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, искусственный интеллект и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе № 3

по курсу «Анализ Алгоитмов»

на тему: «Трудоемкость сортировок»

Студент ИУ7-55Б _____ Романов С. К.
(Группа) (Подпись, дата) (И. О. Фамилия)

Преподаватель _____ Волкова. Л. Л.
(Подпись, дата) (И. О. Фамилия)

2022 г.

Оглавление

1 ВВЕДЕНИЕ

Цель лабораторной работы: изучить, реализовать и протестировать алгоритмы сортировки, оценить их трудоемкость.

Задачи лабораторной работы:

1. Изучить алгоритмы сортировки.
   - Пузырьковая сортировка.
   - Сортировка подсчетом.
   - Итеративная быстрая сортировка.
2. Оценить трудоемкость алгоритмов и сравнивать их временые характеристики экспериментально.
3. Сделать выводы на основе полученных результатов.

Для достижения поставленных целей и задач необходимо:

1. Изучить теоретические основы алгоритмов сортировки.
2. Реализовать алгоритмы сортировки.
3. Протестировать корректность алгоритмов сортировки.
4. Провести экспериментальное исследование.

В ходе работы будут затронуты следующие темы:

1. Алгоритмы сортировки.
2. Сложность алгоритмов.
3. Тестирование алгоритмов

2 Аналитическая часть

2.1 Пузырьковая сортировка

2.1.1 Описание алгоритма

procedure BUBBLESORT(A)
    $n \leftarrow \texttt{len}(A)$
    for $i \leftarrow 1$ **to** n **do**
        for $j \leftarrow 1$ **to** $n - i$ **do**
            if $A[j] > A[j + 1]$ **then**
                $SWAP(A[j], A[j + 1])$
            end if
        end for
    end for
end procedure

2.1.2 Анализ алгоритма

- Время выполнения алгоритма: $O(n^2)$
- Память: $O(1)$

2.2 Сортировка подсчетом

2.2.1 Описание алгоритма

procedure COUNTING SORT(A)
    $n \leftarrow \texttt{len}(A)$
    $C \leftarrow \texttt{new array}(n)$
    for $i \leftarrow 1$ **to** n **do**
        $C[i] \leftarrow 0$
    end for
    for $i \leftarrow 1$ **to** n **do**
        $C[A[i]] \leftarrow C[A[i]] + 1$
    end for

```
    i ← 0
    j ← 0
    while i ≠ len(A)&&j ≠ n do
        if C[j] ≠ 0 then
            A[i] ← j
            C[j] ← C[j] − 1
            i ← i + 1
        else
            j ← j + 1
        end if
    end while
end procedure
```

2.2.2 Анализ алгоритма

- Время выполнения алгоритма: $O(n + k)$
- Память: $O(n + k)$
  - n - размер массива
  - k - максимальное значение элемента массива
- Сортировка подсчетом работает только с целыми числами
- Сортировка подсчетом не является устойчивой
- Сортировка подсчетом не подходит для сортировки больших массивов

2.3 Итеративная быстрая сортировка

2.3.1 Описание алгоритма

```
procedure QuickSort(A)
    n ← len(A)
    S ← {}                                          ▷ Стек
    S ← S ∪ {0, n − 1}
```

while $S \neq \{\}$ **do**
$r \leftarrow S[n_S - 1]$
$l \leftarrow S[n_S - 2]$
$S \leftarrow S \setminus \{r, l\}$
$p \leftarrow \texttt{partition}(A, l, r)$
$S \leftarrow S \cup \{l, p - 1\}$
$S \leftarrow S \cup \{p + 1, r\}$
end while
end procedure

2.3.2 Анализ алгоритма

- Время выполнения алгоритма: $O(n \log n)$
- Память: $O(\log n)$

3 Конструкторская часть

3.1 Разработка алгоритмов

На рисунках 3.1, 3.2 и 3.3 приведены схемы алгоритмов сортировки пузырьком, сортировки подсчетом и быстрой сортировки соответственно.

3.2 Трудоёмкость алгоритмов

3.3 Модель вычислений

Для последующего вычисления трудоемкости введём модель вычислений:

1. Операции из списка (3.1) имеют трудоемкость 1.

$$+, -, /, \%, ==, !=, <, >, <=, >=, [], ++, -- \tag{3.1}$$

2. Трудоемкость оператора выбора if условие then A else B рассчитывается, как (3.2).

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{3.2}$$

3. Трудоемкость цикла рассчитывается, как (3.3).

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \tag{3.3}$$

4. Трудоемкость вызова функции равна 0.

3.4 Трудоёмкость алгоритмов

Пусть размер массивов во всех вычислениях обозначается как N.

3.4.1 Алгоритм сортировки пузырьком

Трудоёмкость алгоритма сортировки пузырьком состоит из:

- трудоёмкость сравнения и инкремента внешнего цикла $i \in [1..N)$ (3.4):

$$f_i = 2 + 2(N-1) \tag{3.4}$$

- суммарная трудоёмкость внутренних циклов, количество итераций которых меняется в промежутке $[1..N-1]$ (3.5):

$$f_j = 3(N-1) + \frac{N \cdot (N-1)}{2} \cdot (3 + f_{if}) \tag{3.5}$$

- трудоёмкость условия во внутреннем цикле (3.6):

$$f_{if} = 4 + \begin{cases} 0, & \text{в лучшем случае} \\ 9, & \text{в худшем случае} \end{cases} \tag{3.6}$$

Трудоёмкость в **лучшем** случае (3.7):

$$f_{best} = \frac{7}{2}N^2 + \frac{3}{2}N - 3 \approx \frac{7}{2}N^2 = O(N^2) \tag{3.7}$$

Трудоёмкость в **худшем** случае (3.8):

$$f_{worst} = 8N^2 - 8N - 3 \approx 8N^2 = O(N^2) \tag{3.8}$$

3.4.2 Алгоритм сортировки подсчетом

Трудоёмкость алгоритма сортировки подсчетом состоит из:

1. нахождение максимального числа в массиве A.
2. трудоёмкость инициализации массива C размером k нулями, где $k = max(A)$.
3. проход по массиву A и подсчет количества вхождений каждого числа в массиве C.

4. проход по массиву C и запись чисел в массив A в соответствии с их количеством в массиве C.

Вычислим каждую трудоёмкость отдельно.

1. Трудоёмкость нахождения максимального числа в массиве A(3.9).

$$f_{max} = 2 + 2(N-1) = 4N - 3 \tag{3.9}$$

2. Трудоёмкость инициализации массива C(3.10).

$$f_C = 2 + 2(k-1) = 4k - 3 \tag{3.10}$$

3. Трудоёмкость подсчета количества вхождений каждого числа в массиве C(3.11).

$$f_{count} = 2 + 2(N-1) = 4N - 3 \tag{3.11}$$

4. проход по массиву C и запись чисел в массив A в соответствии с их количеством в массиве C(3.12).

$$f_{write} = 6 + 7 * N + \begin{cases} 0, & \text{в лучшем случае} \\ 3k, & \text{в худшем случае} \end{cases} \tag{3.12}$$

Трудоёмкость в **лучшем** случае (3.13):

$$f_{best} = 15N + 4k - 3 \approx 15N + 4k = O(N + k) \tag{3.13}$$

Трудоёмкость в **худшем** случае (3.14):

$$f_{worst} = 15N + 7k - 3 \approx 15N + 7k = O(N + k) \tag{3.14}$$

3.4.3 Алгоритм итеративной быстрой сортировки

Трудоёмкость алгоритма итеративной быстрой сортировки (3.15):

$$T(N) = T(J) + T(N - J) + M(N) \tag{3.15}$$

где

1. $T(N)$ - трудоемкость быстрой сортировки массива размера N.
2. $T(J)$ - трудоемкость быстрой сортировки массива размера J.
3. $T(N - J)$ - трудоемкость быстрой сортировки массива размера N-J.
4. $M(N)$ - трудоёмкость разделения массива на две части.

Вычислим для лучшего случая:

- $T(N) = 2T(\frac{N}{2}) + C * N$ - трудоемкость быстрой сортировки массива размера N.
  - $2T(\frac{N}{2})$ поскольку мы разделяем массив на 2 равные части
  - $C * N$ поскольку мы будем проходить все элементы массива на каждом уровне "дерева"
- Следующий шаг - разделить дальше на 4 части ((3.16) и (3.17)):

$$T(N) = 2(2 * T(\frac{N}{4}) + C * N/2) + C * N \tag{3.16}$$

$$T(N) = 4T(\frac{N}{4}) + 2C * N \tag{3.17}$$

- В общем случае (3.18):

$$T(N) = 2^k * T(N/(2^k)) + k * C * N \tag{3.18}$$

- Для лучшего случая - $N = 2^k$ - идеально распределенное дерево (отсюда следует, что $k = log_2(N)$) (3.19)

$$T(N) = 2^k * T(1) + k * C * (2^k) \tag{3.19}$$

– Вычислим T(1) - трудоемкость при массиве длинной 1 - и C - трудоемкость разделения (3.20):

$$T(1) = 6 + 1 * 9 = 15; C = 12 + N/(2^k) * 8 \tag{3.20}$$

- Полная сложность (при $k = log_2(N)$) (3.21):

$$T(N) = 15N + log_2(N) * (12 + 8) = 15N + 20Nlog_2(N) \tag{3.21}$$

Теперь вычислим для худшего случая:

- $T(N) = T(N-1) + C * N$ - трудоемкость быстрой сортировки массива размера N.
  - $T(N-1)$ поскольку мы разделяем массив на 2 неравные части: пустое множество и полное множество за исключением "середины"

- Следующие шаги очевидны (3.22), (3.23):

$$T(N) = T(N-2) + C(N-1) + C * N = T(N-2) + 2C * N - C \tag{3.22}$$

$$T(N) = T(N-3) + 3C * N - 2C * N - C \tag{3.23}$$

- В общем случае (3.24):

$$T(N) = T(N-k) + k * C * N - C(\frac{k(k-1)}{2}) \tag{3.24}$$

- Для худшего случая - $N = k$ - нераспределенное дерево (3.25):

$$T(N) = T(0) + N * C * N - C(\frac{N(N-1)}{2}) \tag{3.25}$$

– Вычислим параметры (3.26):

$$T(0) = 1; C = 12 + (N - k) * 8 \tag{3.26}$$

- Полная сложность (при $N = k$) (3.27):

$$T(N) = 1 + N * N * 12 - 12 * (\frac{N(N-1)}{2}) = 1 + 6N^2 + 6N \tag{3.27}$$

Трудоёмкость в **лучшем** случае (3.28):

$$f_{best} = 15N + 20Nlog_2(N) \approx 20Nlog_2(N) = O(Nlog_2(N)) \tag{3.28}$$

Трудоёмкость в **худшем** случае (3.29):

$$f_{worst} = 1 + 6N^2 + 6N \approx 6N^2 = O(N^2) \tag{3.29}$$

3.5 Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы трёх алгоритмов сортировки. Оценены их трудоёмкости в лучшем и худшем случаях.

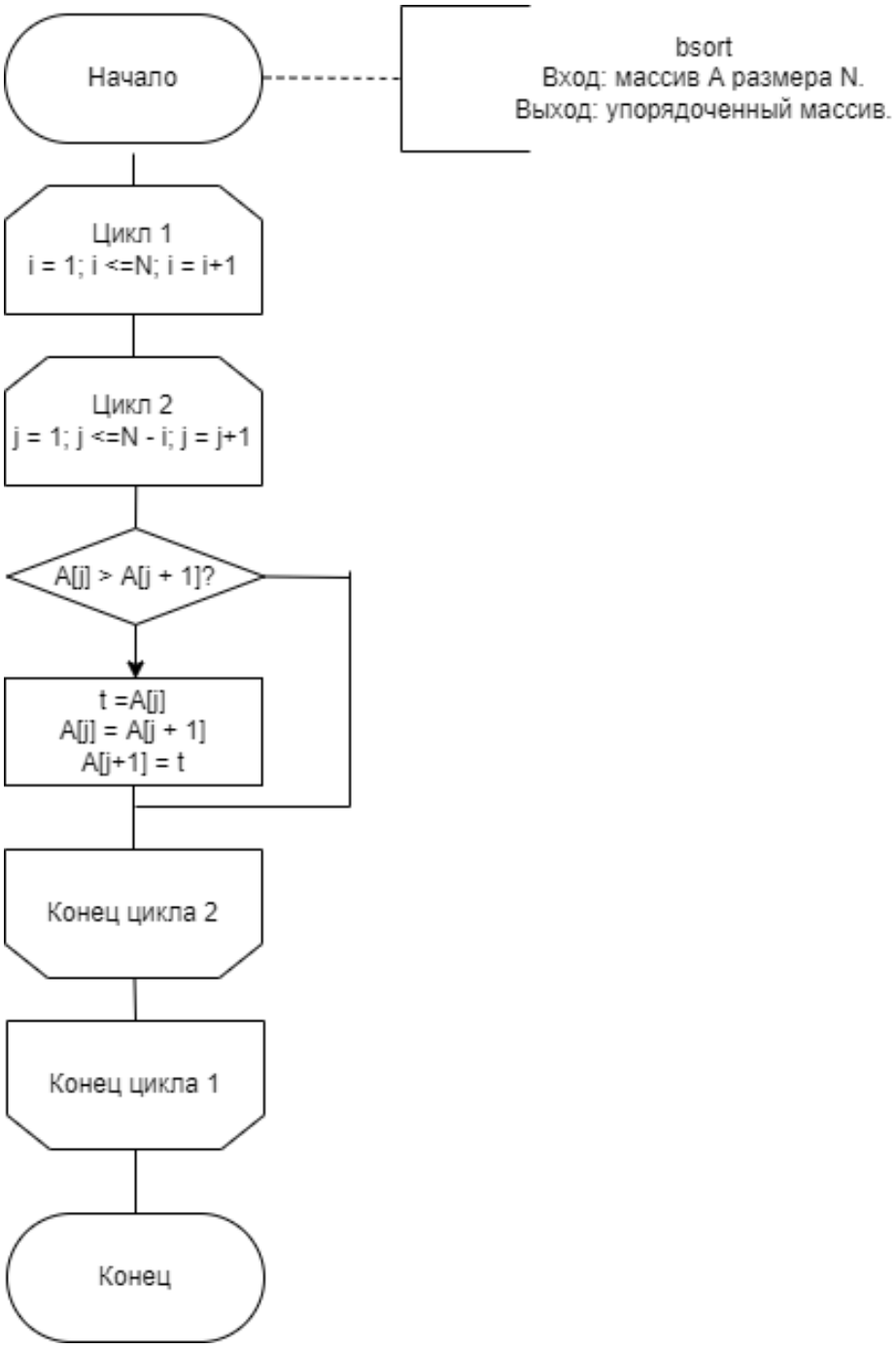


Рисунок 3.1 – Сортирвка пузырьком

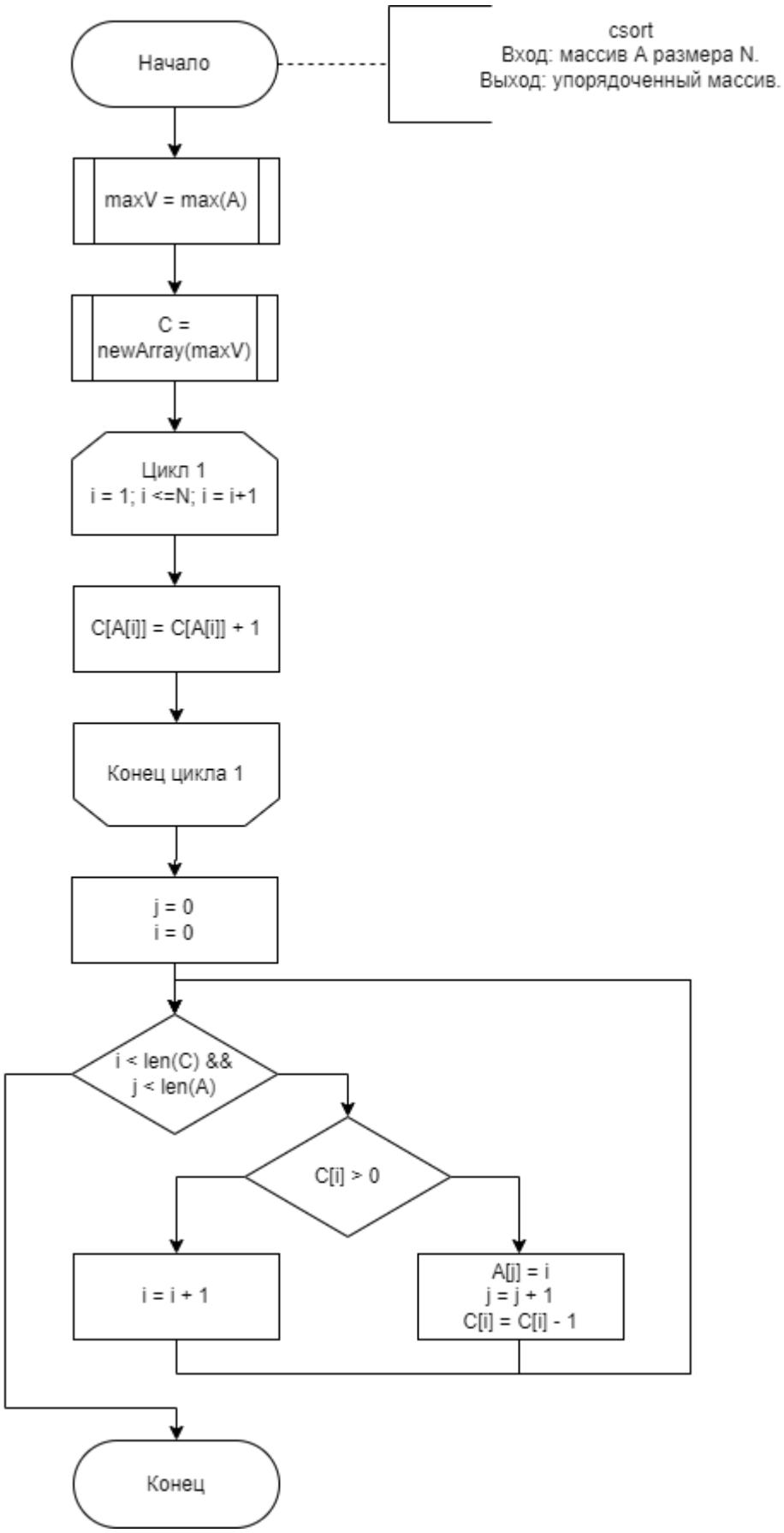


Рисунок 3.2 – Сортировка вставками

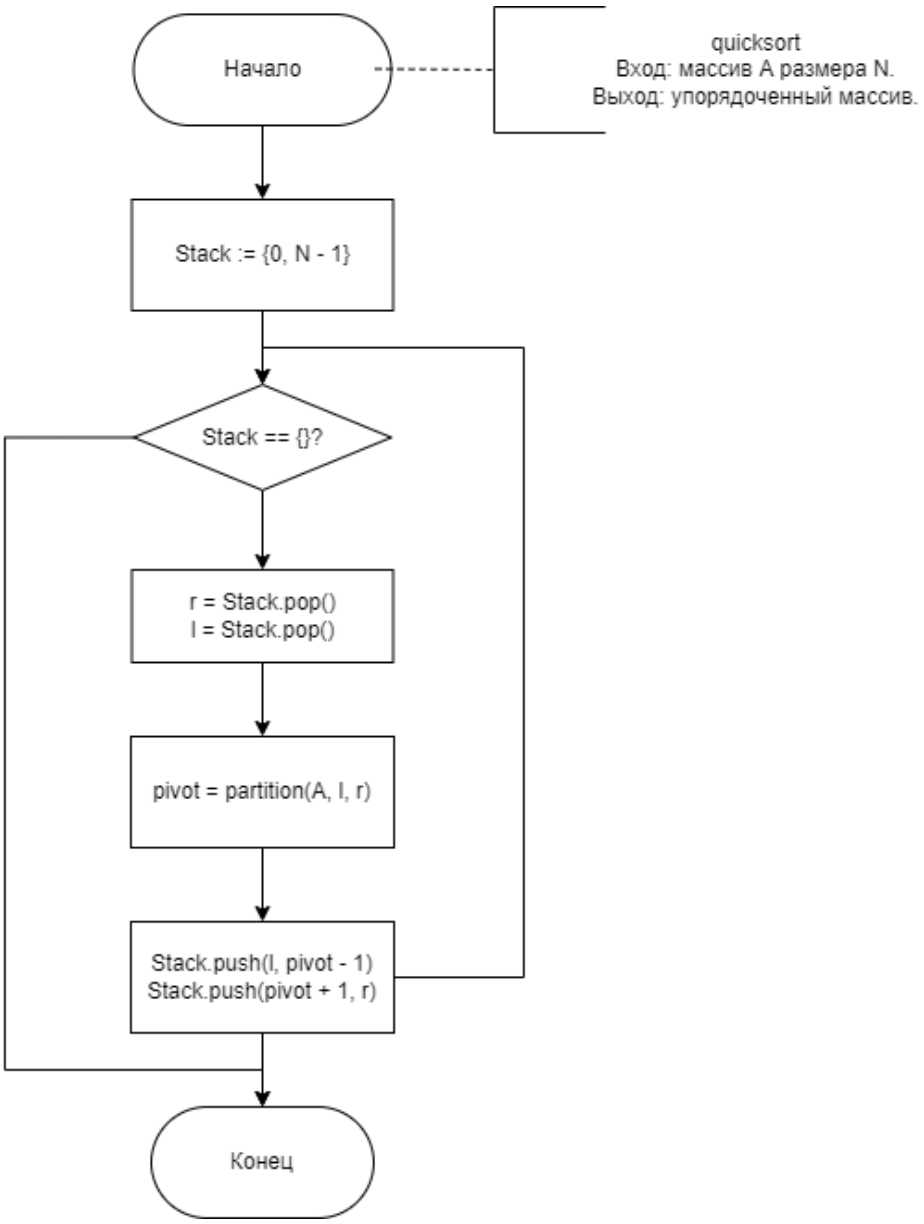


Рисунок 3.3 – Быстрая сортировка

4 Технологическая часть

В данном разделе приведены требования к программному обеспечению, средства реализации и листинги кода.

4.1 Требования к программному обеспечению

К программе предъявляется ряд условий:

– на вход программе подается размер массива (целое число) и массив (целые числа);

– на выходе программа должна выдать отсортированный массив одним из 3 способов;

4.2 Средства реализации

Для реализации данной лабораторной работы необходимо установить следующее программное обеспечение:

- Golang 1.19.1 - язык программирования
- Benchdraw 0.1.0 - Средство визуализации данных
- LaTeX - система документооборота

4.3 Листинги кода

В листингах 4.1, 4.2 и 4.3 приведены реализации алгоритмов сортировки пузырьком, подсчетом и итеративная быстрая соответственно.

Листинг 4.1 – Сортировка массива пузырьком

```
func BubbleSort[T constraints.Ordered](array []T) {
    n := len(array) - 1
    for i := 0; i < n; i++ {
        for j := 0; j < n-i; j++ {
            if array[j] > array[j+1] {
                array[j], array[j+1] = array[j+1], array[j]
            }
        }
    }
}
```

Листинг 4.2 – Сортировка массива подсчетом

```
func CountingSort(array []int) {
    maxVal := array[0]
    for _, val := range array {
        if val > maxVal {
            maxVal = val
        }
    }
    temp := make([]int, maxVal+1)
    for _, val := range array {
        temp[val]++
    }

    j, i := 0, 0

    for i < len(temp) && j < len(array) {
        if temp[i] > 0 {
            array[j] = i
            j++
            temp[i]--
        } else {
            i++
        }
    }
}
```

Листинг 4.3 – Итеративная быстрая сортировка массива

```

func partition[T constraints.Ordered](arr []T, left int, right int) int {
    pivot := arr[right]
    i := left - 1
    for j := left; j <= right-1; j++ {
        if arr[j] <= pivot {
            i++
            arr[i], arr[j] = arr[j], arr[i]
        }
    }
    arr[i+1], arr[right] = arr[right], arr[i+1]

    return i + 1
}

func Quicksort[T constraints.Ordered](array []T) {
    if len(array) < 2 {
        return
```

```
    }

    left := 0
    right := len(array) - 1
    stack := Stack[[2]int]{}
    stack.Push([2]int{left, right})
    for !stack.IsEmpty() {
        lr := stack.Pop()
        left, right = lr[0], lr[1]
        if right <= left {
            continue
        }
        pivot := partition(array, left, right)
        stack.Push([2]int{left, pivot - 1})
        stack.Push([2]int{pivot + 1, right})
    }
}
```

В таблице 4.1 приведены тесты для функций, реализующих алгоритмы сортировки. Все тесты пройдены успешно.

Входной массив	Результат	Ожидаемый результат
$[1, 2, 3, 4, 5]$	$[1, 2, 3, 4, 5]$	$[1, 2, 3, 4, 5]$
$[5, 4, 3, 2, 1]$	$[1, 2, 3, 4, 5]$	$[1, 2, 3, 4, 5]$
$[-1, -2, -3, -2, -5]$	$[-5, -3, -2, -2, -1]$	$[-5, -3, -2, -2, -1]$
$[-2, 7, 0, -1, 3]$	$[-2, -1, 0, 3, 7]$	$[-2, -1, 0, 3, 7]$
$[50]$	$[50]$	$[50]$
$[-40]$	$[-40]$	$[-40]$
Пустой массив	Пустой массив	Пустой массив

Таблица 4.1 – Тестирование функций

4.4 Вывод

В данном разделе были разработаны исходные коды трёх алгоритмов сортировки: пузырьком, вставками и быстрая сортировка.

5 Исследовательская часть

Ниже приведены технические характеристики устройства, на котором было проведено тестирование ПО:

- Операционная система: Windows 11 используя Windows Subsystem for Linux 2 (WSL2) [1] Имитирующей Arch Linux [2] 64-bit.
- Оперативная память: 16 GB.
- Процессор: 11th Gen Intel(R) Core(TM) i5-11320H @ 3.20GHz [3].

5.1 Время выполнения алгоритмов

Алгоритмы тестировались при помощи "бенчмарков" предоставляемых встренными средствами языка Go [4]. Пример такого "бенчмарка" приведен в листинге 5.1. Количество повторов регулируется тестирующей системой самостоятельно, хотя при желание оные можно задать. Точное количество повторов определяется в зависимости от того, был ли получен стабильный результат согласно системе.

Листинг 5.1 – Пример бенчмарка

```
func BenchmarkQuicksortSorted(b *testing.B) {
    for steps, amount := 0, 0; steps < STEPS; steps++ {
        amount += INC
        b.Run(fmt.Sprintf("size=%d", amount), func(b *testing.B) {
            sample := GenerateSortedArray(amount)
            sampleCopy := make([]int, amount)
            b.ResetTimer()
            for i := 0; i < b.N; i++ {
                //b.StopTimer()  // Disabled Due performance reasons
                copy(sampleCopy, sample)
                //b.StartTimer() // Disabled Due performance reasons
                Quicksort(sampleCopy)
            }
        })
    }
}
```

Таблица времени выполнения сортировок на отсортированных данных (в наносекундах)

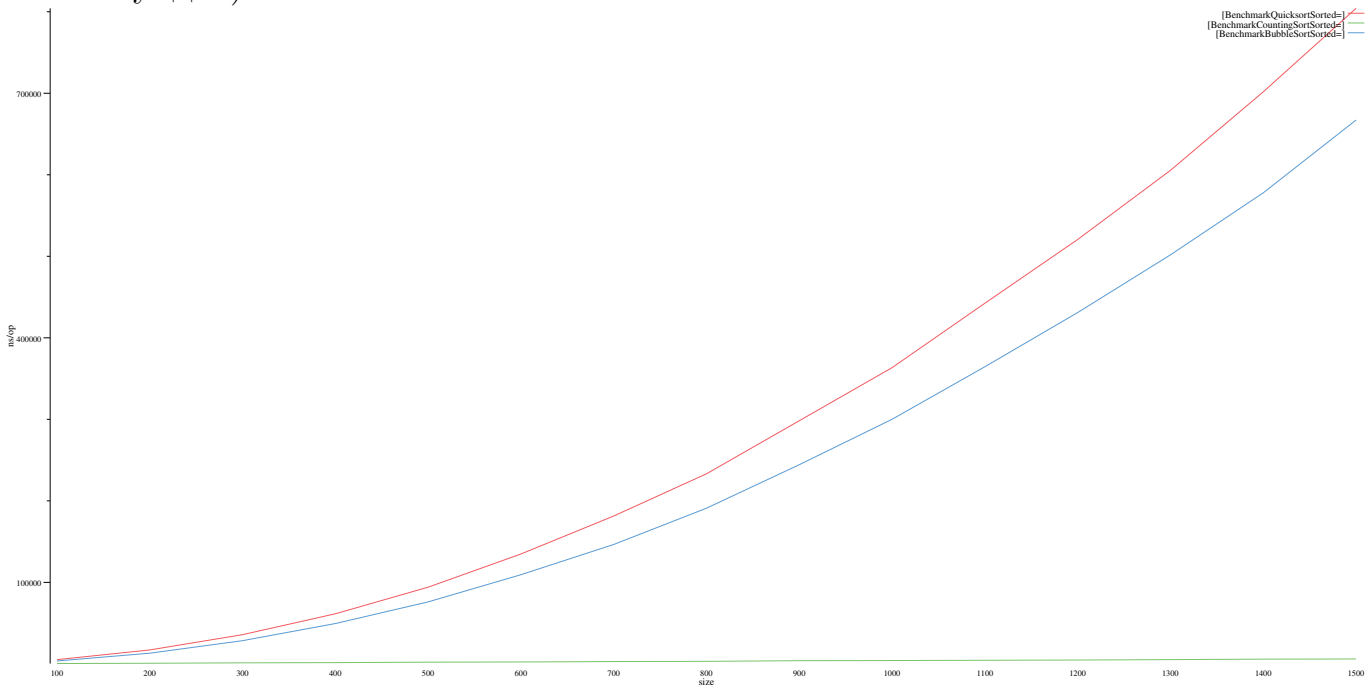


Таблица времени выполнения сортировок на отсортированных данных в обратном порядке (в наносекундах)

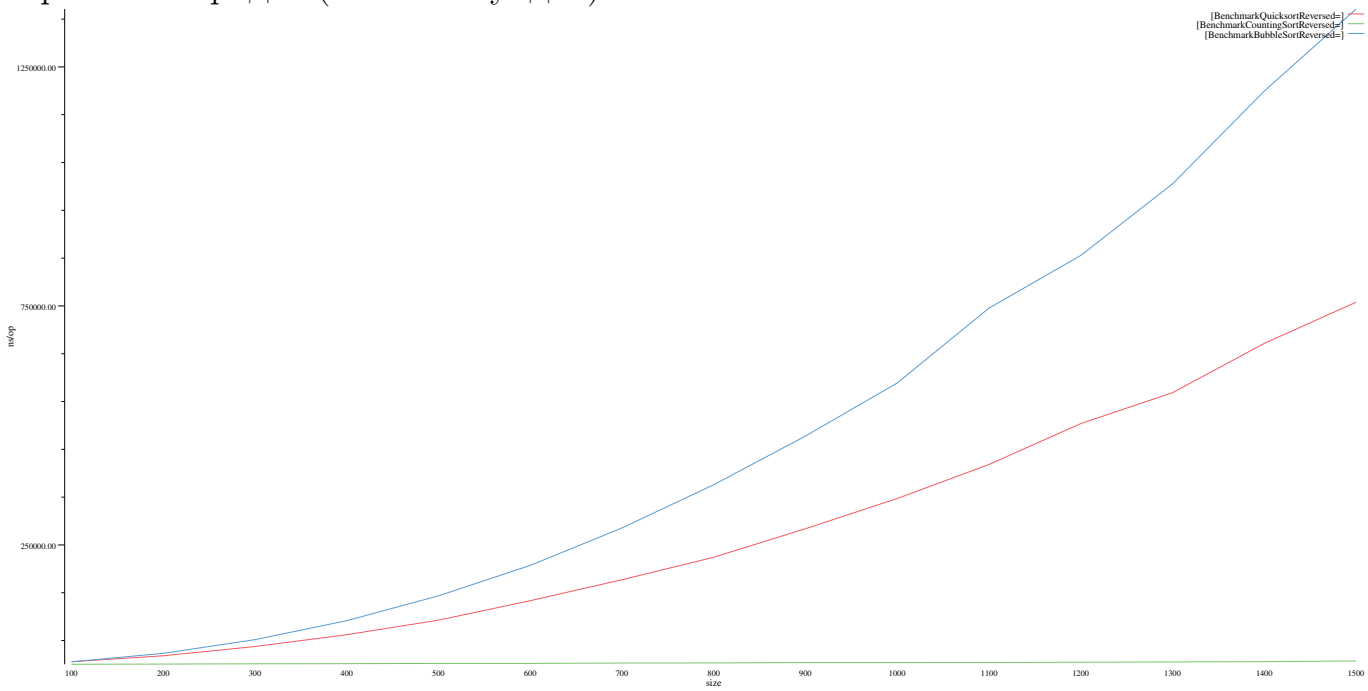
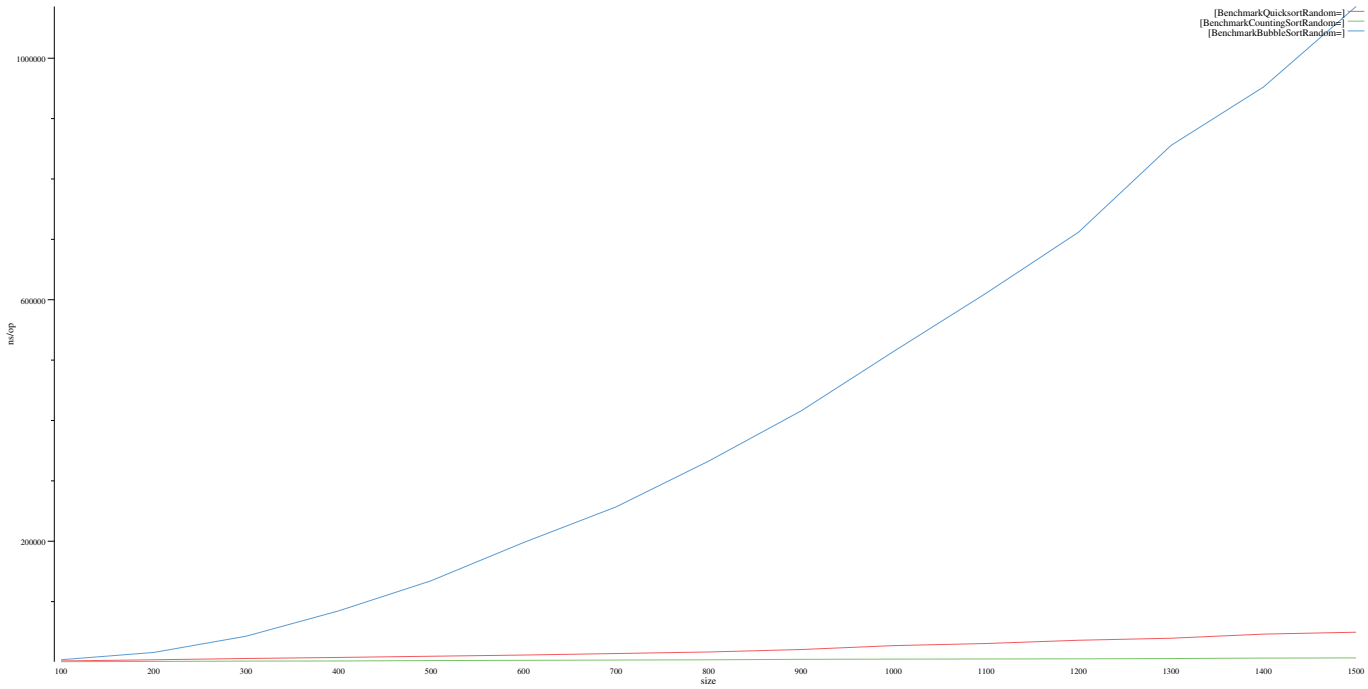


Таблица времени выполнения сортировок на случайных данных (в наносекундах)



5.2 Вывод

Как было и ожидаемо, сортировка подсчетом оказалась самой быстрой, а сортировка пузырьком самой медленной. Быстрая сортировка оказалась быстрой, но при отсортированных данных она оказалась медленнее сортировки пузырьком.

6 ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были достигнуты следующие задачи:

- были изучены и реализованы 3 алгоритма сортировки: пузырёк, подсчетом, быстрая сортировка;
- были приведены аналитические данные о выше перечисленных алгоритмах;
- были приведены подробные блок-схемы, описывающие алгоритмы;
- был проведён сравнительный анализ алгоритмов на основе экспериментальных данных.

Экспериментальные данные показали различные сильные и слабые стороны каждого алгоритма. Так например:

– сортировка пузырьком работает крайне медленно независимо от входных данных;

– быстрая сортировка работает быстрее на случайных данных, поэтому лучше всего подходит как общее решение абстрактной задачи на сортировку данных;

– быстрая сортировка работает медленно на отсортированном массиве.

– сортировка подсчетом работает быстрее всех выше перечисленных алгоритмов, однако требует сильно больше памяти, чем остальные алгоритмы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Windows Subsystem for Linux 2 [Электронный ресурс]. — Дата обращения: 19.09.2022. Режим доступа: `https://learn.microsoft.com/en-us/windows/wsl/about#what-is-wsl-2`.
2. Arch Linux [Электронный ресурс]. — Дата обращения: 19.09.2022. Режим доступа: `https://archlinux.org/`.
3. Процессор Intel® Core™ i5-11320H [Электронный ресурс]. — Дата обращения: 19.09.2022. Режим доступа: `https://ark.intel.com/content/www/ru/ru/ark/products/217183/intel-core-i511320h-processor-8m-cache-up-to-4-50-ghz-with-ipu.html`.
4. Go [Электронный ресурс]. — Дата обращения: 19.09.2022. Режим доступа: `https://go.dev/`.