МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
Институт математики и информационных систем
Факультет автоматики и вычислительной техники
Кафедра электронных вычислительных машин

Дата сдачи на проверку:

«__»_____ 2025 г.

Проверено:

«__»_____ 2025 г.

СОРТИРОВКИ ВСТАВКАМИ И ПИРАМИДАЛЬНАЯ СОРТИРОВКА: ФУНКЦИИ ПО ССЫЛКЕ И ОБРАБОТКА ФАЙЛОВ

Отчёт по лабораторной работе №5

по дисциплине

«Программирование»

Разработал студент гр. ИВТб-1301-05-00 _____/Черкасов А. А./

(подпись)

Заведующая кафедры ЭВМ _____/Должженкова М. Л./

(подпись)

Работа защищена «__»_____ 2025 г.

Киров

2025

Цель

Цель работы: Получить базовые знания о наиболее известных алгоритмах сортировки, изучить принципы работы с текстовыми файлами.

Задания

- Реализовать сортировку данных с помощью вставок.
- Реализовать сортировку данных с помощью пирамидальной сортировки.
- В обоих случаях необходимо предусмотреть возможность изменения компаратора (Реализация компаратора в виде передаваемой в программу функции).
- Считывание и вывод данных необходимо производить из текстового файла.
- Для демонстрации работы программных реализаций самостоятельно подготовить варианты входных данных (при этом объем тестовых файлов должен позволять оценить скорость работы программ).

Решение

Схемы алгоритмов решения задач представлена на рисунках 1 и 2. Исходный код решений представлен в Приложениях А1 и А2.

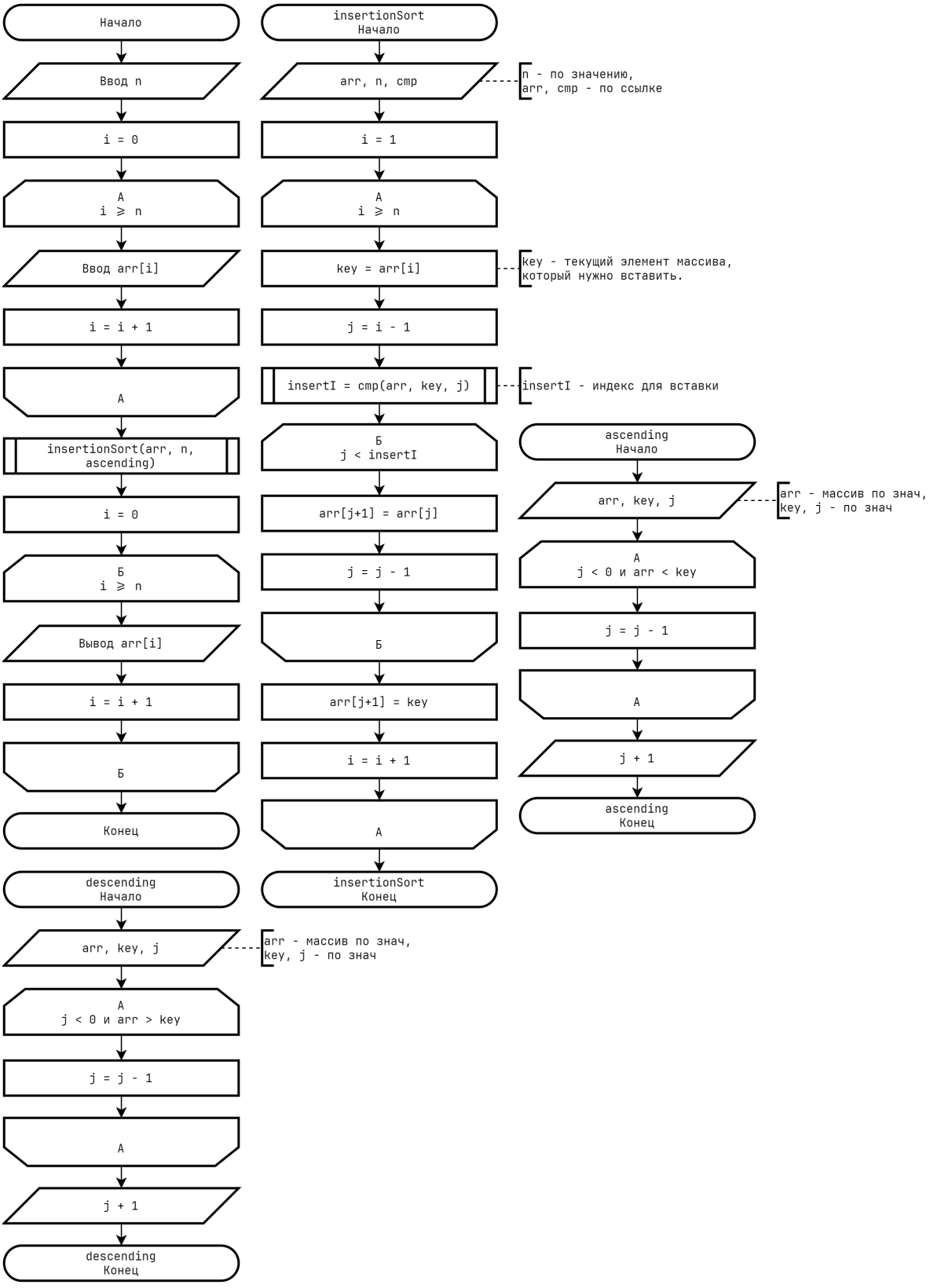

Рисунок 1 - Схема алгоритма Задания 1.

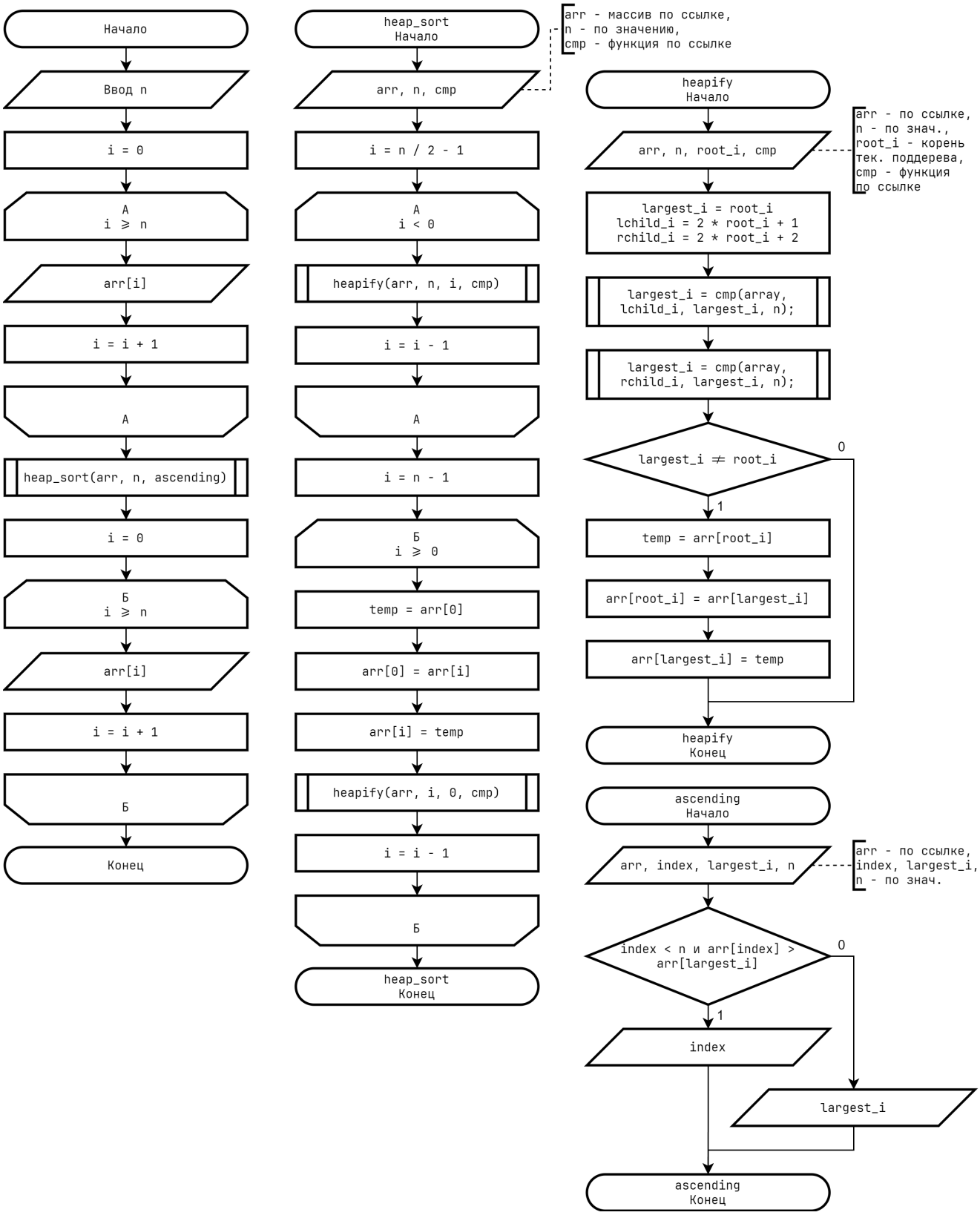

Рисунок 2 - Схема алгоритма Задания 2.

Вывод

В результате работы были реализованы алгоритмы сортировки вставками и пирамидальной сортировки с поддержкой настраиваемого компаратора и обработки текстовых файлов, что позволило оценить их эффективность и сравнить скорость работы.

Приложение А1. Исходный код

```c
#include <stdio.h>
#include <stdlib.h>

const char *in = "./data/gen_input.txt";
const char *out = "./data/output.txt";

int cmp_ascending(int *array, int index, int extreme_index, int size) {
    if (index < size && array[index] > array[extreme_index]) {
        return index;
    }
    return extreme_index;
}

int cmp_descending(int *array, int index, int extreme_index, int size) {
    if (index < size && array[index] < array[extreme_index]) {
        return index;
    }
    return extreme_index;
}

void heapify(int *array, int size, int root_index, int (*cmp)(int *, int, int, int)) {
    int largest_index = root_index;
    int left_child_index = 2 * root_index + 1;
    int right_child_index = 2 * root_index + 2;

    largest_index = cmp(array, left_child_index, largest_index, size);
    largest_index = cmp(array, right_child_index, largest_index, size);

    if (largest_index != root_index) {
        int temp = array[root_index];
        array[root_index] = array[largest_index];
        array[largest_index] = temp;
```

```
        heapify(array, size, largest_index, cmp);
    }
}

void heap_sort(int *array, int n, int (*cmp)(int *, int, int, int)) {
    for (int i = n / 2 - 1; i >= 0; i--) {
        heapify(array, n, i, cmp);
    }

    for (int i = n - 1; i > 0; i--) {
        int temp = array[0];
        array[0] = array[i];
        array[i] = temp;

        heapify(array, i, 0, cmp);
    }
}

int main() {
    FILE *file = fopen(in, "r");
    if (file == NULL) {
        return 1;
    }

    int n;
    fscanf(file, "%d", &n);

    int *arr = malloc(sizeof(int) * n);
    if (arr == NULL) {
        fclose(file);
        return 1;
    }
```

```
    for (int i = 0; i < n; i++) {
        fscanf(file, "%d", &arr[i]);
    }

    fclose(file);

    // heap_sort(arr, n, cmp_ascending);
    heap_sort(arr, n, cmp_descending);

    FILE *output = fopen(out, "w");
    if (output == NULL) {
        free(arr);
        return 1;
    }

    for (int i = 0; i < n; i++) {
        fprintf(output, "%d ", arr[i]);
    }

    fclose(output);
    free(arr);

    return 0;
}
```

Приложение А2. Исходный код

```
#include <stdio.h>
#include <stdlib.h>

int ascending(int *arr, int key, int j) {
  while (j >= 0 && arr[j] > key) {
    j--;
```

```
  }
  return j + 1; // Индекс для вставки
}

int descending(int *arr, int key, int j) {
  while (j >= 0 && arr[j] < key) {
    j--;
  }
  return j + 1;
}

void insertionSort(int *arr, int n, int (*cmp)(int *, int, int)) {
  for (int i = 1; i < n; i++) {
    int key = arr[i];
    int j = i - 1;
    int insertIndex = cmp(arr, key, j);

    while (j >= insertIndex) {
      arr[j + 1] = arr[j];
      j--;
    }
    arr[j + 1] = key;
  }
}

int main() {
  FILE *inputFile = fopen("data/input.txt", "r");
  FILE *outputFile = fopen("data/output.txt", "w");

  if (inputFile == NULL || outputFile == NULL) {
    printf("Error opening file!\n");
    return 1;
  }
```

```
  int n;
  fscanf(inputFile, "%d", &n);

  int *arr = (int *)malloc(n * sizeof(int));
  for (int i = 0; i < n; i++) {
    fscanf(inputFile, "%d", &arr[i]);
  }

  fclose(inputFile);

  // insertionSort(arr, n, ascending);
  insertionSort(arr, n, descending);

  for (int i = 0; i < n; i++) {
    fprintf(outputFile, "%d ", arr[i]);
  }

  fclose(outputFile);
  free(arr);

  return 0;
}
```