную позицию в отсортированную часть массива.

1.1.2 Гномья сортировка

Это метод, которым садовый гном сортирует линию цветочных горшков. По существу он смотрит на текущий и предыдущий садовые горшки: если они в правильном порядке, он шагает на один горшок вперёд, иначе он меняет их местами и шагает на один горшок назад. Граничные условия: если нет предыдущего горшка, он шагает вперёд; если нет следующего горшка, он закончил.

1.1.3 Быстрая сортировка

Массив разбивается на два (возможно пустых) подмассива. Таких, что в одном подмассиве каждый элемент меньше либо равен опорному, и при этом не превышает любой элемент второго подмассива. Опорный элемент вычислыется в ходе процедуры разбиения. Подмассивы сортируются с помощью рекурсивного вызова процедуры быстрой сортировки. Поскольку подмассивы сортируются на месте, для их объединения не трубуются никакие действия.

2 | Конструкторская часть

2.1 Схемы алгоритмов

В данном разделе будут рассмотрены схемы алгоритмов гномьей сортировки (2.1), сортировки вставками (2.2), быстрой сортировки (2.3).

2.2 Трудоемкость алгоритмов

Введем модель трудоемкости для оценки алгоритмов:

1. базовые операции стоимостью 1 — +, -, *, /, =, ==, <=, >=, !=, +=, [], получение полей класса

2. оценка трудоемкости цикла: Fц = a + N*(a + Fтела), где a - условие цикла

3. стоимость условного перехода возьмем за 0, стоимость вычисления условия остаётся.

Далее будет приведены оценки трудоемкости алгоритмов. Построчная оценка трудоемкости гномьей сортировки (Табл. 2.1).

2.2.1 Гномья сортировка

Табл. 2.1 Построчная оценка веса

Код	Вес
int i = 0;	1
while (i < len(arr))	2
if not i or arr[i - 1] <= arr[i]:	2
else :	1
arr[i], arr[i - 1] = arr[i - 1] , arr[i]	2 + 3
i -= 1	1

Лучший случай: Массив отсортирован; не произошло ни одного обмена за 1 проход -> выходим из цикла
Трудоемкость: $1 + 1 + n * (2 + 7 + 1 + 3) + 2 = 13n + 4 = O(n)$
Худший случай: Массив отсортирован в обратном порядке; в каждом случае происходил обмен
Трудоемкость: $1+1+n*(n*(7+5+1+3)+1+1)+2 = n*(16n+2)+4 = 16n^2 + 2n + 4 = O(n^2)$

2.2.2 Сортировка вставками

Лучший случай: отсортированный массив. При этом все внутренние циклы состоят всего из одной итерации.

Трудоемкость: $T(n) = 3n + ((2+2+4+2) * (n-1)) = 3n + 10(n-1) = 13n - 10 = O(n)$

Худший случай: массив отсортирован в обратном нужному порядке. Каждый новый элемент сравнивается со всеми в отсортированной последовательности. Все внутренние циклы будут состоять из j итераций.
Трудоемкость: $T(n) = 3n+(2+2)(n-1)+4\left(\frac{n(n+1)}{2} - 1\right)+5\frac{n(n-1)}{2}+3(n-1) = 3n+4n-4+2n^2+2n-4+2.5n^2-2.5n+3n-3 = 4.5n^2+9.5n-11 = O(n^2)$

2.2.3 Быстрая сортировка

Лучший случай: сбалансированное дерево вызовов $O(n * log(n))$ В наиболее благоприятном случае процедура PARTITION приводит к двум подзадачам, размер каждой из которых не превышает $\frac{n}{2}$, поскольку размер одной из них равен $\frac{n}{2}$, а второй$\frac{n}{2} - 1$. В такой ситуации быстрая сортировка работает намного производительнее, и время ее работы описывается следующим рекуррентным соотношением: $T(n) = 2T(\frac{n}{2}) + O(n)$,где мы не обращаем внимания на неточность, связанную с игнорированием функций “пол” и “потолок”, и вычитанием 1. Это рекуррентное соотношение имеет решение ; $T(n) = O(nlgn)$. При сбалансированности двух частей разбиения на каждом уровне рекурсии мы получаем асимптотически более быстрый алгоритм.

Фактически любое разбиение, характеризующееся конечной константой пропорциональности, приводит к образованию дерева рекурсии высотой $O(lgn)$ со стоимостью каждого уровня, равной $O(n)$. Следовательно, прилюбой постоянной пропорции разбиения полное время работы быстрой сортировки составляет $O(nlgn)$.

Худший случай: несбалансированное дерево $O(n^2)$ Поскольку рекурсивный вызов процедуры разбиения, на вход которой подается массив размером 0, приводит к немедленному возврату из этой процедуры без выполнения каких-ли-бо операций, $T(0) = O(1)$. Таким образом, рекуррентное соотношение, описывающее время работы процедуры в указанном случае, записывается следующим образом: $T(n) = T(n-1) + T(0) + O(n) = T(n-1) + O(n)$. Интуитивно понятно, что при суммировании промежутков времени, затрачиваемых на каждый уровень рекурсии, получается арифметическая прогрессия, что приводит к результату $O(n^2)$.

2.3 Вывод

Гномья сортировка: лучший - $O(n)$, худший - $O(n^2)$
Сортировка вставками: лучший - $O(n)$, худший - $O(n^2)$
Быстрая сортировка: лучший - $O(nlgn)$, худший - $O(n^2)$

При этом сортировка вставками быстрее гномьей с флагом в худшем случае т.к. имеет меньший коэффициент. Вставки $4.5n^2$, гномья сортировка $16n^2$.

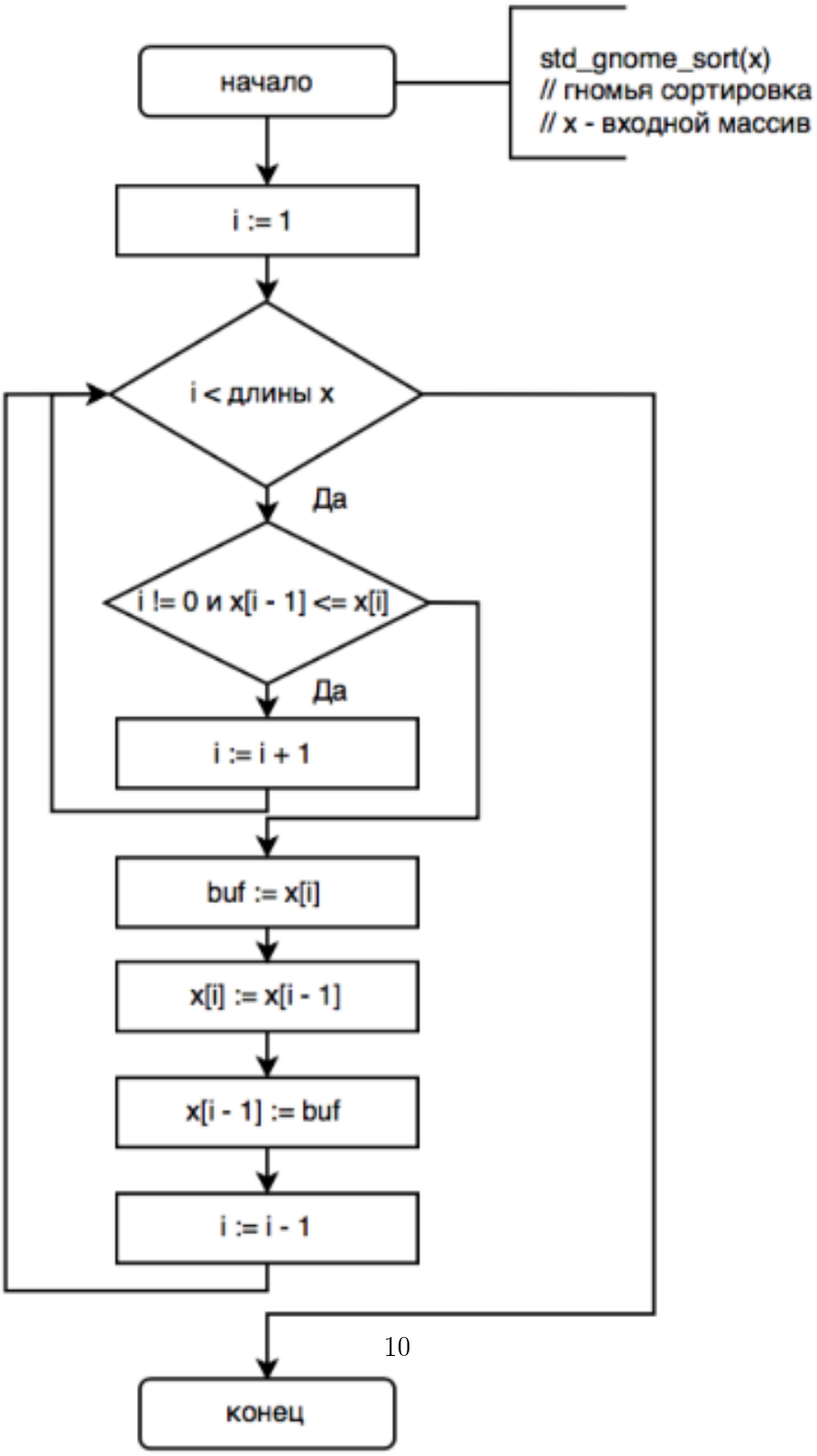

Рис. 2.1: Схема алгоритма гномьей сортировки

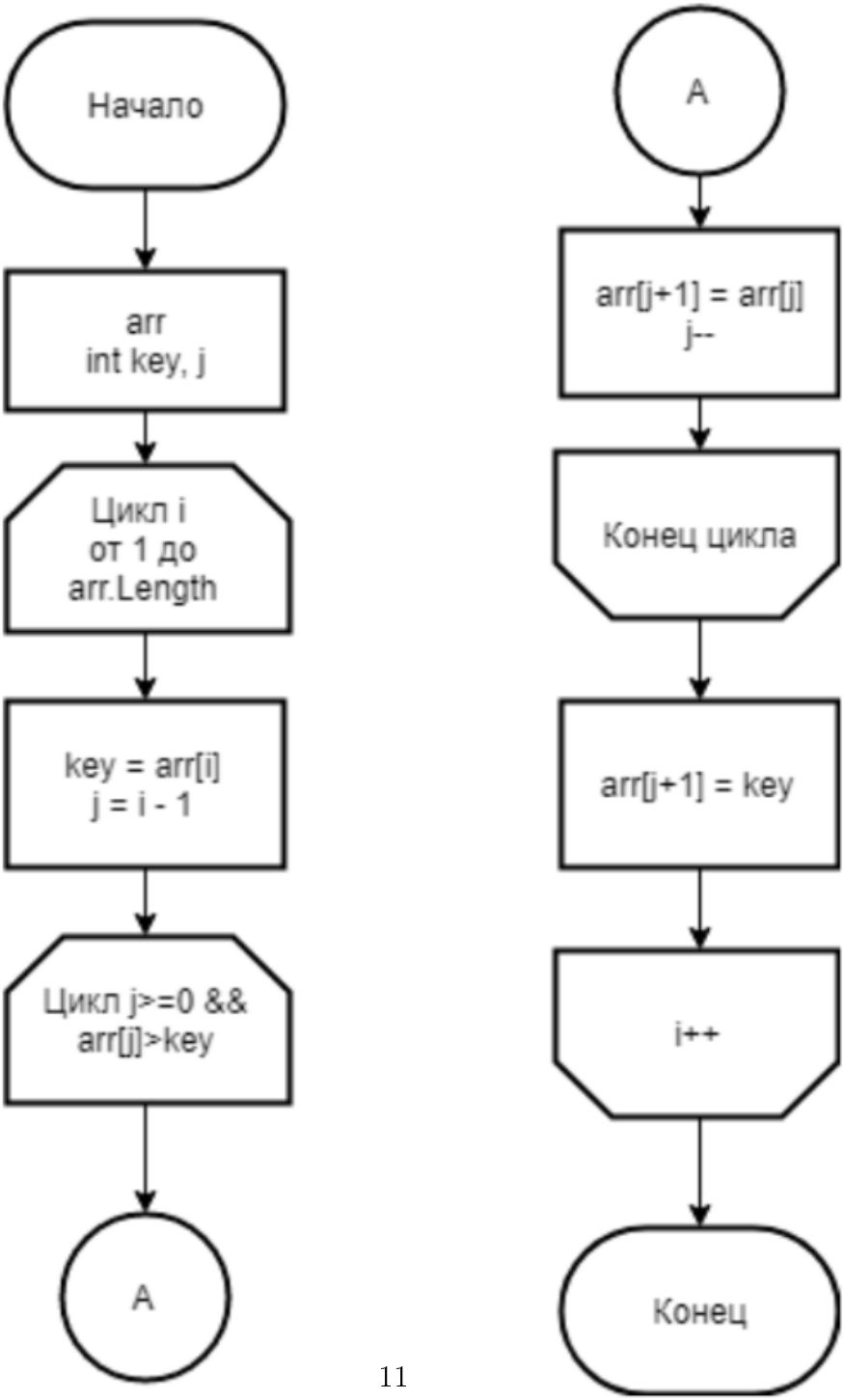

Рис. 2.2: Схема алгоритма сортировки вставками

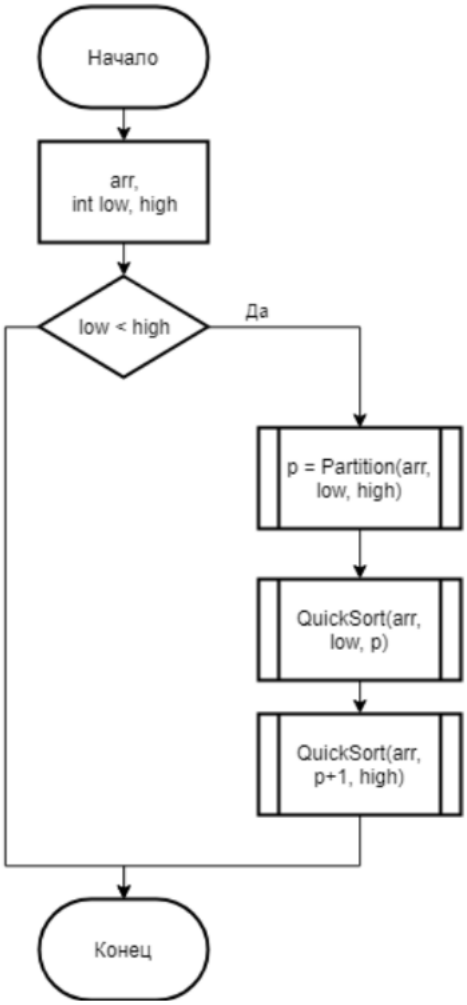

Рис. 2.3: Схема алгоритма быстрой сортировки

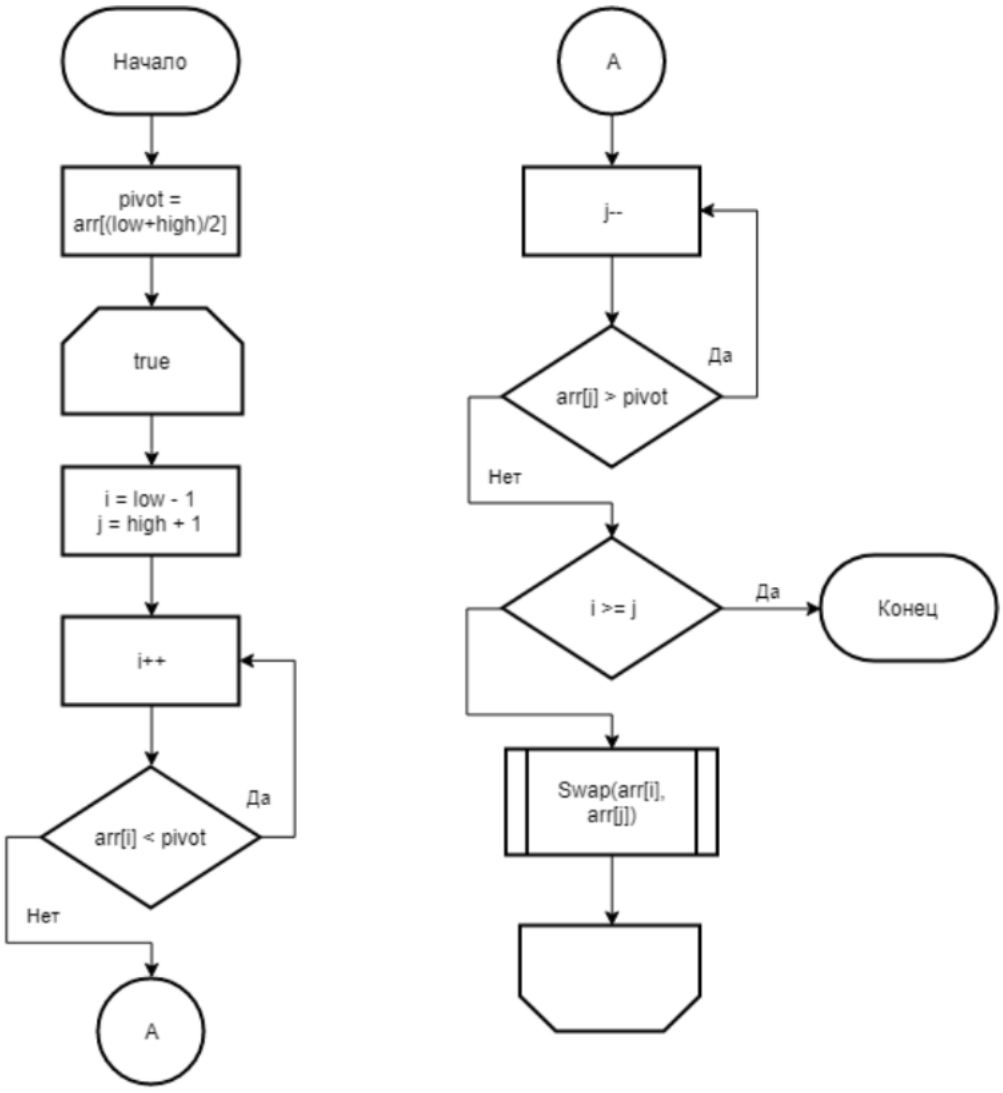

Рис. 2.4: Схема функции partition

3 | Технологическая часть

3.1 Выбор ЯП

В качестве языка программирования был выбран C#, а средой разработки Visual Studio. Время работы алгоритмов было замерено с помощью класса Stopwatch.

3.2 Описание структуры ПО

Ниже представлена IDEF0 диаграмма (3.1), описывающая структуру программы.

Рис. 3.1: Функциональная схема сортировки массива (IDEF0 диаграмма 1 уровня)

3.3 Сведения о модулях программы

Программа состоит из:

- Program.cs - главный файл программы, в котором располагается точка входа в программу и функция замера времени.
- Sortiong.cs - файл класса sorting, в котором находятся алгоримы сортировки

3.4 Листинг кода

Листинг 3.1: Алгоритм гномьей сортировки

```
    def gnome_sort(arr):
      i = 1
      while i < len(arr):
        if not i or arr[i - 1] <= arr[i]:
          i += 1
        else :
          arr[i], arr[i - 1] = arr[i - 1] , arr[i]
          i -= 1
      return arr
```

Листинг 3.2: Алгоритм сортировки вставками

```
    def insertion_sort(arr):
      for i in range(1, len(arr)):
        key = arr[i]
        j = i-1
        while j >=0 and key < arr[j] :
          arr[j+1] = arr[j]
          j -= 1
        arr[j+1] = key
      return arr
```

Листинг 3.3: Алгоритм быстрой сортировки

```
    def quick_sort(nums):
      def _quick_sort(items, low, high):
        if low < high:
          split_index = partition(items, low, high)
          _quick_sort(items, low, split_index)
          _quick_sort(items, split_index + 1, high)
      _quick_sort(nums, 0, len(nums) - 1)
```

Листинг 3.4: Алгоритм поиска опорного элемента

```
    def partition(nums, low, high):
      pivot = nums[(low + high) // 2]
      i = low - 1
      j = high + 1
      while True:
        i += 1
        while nums[i] < pivot:
          i += 1
        j -= 1
        while nums[j] > pivot:
          j -= 1
        if i >= j:
          return j
        nums[i], nums[j] = nums[j], nums[i]
```

3.5 Вывод

В данном разделе были представлены реализации алгоритмов гномьей сортировки, сортировки вставками и быстрой сортировки.

4 | Исследовательская часть

Был проведен замер времени работы каждого из алгоритмов.

4.1 Примеры работы программы

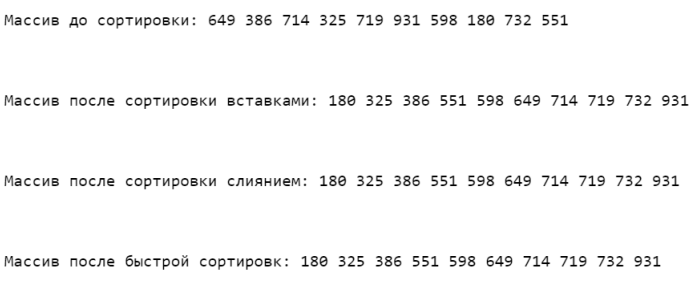

Рис. 4.1: Сортировка массива, заполненного случайными числами

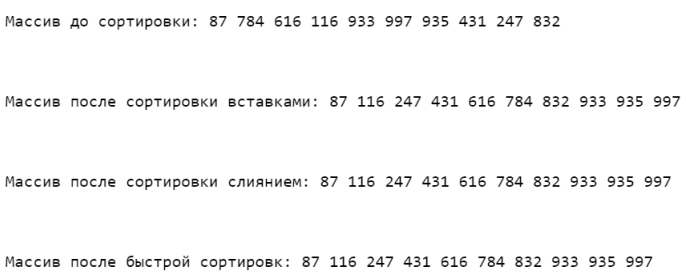

Рис. 4.2: Сортировка массива, заполненного случайными числами

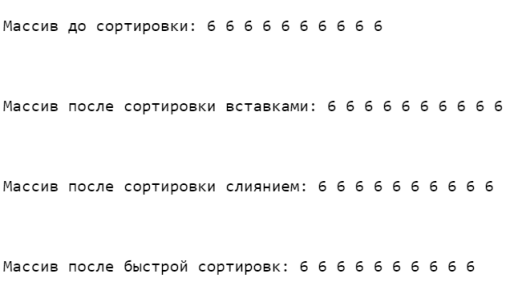

Рис. 4.3: Сортировка массива, заполненного одинаковыми числами

```
Массив до сортировки: 0 0 0 0 0 0 0 0 0 0

Массив после сортировки вставками: 0 0 0 0 0 0 0 0 0 0

Массив после сортировки слиянием: 0 0 0 0 0 0 0 0 0 0

Массив после быстрой сортировк: 0 0 0 0 0 0 0 0 0 0
```

Рис. 4.4: Сортировка пустого массива

4.2 Эксперимент

В рамках данного эксперимента было произведено сравнение времени выполнения трех алгоритмов в лучшем/худшем/случайном случае заполения массива. При длине массивов от 100 до 1000 элементов с шагом 100. На предоставленных ниже графиках (Рис. 4.1), (Рис. 4.2), (Рис. 4.3) по оси l идет длина массива, а по оси t - время сортировки в тиках.

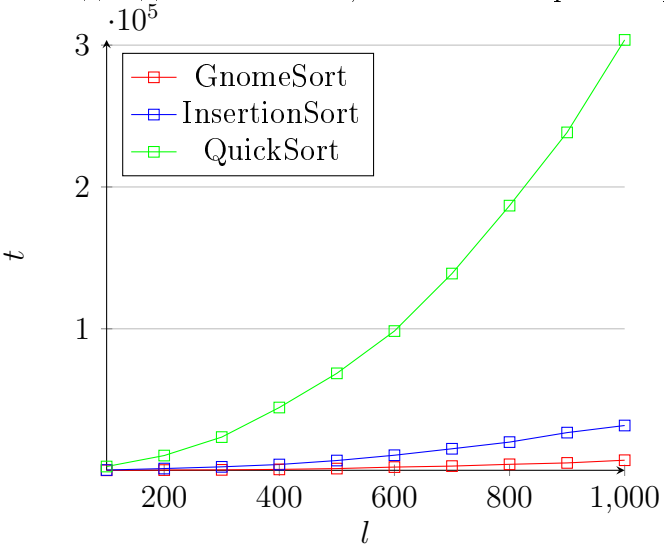

Рис. 4.1: Сравнение времени в лучшем случае

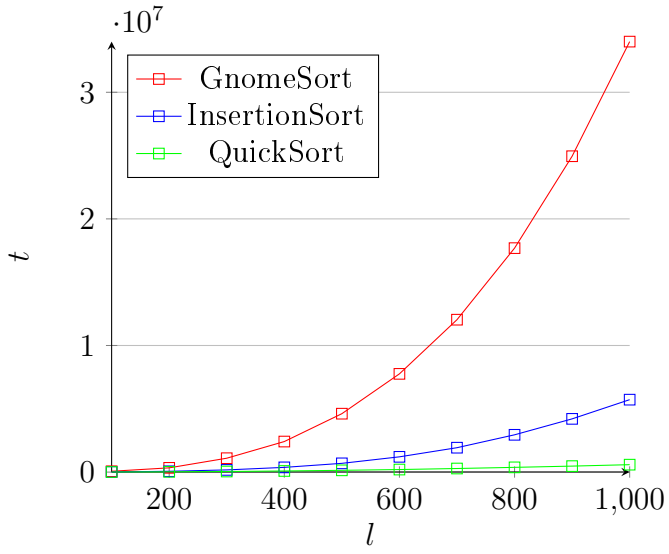

Рис. 4.2: Сравнение времени в худшем случае

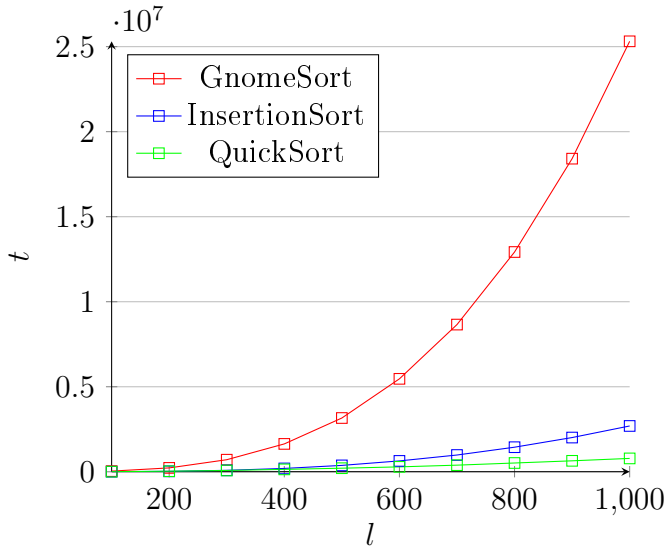

Рис. 4.3: Сравнение времени при случайном заполнении массива

4.3 Вывод

По результатам тестирования выявлено, что все рассматриваемые алгоритмы реализованы правильно. Самым быстрым алгоритмом, при ис-

пользовании случайного заполнения, оказался алгоритм быстрой сортировки, а самым медленным — алгоритм гномьей сортировки.

Заключение

В ходе работы были изучены алгоритмы сортировки массива: гномья, вставки, быстрая сортировка. Выполнено сравнение всех рассматриваемых алгоритмов. В ходе исследования был найден оптимальный алгоритм. Изучены зависимости выполнения алгоритмов от длины массива. Также реализован программный код продукта.

Список литературы

1. Кормен Т. Алгоритмы: построение и анализ [Текст] / Кормен Т. - Вильямс, 2014. - 198 с. - 219 с.