Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Вятский государственный университет»

Колледж ВятГУ

**ОТЧЕТ**

**ПО ДОМАШНЕЙ КОНТРОЛЬНОЙ РАБОТЕ №5**

**«ИССЛЕДОВАНИЕ АЛГОРИТМОВ СОРТИРОВКИ»**

**ПО ДИСЦИПЛИНЕ**

**«ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ»**

Выполнила: студент учебной группы

ИСПк-101-51-00

Семенихин Александр Сергеевич

Преподаватель:

Сергеева Елизавета Григорьевна

Киров

2024

**Цель работы**: получить базовые сведения о наиболее известных алгоритмах сортировки, изучить принципы работы с текстовыми файлами.

1. **Формулировка задания 1. Вариант 14**

**Постановка задачи**

1. Реализовать сортировку данных с помощью вставок.
2. Реализовать сортировку данных с помощью пирамидального алгоритма.
3. В обоих случаях необходимо предусмотреть возможность изменения компаратора (реализация компаратора в виде передаваемой в подпрограмму функции).
4. Считывание и вывод данных необходимо производить из текстового файла.
5. Для демонстрации работы программных реализаций самостоятельно подготовить варианты входных данных (при этом объем текстовых файлов должен позволять оценить скорость работы программ).

**Описание алгоритма**

1. Запрос имени файла: Программа запрашивает у пользователя имя файла, содержащего целые числа.
2. Чтение данных из файла: Программа считывает числа из указанного файла в массив.
3. Проверка содержимого массива: Если массив пуст, программа выводит сообщение об ошибке и завершает выполнение.
4. Сортировка вставками: Массив сортируется с использованием алгоритма сортировки вставками, результат выводится на экран.
5. Пирамидальная сортировка: Массив сортируется с использованием алгоритма пирамидальной сортировки, результат выводится на экран.
6. Завершение работы: Программа завершает выполнение.
7. **Схема алгоритма с комментариями**

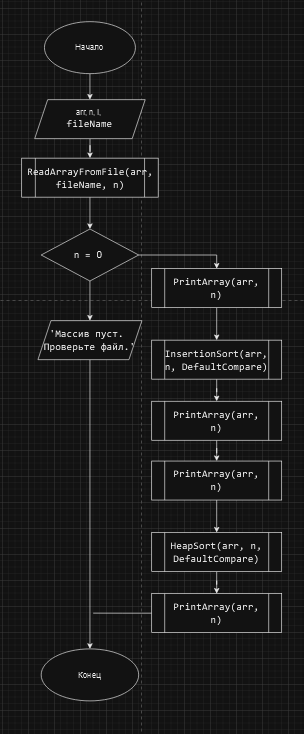
****

Рисунок 1 – Алгоритм решения задачи №1

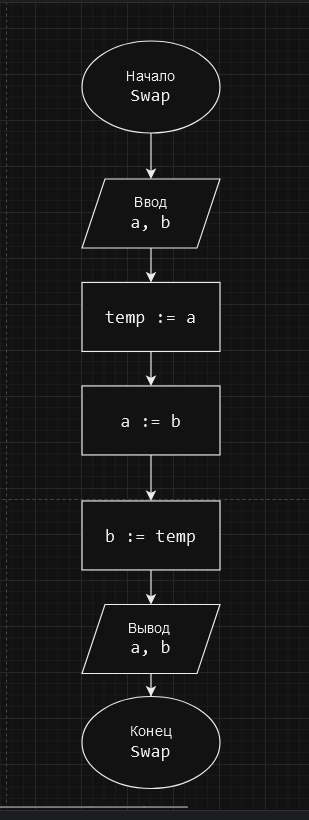


Рисунок 2 – Алгоритм выполнения функции Swap

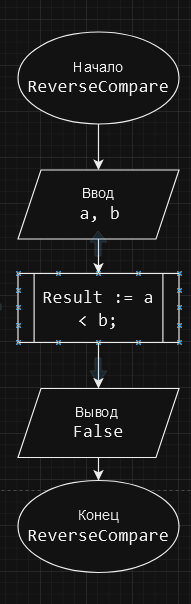


Рисунок 3 – Алгоритм выполнения функций ReverseCompare

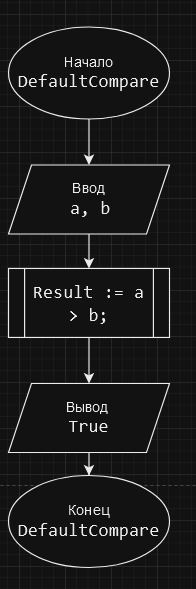


Рисунок 3 – Алгоритм выполнения функций DefaultCompare

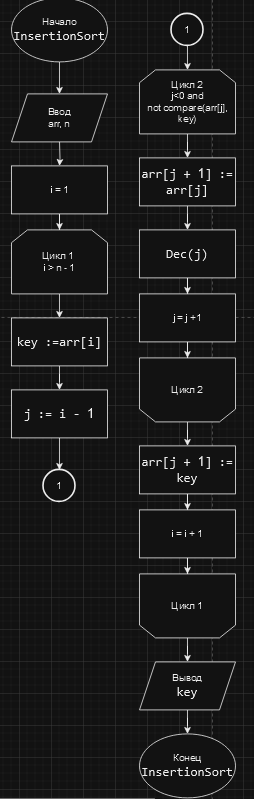


Рисунок 4 – Алгоритм выполнения функции InsertionSort

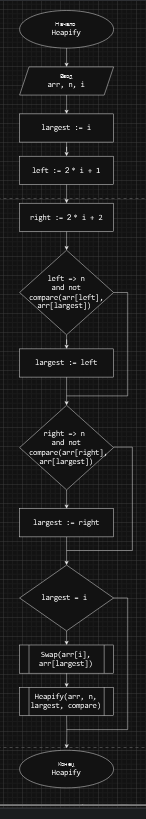


Рисунок 5 – Алгоритм выполнения функции Heapify

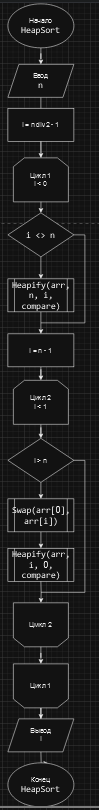


Рисунок 6 – Алгоритм выполнения функции HeapSort

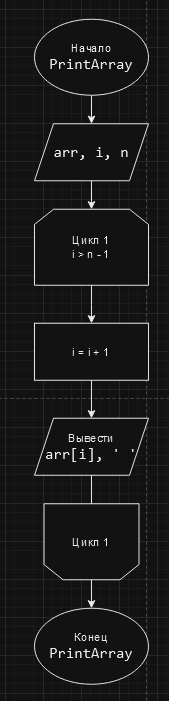
****

Рисунок 7 – Алгоритм выполнения функции PrintArray

****

Рисунок 8 – Алгоритм выполнения функции ReadArrayFromFile

1. **Код программы**
2. **type**
3. TCompareFunc = **function**(a, b: Integer): Boolean;
4. **procedure** Swap(**var** a, b: Integer);
5. **var**
6. temp: Integer;
7. **begin**
8. temp := a;
9. a := b;
10. b := temp;
11. **end**;
12. **function** DefaultCompare(a, b: Integer): Boolean;
13. **begin**
14. Result := a > b; // Сравнение по возрастанию
15. **end**;
16. **function** ReverseCompare(a, b: Integer): Boolean;
17. **begin**
18. Result := a < b; // Сравнение по убыванию
19. **end**;
20. **procedure** InsertionSort(**var** arr: **array of** Integer; n: Integer; compare: TCompareFunc);
21. **var**
22. i, j, key: Integer;
23. **begin**
24. **for** i := 1 **to** n - 1 **do**
25. **begin**
26. key := arr[i];
27. j := i - 1;
28. // Перемещаем элементы arr[0..i-1], которые больше key, на одну позицию вперед
29. **while** (j >= 0) **and** compare(arr[j], key) **do**
30. **begin**
31. arr[j + 1] := arr[j];
32. Dec(j);
33. **end**;
34. arr[j + 1] := key;
35. **end**;
36. **end**;
37. **procedure** Heapify(**var** arr: **array of** Integer; n, i: Integer; compare: TCompareFunc);
38. **var**
39. largest, left, right: Integer;
40. **begin**
41. largest := i; // Инициализируем наибольший элемент как корень
42. left := 2 \* i + 1; // Левый дочерний элемент
43. right := 2 \* i + 2; // Правый дочерний элемент
44. // Если левый дочерний элемент больше корня
45. **if** (left < n) **and** compare(arr[left], arr[largest]) **then**
46. largest := left;
47. // Если правый дочерний элемент больше наибольшего элемента
48. **if** (right < n) **and** compare(arr[right], arr[largest]) **then**
49. largest := right;
50. // Если наибольший элемент не корень
51. **if** largest <> i **then**
52. **begin**
53. Swap(arr[i], arr[largest]); // Меняем местами
54. // Рекурсивно хипифицируем затронутое поддерево
55. Heapify(arr, n, largest, compare);
56. **end**;
57. **end**;
58. **procedure** HeapSort(**var** arr: **array of** Integer; n: Integer; compare: TCompareFunc);
59. **var**
60. i: Integer;
61. **begin**
62. // Построение кучи (перегруппировка массива)
63. **for** i := n **div** 2 - 1 **downto** 0 **do**
64. Heapify(arr, n, i, compare);
65. // Один за другим извлекаем элементы из кучи
66. **for** i := n - 1 **downto** 1 **do**
67. **begin**
68. Swap(arr[0], arr[i]); // Перемещаем текущий корень в конец
69. Heapify(arr, i, 0, compare); // Вызываем хипификацию на уменьшенной куче
70. **end**;
71. **end**;
72. **procedure** PrintArray(arr: **array of** Integer; n: Integer);
73. **var**
74. i: Integer;
75. **begin**
76. **for** i := 0 **to** n - 1 **do**
77. Write(arr[i], ' ');
78. Writeln;
79. **end**;
80. **procedure** ReadArrayFromFile(**var** arr: **array of** Integer; **const** fileName: string; **var** n: Integer);
81. **var**
82. f: TextFile;
83. temp: Integer;
84. **begin**
85. AssignFile(f, fileName);
86. Reset(f);
88. n := 0; // Начальное количество элементов
89. SetLength(arr, 0); // Изначально массив пустой
91. // Считываем данные из файла
92. **while not** Eof(f) **do**
93. **begin**
94. Read(f, temp);
95. SetLength(arr, n + 1); // Увеличиваем размер массива
96. arr[n] := temp; // Записываем в массив
97. Inc(n); // Увеличиваем количество элементов
98. **end**;
99. CloseFile(f);
100. **end**;
101. **var**
102. arr: **array of** Integer;
103. n, i: Integer;
104. fileName: string;
105. **begin**
106. Write('Введите имя файла: ');
107. ReadLn(fileName);
109. // Считываем массив из файла
110. ReadArrayFromFile(arr, fileName, n);
111. // Проверяем, были ли считаны данные
112. **if** n = 0 **then**
113. **begin**
114. Writeln('Массив пуст. Проверьте файл.');
115. **Exit**;
116. **end**;
117. // Сортировка вставками
118. Writeln('Массив до сортировки вставками:');
119. PrintArray(arr, n);
120. InsertionSort(arr, n, DefaultCompare);
121. Writeln('Массив после сортировки вставками:');
122. PrintArray(arr, n);
123. // Сортировка кучей
124. Writeln('Массив до пирамидальной сортировки:');
125. PrintArray(arr, n);
126. HeapSort(arr, n, DefaultCompare);
127. Writeln('Массив после пирамидальной сортировки:');
128. PrintArray(arr, n);
129. **end**.
130. **Результат выполнения программы**

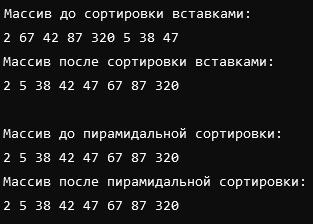
****

Рисунок 9 – Результаты тестирования задачи

**Вывод**

В процессе выполнения домашней контрольной работы мы изучили основные сведения о наиболее известных алгоритмах сортировки, а также принципы работы с текстовыми файлами. Для разработки алгоритмических схем мы освоили правила написания письменных алгоритмов, которые помогают достичь правильного решения задач. Кроме того, мы научились создавать схемы в программе draw.io — бесплатном онлайн-сервисе для создания и совместного редактирования диаграмм, схем и других визуальных представлений данных. В ходе работы возникли трудности. Например, при создании алгоритмических схем сложно было правильно оформить все функции из-за их большого количества. Однако, благодаря изучению материала и исправлению допущенных ошибок, нам удалось успешно решить поставленную задачу. Мы разработали код, реализующий алгоритмы сортировки вставками и пирамидальной сортировки, а также углубили свои знания в программировании на языке Pascal.