МГТУ им. Баумана

Лабораторная работа №3

По курсу: "Анализ алгоритмов"

Сортировки

Работу выполнил: Рязанов М. С., ИУ7-52Б

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

Москва, 2019

Оглавление

Введение

Алгоритм сортировки - это алгоритм упорядочивания элементов в списке в соответсвии с определенным на множестве значений отношением порядка. Свойство упорядоченности элеметов применяется является необходимым критерием некоторых алгоритмов(двоичный поиск), а также может быть полезно при анализе данных, так как облегчает восприятие данных. Поэтому оптимизиция скорости работы алгоритмов сортировок является важной задачей.

Целью данной лабораторной работы является изучение и сравнения алгоритмов сортировок. Для достижения поставленной цели необходимо решить следующие задачи:

- проанализировать существующие методы решения задачи;
- разработать алгоритмы решения задачи;
- оптимизировать алгоритмы и расчитать их трудоемокость;
- выбрать технологии для последующей реализации и иследования алгоритмов;
- реализовать разработанные алгоритмы;
- произвести тестирование корректности работы реализаций;
- сравнить быстродействие реализаций;
- описать и обосновать полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1 | Аналитическая часть

В данном разделе рассматриваются основные типы сортировок, а также подробнее анализируются следующие алгоритмы:

- сортировка вставками;
- сортировка слиянием;
- поразрядная сортировка по младшему разряду.

1.1 Классификация сортировок

1.1.1 Устойчивость

Свойство устойчивости сортировки означает, что сортировка не меняет относительный порядок сортируемых элементов, имеющих одинаковые ключи, по которым происходит сортировка. В соответсвии с наличием или отсутсвием данного свойства различают следующие виды сортировок:

- устойчивые;
- неустойчивые.

1.1.2 Использование дополнительной памяти

Для сортировки элемнтов алгоритм может использовать дополнительную память. В зависимости от объема используемой алгоритмом сортировки различают:

- локальные - используют $O(1)$ дополнительной памяти;

- нелокальные - используют дополнительный объем пямяти, зависящий от числа сортируемых элементов

1.1.3 Использование операций сравнения

Алгоритмы, использующие для сортировки сравнение элементов между собой, называются основанными на сравнениях. Минимальная трудоемкость худшего случая для этих алгоритмов составляет $O(n \cdot logn)$, но они отличаются гибкостью применения. Для специальных случаев (типов данных) существуют более эффективные алгоритмы.

1.1.4 Сортировка вставками

В данном алгоритме сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов. В соответсвии с приведенной выше классификацией данный алгоритм можно отнести к следующим классам:

- устойчивый;
- локальный;
- использующий сравнение;

1.1.5 Сортировка слиянием

Сортировка базируется на методе "Разделяй и властвуй задача сортировки решается для подмассивов меньшего размера, а затем отсортированные массивы сливаются в один. В соответсвии с приведенной выше классификацией данный алгоритм можно отнести к следующим классам:

- устойчивый;
- нелокальный;
- использующий сравнение;

1.1.6 Поразрядная сортировка по младшему разряду

Элементы перебираются по порядку и группируются по самому младшему разряду (сначала все, заканчивающиеся на 0, затем заканчивающиеся на 1, ..., заканчивающиеся на 9). Возникает новая последовательность. Затем группируются по следующему разряду с конца, затем по следующему и т.д. пока не будут перебраны все разряды, от младших к старшим. Является модификацией алгоритма сортировки подсчетом, но расширяет диапозон, который могу принимать элементы сортируемого множества, так как не требует создания массива размером с диапозон принимаемы значений. В соответсвии с приведенной выше классификацией данный алгоритм можно отнести к следующим классам:

- неустойчивый;
- нелокальный;
- не использующий сравнение;

1.1.7 Выводы

В аналитическом разделе выполнено следующее:

- расмотрены основные типы сортировок;
- рассмотрены алгоритмы сортировок вставками, слиянием, поразрядной сортировки.

2 | Конструкторская часть

2.1 Разработка алгоритмов

2.1.1 Алгоритм сортировки вставками

В данном алгоритме значения просматриваются поочередно, и каждый последующий элемент добавляется в необходимое место среди уже добавленных. На рисинуке 2.1 приведна схема алгоритма сортировки вставка-

ми

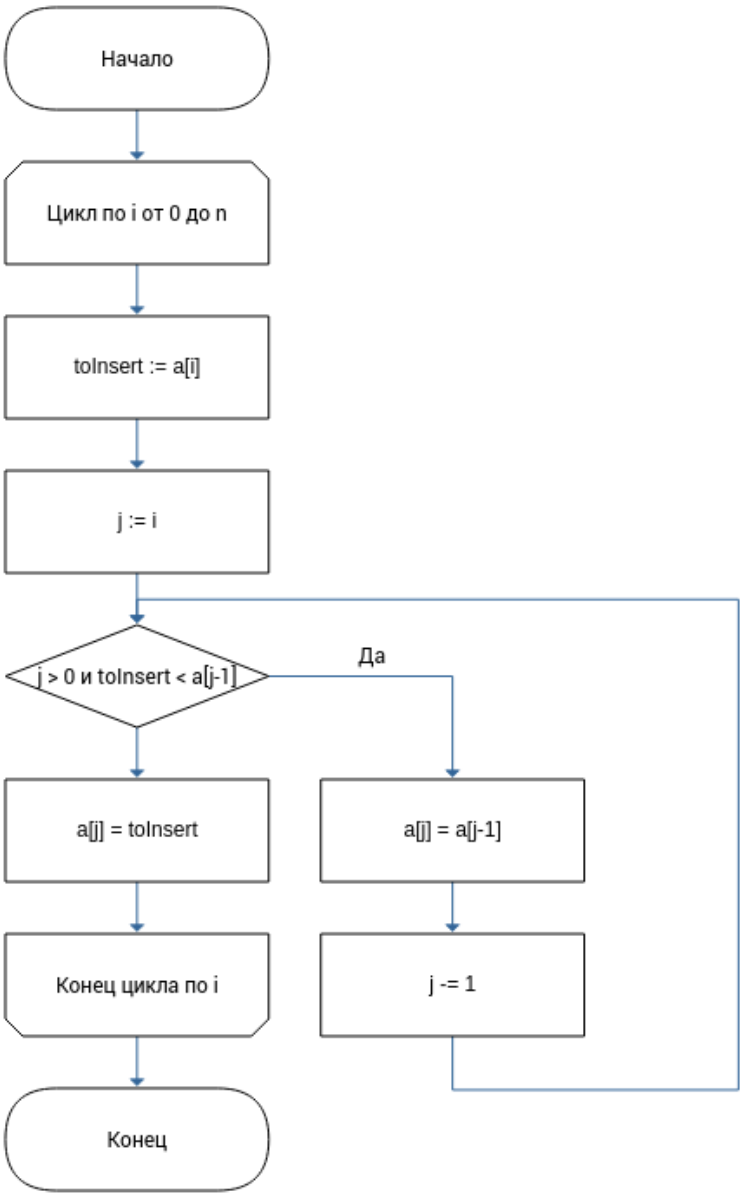

Рис. 2.1: Схема алгоритма сортировки вставками

2.1.2 Алгоритм сортировки слиянием

Данный алгоритм базируется на методе "Разделяй и властвуй" - задача сначала решается для подмассивов меньшего размера, которые затем сливаются в один отсортированный. На рисунке 2.2, 2.3 приведена схема алгоритма сортировки слиянием.

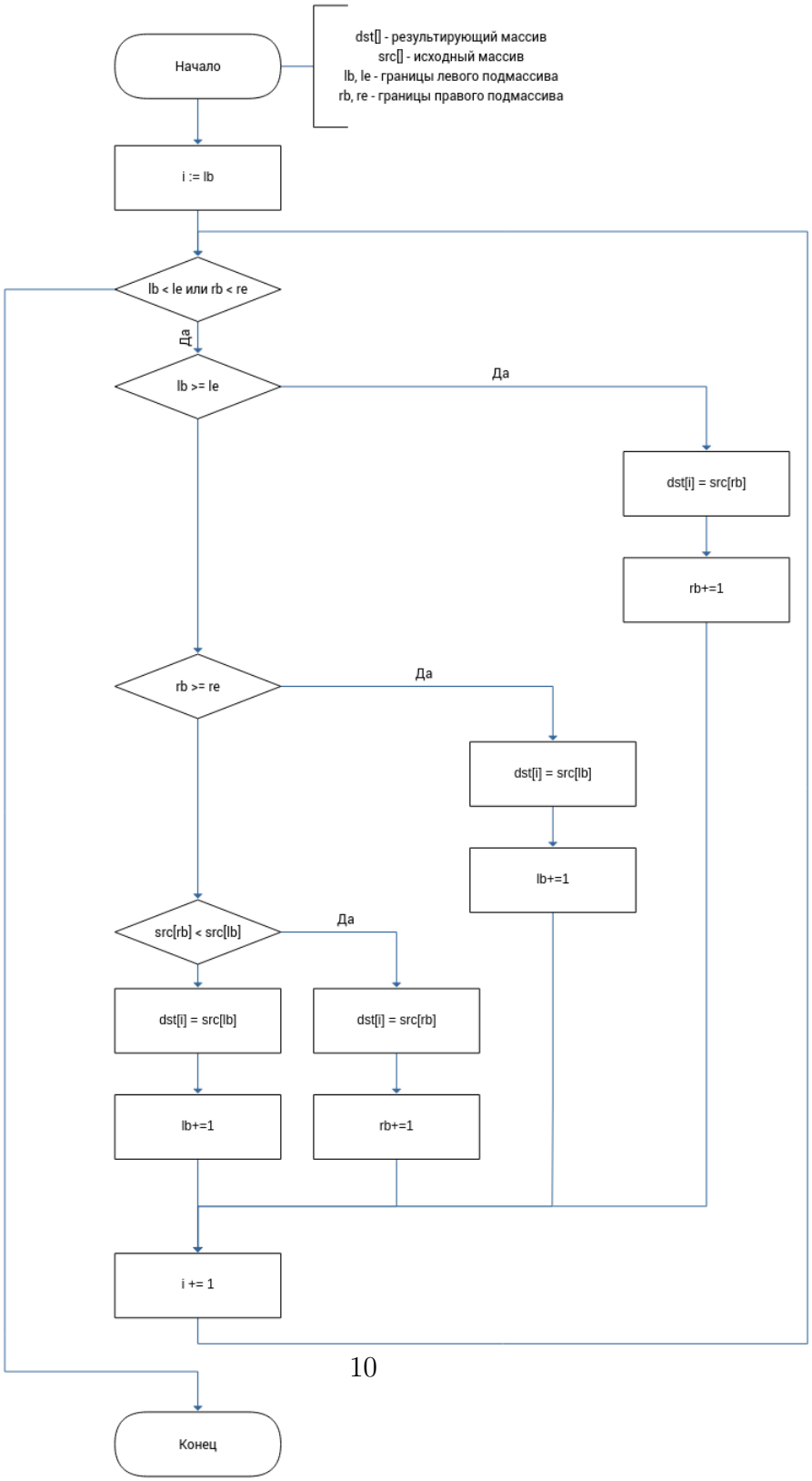

Рис. 2.2: Схема алгоритма merge

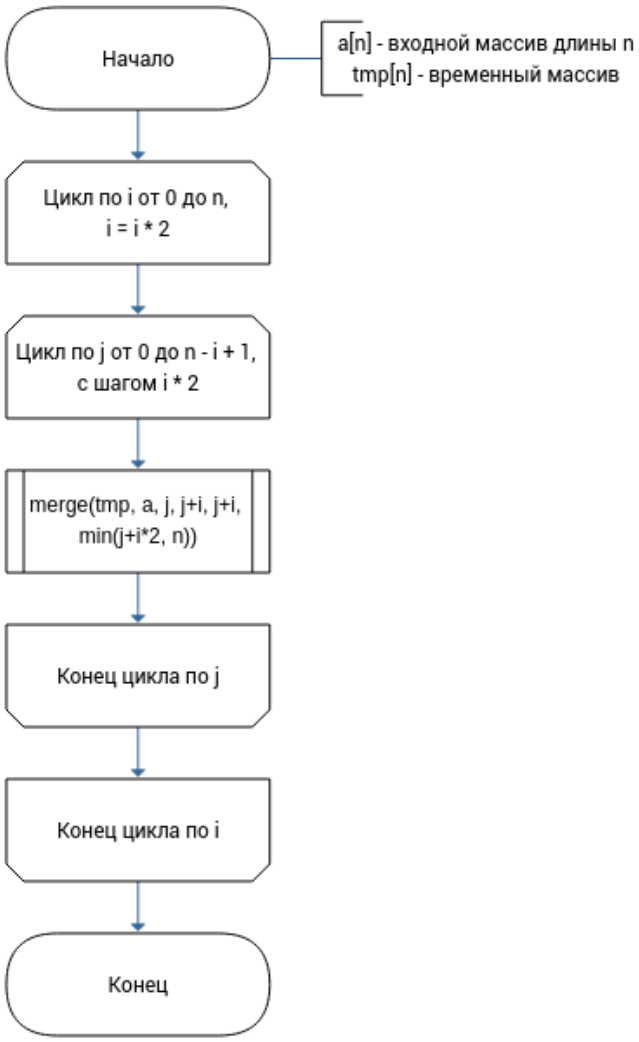

Рис. 2.3: Схема алгоритма сортировки слиянием

2.1.3 Алгоритм поразрядной сортировки по младшему разряду

В данном алгоритме элементы перебираются по порядку и группируются по самому младшему разряду (сначала все, заканчивающиеся на 0, затем заканчивающиеся на 1, ..., заканчивающиеся на 9). Возникает но-

вая последовательность. Затем группируются по следующему разряду с конца, затем по следующему и т.д. пока не будут перебраны все разряды, от младших к старшим. Так как числа хранятся в виде последовательности бит, то удобно рассматривать в качестве разряда несколько соседних бит числа. На рисунке 2.4 приведена схема алгоритма поразрядной сортировки.

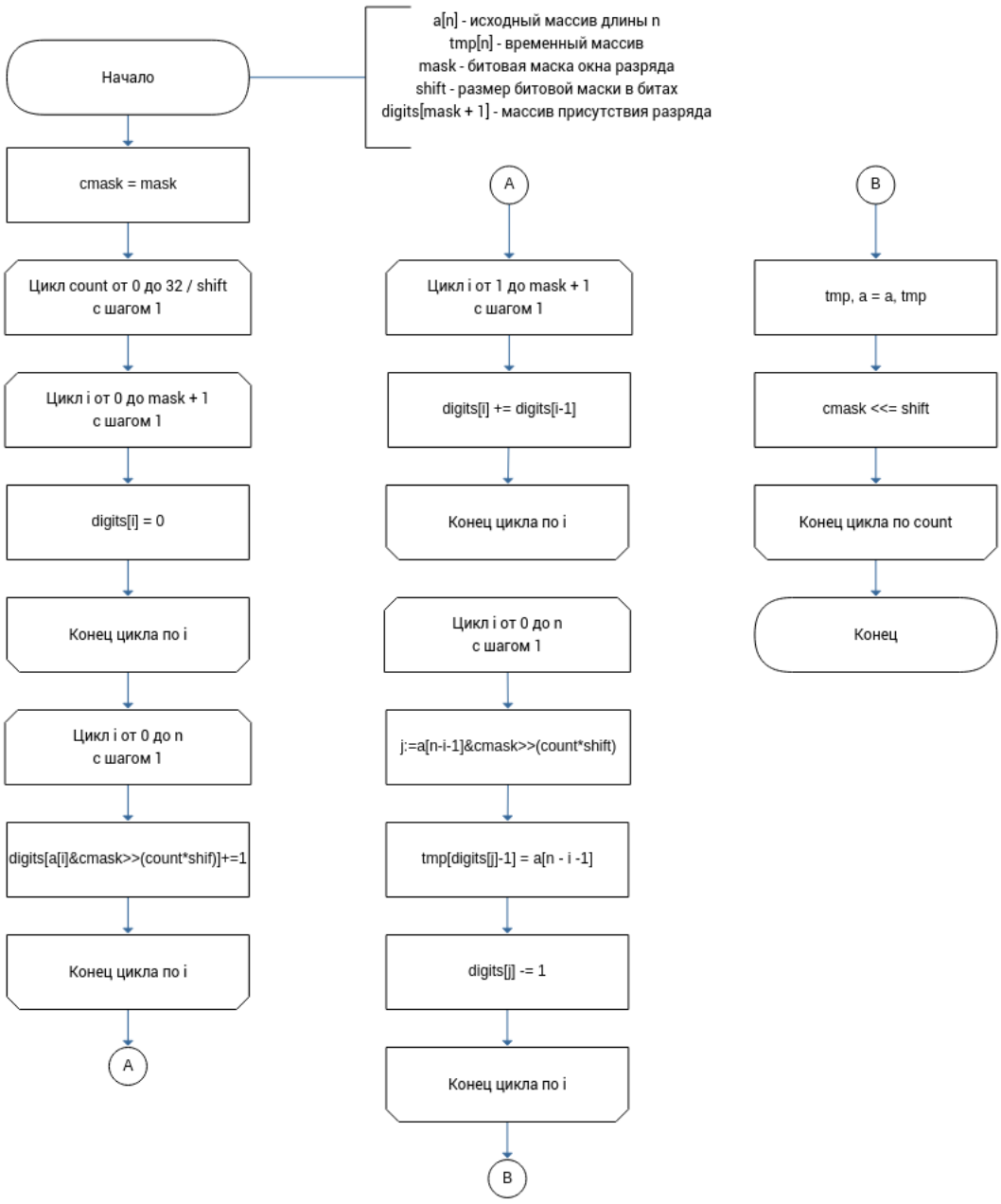

Рис. 2.4: Схема алгоритма поразрядной сортировки

Выводы: В данном разделе были разработаны схемы следующих алгоритмов сортировок:

- сортировка вставками (рисунок 2.1);
- сортировка слиянием (рисунок 2.3);
- поразрядная сортировка по младшему разряду(рисунок 2.4).

3 | Технологическая часть

3.1 Выбор ЯП

В качестве языка программирования был выбран go[1] т.к он сочетает в себе эффективность, которая позволит оценить характеристики работы алгоритмов максимально точно, и удобство реализации. Также имеет встроенную библиотеку для проведение модульного тестирования.

Время работы алгоритмов было замерено с помощью функции Now() из пакета time. Для генерации случайных чисел использовалась функция Int() из пакета rand.

3.2 Сведения о модулях программы

Программа состоит из:

- main.go - главный файл программы - точка входа программы, обработка входных данных;
- sort.go - файл с реализацией алгоритмов;
- sort_test.go - файл с тестами.

Листинг 3.1: Функция сортировки вставками

```
1 func InsertSort(a []int) {
2   for i := 0; i < len(a); i++ {
3     toInsert := a[i]
4     j := i
5     for ; j > 0 && toInsert < a[j-1]; j-- {
```

```
      a[j] = a[j-1]
    }
    a[j] = toInsert
  }
}
```

Листинг 3.2: Функция слияния

```
func merge(dst, src []int, lb, le, rb, re int) {
  for i := lb; lb < le || rb < re; i++ {
    if lb >= le {
      dst[i] = src[rb]
      rb++
    } else if rb >= re {
      dst[i] = src[lb]
      lb++
    } else if src[rb] < src[lb] {
      dst[i] = src[rb]
      rb++
    } else {
      dst[i] = src[lb]
      lb++
    }
  }
}
```

Листинг 3.3: Функция сортировки слиянием

```
func MergeSort(a []int) {
  tmp := make([]int, len(a))
  for k := 1; k < len(a); k <<= 1 {
    for i := 0; i < len(a)-k+1; i += k << 1 {
      merge(tmp, a, i, i+k, i+k, min(i+k<<1, len(a)))
    }
    copy(a, tmp)
  }
}
```

Листинг 3.4: Функция поразрядной сортировки по младшему разряду

```
func RadixSort(a []int) {
  digits := make([]int, 4096)
  anses := make([]int, len(a))
  nDigits := len(digits)
  mask := 0xFFF
  shift := uint(12)
  count := uint(0)

  for mask > 0 {
    // Обнуляем количество чисел
    for i := range digits {
      digits[i] = 0
    }
    // Подсчитываем число чисел
    for _, val := range a {
      digits[val&mask>>(count*shift)]++
    }
    // Пересчитываем границы диапозона
    for i := 1; i < nDigits; i++ {
      digits[i] += digits[i-1]
    }
    // Упорядочивем значения
    for i := range a {
      anses[digits[a[len(a)-i-1]&mask>>(count*shift)]-1] =
   a[len(a)-i-1]
      digits[a[len(a)-i-1]&mask>>(count*shift)]--
    }
    a, anses = anses, a
    mask <<= shift
    count++
  }
}
```

3.3 Тестирование

Тестирование было организовано с помощью пакета **testing**.

Для тестирования функций сортировок были составлены следующие тестовые наборы

- массив отсортированных элементов;
- массив элементов отсортированных в обратном порядке;
- массив из 1 элемента;
- массив с отрицательными элементами(для сортировок слиянием и вставками);
- массив с элементами с различным числом разрядов(для поразрядной сортировки).

3.4 Сравнительный анализ реализаций

Примем следующую теоретическую модель вычислений для оценки трудуемкости алгоритмов:

- операции единичной стоимости(арифмитические, сравнения, обращение по адресу)
- Циклы определяются формулой:

$$f_{cycle} = f_{init} + f_{cmp} + n \times (f_{inner} + f_{inc} + f_{cmp}), \tag{3.1}$$

  где
  n - количество итераций цикла,
  f_{cycle} - трудоемкость цикла,
  f_{init} - трудоемкость инициализации,
  f_{cmp} - трудоемкость сравнения,
  f_{inner} - трудоемкость тела цикла,
  f_{inc} - трудоемкость инкремента;

- переход по условию в условном операторе равен нулю, но при этом в расчет входит трудоемкость вычисления условия.

Пусть дан массив неотрицательных целых чисел A размерностью n и k - максимальное число разрядов в числах массива, $shift$ - размер разряда

3.4.1 Трудоемкость алгоритма поразрядной сортировки

Опираясь на указанные выше правила, подсчитаем трудоемкость стандартного алгоритма умножения матриц:

$$f_{rad} = 7 + \frac{k}{shift} \cdot (6 + f_1 + f_2 + f_3 + f_4), \tag{3.2}$$

где f_i - трудоемкость соответсвующего цикла

$$f_1 = 2 + 2^{shift} \cdot (2 + 2) = 2 + 2^{shift+2} \tag{3.3}$$

$$f_2 = 2 + n \cdot (2 + 5) = 7n + 2 \tag{3.4}$$

$$f_3 = 2 + 2^{shift} \cdot (2 + 4) - 6 = 6 \cdot 2^{shift} - 4 \tag{3.5}$$

$$f_4 = 2 + n \cdot (2 + 9 + 15) = 26n + 2 \tag{3.6}$$

$$f_{rad} = 7 + \frac{k}{shift} \cdot (8 + 10 \cdot 2^{shift} + 33n) = \\ = \frac{33k}{shift}n + \frac{10k}{shift}2^{shift} + \frac{8k}{shift} + 7 \tag{3.7}$$

3.4.2 Трудоемкость алгоритма сортировки вставками

Данный алгоритм имеет трудоемкость порядка $O(n^2)$ в худшем и среднем случае и порядка $O(n)$ в лучшем случае. Где худший случай - это массив отсортированный в обратном порядке, а лучший - массив отсортированный в необходимом порядке.

3.4.3 Трудоемкость алгоритма сортировки слиянием

Данный алгоритм имеет трудоемкость порядка $O(n \cdot logn)$ всегда, так как производит решение подзадач меньшего порядка и слияние в любом случае.

Выводы:

- В качестве языка программирования выбран Golang, так как прост в использовании, а также достаточно эффективен для проведения анализа времени работы разработанных алгоритмов
- Алгоритмы реализованы в виде подпрограмм, код которых приведен в листингах : 3.1, 3.2, 3.4
- Разработаны тестовые случаи для проверки корректности работы функций
- Был выбран консольный интерфейс программы, так как прост в реализации и использовании при небольшом функционале программмы.
- Были расчитаны трудоемкости реализованных алгоритмов

4 | Исследовательская часть

Для оценки эффективности реализованных функций были произведены замеры времени выполнения на целочисленных массивах разного размера. Так как поразрядая сортировка и сортировка имеют одинаковую трудоемкость при входных данных разной природу, тестирование проводилась на следующих типах массива с целью покрыть лучший, худший и средний случай для сортирвки вставками:

- отсортированный массив;
- массив, отсортированный в обратном порядке;
- массив, заполненый с помощью генератора псевдослучайных чисел.

На рисунках 4.1 и 4.2 приведены графики зависимости времени сортировки от размера массива на отсортированных данных.

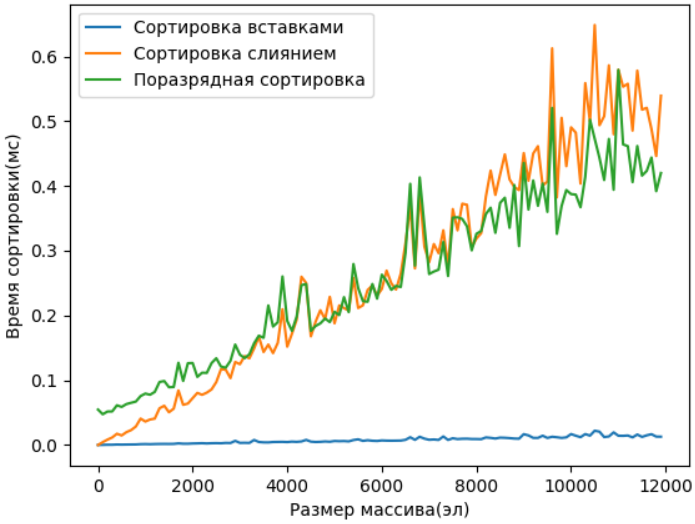

Рис. 4.1: Время выполнения на отсортированных массивах

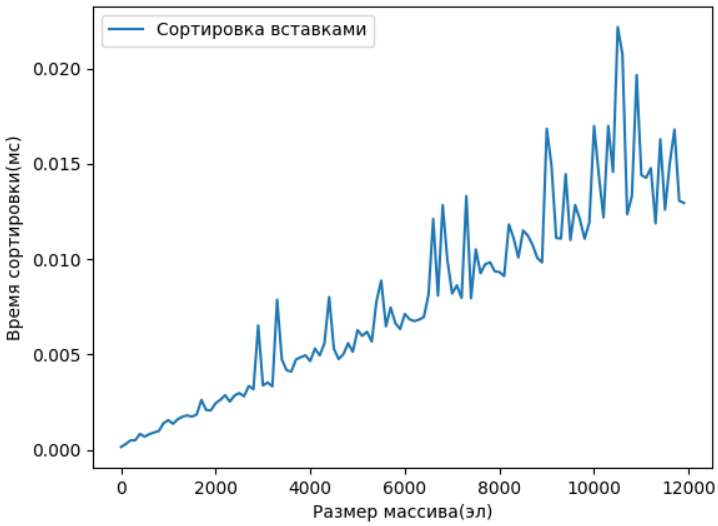

Рис. 4.2: Время выполнения сортировки вставками на отсортированных массивах

Из приведенных графиков видно, что кривая времени выполения сортировки вставки и поразрядной сортировки имеет линейную зависимость

от размера массива, сортировка слиянием имеет зависимость $nlogn$, но логарифмическая функциия растет медленно, поэтому она показывает лучший результат чем поразрядная сортировка на массивах длины меньше 6000. В тоже время алгоритмы имеют разный коэфициент при линейном члене. Из графиков видно, что наименьший коэфициент имеет сортировка вставками.

На рисунках 4.3 и 4.4 приведены графики зависимости времени выполнения сортировки от размера массива, отсортированного в обратном порядке.

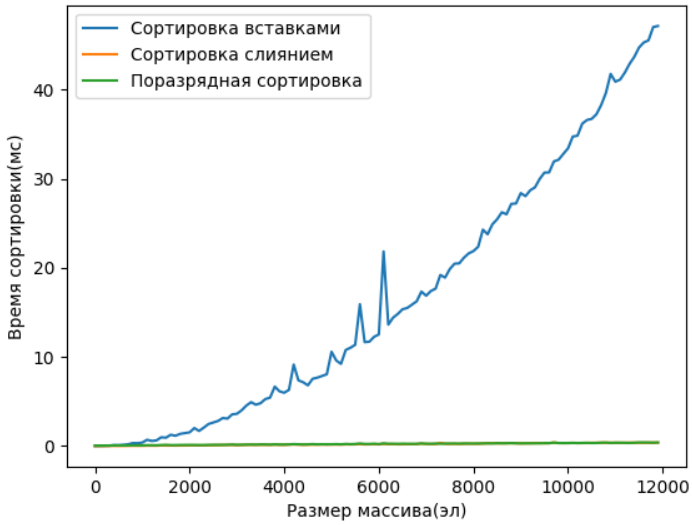

Рис. 4.3: Время выполнения сортировок на массивах, отсортированных в обратном порядке

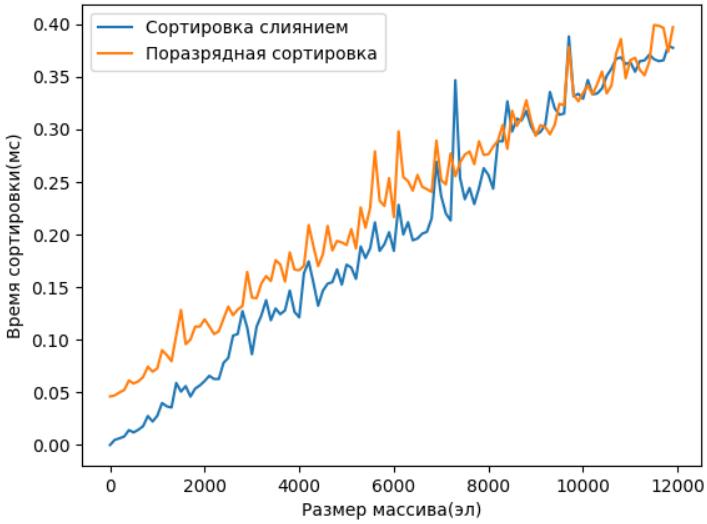

Рис. 4.4: Время выполнения поразрядной сортировки и сортировки слиянием на массивах, отсортированных в обратном порядке

Из графиков видно, что время выполнения сортировки вставками имеет квадратичную зависимость от размера массивов и на массивах размером более 1000 работает заметно медленее. Также видно, что время работы поразрядной сортировки имеет линейную зависимость от размера массива, но из-за большей константы паритет времени выполнения с сортировкой слиянием достигается лишь при массивах размером 10000.

На рисунках 4.5 и 4.6 приведены графики зависимости времени выполнения сортировки от размера массива, заполненного случайным образом.

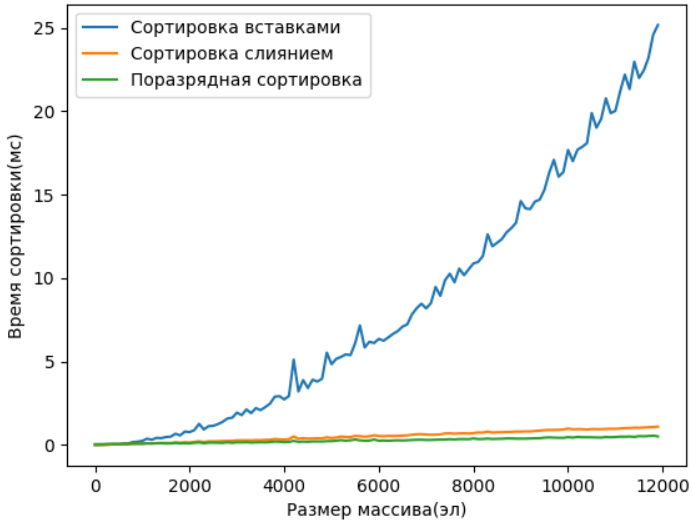

Рис. 4.5: Время выполнения сортировок на случайных массивах

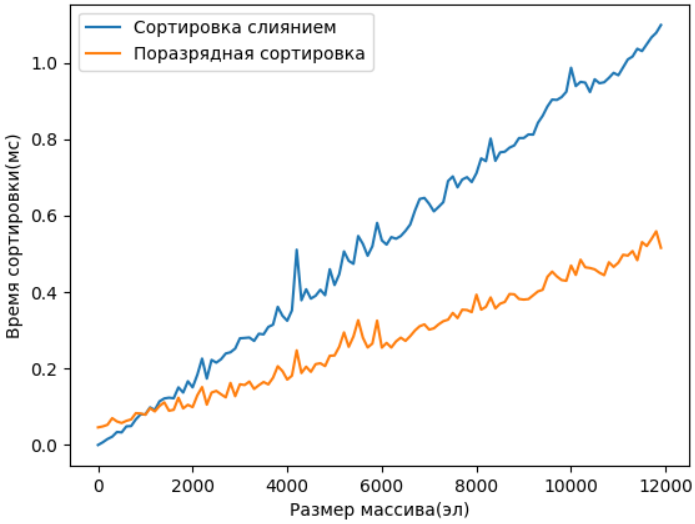

Рис. 4.6: Время выполнения поразрядной сортировки и сортировки слиянием на случайных массивах

Из графиков видно, что время выполнения сортировки вставками квадратично зависит от размера массива и на массивах размером более

1000 заметно превышает время выполнения сортировки слиянием и поразрядной сортировки. Также видно, что поразрядная сортировка имеет линейную сложность и на массивах размера более 1000 имеет время выполния меньше чем сортировка слиянием, имеющая сложность $nlogn$.

Выводы:

- на почти отсортированных данных, и данных малого размера следуюет использовать сортировку вставками, так как она имеет меньшую константу и линейно зависит от размера входных данных;
- в случае неотсортированных данных выбор зависит от типа данных, так для целых неотрицательных чисел лучшее время выполнения на больших объемах данных показывает поразрядная сортировка, если же нужна универсальная и устойчивая сортировка, работающая с произвольным ключом, по которому происходит сравнение, то среди рассмотренных наилучший результат покажет сортировка слиянием.

Заключение

Был изучены и реализованы алгоритмы сортировок: вставками, слиянием, поразрядная. Для поразрядной сортировки была расчитана трудоемкость.

Экспериментально было подтверждено различие во временной эффективности разных алгоритмов, обоснован выбор того или иного алгоритма на разных типах данных.

Список литературы

[1] *Golang documentation*. URL: `https://godoc.org/`. (accessed: 02.10.2019).