

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

КУРСОВАЯ РАБОТА
по дисциплине «Алгоритмы и структуры данных»
Тема: Слабые кучи

Студент гр. 9381

Щеглов Д.А.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2021

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Щеглов Д.А.

Группа 9381

Тема работы: Сортировка слабой кучей. Демонстрация.

Исходные данные:

На вход программе подаётся количество элементов массива и сам целочисленный массив, элементы массива разделены пробелом.

Содержание пояснительной записки:

«Содержание», «Введение», «Ход выполнения работы», «Заключение», «Список использованных источников».

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 31.10.2020

Дата сдачи реферата: 03.03.2021

Дата защиты реферата: 03.03.2021

Студент

Щеглов Д.А.

Преподаватель

Фирсов М.А.

АННОТАЦИЯ

В курсовой работе происходит сортировка массива. Программа демонстрирует процесс сортировки массива слабой кучей при помощи вывода на экран состояния элементов на каждом шаге. Результатом работы программы является отсортированный, с помощью слабой кучи, массив. Для написания программа потребовалось изучить структуру слабой кучи, алгоритм построения, алгоритм сортировки с помощью неё, а также придумать визуализацию работы алгоритма. Результатом является программа, которая считывает и сортирует исходный целочисленный массив, визуализируя работу алгоритма.

SUMMARY

In the course work, the array is sorted. The program demonstrates the process of sorting a weak heap by displaying the status of items at each step. The result of the program's work is an array sorted using the weak heap. To write the program, it was required to study the weak heap, the algorithm of the algorithm, the sorting algorithm using the algorithm, and also come up with a visualization of the algorithm's operation. The result is a program that reads and sorts the original integer array, visualizing the operation of the algorithm.

СОДЕРЖАНИЕ

	Введение	5
1.	Задание	7
2.	Описание классов и функций	7
2.1.	Класс слабой кучи <i>WeakHeap</i>	7
2.2.	Описание метода <i>void DisplayArray()</i>	7
2.3	Описание метода <i>void WeakHeapMerge(int *bit_array, inti, int j)</i>	8
2.4	Описание <i>void WeakHeapSort()</i>	8
2.5	Описание метода <i>WeakHeap* InputHeap()</i>	9
2.6	Описание деструктора <i>~WeakHeap()</i>	9
2.7	Описание <i>void weakHeapSort()</i>	9
2.8	Описание функции <i>double log(int a, int b)</i>	9
3.	Описание алгоритма сортировки	9
4.	Тестирование	10
5.	Демонстрация	65
	Заключение	67
	Список использованных источников	68
	Приложение. Исходный код программы.	69

ВВЕДЕНИЕ

Обычная куча представляет собой сортирующее дерево, в котором любой родитель больше (или равен) чем любой из его потомков. В слабой куче это требование ослаблено — любой родитель больше (или равен) любого потомка только из своего правого поддеревя. В левом поддереве потомки могут быть и меньше и больше родителя. Такой подход позволяет значительно сократить издержки по поддержанию набора данных в состоянии кучи. Ведь нужно обеспечить принцип «потомок не больше родителя» не для всей структуры, а только её половины. При этом слабая куча, не являясь на 100% сортирующим деревом, сортирует не хуже обычной кучи, а в чём-то даже и лучше. Поскольку в корне кучи, даже слабой, нужен именно максимальный по величине элемент, у корня левого поддеревя нет.

Цель работы

Целью работы является изучение сортировки методом слабой кучи и написание программы, которая будет сортировать входной массив.

Основные теоретические положения.

Есть теоретический нижний предел для оценки минимального количества сравнений (в тех сортировках, в которых эти сравнения широко используются):

$$\log n! = n \log n - n / \ln 2 + O(\log n), \text{ где } 1 / \ln 2 = 1.4426$$

В сортировке слабой кучей количество сравнений минимизировано и достаточно близко приближено к нижнему пределу. Это может иметь практическое значение, если нужно упорядочить объекты, сравнение которых дорого обходится, например, если речь идёт о сортировке длинных строк.

Итоговый алгоритм сортировки слабой кучей.

Формируем из массива слабую кучу:

I.1. Перебираем элементы массива слева-направо.

I.2. Для текущего элемента поднимаемся вверх по родительской ветке до ближайшего «правого» родителя.

I.3. Сравниваем текущий элемент и ближайшего правого родителя.

I.4. Если ближайший правый родитель меньше текущего элемента, то:

I.4.a. Меняем местами (левый \Leftrightarrow правый) поддеревья с потомками для узла, в котором находится текущий элемент.

I.4.b. Меняем значениями ближайший «правый» родитель и узел с текущим элементом.

II. Из корня кучи текущий максимальный элемент перемещаем в конец неотсортированной части массива, после чего восстанавливаем слабую кучу:

II.1. В корне кучи находится текущий максимальный элемент для неотсортированной части массива.

II.2. Меняем местами максимум из корня кучи и последний элемент в неотсортированной части массива. Последний элемент с максимумом перестаёт быть узлом слабой кучи.

II.3. После этого обмена дерево перестало быть слабой кучей, так как в корне оказался не максимальный элемент. Поэтому делаем просейку:

II.3.a. Опускаемся из корня кучи по левым потомкам как можно ниже.

II.3.b. Поднимаемся по левым потомкам обратно к корню кучи, сравнивая каждый левый потомок с корнем.

II.3.v. Если элемент в корне меньше, чем очередной левый потомок, то:

II.3.v.1. Меняем местами (левый \Leftrightarrow правый) поддеревья с потомками для узла, в котором находится текущий левый потомок.

II.3.v.2. Меняем значениями корень кучи и узел с текущим левым потомком.

II.4. В корне слабой кучи снова находится максимальный элемент для оставшейся неотсортированной части массива. Возвращаемся в пункт II.1 и повторяем процесс, пока не будут отсортированы все элементы.

1. ЗАДАНИЕ

Вариант № 31. Сортировка слабой кучей. Демонстрация.

2. ОПИСАНИЕ КЛАССОВ И ФУНКЦИЙ

Класс слабой кучи **WeakHeap**

Для работы со слабой кучей был создан класс слабой кучи **WeakHeap**.

Публичными полями класса являются:

vector <int> wheap — хранятся элементы введённого массива, это вектор, встроенная возможность языка программирования, хранится в библиотеке *<vector>*

int size_array — хранится количество элементов *wheap*,

int bit_array* — массив для хранения информации об обмене поддеревьями слабой кучи,

int size_array — размер массива бит.

Для класса реализованы публичные методы для работы со слабой кучей.

Описание метода *void DisplayArray()*

Метод выводит элементы массива, который хранится в поле класса *vector <int> wheap*. Таким образом, с помощью него отсортированный массив выводится на экран.

Описание метода *void DisplayHeap()*

Записываем элементы в кучу в порядке, удобном для вывода на экран. Затем вычисляем глубину кучи при помощи функции $\log(2, \text{size})$. Затем проходимся по элементам, выводим их в наглядном виде, в виде бинарного дерева. Счётчик *k* нужен для случая, когда количество элементов в слабой куче массива не будет равно степени двойки, без него вместо отсутствующих элементов на последнем уровне выведутся нули.

Переменные:

int depth – переменная, для вывода кучи по уровням.

int idt – переменная, для изменения отступов от левой границы консоли.

Используется для наглядного отображения кучи на экран.

void DisplayHeap(int i_1, int j_1, int num)- перегрузка метода displayHeap, используется для того, чтобы можно было выделять меняющиеся местами элементы.

int_i, int_j – элементы, которые надо выделить.

int num – количество отсортированных элементов в массиве.

Описание метода *void WeakHeapMerge(unsigned char* r, int i, int j)*;

unsigned char r* – массив для обозначения, какой потомок у элемента правый, а какой левый.

int i – индекс потомка.

int j – индекс суперродителя.

Метод принимает на вход массив, который обозначает правых и левых потомков элемента и их индексы. Если суперродитель меньше потомка, то для потомка переопределяем, порядок его потомков, затем меняем значения суперродителя и потомка при помощи *swap()*, выводим слабую кучу на экран для демонстрации, чтобы было видно, какие элементы могли поменяться.

Описание *void WeakHeapSort()*

Метод, в котором происходит сортировка. Реализуется обход бинарного дерева с помощью битовых операций.

Переменные:

int n – переменная, в которой хранится размер кучи.

unsigned char r* – массив для обозначения, какой потомок у элемента правый, а какой левый.

Gparent – переменная для хранения суперродителя.

int i, j, x, y – переменные, для обхода кучи

Описание метода *WeakHeap* InputHeap()*

Данный метод создаёт новый объект слабой кучи при вводе с консоли.

int count – количество считываемых элементов.

int elem – переменная, для считывания элементов.

Описание деструктора *~WeakHeap()*

Деструктор класса, очищает вектор *wheap*, в котором хранятся элементы.

Описание функции *double logarifm(int a, int b)*

Функция принимает целые значения, которые соответствуют логарифму по основанию *a* от *b*, и возвращает значение самого логарифма.

int a – основание логарифма.

int b – показатель степени логарифма.

Для реализации использованная функция $\log(val)$, которая вычисляет значение натурального логарифма от *val* и возвращается его.

3. ОПИСАНИЕ АЛГОРИТМА СОРТИРОВКИ

Сначала формируем из массива слабую кучу. Перебираем элементы массива слева-направо, для текущего элемента поднимаемся вверх по родительской ветке до ближайшего «правого» родителя, сравниваем текущий элемент и ближайшего правого родителя. Если ближайший правый родитель меньше текущего элемента, то меняем местами поддерева с потомками для узла, в котором находится текущий элемент, меняем значениями ближайший «правый» родитель и узел с текущим элементом.

Затем из корня кучи текущий максимальный элемент перемещаем в конец не отсортированной части массива, после чего восстанавливаем слабую кучу: в корне кучи находится текущий максимальный элемент для не отсортированной части массива, меняем местами максимум из корня кучи и последний элемент в неотсортированной части массива. Последний элемент с максимумом перестает быть узлом слабой кучи. После этого обмена дерево перестало быть слабой

кучей, так как в корне оказался не максимальный элемент. Поэтому делаем просейку: опускаемся из корня кучи по левым потомкам как можно ниже. Поднимаемся по левым потомкам обратно к корню кучи, сравнивая каждый левый потомок с корнем. Если элемент в корне меньше, чем очередной левый потомок, то: меняем местами поддерева с потомками для узла, в котором находится текущий левый потомок. Меняем значениями корень кучи и узел с текущим левым потомком. В корне слабой кучи снова находится максимальный элемент для оставшейся не отсортированной части массива. Затем снова из корня кучи текущий максимальный элемент перемещаем в конец не отсортированной части массива, восстанавливаем слабую кучу, повторяем процесс, пока не будут отсортированы все элементы.

4. ТЕСТИРОВАНИЕ

Номер теста	Входные данные	Выходные данные
1	16 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1	<p>Программа для визуализации сортировки слабой кучей.</p> <p>На примере данной программы, можно увидеть, как происходит сортировка слабой кучей.</p> <p>Список команд:</p> <ol style="list-style-type: none"> 1.Ввести массив с клавиатуры 2.Завершить работу программы <p>Введите номер команды:</p> <p>1</p> <p>Введите количество элементов массива:</p>

		<p>16</p> <p>Введите через пробел элементы массива:</p> <p>16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1</p> <p>1 ЭТАП 1 ЭТАП 1 ЭТАП 1 ЭТАП 1 ЭТАП</p> <p>Построение первоначальной слабой кучи:</p> <p>Выделяем рассматриваемые элементы!</p> <p>16</p> <p>15</p> <p>14 13</p> <p>12 11 10 (9)</p> <p>8 7 6 5 4 3 2 (1)</p> <hr/> <p>СупперРодитель больше и равен потомку, оставляем всё, как есть.</p> <p>Выделяем рассматриваемые элементы!</p> <p>16</p> <p>15</p> <p>14 (13)</p> <p>12 11 10 9</p> <p>8 7 6 5 4 3 (2) 1</p> <hr/> <p>СупперРодитель больше и равен потомку, оставляем всё, как есть.</p> <p>Выделяем рассматриваемые элементы!</p> <p>16</p> <p>15</p>
--	--	---

		14 13 12 11 (10) 9 8 7 6 5 4 (3) 2 1 <hr/> <hr/> СупперРодитель больше и равен потомку, оставляем всё, как есть. Выделяем рассматриваемые элементы! 16 (15) 14 13 12 11 10 9 8 7 6 5 (4) 3 2 1 <hr/> <hr/> СупперРодитель больше и равен потомку, оставляем всё, как есть. Выделяем рассматриваемые элементы! 16 15 14 13 12 (11) 10 9 8 7 6 (5) 4 3 2 1 <hr/> <hr/> СупперРодитель больше и равен потомку, оставляем всё, как есть. Выделяем рассматриваемые элементы! 16
--	--	--

		<p>15</p> <p>(14) 13</p> <p>12 11 10 9</p> <p>8 7 (6) 5 4 3 2 1</p> <hr/> <hr/> <p>СупперРодитель больше и равен потомку, оставляем всё, как есть.</p> <p>Выделяем рассматриваемые элементы!</p> <p>16</p> <p>15</p> <p>14 13</p> <p>(12) 11 10 9</p> <p>8 (7) 6 5 4 3 2 1</p> <hr/> <hr/> <p>СупперРодитель больше и равен потомку, оставляем всё, как есть.</p> <p>Выделяем рассматриваемые элементы!</p> <p>(16)</p> <p>15</p> <p>14 13</p> <p>12 11 10 9</p> <p>(8) 7 6 5 4 3 2 1</p> <hr/> <hr/> <p>СупперРодитель больше и равен потомку, оставляем всё, как есть.</p> <p>Выделяем рассматриваемые элементы!</p>
--	--	--

		16 15 14 (13) 12 11 10 (9) 8 7 6 5 4 3 2 1 <hr/> <hr/> СупперРодитель больше и равен потомку, оставляем всё, как есть. Выделяем рассматриваемые элементы! 16 (15) 14 13 12 11 (10) 9 8 7 6 5 4 3 2 1 <hr/> <hr/> СупперРодитель больше и равен потомку, оставляем всё, как есть. Выделяем рассматриваемые элементы! 16 15 (14) 13 12 (11) 10 9 8 7 6 5 4 3 2 1 <hr/> <hr/> СупперРодитель больше и равен потомку, оставляем всё, как есть.
--	--	--

		<p>Выделяем рассматриваемые элементы!</p> <p>(16)</p> <p>15</p> <p>14 13</p> <p>(12) 11 10 9</p> <p>8 7 6 5 4 3 2 1</p> <hr/> <hr/> <p>СупперРодитель больше и равен потомку, оставляем всё, как есть.</p> <p>Выделяем рассматриваемые элементы!</p> <p>16</p> <p>(15)</p> <p>14 (13)</p> <p>12 11 10 9</p> <p>8 7 6 5 4 3 2 1</p> <hr/> <hr/> <p>СупперРодитель больше и равен потомку, оставляем всё, как есть.</p> <p>Выделяем рассматриваемые элементы!</p> <p>(16)</p> <p>15</p> <p>(14) 13</p> <p>12 11 10 9</p> <p>8 7 6 5 4 3 2 1</p> <hr/> <hr/> <p>СупперРодитель больше и равен потомку,</p>
--	--	---

оставляем всё, как есть.

Выделяем рассматриваемые элементы!

(16)

(15)

14 13

12 11 10 9

8 7 6 5 4 3 2 1

СупперРодитель больше и равен потомку,

оставляем всё, как есть.

Слабая куча построена:

16

15

14 13

12 11 10 9

8 7 6 5 4 3 2 1

2 ЭТАП 2 ЭТАП 2 ЭТАП 2 ЭТАП 2 ЭТАП

Переносим максимум из корня, применяем

слабую просейку:

(16)

15

14 13

12 11 10 9

8 7 6 5 4 3 2 (1)

		<p>Переместили корень 16 и элемент из конца неотсортированной части 1</p> <p>Отсортированная часть массива: 16</p> <p>Неотсортированная часть массива: 1 15 14 13 12 11 10 9 8 7 6 5 4 3 2</p> <p>(1)</p> <p>15</p> <p>14 13</p> <p>12 11 10 9</p> <p>8 7 6 5 4 3 2 (16)</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(1)</p> <p>15</p> <p>14 13</p> <p>12 11 10 9</p> <p>(8) 7 6 5 4 3 2 16</p> <hr/> <hr/> <p>Суперродитель 1 меньше потомка 8, меняем их местами</p> <p>(8)</p> <p>15</p> <p>14 13</p> <p>12 11 10 9</p> <p>(1) 7 6 5 4 3 2 16</p> <hr/>
--	--	---

Выделяем рассматриваемые элементы!

(8)

15

14 13

(12) 11 10 9

1 7 6 5 4 3 2 16

Суперродитель 8 меньше потомка 12, меняем их местами

(12)

15

14 13

(8) 11 10 9

1 7 6 5 4 3 2 16

Выделяем рассматриваемые элементы!

(12)

15

(14) 13

8 11 10 9

1 7 6 5 4 3 2 16

Суперродитель 12 меньше потомка 14, меняем их местами

(14)

		<p>15</p> <p>(12) 13</p> <p>8 11 10 9</p> <p>1 7 6 5 4 3 2 16</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(14)</p> <p>(15)</p> <p>12 13</p> <p>8 11 10 9</p> <p>1 7 6 5 4 3 2 16</p> <hr/> <hr/> <p>Суперродитель 14 меньше потомка 15, меняем их местами</p> <p>(15)</p> <p>(14)</p> <p>12 13</p> <p>8 11 10 9</p> <p>1 7 6 5 4 3 2 16</p> <hr/> <hr/> <p>Переносим максимум из корня, применяем слабую просейку:</p> <p>(15)</p> <p>14</p> <p>12 13</p> <p>8 11 10 9</p>
--	--	---

		<p>1 7 6 5 4 3 (2)</p> <hr/> <hr/> <p>Переместили корень 15 и элемент из конца неотсортированной части 2</p> <p>Отсортированная часть массива: 15 16</p> <p>Неотсортированная часть массива: 2 14 12 13 8 11 10 9 1 7 6 5 4 3</p> <p>(2)</p> <p>14</p> <p>12 13</p> <p>8 11 10 9</p> <p>1 7 6 5 4 3 (15)</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(2)</p> <p>14</p> <p>12 13</p> <p>8 11 10 9</p> <p>1 7 6 5 (4) 3 15</p> <hr/> <hr/> <p>Суперродитель 2 меньше потомка 4, меняем их местами</p> <p>(4)</p> <p>14</p> <p>12 13</p>
--	--	--

		<p>8 11 10 9</p> <p>1 7 6 5 (2) 3 15</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(4)</p> <p>14</p> <p>12 13</p> <p>8 11 (10) 9</p> <p>1 7 6 5 2 3 15</p> <hr/> <hr/> <p>Суперродитель 4 меньше потомка 10, меняем их местами</p> <p>(10)</p> <p>14</p> <p>12 13</p> <p>8 11 (4) 9</p> <p>1 7 6 5 2 3 15</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(10)</p> <p>14</p> <p>12 (13)</p> <p>8 11 4 9</p> <p>1 7 6 5 2 3 15</p> <hr/> <hr/>
--	--	---

		<p>Суперродитель 10 меньше потомка 13, меняем их местами</p> <p>(13)</p> <p>14</p> <p>12 (10)</p> <p>8 11 4 9</p> <p>1 7 6 5 2 3 15</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(13)</p> <p>(14)</p> <p>12 10</p> <p>8 11 4 9</p> <p>1 7 6 5 2 3 15</p> <hr/> <hr/> <p>Суперродитель 13 меньше потомка 14, меняем их местами</p> <p>(14)</p> <p>(13)</p> <p>12 10</p> <p>8 11 4 9</p> <p>1 7 6 5 2 3 15</p> <hr/> <hr/> <p>Переносим максимум из корня, применяем слабую просейку:</p> <p>(14)</p>
--	--	---

		<p>13</p> <p>12 10</p> <p>8 11 4 9</p> <p>1 7 6 5 2 (3)</p> <hr/> <hr/> <p>Переместили корень 14 и элемент из конца неотсортированной части 3</p> <p>Отсортированная часть массива: 14 15 16</p> <p>Неотсортированная часть массива: 3 13 12 10 8 11 4 9 1 7 6 5 2</p> <p>(3)</p> <p>13</p> <p>12 10</p> <p>8 11 4 9</p> <p>1 7 6 5 2 (14)</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(3)</p> <p>13</p> <p>12 10</p> <p>8 11 4 9</p> <p>1 7 (6) 5 2 14</p> <hr/> <hr/> <p>Суперродитель 3 меньше потомка 6, меняем их местами</p>
--	--	---

		<p>(6)</p> <p>13</p> <p>12 10</p> <p>8 11 4 9</p> <p>1 7 (3) 5 2 14</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(6)</p> <p>13</p> <p>12 10</p> <p>8 (11) 4 9</p> <p>1 7 3 5 2 14</p> <hr/> <hr/> <p>Суперродитель 6 меньше потомка 11, меняем их местами</p> <p>(11)</p> <p>13</p> <p>12 10</p> <p>8 (6) 4 9</p> <p>1 7 3 5 2 14</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(11)</p> <p>13</p> <p>(12) 10</p> <p>8 6 4 9</p>
--	--	---

		<div>1 7 3 5 2 14</div> <hr/> <div>Суперродитель 11 меньше потомка 12, меняем их местами</div> <div>(12)</div> <div>13</div> <div>(11) 10</div> <div>8 6 4 9</div> <div>1 7 3 5 2 14</div> <hr/> <div>Выделяем рассматриваемые элементы!</div> <div>(12)</div> <div>(13)</div> <div>11 10</div> <div>8 6 4 9</div> <div>1 7 3 5 2 14</div> <hr/> <div>Суперродитель 12 меньше потомка 13, меняем их местами</div> <div>(13)</div> <div>(12)</div> <div>11 10</div> <div>8 6 4 9</div> <div>1 7 3 5 2 14</div> <hr/>
--	--	---

		<p>Переносим максимум из корня, применяем слабую просейку:</p> <p>(13)</p> <p>12</p> <p>11 10</p> <p>8 6 4 9</p> <p>1 7 3 5 (2)</p> <hr/> <hr/> <p>Переместили корень 13 и элемент из конца неотсортированной части 2</p> <p>Отсортированная часть массива: 13 14 15 16</p> <p>Неотсортированная часть массива: 2 12 11 10 8 6 4 9 1 7 3 5</p> <p>(2)</p> <p>12</p> <p>11 10</p> <p>8 6 4 9</p> <p>1 7 3 5 (13)</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(2)</p> <p>12</p> <p>11 10</p> <p>8 6 4 (9)</p> <p>1 7 3 5 13</p> <hr/>
--	--	--

Суперродитель 2 меньше потомка 9, меняем их местами

(9)

12

11 10

8 6 4 (2)

1 7 3 5 13

Выделяем рассматриваемые элементы!

(9)

12

11 (10)

8 6 4 2

1 7 3 5 13

Суперродитель 9 меньше потомка 10, меняем их местами

(10)

12

11 (9)

8 6 4 2

1 7 3 5 13

Выделяем рассматриваемые элементы!

(10)

		<p>(12)</p> <p>11 9</p> <p>8 6 4 2</p> <p>1 7 3 5 13</p> <hr/> <hr/> <p>Суперродитель 10 меньше потомка 12, меняем их местами</p> <p>(12)</p> <p>(10)</p> <p>11 9</p> <p>8 6 4 2</p> <p>1 7 3 5 13</p> <hr/> <hr/> <p>Переносим максимум из корня, применяем слабую просейку:</p> <p>(12)</p> <p>10</p> <p>11 9</p> <p>8 6 4 2</p> <p>1 7 3 (5)</p> <hr/> <hr/> <p>Переместили корень 12 и элемент из конца неотсортированной части 5</p> <p>Отсортированная часть массива: 12 13 14 15</p> <p>16</p> <p>Неотсортированная часть массива: 5 10 11 9 8</p>
--	--	---

		<p>6 4 2 1 7 3</p> <p>(5)</p> <p>10</p> <p>11 9</p> <p>8 6 4 2</p> <p>1 7 3 (12)</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(5)</p> <p>10</p> <p>11 9</p> <p>8 6 4 2</p> <p>1 (7) 3 12</p> <hr/> <hr/> <p>Суперродитель 5 меньше потомка 7, меняем их местами</p> <p>(7)</p> <p>10</p> <p>11 9</p> <p>8 6 4 2</p> <p>1 (5) 3 12</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(7)</p> <p>10</p>
--	--	--

		<p>11 9</p> <p>(8) 6 4 2</p> <p>1 5 3 12</p> <hr/> <hr/> <p>Суперродитель 7 меньше потомка 8, меняем их местами</p> <p>(8)</p> <p>10</p> <p>11 9</p> <p>(7) 6 4 2</p> <p>1 5 3 12</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(8)</p> <p>10</p> <p>(11) 9</p> <p>7 6 4 2</p> <p>1 5 3 12</p> <hr/> <hr/> <p>Суперродитель 8 меньше потомка 11, меняем их местами</p> <p>(11)</p> <p>10</p> <p>(8) 9</p> <p>7 6 4 2</p> <p>1 5 3 12</p>
--	--	---

Выделяем рассматриваемые элементы!

(11)

(10)

8 9

7 6 4 2

1 5 3 12

СупперРодитель больше и равен потомку,
оставляем всё, как есть.

Переносим максимум из корня, применяем
слабую просейку:

(11)

10

8 9

7 6 4 2

1 5 (3)

Переместили корень 11 и элемент из конца
неотсортированной части 3

Отсортированная часть массива: 11 12 13 14
15 16

Неотсортированная часть массива: 3 10 8 9 7 6
4 2 1 5

(3)

		10 8 9 7 6 4 2 1 5 (11) <hr/> <hr/> Выделяем рассматриваемые элементы! (3) 10 8 9 7 (6) 4 2 1 5 11 <hr/> <hr/> Суперродитель 3 меньше потомка 6, меняем их местами (6) 10 8 9 7 (3) 4 2 1 5 11 <hr/> <hr/> Выделяем рассматриваемые элементы! (6) 10 (8) 9 7 3 4 2 1 5 11
--	--	--

Суперродитель 6 меньше потомка 8, меняем их местами

(8)

10

(6) 9

7 3 4 2

1 5 11

Выделяем рассматриваемые элементы!

(8)

(10)

6 9

7 3 4 2

1 5 11

Суперродитель 8 меньше потомка 10, меняем их местами

(10)

(8)

6 9

7 3 4 2

1 5 11

Переносим максимум из корня, применяем

		<p>слабую просейку:</p> <p>(10)</p> <p>8</p> <p>6 9</p> <p>7 3 4 2</p> <p>1 (5)</p> <hr/> <hr/> <p>Переместили корень 10 и элемент из конца неотсортированной части 5</p> <p>Отсортированная часть массива: 10 11 12 13 14 15 16</p> <p>Неотсортированная часть массива: 5 8 6 9 7 3 4 2 1</p> <p>(5)</p> <p>8</p> <p>6 9</p> <p>7 3 4 2</p> <p>1 (10)</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(5)</p> <p>8</p> <p>6 9</p> <p>7 3 (4) 2</p> <p>1 10</p> <hr/>
--	--	--

СупперРодитель больше и равен потомку,
оставляем всё, как есть.

Выделяем рассматриваемые элементы!

(5)

8

6 (9)

7 3 4 2

1 10

Суперродитель 5 меньше потомка 9, меняем их
местами

(9)

8

6 (5)

7 3 4 2

1 10

Выделяем рассматриваемые элементы!

(9)

(8)

6 5

7 3 4 2

1 10

СупперРодитель больше и равен потомку,

		<p>оставляем всё, как есть.</p> <p>Переносим максимум из корня, применяем слабую просейку:</p> <p>(9)</p> <p>8</p> <p>6 5</p> <p>7 3 4 2</p> <p>(1)</p> <hr/> <hr/> <p>Переместили корень 9 и элемент из конца неотсортированной части 1</p> <p>Отсортированная часть массива: 9 10 11 12 13 14 15 16</p> <p>Неотсортированная часть массива: 1 8 6 5 7 3 4 2</p> <p>(1)</p> <p>8</p> <p>6 5</p> <p>7 3 4 2</p> <p>(9)</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(1)</p> <p>8</p> <p>6 5</p> <p>7 3 4 (2)</p>
--	--	--

		<div>9</div> <div></div> <div></div> <div>Суперродитель 1 меньше потомка 2, меняем их местами</div> <div>(2)</div> <div>8</div> <div>6 5</div> <div>7 3 4 (1)</div> <div>9</div> <div></div> <div></div> <div>Выделяем рассматриваемые элементы!</div> <div>(2)</div> <div>8</div> <div>6 (5)</div> <div>7 3 4 1</div> <div>9</div> <div></div> <div></div> <div>Суперродитель 2 меньше потомка 5, меняем их местами</div> <div>(5)</div> <div>8</div> <div>6 (2)</div> <div>7 3 4 1</div> <div>9</div> <div></div> <div></div>
--	--	---

		<p>Выделяем рассматриваемые элементы!</p> <p>(5)</p> <p>(8)</p> <p>6 2</p> <p>7 3 4 1</p> <p>9</p> <hr/> <hr/> <p>Суперродитель 5 меньше потомка 8, меняем их местами</p> <p>(8)</p> <p>(5)</p> <p>6 2</p> <p>7 3 4 1</p> <p>9</p> <hr/> <hr/> <p>Переносим максимум из корня, применяем слабую просейку:</p> <p>(8)</p> <p>5</p> <p>6 2</p> <p>7 3 4 (1)</p> <hr/> <hr/> <p>Переместили корень 8 и элемент из конца неотсортированной части 1</p> <p>Отсортированная часть массива: 8 9 10 11 12</p> <p>13 14 15 16</p>
--	--	--

		<p>Неотсортированная часть массива: 1 5 6 2 7 3 4</p> <p>(1)</p> <p>5</p> <p>6 2</p> <p>7 3 4 (8)</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(1)</p> <p>5</p> <p>6 2</p> <p>(7) 3 4 8</p> <hr/> <hr/> <p>Суперродитель 1 меньше потомка 7, меняем их местами</p> <p>(7)</p> <p>5</p> <p>6 2</p> <p>(1) 3 4 8</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(7)</p> <p>5</p> <p>(6) 2</p> <p>1 3 4 8</p> <hr/>
--	--	--

СупперРодитель больше и равен потомку,
оставляем всё, как есть.

Выделяем рассматриваемые элементы!

(7)

(5)

6 2

1 3 4 8

СупперРодитель больше и равен потомку,
оставляем всё, как есть.

Переносим максимум из корня, применяем
слабую просейку:

(7)

5

6 2

1 3 (4)

Переместили корень 7 и элемент из конца
неотсортированной части 4

Отсортированная часть массива: 7 8 9 10 11
12 13 14 15 16

Неотсортированная часть массива: 4 5 6 2 1 3

(4)

5

6 2

		<p>1 3 (7)</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(4)</p> <p>5</p> <p>6 2</p> <p>(1) 3 7</p> <hr/> <hr/> <p>СупперРодитель больше и равен потомку, оставляем всё, как есть.</p> <p>Выделяем рассматриваемые элементы!</p> <p>(4)</p> <p>5</p> <p>(6) 2</p> <p>1 3 7</p> <hr/> <hr/> <p>Суперродитель 4 меньше потомка 6, меняем их местами</p> <p>(6)</p> <p>5</p> <p>(4) 2</p> <p>1 3 7</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(6)</p>
--	--	---

		<p>(5)</p> <p>4 2</p> <p>1 3 7</p> <hr/> <hr/> <p>СупперРодитель больше и равен потомку, оставляем всё, как есть.</p> <p>Переносим максимум из корня, применяем слабую просейку:</p> <p>(6)</p> <p>5</p> <p>4 2</p> <p>1 (3)</p> <hr/> <hr/> <p>Переместили корень 6 и элемент из конца неотсортированной части 3</p> <p>Отсортированная часть массива: 6 7 8 9 10 11 12 13 14 15 16</p> <p>Неотсортированная часть массива: 3 5 4 2 1</p> <p>(3)</p> <p>5</p> <p>4 2</p> <p>1 (6)</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(3)</p>
--	--	---

		<p>5</p> <p>(4) 2</p> <p>1 6</p> <hr/> <hr/> <p>Суперродитель 3 меньше потомка 4, меняем их местами</p> <p>(4)</p> <p>5</p> <p>(3) 2</p> <p>1 6</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(4)</p> <p>(5)</p> <p>3 2</p> <p>1 6</p> <hr/> <hr/> <p>Суперродитель 4 меньше потомка 5, меняем их местами</p> <p>(5)</p> <p>(4)</p> <p>3 2</p> <p>1 6</p> <hr/> <hr/> <p>Переносим максимум из корня, применяем</p>
--	--	--

		<p>слабую просейку:</p> <p>(5)</p> <p>4</p> <p>3 2</p> <p>(1)</p> <hr/> <hr/> <p>Переместили корень 5 и элемент из конца неотсортированной части 1</p> <p>Отсортированная часть массива: 5 6 7 8 9 10</p> <p>11 12 13 14 15 16</p> <p>Неотсортированная часть массива: 1 4 3 2</p> <p>(1)</p> <p>4</p> <p>3 2</p> <p>(5)</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(1)</p> <p>4</p> <p>3 (2)</p> <p>5</p> <hr/> <hr/> <p>Суперродитель 1 меньше потомка 2, меняем их местами</p> <p>(2)</p>
--	--	--

		<p>4</p> <p>3 (1)</p> <p>5</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(2)</p> <p>(4)</p> <p>3 1</p> <p>5</p> <hr/> <hr/> <p>Суперродитель 2 меньше потомка 4, меняем их местами</p> <p>(4)</p> <p>(2)</p> <p>3 1</p> <p>5</p> <hr/> <hr/> <p>Переносим максимум из корня, применяем слабую просейку:</p> <p>(4)</p> <p>2</p> <p>3 (1)</p> <hr/> <hr/> <p>Переместили корень 4 и элемент из конца неотсортированной части 1</p>
--	--	--

		<p>Отсортированная часть массива: 4 5 6 7 8 9 10 11 12 13 14 15 16</p> <p>Неотсортированная часть массива: 1 2 3</p> <p>(1) 2 3 (4)</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(1) 2 (3) 4</p> <hr/> <hr/> <p>Суперродитель 1 меньше потомка 3, меняем их местами</p> <p>(3) 2 (1) 4</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(3) (2) 1 4</p> <hr/> <hr/> <p>СуперРодитель больше и равен потомку,</p>
--	--	--

		<p>оставляем всё, как есть.</p> <p>Переносим максимум из корня, применяем слабую просейку:</p> <p>(3)</p> <p>2</p> <p>(1)</p> <hr/> <hr/> <p>Переместили корень 3 и элемент из конца неотсортированной части 1</p> <p>Отсортированная часть массива: 3 4 5 6 7 8 9</p> <p>10 11 12 13 14 15 16</p> <p>Неотсортированная часть массива: 1 2</p> <p>(1)</p> <p>2</p> <p>(3)</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(1)</p> <p>(2)</p> <p>3</p> <hr/> <hr/> <p>Суперродитель 1 меньше потомка 2, меняем их местами</p> <p>(2)</p> <p>(1)</p>
--	--	---

		<p>3</p> <hr/> <hr/> <p>(2)</p> <p>(1)</p> <hr/> <hr/> <p>Меняем местами корень 2 и следующий за ним элемент 1</p> <p>(1)</p> <p>(2)</p> <hr/> <hr/> <p>В результате сортировки массив примет вид: Итоговый массив: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16</p> <p>Список команд:</p> <p>1.Ввести массив с клавиатуры</p> <p>2.Завершить работу программы</p> <p>Введите номер команды:</p>
2	<p>9</p> <p>1 2 3 4 5 6 7 8 9</p>	<p>Программа для визуализации сортировки слабой кучей.</p> <p>На примере данной программы, можно увидеть, как происходит сортировка слабой кучей.</p>

Список команд:

1. Ввести массив с клавиатуры
2. Завершить работу программы

Введите номер команды:

1

Введите количество элементов массива:

9 8 9

Введите через пробел элементы массива:

1 2 3 4 5 6 7 8 9

1 ЭТАП 1 ЭТАП 1 ЭТАП 1 ЭТАП 1 ЭТАП

Построение первоначальной слабой кучи:

Выделяем рассматриваемые элементы!

(1)

2

3 4

5 6 7 8

(9)

Суперродитель 1 меньше потомка 9, меняем их местами

(9)

2

3 4

5 6 7 8

(1)

Выделяем рассматриваемые элементы!

9

2

3 (4)

5 6 7 (8)

1

Суперродитель 4 меньше потомка 8, меняем их местами

9

2

3 (8)

5 6 7 (4)

1

Выделяем рассматриваемые элементы!

9

(2)

3 8

5 6 (7) 4

1

Суперродитель 2 меньше потомка 7, меняем их местами

		<p>9</p> <p>(7)</p> <p>3 8</p> <p>5 6 (2) 4</p> <p>1</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>9</p> <p>7</p> <p>(3) 8</p> <p>5 (6) 2 4</p> <p>1</p> <hr/> <hr/> <p>Суперродитель 3 меньше потомка 6, меняем их местами</p> <p>9</p> <p>7</p> <p>(6) 8</p> <p>5 (3) 2 4</p> <p>1</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(9)</p> <p>7</p> <p>6 8</p> <p>(5) 3 2 4</p>
--	--	---

		<div>1</div> <hr/> <div>СупперРодитель больше и равен потомку, оставляем всё, как есть.</div> <div>Выделяем рассматриваемые элементы!</div> <div>9</div> <div>(7)</div> <div>6 (8)</div> <div>5 3 2 4</div> <div>1</div> <hr/> <div>Суперродитель 7 меньше потомка 8, меняем их местами</div> <div>9</div> <div>(8)</div> <div>6 (7)</div> <div>5 3 2 4</div> <div>1</div> <hr/> <div>Выделяем рассматриваемые элементы!</div> <div>(9)</div> <div>8</div> <div>(6) 7</div> <div>5 3 2 4</div> <div>1</div> <hr/>
--	--	---

СупперРодитель больше и равен потомку,
оставляем всё, как есть.

Выделяем рассматриваемые элементы!

(9)

(8)

6 7

5 3 2 4

1

СупперРодитель больше и равен потомку,
оставляем всё, как есть.

Слабая куча построена:

9

8

6 7

5 3 2 4

1

2 ЭТАП 2 ЭТАП 2 ЭТАП 2 ЭТАП 2 ЭТАП

Переносим максимум из корня, применяем
слабую просейку:

(9)

8

6 7

5 3 2 4

(1)

Переместили корень 9 и элемент из конца
неотсортированной части 1

Отсортированная часть массива: 9

Неотсортированная часть массива: 1 8 6 7 5 3 2
4

(1)

8

6 7

5 3 2 4

(9)

Выделяем рассматриваемые элементы!

(1)

8

6 7

(5) 3 2 4

9

Суперродитель 1 меньше потомка 5, меняем их
местами

(5)

8

6 7

(1) 3 2 4

9

Выделяем рассматриваемые элементы!

(5)

8

(6) 7

1 3 2 4

9

Суперродитель 5 меньше потомка 6, меняем их местами

(6)

8

(5) 7

1 3 2 4

9

Выделяем рассматриваемые элементы!

(6)

(8)

5 7

1 3 2 4

9

Суперродитель 6 меньше потомка 8, меняем их местами

		<p>(8)</p> <p>(6)</p> <p>5 7</p> <p>1 3 2 4</p> <p>9</p> <hr/> <hr/> <p>Переносим максимум из корня, применяем слабую просейку:</p> <p>(8)</p> <p>6</p> <p>5 7</p> <p>1 3 2 (4)</p> <hr/> <hr/> <p>Переместили корень 8 и элемент из конца неотсортированной части 4</p> <p>Отсортированная часть массива: 8 9</p> <p>Неотсортированная часть массива: 4 6 5 7 1 3 2</p> <p>(4)</p> <p>6</p> <p>5 7</p> <p>1 3 2 (8)</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(4)</p> <p>6</p>
--	--	--

		<p>5 (7)</p> <p>1 3 2 8</p> <hr/> <hr/> <p>Суперродитель 4 меньше потомка 7, меняем их местами</p> <p>(7)</p> <p>6</p> <p>5 (4)</p> <p>1 3 2 8</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(7)</p> <p>(6)</p> <p>5 4</p> <p>1 3 2 8</p> <hr/> <hr/> <p>СупперРодитель больше и равен потомку, оставляем всё, как есть.</p> <p>Переносим максимум из корня, применяем слабую просейку:</p> <p>(7)</p> <p>6</p> <p>5 4</p> <p>1 3 (2)</p> <hr/> <hr/>
--	--	--

		<p>Переместили корень 7 и элемент из конца неотсортированной части 2</p> <p>Отсортированная часть массива: 7 8 9</p> <p>Неотсортированная часть массива: 2 6 5 4 1 3</p> <p>(2)</p> <p>6</p> <p>5 4</p> <p>1 3 (7)</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(2)</p> <p>6</p> <p>5 (4)</p> <p>1 3 7</p> <hr/> <hr/> <p>Суперродитель 2 меньше потомка 4, меняем их местами</p> <p>(4)</p> <p>6</p> <p>5 (2)</p> <p>1 3 7</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(4)</p> <p>(6)</p>
--	--	---

		<p>5 2</p> <p>1 3 7</p> <hr/> <hr/> <p>Суперродитель 4 меньше потомка 6, меняем их местами</p> <p>(6)</p> <p>(4)</p> <p>5 2</p> <p>1 3 7</p> <hr/> <hr/> <p>Переносим максимум из корня, применяем слабую просейку:</p> <p>(6)</p> <p>4</p> <p>5 2</p> <p>1 (3)</p> <hr/> <hr/> <p>Переместили корень 6 и элемент из конца неотсортированной части 3</p> <p>Отсортированная часть массива: 6 7 8 9</p> <p>Неотсортированная часть массива: 3 4 5 2 1</p> <p>(3)</p> <p>4</p> <p>5 2</p> <p>1 (6)</p>
--	--	--

Выделяем рассматриваемые элементы!

(3)

4

(5) 2

1 6

Суперродитель 3 меньше потомка 5, меняем их местами

(5)

4

(3) 2

1 6

Выделяем рассматриваемые элементы!

(5)

(4)

3 2

1 6

СупперРодитель больше и равен потомку, оставляем всё, как есть.

Переносим максимум из корня, применяем слабую просейку:

(5)

		<p>4</p> <p>3 2</p> <p>(1)</p> <hr/> <hr/> <p>Переместили корень 5 и элемент из конца неотсортированной части 1</p> <p>Отсортированная часть массива: 5 6 7 8 9</p> <p>Неотсортированная часть массива: 1 4 3 2</p> <p>(1)</p> <p>4</p> <p>3 2</p> <p>(5)</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(1)</p> <p>4</p> <p>(3) 2</p> <p>5</p> <hr/> <hr/> <p>Суперродитель 1 меньше потомка 3, меняем их местами</p> <p>(3)</p> <p>4</p> <p>(1) 2</p> <p>5</p>
--	--	--

Выделяем рассматриваемые элементы!

(3)

(4)

1 2

5

Суперродитель 3 меньше потомка 4, меняем их местами

(4)

(3)

1 2

5

Переносим максимум из корня, применяем слабую просейку:

(4)

3

1 (2)

Переместили корень 4 и элемент из конца неотсортированной части 2

Отсортированная часть массива: 4 5 6 7 8 9

Неотсортированная часть массива: 2 3 1

		<p>(2)</p> <p>3</p> <p>1 (4)</p> <hr/> <hr/> <p>Выделяем рассматриваемые элементы!</p> <p>(2)</p> <p>(3)</p> <p>1 4</p> <hr/> <hr/> <p>Суперродитель 2 меньше потомка 3, меняем их местами</p> <p>(3)</p> <p>(2)</p> <p>1 4</p> <hr/> <hr/> <p>Переносим максимум из корня, применяем слабую просейку:</p> <p>(3)</p> <p>2</p> <p>(1)</p> <hr/> <hr/> <p>Переместили корень 3 и элемент из конца неотсортированной части 1</p> <p>Отсортированная часть массива: 3 4 5 6 7 8 9</p> <p>Неотсортированная часть массива: 1 2</p>
--	--	--

(1)

2

(3)

Выделяем рассматриваемые элементы!

(1)

(2)

3

Суперродитель 1 меньше потомка 2, меняем их местами

(2)

(1)

3

(2)

(1)

Меняем местами корень 2 и следующий за ним элемент 1

(1)

(2)

		<p>В результате сортировки массив примет вид:</p> <p>Итоговый массив: 1 2 3 4 5 6 7 8 9</p> <p>Список команд:</p> <p>1.Ввести массив с клавиатуры</p> <p>2.Завершить работу программы</p> <p>Введите номер команды:</p>
3	10 51 89 12 70 25 41 96 22 35 76	Итоговый массив: 12 22 25 35 41 51 70 76 89 96
4	8 8 7 6 5 4 3 2 1	Итоговый массив: 1 2 3 4 5 6 7 8
5	5 -9 4 3 2 -7	Итоговый массив: -9 -7 2 3 4
6	14 37 -8 10 1 15 26 500 120 11 47 -29 - 55 0 17	Итоговый массив: -55 -29 -8 0 1 10 11 15 17 26 37 47 120 500
7	1 120	Итоговый массив: 120

5. ДЕМОНСТРАЦИЯ

В начале программы пользователю предложено небольшое меню, в котором можно выбрать из двух команд: запуск или завершение программы. Им соответствуют значения 1 и 2:

1.Ввести массив с клавиатуры

2.Завершить работу программы

Такая реализация позволяет несколько раз вводить массив с клавиатуры, не прерывая выполнения программы.

После выбора команды 1, пользователю предлагается ввести количество элементов сортируемого массива. После ввода количества элементов массива, предлагается ввести все элементы этого массива через пробел.

Далее с помощью команды вывода на экран *cout* продемонстрирован ход работы программы.

После выбора команды 2 программа тут же завершается, с её помощью пользователь может выбирать удобный момент для завершения программы.

ЗАКЛЮЧЕНИЕ

В результате выполнения работы была изучена сортировка методом слабой кучи. Была изучена структура слабой кучи, а также алгоритм её построения. Реализован алгоритм сортировки с помощью слабой кучи, а также визуализирована его работа.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Habr. URL: <https://habr.com/en/company/edison/blog/499786/>
2. https://en.wikipedia.org/wiki/weak_heap#weak-heap_sort

ПРИЛОЖЕНИЕ

ИСХОДНЫЙ КОД ПРОГРАММЫ.

Название файла main.cpp

```
#include <iostream>
#include <cstdlib>
#include <vector>
#include <algorithm>
#include <iterator>
#include <fstream>
#include <cstring>
#include <cmath>
#include <limits>
#include <cstring>
#include <cctype>

#define GETFLAG(r, x) ((r[(x) >> 3] >> ((x) & 7)) & 1) //если в
качестве "левого" потомка родителя
#define TOGGLEFLAG(r, x) (r[(x) >> 3] ^= 1 << ((x) & 7)) //Для
потомка переопределяем, порядок его потомков
//(кто "левый", а кто "правый")

using namespace std;

class WeakHeap {

public:

    vector<int> wheap;
    int size_of_heap = 0;
    unsigned char* r = nullptr;
    int s;
    WeakHeap() {}
    void DisplayArray();
    void DisplayHeap(int i, int j, int num);
    void DisplayHeap();
    void WeakHeapMerge(unsigned char* r, int i, int j, int num);
    void WeakHeapSort();
    WeakHeap* InputHeap();
    ~WeakHeap() {
        if (!wheap.empty())
            wheap.clear();
    }
};

//вычисляет логарифм от b по основанию a
```

```

double log(int a, int b)
{
    return log(b) / log(a);
}

void WeakHeap::DisplayHeap(int i_1, int j_1, int num) {

    if (i_1 == 0 || j_1 == 0)
        std::cout << "(" << wheap[0] << ")";
    else
        std::cout << wheap[0];

    cout << "\n";
    int new_size = size_of_heap - num;

    int depth = (int)log(2, new_size); //вычисляем глубину дерева
    if ((int)log(2, new_size) != pow(2, depth))
        depth += 1;

    int k = 0;
    double idt = depth * 6 / 4;

    for (int i = 0; i < depth; i++) {

        for (int iter = 0; iter < idt; iter++)
            cout << " ";
        idt = idt / 5 * 2;

        for (int j = 0; j < pow(2, i); j++) {
            if (k < new_size)
                if (k + 1 == i_1 || k + 1 == j_1) {
                    cout << "(" << wheap[k + 1] << ")" << " ";
                }
            else {
                cout << wheap[k + 1] << " ";
            }

            k++;
        }

        cout << "\n";
    }

    for (int it = 0; it < 65; it++)

```

```

        cout << "_";
        cout << "\n";
    }

void WeakHeap::DisplayHeap() {
    cout << wheap[0];
    cout << "\n";

    int depth = (int)log(2, size_of_heap); //вычисляем глубину
    дерева
    if ((int)log(2, size_of_heap) != pow(2, depth))
        depth += 1;

    int k = 0;
    double idt = depth * 2;

    for (int i = 0; i < depth; i++) {

        for (int iter = 0; iter < idt; iter++)
            cout << " ";
        idt = idt / 2;

        for (int j = 0; j < pow(2, i); j++) {
            if (k < size_of_heap - 1)
                cout << wheap[k + 1] << " ";
            k++;
        }
        cout << "\n";
    }
}

void WeakHeap::WeakHeapMerge(unsigned char* r, int i, int j, int
num) {

    cout << "Выделяем рассматриваемые элементы!\n";
    if (wheap[i] < wheap[j]) { //"Суперродитель" меньше потомка?
        //Для потомка переопределяем, порядок его потомков
        //(кто "левый", а кто "правый")

        TOGGLEFLAG(r, j);
        //Меняем значения "суперродителя" и потомка

        this->DisplayHeap(i, j, num);
        cout << "Суперродитель " << wheap[i] << " меньше потомка "
<< wheap[j] << ", меняем их местами\n";
        swap(wheap[i], wheap[j]);
        this->DisplayHeap(i, j, num);

    }
    else {
        this->DisplayHeap(i, j, num);
    }
}

```

```

        cout << "СупперРодитель больше и равен потомку, оставляем
всё, как есть.\n";
    }
}

void WeakHeap::WeakHeapSort() {
    int n = size_of_heap;
    int lef;
    int per;
    if (n > 1) {

        int i, j, x, y, Gparent;
        s = (n + 7) / 8;
        r = new unsigned char[s];

        //Массив для обозначения, какой у элемента
        //потомок "левый", а какой "правый"
        for (i = 0; i < n / 8; ++i)
            r[i] = 0;

        cout << "\n1 ЭТАП 1 ЭТАП 1 ЭТАП 1 ЭТАП\n";
        cout << "Построение первоначальной слабой кучи:\n";

        //Построение первоначальной слабой кучи
        for (i = n - 1; i > 0; --i) {
            j = i;
            //Поднимаемся на сколько возможно вверх,
            //если в качестве "левого" потомка родителя
            lef = GETFLAG(r, j >> 1);
            while ((j & 1) == lef) {
                j = j >> 1;
                lef = GETFLAG(r, j >> 1);
            }
            //И ещё на один уровень вверх как "правый" потомок
            родителя
            Gparent = j >> 1;
            //Слияние начального элемента, с которого
            //начали восхождение до "суперродителя"
            WeakHeapMerge(r, Gparent, i, 1);
        }

        //Перенос максимума из корня в конец -->
        //слабая просейка --> и всё по новой
        cout << "Слабая куча построена:\n";
        this->DisplayHeap();
        for (int it = 0; it < 65; it++)
            cout << " ";

        cout << "\n2 ЭТАП 2 ЭТАП 2 ЭТАП 2 ЭТАП\n";

        for (i = n - 1; i >= 2; --i) {

```



```

        cout << "Переносим максимум из корня, применяем слабую
просейку:\n";
        //Максимум отправляем в конец неотсортированной части
массива
        //Элемент из конца неотсортированной части попадает в
корень
        this->DisplayHeap(0, i, n - i);
        cout << "Переместили корень " << wheap[0] << " и
элемент из конца неотсортированной части " << wheap[i] << "\n";

        swap(wheap[0], wheap[i]);

        cout << "Отсортированная часть массива: ";
        for (int j = i; j < size_of_heap; j++)
            cout << wheap[j] << " ";
        cout << "\n";

        cout << "Неотсортированная часть массива: ";
        for (int j = 0; j < i; j++)
            cout << wheap[j] << " ";
        cout << "\n\n";

        this->DisplayHeap(0, i, n - i);
        x = 1;
        //Опускаемся жадно вниз по "левым" веткам
        lef = GETFLAG(r, x);
        while ((y = 2 * x + lef) < i) {
            x = y;
            lef = GETFLAG(r, x);
        }
        //Поднимаемся по "левой" ветке обратно до самого
вверха
        //попутно по дороге делаем слияние каждого узла с
корнем
        while (x > 0) {
            WeakHeapMerge(r, 0, x, n - i);
            x >>= 1;
        }
    }
    //Последнее действие - меняем местами корень
    //и следующий за ним элемент
    this->DisplayHeap(0, 1, n - 1);
    cout << "Меняем местами корень " << wheap[0] << " и
следующий за ним элемент " << wheap[1] << "\n";
    swap(wheap[0], wheap[1]);
    this->DisplayHeap(0, 1, n - 1);
    delete[] r;
}
}
void WeakHeap::DisplayArray()
{
    for (int i = 0; i < size_of_heap; i++)
        std::cout << wheap[i] << " ";

```

```

        std::cout << "\n\n";
    }

WeakHeap* WeakHeap::InputHeap() {
    int count;
    cin >> count;
    int elem;
    WeakHeap* wh = new WeakHeap();
    cout << "Введите через пробел элементы массива:\n";
    while (wh->size_of_heap != count) {
        cin >> elem;
        wh->wheap.push_back(elem);
        wh->size_of_heap++;
    }

    return wh;
}

int main()
{
    setlocale(LC_ALL, "rus");
    cout << "Программа для визуализации сортировки слабой
кучей.\n\n";
    cout << "На примере данной программы, можно увидеть, как
происходит сортировка слабой кучей.\n\n";
    int command;
    WeakHeap* wh = nullptr;
    int flag = 0;
    while (!flag) {
        cout << "Список команд:\n 1.Ввести массив с клавиатуры\n
2.Завершить работу программы\n\n";
        cout << "Введите номер команды: \n";
        cin >> command;

        if (command == 1) {
            cout << "Введите количество элементов массива:\n";
            wh = wh->InputHeap();
            if (wh) {
                wh->WeakHeapSort();
                cout << "\nВ результате сортировки массив примет
вид:\n";

                cout << "Итоговый массив: ";
                wh->DisplayArray();
                delete wh;
            }
        }
        else if (command == 2) {
            cout << "Работа программы завершена!\n";
            flag = 1;
        }
        else {

```

```
        cout << "Введите корректную команду! \n" << command <<
"\n";
    }
}

return 0;
}
```