Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет безопасности информационных технологий

Дисциплина:
«Алгоритмы и структуры данных»

Лабораторная работа №1

Выполнил:

Беляков Никита Андреевич N3245

_____
(подпись)

Проверил:

Еврофеев С.А.

_____
(отметка о выполнении)

_____
(подпись)

Санкт-Петербург
2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данной лабораторной работе мы рассмотрим алгоритм сортировки, известный как "бетонная сортировка"(Betonic Sort). Этот алгоритм относится к семейству сортировок, основанных на сети сравнителей, и обладает временной сложностью $O(nlog^2n)$.

Целью данной работы является реализация алгоритма бетонной сортировки на языке программирования $Go(Golang)$ и его применение для сортировки массива строк определенной длины, где длина массива является степенью двойки.

В ходе выполнения работы мы разработаем алгоритм бетонной сортировки, реализуем его в виде функции на языке Go, а также создадим простое консольное приложение для взаимодействия с пользователем. Для удобства ввода данных мы предоставим выбор между вводом массива с клавиатуры и чтением массива из файла.

1 ХОД РАБОТЫ

1.1 Блок схема

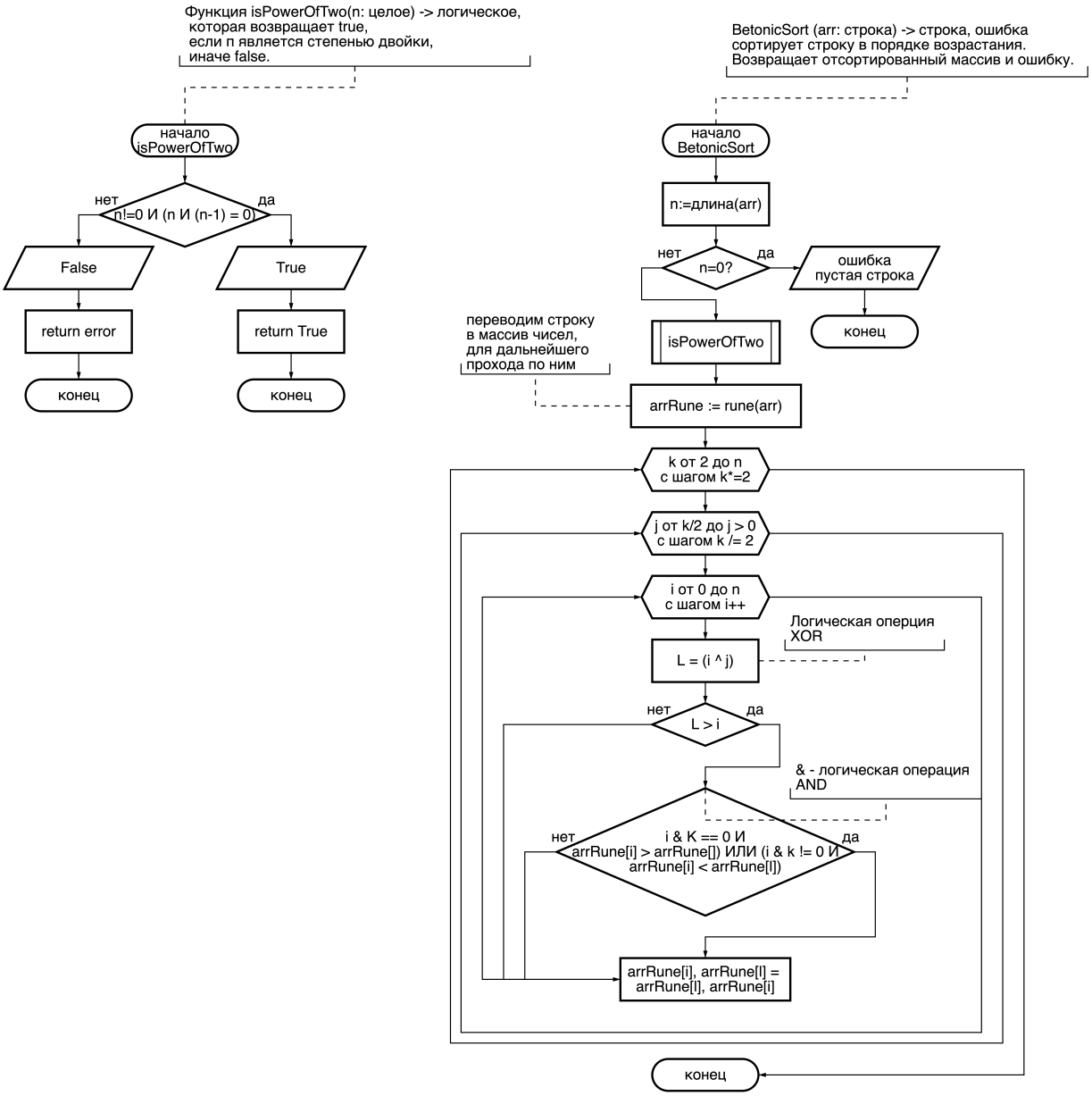

Рисунок 1.1 — Блок схема Betonic Sort.

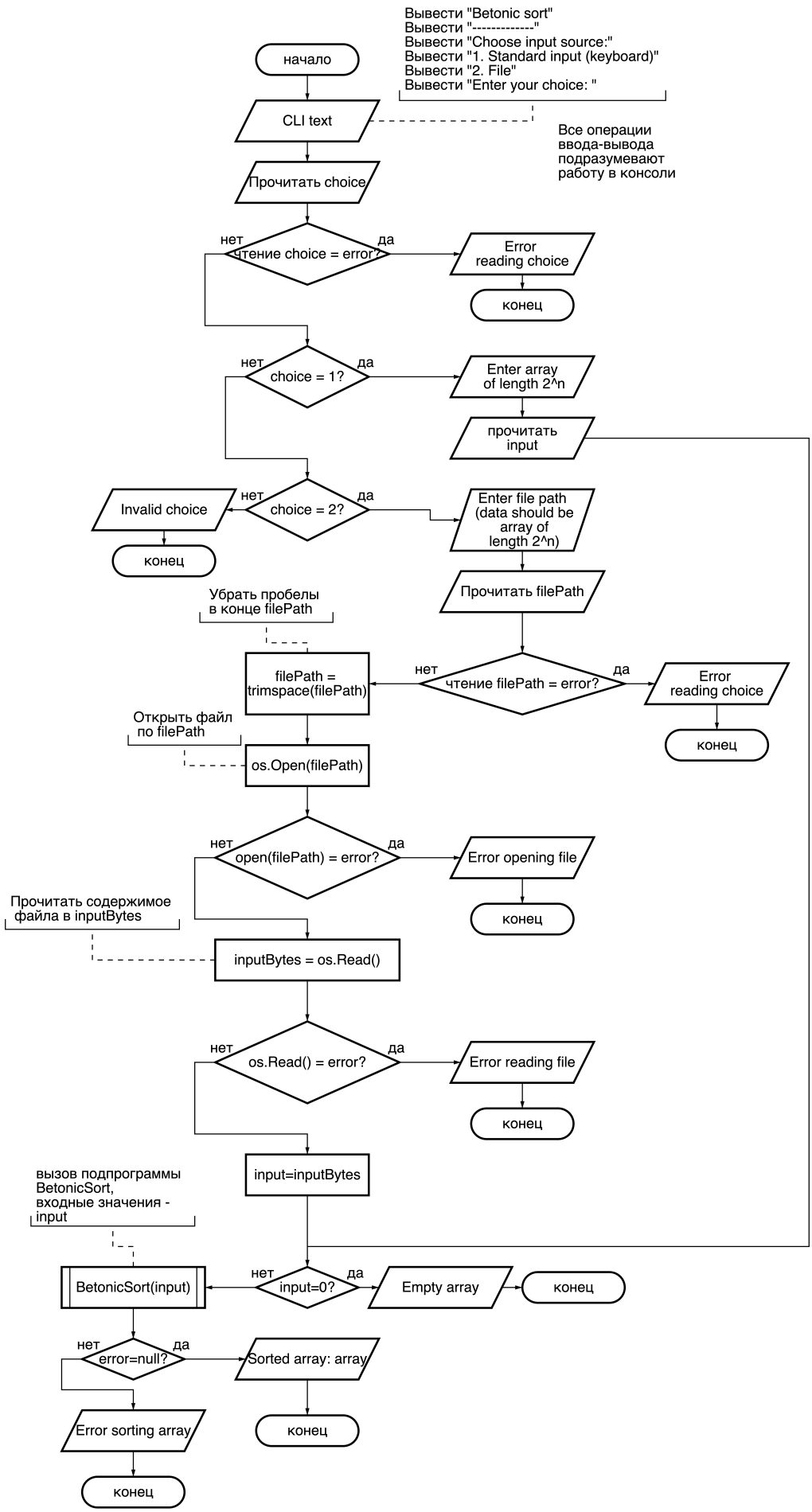

Рисунок 1.2 — Блок схема main.

1.2 Псевдокод и маршрут

```
Функция isPowerOfTwo(n: Целое) -> Логическое:
    Возвратить (n != 0) И (n И (n-1) = 0)

Функция BetonicSort(arr: Строка) -> (Строка, Ошибка):
    n := ДлинаСтроки(arr) // длина массива

    // проверка на пустоту массива
    Если n = 0 Тогда
        Возвратить "", Ошибка("пустой массив")
    Конец Если

    // проверка, является ли n степенью двойки
    Если НЕ isPowerOfTwo(n) Тогда
        Возвратить "", Ошибка("длина массива должна быть степенью двойки")
    Конец Если

    // преобразование строки в массив рун
    arrRune := ПреобразоватьСтрокуВМассивРун(arr)

    // сортировка по алгоритму BetonicSort
    Для k от 2 до n с шагом k *= 2 Цикл:
        Для j от k / 2 до 1 с шагом j /= 2 Цикл:
            Для i от 0 до ДлинаМассиваРун Цикл:
                l := i ИЛИ j
                Если l > i И l < ДлинаМассиваРун Тогда
                    Если ((i И k = 0) И (arrRune[i] > arrRune[l]))
                    ИЛИ ((i И k != 0) И (arrRune[i] < arrRune[l]))
                    Тогда
                        ПоменятьМестами(arrRune[i], arrRune[l])
                    Конец Если
                Конец Если
            Конец Цикла
        Конец Цикла
    Конец Цикла
    // преобразование массива рун обратно в строку
    Возвратить ПреобразоватьМассивРунВСтроку(arrRune)
```

Маршрут выполнения программы на основе данного кода:

- Программа выводит на экран меню выбора источника ввода.
- Пользователь делает выбор между стандартным вводом (клавиатурой) и файлом.
- В зависимости от выбора, программа либо считывает массив с клавиатуры, либо запрашивает путь к файлу и считывает массив из файла.
- Проверяется, не является ли массив пустым. Если да, программа завершается с сообщением об ошибке.
- Проверяется, является ли длина массива степенью двойки. Если нет, программа завершается с сообщением об ошибке.
- Происходит сортировка массива с использованием алгоритма BetonicSort.
- Отсортированный массив выводится на экран.

1.3 Спецификация переменных

Ниже представлена спецификация переменных в формате переменная:тип:мин:макс.

1. **arr**:$string$:0:2^{31}
2. **n**:int:-2147483648:2147483647.
3. **k**:int:-2147483648:2147483647.
4. **j**:int:-2147483648:2147483647.
5. **i**:int:-2147483648:2147483647.
6. **l**:int:-2147483648:2147483647.
7. **choice**:int:-2147483648:2147483647.
8. **input**:$string$:0:2^{31}.
9. **filePath**:$string$:0:2^{31}.
10. **inputBytes**:$[]byte$:0:2^{31}.
11. **sortedArr**:$string$:0:2^{31}.

1.4 Листинг программы

Ниже представлен листинг файла BetonicSort.go

```
package betonicSort

import (
        "errors"
)

// isPowerOfTwo returns true if n is a power of 2, false otherwise.
func isPowerOfTwo(n int) bool {
        return (n != 0) && (n&(n-1) == 0)
}

// BetonicSort sorts the string in ascending order. Returns sorted array and error.
func BetonicSort(arr string) (string, error) {
        n := len(arr) // length of array

        // check if array is empty
        if n == 0 {
                return "", errors.New("empty array")
        }

        // check if n is a power of 2
        if !isPowerOfTwo(n) {
                return "", errors.New("array length must be a power of 2")
        }

        // convert string to array of runes
        arrRune := []rune(arr)

        // betonic sort
        for k := 2; k <= n; k *= 2 {
                for j := k / 2; j > 0; j /= 2 {
                        for i := 0; i < len(arrRune); i++ {
                                l := i ^ j
                                if l > i && l < len(arrRune) {
                                        if (i&k == 0) && (arrRune[i] > arrRune[l]) || (i&k != 0)
                    && (arrRune[i] < arrRune[l]) {
                                                arrRune[i], arrRune[l] = arrRune[l], arrRune[i]
                                        }
                                }
                        }
```

```
                }
        }

        // convert array of runes to string
        return string(arrRune), nil
}
```

Ниже представлен листинг файла main.go

```
package main

import (
        "algosITMO/labs/lab2/betonicSort"
        "bufio"
        "fmt"
        "io"
        "log"
        "os"
        "strings"
)

func main() {
        // CLI UI
        fmt.Println("Betonic sort")
        fmt.Println("-------------")
        fmt.Println("Choose input source:")
        fmt.Println("1. Standard input (keyboard)")
        fmt.Println("2. File")

        reader := bufio.NewReader(os.Stdin)

        var choice int
        fmt.Print("Enter your choice: ")
        _, err := fmt.Scan(&choice)
        if err != nil {
                log.Fatal("Error reading choice: ", err)
        }

        var input string

        switch choice {
        case 1: // read from keyboard
                fmt.Println("Enter array of length 2^n:")
                input, err = reader.ReadString('\n')
                if err != nil {
                        log.Fatal("Error reading from keyboard: ", err)
                }
                input = strings.TrimSpace(input)
        case 2: // read from file
                fmt.Print("Enter file path (data should be array of length 2^n): ")
                filePath, err := reader.ReadString('\n')

                if err != nil {
                        log.Fatal("Error reading file path: ", err)
                }
                filePath = strings.TrimSpace(filePath) // remove trailing newline character

                file, err := os.Open(filePath)
                if err != nil {
                        log.Fatal("Error opening file: ", err)
                }
                // close file on exit
                defer func() {
                        if err := file.Close(); err != nil {
```

```
                        log.Fatal("Error closing file: ", err)
                }
        }()
        inputBytes, err := io.ReadAll(file)
        if err != nil {
                log.Fatal("Error reading file: ", err)
        }
        input = string(inputBytes)

    default: // invalid choice
        log.Fatal("Invalid choice")
    }

    if len(input) == 0 {
        fmt.Println("Empty array. Exit.")
        return
    }

    // sort array
    sortedArr, err := betonicSort.BetonicSort(input)
    if err != nil {
        log.Fatal("Error sorting array: ", err)
    }

    // print sorted array
    fmt.Println("Sorted array:", sortedArr)
}
```

1.5 Разбор цикла по шагам

case1	case2	case3	case4	case5
47384920	3647383917253679	9443	35748234	fdsjkfsdklf2213f
Step 0: 47834920	Step 0: 3674383917253679	Step 0: 4943	Step 0: 35742834	Step 0: dfsjkfsdklf2213f
Step 1: 43874920	Step 1: 3674389317253679	Step 1: 4349	Step 1: 35742843	Step 1: dfsjfksdklf2213f
Step 2: 34874920	Step 2: 3674389317523679	Step 2: 3449	Step 2: 34752843	Step 2: dfsjfksdklf2123f
Step 3: 34784920	Step 3: 3674389317523697	Sorted array: 3449	Step 3: 34754823	Step 3: dfsjfksdklf212f3
Step 4: 34789420	Step 4: 3476389317523697		Step 4: 34574823	Step 4: dfsjskfdklf212f3
Step 5: 34289470	Step 5: 3476983317523697		Step 5: 34578423	Step 5: dfsjskfdflk212f3
Step 6: 34209478	Step 6: 3476983312573697		Step 6: 34578432	Step 6: dfsjskfdf2kl12f3
Step 7: 24309478	Step 7: 3476983312579637		Step 7: 34378452	Step 7: dfsjskfdf2klf213
Step 8: 20349478	Step 8: 3476983312579736		Step 8: 34328457	Step 8: dfsjskfdf2klf312
Step 9: 20347498	Step 9: 3467983312579736		Step 9: 32348457	Step 9: dfjsskfdf2klf312
Step 10: 02347498	Step 10: 3467983312579763		Step 10: 32345487	Step 10: dfjsskfd2fklf312
Step 11: 02344798	Step 11: 3437986312579763		Step 11: 23345487	Step 11: dfjsskfd2fklf321
Step 12: 02344789	Step 12: 3433986712579763		Step 12: 23344587	Step 12: dffsskjd2fklf321
Sorted array: 02344789	Step 13: 3433986792571763		Step 13: 23344578	Step 13: dffdskjs2fklf321
	Step 14: 3433986797571263		Sorted array: 23344578	Step 14: dffdskjsffkl2321
	Step 15: 3433986797671253			Step 15: ddffskjsffkl2321
	Step 16: 3334986797671253			Step 16: ddffjkssffkl2321
	Step 17: 3334689797671253			Step 17: ddff jksskffl2321
	Step 18: 3334679897671253			Step 18: ddffjkssklff2321
	Step 19: 3334679897675213			Step 19: ddffjksslkff2321
	Step 20: 3334679897675312			Step 20: ddffjkssIkff3221
	Step 21: 3334678997675312			Step 21: ddff3kssIkffj221
	Step 22: 3334678997765312			Step 22: ddff32sslkffjk21
	Step 23: 3334678997765321			Step 23: ddff322slkffjks1
	Step 24: 3334578997766321			Step 24: ddff3221lkffjkss
	Step 25: 3334538997766721			Step 25: 3dffd221lkffjkss
	Step 26: 3334532997766781			Step 26: 32ffdd21lkffjkss
	Step 27: 3334532197766789			Step 27: 322fddf1lkffjkss
	Step 28: 3324533197766789			Step 28: 3221ddfflkffjkss
	Step 29: 3321533497766789			Step 29: 3221ddffjkfflkss
	Step 30: 3321533467769789			Step 30: 2231ddffjkfflkss
	Step 31: 2331533467769789			Step 31: 2132ddffjkfflkss
	Step 32: 2133533467769789			Step 32: 2132ddfffkjflkss
	Step 33: 2133335467769789			Step 33: 2132ddffffjklkss
	Step 34: 2133335466779789			Step 34: 1232ddffffjklkss
	Step 35: 2133335466778799			Step 35: 1223ddffffjklkss
	Step 36: 1233335466778799			Step 36: 1223ddffffjkklss
	Sorted array: 1233334566778799			Sorted array: 1223ddffffjkklss

1.6 Расчет сложности алгоритма

Сложность алгоритма битонической сортировки определяется числом раундов параллельных сравнений, которое вычисляется по формуле:

Пусть $p = \lfloor \log_2 n \rfloor$ и $q = \lceil \log_2 n \rceil$.

$$\text{число раундов} = \frac{q(q+1)}{2},$$

Таким образом, количество сравнений c ограничено следующим неравенством:

$$\frac{2^{p-1} \cdot p \cdot (p+1)}{2} \leq c \leq \left\lfloor \frac{n}{2} \right\rfloor \cdot \frac{q \cdot (q+1)}{2}.$$

Это неравенство устанавливает точное значение количества сравнений c в случае, когда n является степенью двойки.

1.7 Тесты

```
package betonicSort_test

import (
        "algosITMO/labs/lab2/betonicSort"
        "errors"
        "testing"
)

func TestBetonicSort(t *testing.T) {
        tests := []struct {
                input string
                want  string
                err   error
        }{
                {input: "1234", want: "1234", err: nil},
                {input: "4321", want: "1234", err: nil},
                {input: "1243", want: "1234", err: nil},
                {input: "3214", want: "1234", err: nil},
                {input: "12345678", want: "12345678", err: nil},
                {input: "87654321", want: "12345678", err: nil},
                {input: "12435678", want: "12345678", err: nil},
                {input: "87543212", want: "12234578", err: nil},
                {input: "1232332456789", want: "", err: errors.New("array length
            must be a power of 2")},
                {input: "", want: "", err: errors.New("empty array")},
        }
        for _, test := range tests {
                got, err := betonicSort.BetonicSort(test.input)
                if (err == nil && test.err != nil) || (err != nil && test.err ==
            nil) || (err != nil && test.err != nil && err.Error() !=
            test.err.Error()) {
                        t.Errorf("betonicSort(%q) = %q, %v; want %q, %v", test.input,
            got, err, test.want, test.err)
                }
                if got != test.want {
                        t.Errorf("betonicSort(%q) = %q, want %q, error: %v",
            test.input, got, test.want, test.err)
                }
        }
}
```

```
=== RUN   TestBetonicSort
--- PASS: TestBetonicSort (0.00s)
PASS
ok      algosITMO/labs/lab2/betonicSort (cached)
```

Рисунок 1.3 — Запуск тестов.

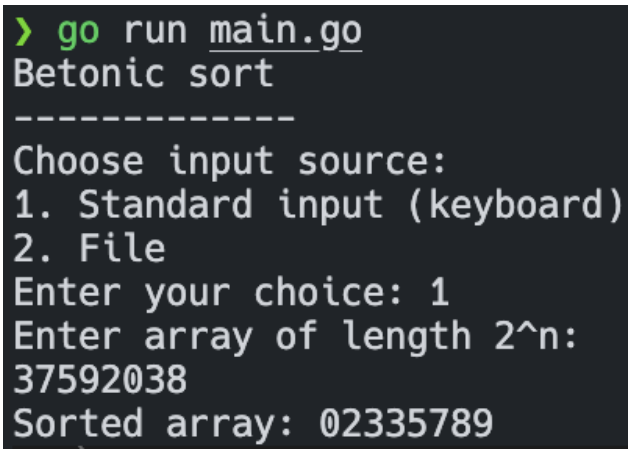

Рисунок 1.4 — Обычная ситуация для ввода с клавиатуры.

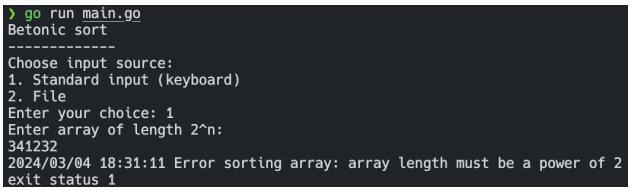

Рисунок 1.5 — Ввод массива не равным 2^n.

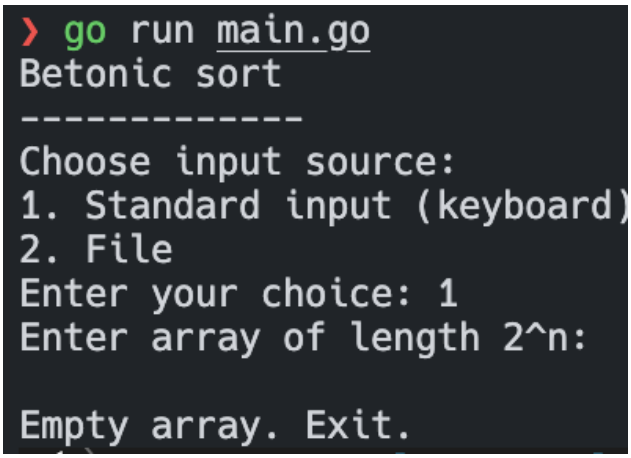

Рисунок 1.6 — Пустой ввод.

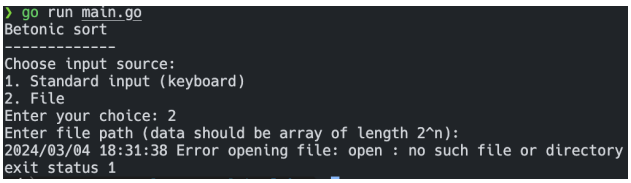

Рисунок 1.7 — Пустой ввод пути файла.

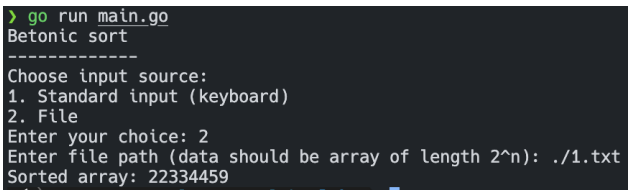

Рисунок 1.8 — Обычная ситуация для ввода с файла.

ЗАКЛЮЧЕНИЕ

Задача: Реализовать программу для бетонической сортировки.
Реализованы функции:

- Ввод массива чисел (по выбору пользователя, как через файл, так через консоль) и его проверка на корректность.
- Преобразование строковых значений в числа int.
- Выполнение бетонической сортировки (в функции $BitonicSort$).

Среда запуска: $Golang/go$ 1.22.0 (выпущена 2024-02-06)
Редактор: $VSCode$
Все тесты были успешно пройдены.