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 40 страниц.

Дата выдачи задания: 31.10.2020

Дата сдачи реферата: 24.03.2021

Дата защиты реферата: 30.03.2021

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

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 поддерева. 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 поддерева 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.а. Меняем местами (левый ⇔ правый) поддеревья с потомками для узла, в котором находится текущий элемент.
>
> I.4.б. 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.а. Опускаемся из корня кучи по левым потомкам как можно ниже.
>
> II.3.б. Поднимаемся по левым потомкам обратно к корню кучи, сравнивая каждый левый потомок с корнем.
>
> II.3.в. Если элемент в корне меньше, чем очередной левый потомок, то:
>
> II.3.в.1. Меняем местами (левый ⇔ правый) поддеревья с потомками для узла, в котором находится текущий левый потомок.
>
> II.3.в.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, СТРУКТУР, 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_of_heap - количество элементов в куче.

int size_of_array - переменная для хранения размера кучи при её поэтапном выводе на экран.

int bit_array* - массив бит для хранения информации об обмене поддеревьями слабой кучи.

int s - размер массива бит.

int flag - переменная, для выбора режима вывода кучи на экран.

int flag2 - переменная, которая нужна для того, что элемент, переставший быть частью кучи не выводился на экран.

int flag4 - переменная, для корректного отображения текущего размера кучи.

int nsize - переменная, использующаяся для корректного отображения неполных бинарных деревьев.

int bit* – массив для дублирования массива бит, позволяет выводить кучу на экран, не изменяя состояние массива бит.

int mas* – вспомогательный массив, в котором хранится первоначальное состояние массива бит, на момент начала вывода кучи на экран.

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.

void DisplayHeap(int i_1, int j_1, int num);

Записываем элементы в кучув порядке, удобном для вывода на экран. Затем вычисляем глубину кучи при помощи функции log(2, size). Затем проходимся по элементам, выводим их в наглядном виде, в виде бинарного дерева.

int idt - переменная, для изменения отступов от левой границы консоли. Используется для наглядного отображения кучи на экран.

int_i, int_j - элементы, которые надо выделить.

int num - количество отсортированных элементов в массиве.

vector <int> res - переменная для хранения изначального состояния массива wheap, в конце метода массиву wheap присваивается его значение, чтобы куча продолжала правильно храниться в массиве.

num_elem_in_str - переменная, в которой хранится номер расположения текущего элемента в строке, во время вывода кучи на экран.

int current_size - переменная, которая используется для отображения текущего размера кучи при выводе на экран.

int flag3 - переменная для корректного выделения обмениваемого элемента, который является единственным потомком.

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 соответсвуют 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.

Функция *main()*

int command – переменная, в которую записывается введённая пользователем команда. С помощью неё реализовано простейшее меню программы, которое позволяет сортировать сколько угодно массивов, пока пользователь не решит завершить программу.

В функции main() создаётся объект класса *WeekHeep* wh. С его помощью происходит сортировка массива, введенного пользователем и демонстрация сортировки слабой кучи.

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 поддеревья 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 конецне 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 внеотсортированной части массива. Последний элемент с максимумом перестаётбыть 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 поддеревья 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 находитсямаксимальный элемент для оставшейся не отсортированной части массива.

Затем снова из корня кучи текущий максимальный элемент перемещаем вконец 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 59 52 40 29 87 16 30 86 45 49 50 22 92 17 60 91	Программа для визуализации сортировки слабой кучей. На примере данной программы, можно увидеть, как происходит сортировка слабой кучей. Список команд: 1.Ввести массив с клавиатуры 2.Завершить работу программы Введите номер команды: 1 Введите количество элементов массива:

		16 Введите через пробел элементы массива: 59 52 40 29 87 16 30 86 45 49 50 22 92 17 60 91 1 ЭТАП 1 ЭТАП 1 ЭТАП 1 ЭТАП 1 ЭТАП Построение первоначальной слабой кучи: Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 0 0 0 Так как бит предка 52 равен 0, его потомки должны быть на месте: Так как бит предка 40 равен 0, его потомки должны быть на месте: Так как бит предка 29 равен 0, его потомки должны быть на месте: Так как бит предка 87 равен 0, его потомки должны быть на месте: Так как бит предка 16 равен 0, его потомки должны быть на месте: Так как бит предка 30 равен 0, его потомки должны быть на месте: Так как бит предка 86 равен 0, его потомки должны быть на месте: 59 52 40 29 87 16 30 (86) 45 49 50 22 92 17 60 (91) _____ _____

		Суперродитель 86 меньше потомка 91, меняем их местами. Текущее состояние массива бит: 0 0 0 0 0 0 0 Так как бит предка 52 равен 0, его потомки должны быть на месте: Так как бит предка 40 равен 0, его потомки должны быть на месте: Так как бит предка 29 равен 0, его потомки должны быть на месте: Так как бит предка 87 равен 0, его потомки должны быть на месте: Так как бит предка 16 равен 0, его потомки должны быть на месте: Так как бит предка 30 равен 0, его потомки должны быть на месте: Так как бит предка 91 равен 0, его потомки должны быть на месте: 59 52 40 29 87 16 30 (91) 45 49 50 22 92 17 60 (86) _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 0 0 0 Так как бит предка 52 равен 0, его потомки должны быть на месте: Так как бит предка 40 равен 0, его потомки

		должны быть на месте: Так как бит предка 29 равен 0, его потомки должны быть на месте: Так как бит предка 87 равен 0, его потомки должны быть на месте: Так как бит предка 16 равен 0, его потомки должны быть на месте: Так как бит предка 30 равен 0, его потомки должны быть на месте: Так как бит предка 91 равен 0, его потомки должны быть на месте: 59 52 40 (29) 87 16 30 91 45 49 50 22 92 17 (60) 86 _____ _____ Суперродитель 29 меньше потомка 60, меняем их местами. Текущее состояние массива бит: 0 0 0 0 0 0 0 Так как бит предка 52 равен 0, его потомки должны быть на месте: Так как бит предка 40 равен 0, его потомки должны быть на месте: Так как бит предка 60 равен 0, его потомки должны быть на месте: Так как бит предка 87 равен 0, его потомки должны быть на месте:

		Так как бит предка 16 равен 0, его потомки должны быть на месте: Так как бит предка 30 равен 0, его потомки должны быть на месте: Так как бит предка 91 равен 0, его потомки должны быть на месте: 59 52 40 (60) 87 16 30 91 45 49 50 22 92 17 (29) 86 _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 0 0 0 Так как бит предка 52 равен 0, его потомки должны быть на месте: Так как бит предка 40 равен 0, его потомки должны быть на месте: Так как бит предка 60 равен 0, его потомки должны быть на месте: Так как бит предка 87 равен 0, его потомки должны быть на месте: Так как бит предка 16 равен 0, его потомки должны быть на месте: Так как бит предка 30 равен 0, его потомки должны быть на месте: Так как бит предка 91 равен 0, его потомки должны быть на месте:

		59 52 40 60 87 16 (30) 91 45 49 50 22 92 (17) 29 86 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 0 0 0 Так как бит предка 52 равен 0, его потомки должны быть на месте: Так как бит предка 40 равен 0, его потомки должны быть на месте: Так как бит предка 60 равен 0, его потомки должны быть на месте: Так как бит предка 87 равен 0, его потомки должны быть на месте: Так как бит предка 16 равен 0, его потомки должны быть на месте: Так как бит предка 30 равен 0, его потомки должны быть на месте: Так как бит предка 91 равен 0, его потомки должны быть на месте: 59 (52) 40 60 87 16 30 91

		45 49 50 22 (92) 17 29 86 _____ _____ Суперродитель 52 меньше потомка 92, меняем их местами. Текущее состояние массива бит: 0 0 0 0 0 0 0 Так как бит предка 92 равен 0, его потомки должны быть на месте: Так как бит предка 40 равен 0, его потомки должны быть на месте: Так как бит предка 60 равен 0, его потомки должны быть на месте: Так как бит предка 87 равен 0, его потомки должны быть на месте: Так как бит предка 16 равен 0, его потомки должны быть на месте: Так как бит предка 30 равен 0, его потомки должны быть на месте: Так как бит предка 91 равен 0, его потомки должны быть на месте: 59 (92) 40 60 87 16 30 91 45 49 50 22 (52) 17 29 86 _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 0 0 0

		Так как бит предка 92 равен 0, его потомки должны быть на месте: Так как бит предка 40 равен 0, его потомки должны быть на месте: Так как бит предка 60 равен 0, его потомки должны быть на месте: Так как бит предка 87 равен 0, его потомки должны быть на месте: Так как бит предка 16 равен 0, его потомки должны быть на месте: Так как бит предка 30 равен 0, его потомки должны быть на месте: Так как бит предка 91 равен 0, его потомки должны быть на месте: 59 92 40 60 87 (16) 30 91 45 49 50 (22) 52 17 29 86 _____ _____ Суперродитель 16 меньше потомка 22, меняем их местами. Текущее состояние массива бит: 0 0 0 0 0 0 0 Так как бит предка 92 равен 0, его потомки должны быть на месте: Так как бит предка 40 равен 0, его потомки должны быть на месте: Так как бит предка 60 равен 0, его потомки

		должны быть на месте: Так как бит предка 87 равен 0, его потомки должны быть на месте: Так как бит предка 22 равен 0, его потомки должны быть на месте: Так как бит предка 30 равен 0, его потомки должны быть на месте: Так как бит предка 91 равен 0, его потомки должны быть на месте: 59 92 40 60 87 (22) 30 91 45 49 50 (16) 52 17 29 86 _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 0 0 0 Так как бит предка 92 равен 0, его потомки должны быть на месте: Так как бит предка 40 равен 0, его потомки должны быть на месте: Так как бит предка 60 равен 0, его потомки должны быть на месте: Так как бит предка 87 равен 0, его потомки должны быть на месте: Так как бит предка 22 равен 0, его потомки должны быть на месте: Так как бит предка 30 равен 0, его потомки

		должны быть на месте: Так как бит предка 91 равен 0, его потомки должны быть на месте: 59 92 (40) 60 87 22 30 91 45 49 (50) 16 52 17 29 86 _____ _____ Суперродитель 40 меньше потомка 50, меняем их местами. Текущее состояние массива бит: 0 0 0 0 0 0 0 Так как бит предка 92 равен 0, его потомки должны быть на месте: Так как бит предка 50 равен 0, его потомки должны быть на месте: Так как бит предка 60 равен 0, его потомки должны быть на месте: Так как бит предка 87 равен 0, его потомки должны быть на месте: Так как бит предка 22 равен 0, его потомки должны быть на месте: Так как бит предка 30 равен 0, его потомки должны быть на месте: Так как бит предка 91 равен 0, его потомки должны быть на месте: 59 92

		(50) 60 87 22 30 91 45 49 (40) 16 52 17 29 86 _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 0 0 0 Так как бит предка 92 равен 0, его потомки должны быть на месте: Так как бит предка 50 равен 0, его потомки должны быть на месте: Так как бит предка 60 равен 0, его потомки должны быть на месте: Так как бит предка 87 равен 0, его потомки должны быть на месте: Так как бит предка 22 равен 0, его потомки должны быть на месте: Так как бит предка 30 равен 0, его потомки должны быть на месте: Так как бит предка 91 равен 0, его потомки должны быть на месте: 59 92 50 60 (87) 22 30 91 45 (49) 40 16 52 17 29 86 _____ _____ Суперродитель больше или равен потомка,

		оставляем всё, как есть. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 0 0 0 Так как бит предка 92 равен 0, его потомки должны быть на месте: Так как бит предка 50 равен 0, его потомки должны быть на месте: Так как бит предка 60 равен 0, его потомки должны быть на месте: Так как бит предка 87 равен 0, его потомки должны быть на месте: Так как бит предка 22 равен 0, его потомки должны быть на месте: Так как бит предка 30 равен 0, его потомки должны быть на месте: Так как бит предка 91 равен 0, его потомки должны быть на месте: (59) 92 50 60 87 22 30 91 (45) 49 40 16 52 17 29 86 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 0 0 0 Так как бит предка 92 равен 0, его потомки

		должны быть на месте: Так как бит предка 50 равен 0, его потомки должны быть на месте: Так как бит предка 60 равен 0, его потомки должны быть на месте: Так как бит предка 87 равен 0, его потомки должны быть на месте: Так как бит предка 22 равен 0, его потомки должны быть на месте: Так как бит предка 30 равен 0, его потомки должны быть на месте: Так как бит предка 91 равен 0, его потомки должны быть на месте: 59 92 50 (60) 87 22 30 (91) 45 49 40 16 52 17 29 86 _____ _____ Суперродитель 60 меньше потомка 91, меняем их местами. Меняем местами поддеревья с потомками для узла, в котором находится текущий элемент. Текущее состояние массива бит: 0 0 0 0 0 0 1 Так как бит предка 92 равен 0, его потомки должны быть на месте: Так как бит предка 50 равен 0, его потомки

		должны быть на месте: Так как бит предка 91 равен 0, его потомки должны быть на месте: Так как бит предка 87 равен 0, его потомки должны быть на месте: Так как бит предка 22 равен 0, его потомки должны быть на месте: Так как бит предка 30 равен 0, его потомки должны быть на месте: Так как бит предка 60 равен 1, его потомки должны быть переставлены: 59 92 50 (91) 87 22 30 (60) 45 49 40 16 52 17 86 29 _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 0 0 1 Так как бит предка 92 равен 0, его потомки должны быть на месте: Так как бит предка 50 равен 0, его потомки должны быть на месте: Так как бит предка 91 равен 0, его потомки должны быть на месте: Так как бит предка 87 равен 0, его потомки должны быть на месте: Так как бит предка 22 равен 0, его потомки

		должны быть на месте: Так как бит предка 30 равен 0, его потомки должны быть на месте: Так как бит предка 60 равен 1, его потомки должны быть переставлены: 59 (92) 50 91 87 22 (30) 60 45 49 40 16 52 17 86 29 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 0 0 1 Так как бит предка 92 равен 0, его потомки должны быть на месте: Так как бит предка 50 равен 0, его потомки должны быть на месте: Так как бит предка 91 равен 0, его потомки должны быть на месте: Так как бит предка 87 равен 0, его потомки должны быть на месте: Так как бит предка 22 равен 0, его потомки должны быть на месте: Так как бит предка 30 равен 0, его потомки должны быть на месте: Так как бит предка 60 равен 1, его потомки

		должны быть переставлены: 59 92 (50) 91 87 (22) 30 60 45 49 40 16 52 17 86 29 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 0 0 1 Так как бит предка 92 равен 0, его потомки должны быть на месте: Так как бит предка 50 равен 0, его потомки должны быть на месте: Так как бит предка 91 равен 0, его потомки должны быть на месте: Так как бит предка 87 равен 0, его потомки должны быть на месте: Так как бит предка 22 равен 0, его потомки должны быть на месте: Так как бит предка 30 равен 0, его потомки должны быть на месте: Так как бит предка 60 равен 1, его потомки должны быть переставлены: (59) 92 50 91

		(87) 22 30 60 45 49 40 16 52 17 86 29 _____ _____ Суперродитель 59 меньше потомка 87, меняем их местами. Меняем местами поддеревья с потомками для узла, в котором находится текущий элемент. Текущее состояние массива бит: 0 0 0 1 0 0 1 Так как бит предка 92 равен 0, его потомки должны быть на месте: Так как бит предка 50 равен 0, его потомки должны быть на месте: Так как бит предка 91 равен 0, его потомки должны быть на месте: Так как бит предка 59 равен 1, его потомки должны быть переставлены: Так как бит предка 22 равен 0, его потомки должны быть на месте: Так как бит предка 30 равен 0, его потомки должны быть на месте: Так как бит предка 60 равен 1, его потомки должны быть переставлены: (87) 92 50 91 (59) 22 30 60 49 45 40 16 52 17 86 29

		--- --- Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 1 0 0 1 Так как бит предка 92 равен 0, его потомки должны быть на месте: Так как бит предка 50 равен 0, его потомки должны быть на месте: Так как бит предка 91 равен 0, его потомки должны быть на месте: Так как бит предка 59 равен 1, его потомки должны быть переставлены: Так как бит предка 22 равен 0, его потомки должны быть на месте: Так как бит предка 30 равен 0, его потомки должны быть на месте: Так как бит предка 60 равен 1, его потомки должны быть переставлены: 87 (92) 50 (91) 59 22 30 60 49 45 40 16 52 17 86 29 --- --- Суперродитель больше или равен потомка, оставляем всё, как есть. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 1 0 0 1

		Так как бит предка 92 равен 0, его потомки должны быть на месте: Так как бит предка 50 равен 0, его потомки должны быть на месте: Так как бит предка 91 равен 0, его потомки должны быть на месте: Так как бит предка 59 равен 1, его потомки должны быть переставлены: Так как бит предка 22 равен 0, его потомки должны быть на месте: Так как бит предка 30 равен 0, его потомки должны быть на месте: Так как бит предка 60 равен 1, его потомки должны быть переставлены: (87) 92 (50) 91 59 22 30 60 49 45 40 16 52 17 86 29 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 1 0 0 1 Так как бит предка 92 равен 0, его потомки должны быть на месте: Так как бит предка 50 равен 0, его потомки должны быть на месте:

		Так как бит предка 91 равен 0, его потомки должны быть на месте: Так как бит предка 59 равен 1, его потомки должны быть переставлены: Так как бит предка 22 равен 0, его потомки должны быть на месте: Так как бит предка 30 равен 0, его потомки должны быть на месте: Так как бит предка 60 равен 1, его потомки должны быть переставлены: (87) (92) 50 91 59 22 30 60 49 45 40 16 52 17 86 29 _____ _____ Суперродитель 87 меньше потомка 92, меняем их местами. Меняем местами поддеревья с потомками для узла, в котором находится текущий элемент. Текущее состояние массива бит: 1 0 0 1 0 0 1 Так как бит предка 87 равен 1, его потомки должны быть переставлены: Так как бит предка 91 равен 0, его потомки должны быть на месте: Так как бит предка 50 равен 0, его потомки должны быть на месте:

		Так как бит предка 30 равен 0, его потомки должны быть на месте: Так как бит предка 60 равен 1, его потомки должны быть переставлены: Так как бит предка 59 равен 1, его потомки должны быть переставлены: Так как бит предка 22 равен 0, его потомки должны быть на месте: (92) (87) 91 50 30 60 59 22 52 17 86 29 49 45 40 16 _____ _____ Теперь строим итоговую слабую кучу: На данном этапе имеем: 1|0|0|1|0|0|1| массив бит 92|87|50|91|59|22|30|60|45|49|40|16|52|17|29|86| массив чисел Вывод итоговой слабой кучи с использованием массива: Текущее состояние массива бит: 1 0 0 1 0 0 1 Так как бит предка 87 равен 1, его потомки должны быть переставлены: Так как бит предка 91 равен 0, его потомки должны быть на месте:

		Так как бит предка 50 равен 0, его потомки должны быть на месте: Так как бит предка 30 равен 0, его потомки должны быть на месте: Так как бит предка 60 равен 1, его потомки должны быть переставлены: Так как бит предка 59 равен 1, его потомки должны быть переставлены: Так как бит предка 22 равен 0, его потомки должны быть на месте: 92 87 91 50 30 60 59 22 52 17 86 29 49 45 40 16 _____ _____ ^^^^^^^^^^^^^^^^^^^^^^ Получившаяся слабая куча выведена на экран! 92|87|50|91|59|22|30|60|45|49|40|16|52|17|29|86| массив чисел 2 ЭТАП 2 ЭТАП 2 ЭТАП 2 ЭТАП 2 ЭТАП На 2 этапе из корня кучи текущий максимальный элемент перемещаем в конец неотсортированной части массива, после чего восстанавливаем слабую кучу Переносим максимум из корня, применяем

		слабую просейку: Текущее состояние массива бит: 1 0 0 1 0 0 1 Текущий размер кучи: 16 (92) 87 91 50 30 60 59 22 52 17 (86) 29 49 45 40 16 _____ _____ Переместили корень 92 и элемент из конца неотсортированной части 86 Отсортированная часть массива: 92 Неотсортированная часть массива: 86 87 50 91 59 22 30 60 45 49 40 16 52 17 29 Текущее состояние массива бит: 1 0 0 1 0 0 1 Текущий размер кучи: 16 (86) 87 91 50 30 60 59 22 52 17 (92) 29 49 45 40 16 _____ _____ Последний элемент с максимумом перестаёт быть узлом слабой кучи.

		Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 0 0 1 0 0 1 Текущий размер кучи: 15 (86) 87 91 50 30 60 59 22 (52) 17 29 49 45 40 16 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 0 0 1 0 0 1 Текущий размер кучи: 15 (86) 87 91 50 (30) 60 59 22 52 17 29 49 45 40 16 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 0 0 1 0 0 1

		Текущий размер кучи: 15 (86) 87 (91) 50 30 60 59 22 52 17 29 49 45 40 16 _____ _____ Суперродитель 86 меньше потомка 91, меняем их местами. Меняем местами поддеревья с потомками для узла, в котором находится текущий левый потомок. Текущее состояние массива бит: 1 0 1 1 0 0 1 Текущий размер кучи: 15 (91) 87 (86) 50 60 30 59 22 29 52 17 49 45 40 16 _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 0 1 1 0 0 1 Текущий размер кучи: 15 (91)

		(87) 86 50 60 30 59 22 29 52 17 49 45 40 16 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Переносим максимум из корня, применяем слабую просейку: Текущее состояние массива бит: 1 0 1 1 0 0 1 Текущий размер кучи: 15 (91) 87 86 50 60 30 59 22 (29) 52 17 49 45 40 16 _____ _____ Переместили корень 91 и элемент из конца неотсортированной части 29 Отсортированная часть массива: 91 92 Неотсортированная часть массива: 29 87 50 86 59 22 30 60 45 49 40 16 52 17 Текущее состояние массива бит: 1 0 1 1 0 0 1 Текущий размер кучи: 15

		(29) 87 86 50 60 30 59 22 (91) 52 17 49 45 40 16 _____ _____ Последний элемент с максимумом перестаёт быть узлом слабой кучи. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 0 1 1 0 0 1 Текущий размер кучи: 14 (29) 87 86 50 (60) 30 59 22 52 17 49 45 40 16 _____ _____ Суперродитель 29 меньше потомка 60, меняем их местами. Текущее состояние массива бит: 1 0 1 1 0 0 0 Текущий размер кучи: 14 (60) 87 86 50

		(29) 30 59 22 52 17 49 45 40 16 _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 0 1 1 0 0 0 Текущий размер кучи: 14 (60) 87 (86) 50 29 30 59 22 52 17 49 45 40 16 _____ _____ Суперродитель 60 меньше потомка 86, меняем их местами. Меняем местами поддеревья с потомками для узла, в котором находится текущий левый потомок. Текущее состояние массива бит: 1 0 0 1 0 0 0 Текущий размер кучи: 14 (86) 87 (60) 50 30 29 59 22 52 17 49 45 40 16 _____

		_____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 0 0 1 0 0 0 Текущий размер кучи: 14 (86) (87) 60 50 30 29 59 22 52 17 49 45 40 16 _____ _____ Суперродитель 86 меньше потомка 87, меняем их местами. Меняем местами поддеревья с потомками для узла, в котором находится текущий левый потомок. Текущее состояние массива бит: 0 0 0 1 0 0 0 Текущий размер кучи: 14 (87) (86) 50 60 59 22 30 29 49 45 40 16 52 17 _____ _____ Переносим максимум из корня, применяем слабую просейку:

		Текущее состояние массива бит: 0 0 0 1 0 0 0 Текущий размер кучи: 14 (87) 86 50 60 59 22 30 29 49 45 40 16 52 (17) _____ _____ Переместили корень 87 и элемент из конца неотсортированной части 17 Отсортированная часть массива: 87 91 92 Неотсортированная часть массива: 17 86 50 60 59 22 30 29 45 49 40 16 52 Текущее состояние массива бит: 0 0 0 1 0 0 0 Текущий размер кучи: 14 (17) 86 50 60 59 22 30 29 49 45 40 16 52 (87) _____ _____ Последний элемент с максимумом перестаёт быть узлом слабой кучи.

		Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 1 0 0 0 Текущий размер кучи: 13 (17) 86 50 60 59 22 30 29 (49) 45 40 16 52 _____ _____ Суперродитель 17 меньше потомка 49, меняем их местами. Текущее состояние массива бит: 0 0 0 1 0 0 0 Текущий размер кучи: 13 (49) 86 50 60 59 22 30 29 (17) 45 40 16 52 _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 1 0 0 0 Текущий размер кучи: 13 (49) 86

		50 60 (59) 22 30 29 17 45 40 16 52 _____ _____ Суперродитель 49 меньше потомка 59, меняем их местами. Меняем местами поддеревья с потомками для узла, в котором находится текущий левый потомок. Текущее состояние массива бит: 0 0 0 0 0 0 0 Текущий размер кучи: 13 (59) 86 50 60 (49) 22 30 29 45 17 40 16 52 _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 0 0 0 Текущий размер кучи: 13 (59) 86 (50) 60 49 22 30 29 45 17 40 16 52

		_____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 0 0 0 Текущий размер кучи: 13 (59) (86) 50 60 49 22 30 29 45 17 40 16 52 _____ _____ Суперродитель 59 меньше потомка 86, меняем их местами. Меняем местами поддеревья с потомками для узла, в котором находится текущий левый потомок. Текущее состояние массива бит: 1 0 0 0 0 0 0 Текущий размер кучи: 13 (86) (59) 60 50 30 29 49 22 52 45 17 40 16 _____

		_____ Переносим максимум из корня, применяем слабую просейку: Текущее состояние массива бит: 1 0 0 0 0 0 0 Текущий размер кучи: 13 (86) 59 60 50 30 29 49 22 (52) 45 17 40 16 _____ _____ Переместили корень 86 и элемент из конца неотсортированной части 52 Отсортированная часть массива: 86 87 91 92 Неотсортированная часть массива: 52 59 50 60 49 22 30 29 45 17 40 16 Текущее состояние массива бит: 1 0 0 0 0 0 0 Текущий размер кучи: 13 (52) 59 60 50 30 29 49 22 (86) 45 17 40 16 _____ _____

		Последний элемент с максимумом перестаёт быть узлом слабой кучи. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 0 0 0 0 0 0 Текущий размер кучи: 12 (52) 59 60 50 (30) 29 49 22 45 17 40 16 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 0 0 0 0 0 0 Текущий размер кучи: 12 (52) 59 (60) 50 30 29 49 22 45 17 40 16 _____ _____ Суперродитель 52 меньше потомка 60, меняем их местами.

		Меняем местами поддеревья с потомками для узла, в котором находится текущий левый потомок. Текущее состояние массива бит: 1 0 1 0 0 0 0 Текущий размер кучи: 12 (60) 59 (52) 50 29 30 49 22 45 17 40 16 _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 0 1 0 0 0 0 Текущий размер кучи: 12 (60) (59) 52 50 29 30 49 22 45 17 40 16 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Переносим максимум из корня, применяем слабую просейку: Текущее состояние массива бит: 1 0 1 0 0 0 0

		Текущий размер кучи: 12 (60) 59 52 50 29 30 49 22 45 17 40 (16) _____ _____ Переместили корень 60 и элемент из конца неотсортированной части 16 Отсортированная часть массива: 60 86 87 91 92 Неотсортированная часть массива: 16 59 50 52 49 22 30 29 45 17 40 Текущее состояние массива бит: 1 0 1 0 0 0 0 Текущий размер кучи: 12 (16) 59 52 50 29 30 49 22 45 17 40 (60) _____ _____ Последний элемент с максимумом перестаёт быть узлом слабой кучи.

		Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 0 1 0 0 0 0 Текущий размер кучи: 11 (16) 59 52 50 (29) 30 49 22 45 17 40 _____ _____ Суперродитель 16 меньше потомка 29, меняем их местами. Текущее состояние массива бит: 1 0 1 0 0 0 1 Текущий размер кучи: 11 (29) 59 52 50 (16) 30 49 22 45 17 40 _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 0 1 0 0 0 1 Текущий размер кучи: 11 (29) 59

		(52) 50 16 30 49 22 45 17 40 _____ _____ Суперродитель 29 меньше потомка 52, меняем их местами. Меняем местами поддеревья с потомками для узла, в котором находится текущий левый потомок. Текущее состояние массива бит: 1 0 0 0 0 0 1 Текущий размер кучи: 11 (52) 59 (29) 50 30 16 49 22 45 17 40 _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 0 0 0 0 0 1 Текущий размер кучи: 11 (52) (59) 29 50 30 16 49 22 45 17 40

		_____ _____ Суперродитель 52 меньше потомка 59, меняем их местами. Меняем местами поддеревья с потомками для узла, в котором находится текущий левый потомок. Текущее состояние массива бит: 0 0 0 0 0 0 1 Текущий размер кучи: 11 (59) (52) 50 29 49 22 30 16 45 17 40 _____ _____ Переносим максимум из корня, применяем слабую просейку: Текущее состояние массива бит: 0 0 0 0 0 0 1 Текущий размер кучи: 11 (59) 52 50 29 49 22 30 16 45 17 (40) _____ _____

		Переместили корень 59 и элемент из конца неотсортированной части 40 Отсортированная часть массива: 59 60 86 87 91 92 Неотсортированная часть массива: 40 52 50 29 49 22 30 16 45 17 Текущее состояние массива бит: 0 0 0 0 0 0 1 Текущий размер кучи: 11 (40) 52 50 29 49 22 30 16 45 17 (59) _____ _____ Последний элемент с максимумом перестаёт быть узлом слабой кучи. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 0 0 1 Текущий размер кучи: 10 (40) 52 50 29 49 22 30 16 (45) 17

		--- --- Суперродитель 40 меньше потомка 45, меняем их местами. Текущее состояние массива бит: 0 0 0 0 0 0 1 Текущий размер кучи: 10 (45) 52 50 29 49 22 30 16 (40) 17 --- --- Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 0 0 1 Текущий размер кучи: 10 (45) 52 50 29 (49) 22 30 16 40 17 --- --- Суперродитель 45 меньше потомка 49, меняем их местами. Меняем местами поддеревья с потомками для узла,

		в котором находится текущий левый потомок. Текущее состояние массива бит: 0 0 0 1 0 0 1 Текущий размер кучи: 10 (49) 52 50 29 (45) 22 30 16 17 40 _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 1 0 0 1 Текущий размер кучи: 10 (49) 52 (50) 29 45 22 30 16 17 40 _____ _____ Суперродитель 49 меньше потомка 50, меняем их местами. Меняем местами поддеревья с потомками для узла, в котором находится текущий левый потомок. Текущее состояние массива бит: 0 1 0 1 0 0 1 Текущий размер кучи: 10

		(50) 52 (49) 29 22 45 30 16 17 40 _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 1 0 1 0 0 1 Текущий размер кучи: 10 (50) (52) 49 29 22 45 30 16 17 40 _____ _____ Суперродитель 50 меньше потомка 52, меняем их местами. Меняем местами поддеревья с потомками для узла, в котором находится текущий левый потомок. Текущее состояние массива бит: 1 1 0 1 0 0 1 Текущий размер кучи: 10 (52) (50)

		29 49 30 16 22 45 17 40 _____ _____ Переносим максимум из корня, применяем слабую просейку: Текущее состояние массива бит: 1 1 0 1 0 0 1 Текущий размер кучи: 10 (52) 50 29 49 30 16 22 45 (17) 40 _____ _____ Переместили корень 52 и элемент из конца неотсортированной части 17 Отсортированная часть массива: 52 59 60 86 87 91 92 Неотсортированная часть массива: 17 50 49 29 45 22 30 16 40 Текущее состояние массива бит: 1 1 0 1 0 0 1 Текущий размер кучи: 10 (17) 50

		29 49 30 16 22 45 (52) 40 _____ _____ Последний элемент с максимумом перестаёт быть узлом слабой кучи. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 1 0 1 0 0 1 Текущий размер кучи: 9 (17) 50 29 49 (30) 16 22 45 40 _____ _____ Суперродитель 17 меньше потомка 30, меняем их местами. Текущее состояние массива бит: 1 1 0 1 0 1 1 Текущий размер кучи: 9 (30) 50 29 49 (17) 16 22 45 40

		--- --- Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 1 0 1 0 1 1 Текущий размер кучи: 9 (30) 50 (29) 49 17 16 22 45 40 --- --- Суперродитель больше или равен потомка, оставляем всё, как есть. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 1 0 1 0 1 1 Текущий размер кучи: 9 (30) (50) 29 49 17 16 22 45 40 --- --- Суперродитель 30 меньше потомка 50, меняем их местами. Меняем местами поддеревья с потомками для

		узла, в котором находится текущий левый потомок. Текущее состояние массива бит: 0 1 0 1 0 1 1 Текущий размер кучи: 9 (50) (30) 49 29 22 45 17 16 40 _____ _____ Переносим максимум из корня, применяем слабую просейку: Текущее состояние массива бит: 0 1 0 1 0 1 1 Текущий размер кучи: 9 (50) 30 49 29 22 45 17 16 (40) _____ _____ Переместили корень 50 и элемент из конца неотсортированной части 40 Отсортированная часть массива: 50 52 59 60 86 87 91 92 Неотсортированная часть массива: 40 30 49 29

		45 22 17 16 Текущее состояние массива бит: 0 1 0 1 0 1 1 Текущий размер кучи: 9 (40) 30 49 29 22 45 17 16 (50) _____ _____ Последний элемент с максимумом перестаёт быть узлом слабой кучи. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 1 0 1 0 1 1 Текущий размер кучи: 8 (40) 30 49 29 (22) 45 17 16 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Выделяем рассматриваемые элементы!

		Текущее состояние массива бит: 0 1 0 1 0 1 1 Текущий размер кучи: 8 (40) 30 (49) 29 22 45 17 16 _____ _____ Суперродитель 40 меньше потомка 49, меняем их местами. Меняем местами поддеревья с потомками для узла, в котором находится текущий левый потомок. Текущее состояние массива бит: 0 0 0 1 0 1 1 Текущий размер кучи: 8 (49) 30 (40) 29 45 22 17 16 _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 1 0 1 1 Текущий размер кучи: 8

		(49) (30) 40 29 45 22 17 16 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Переносим максимум из корня, применяем слабую просейку: Текущее состояние массива бит: 0 0 0 1 0 1 1 Текущий размер кучи: 8 (49) 30 40 29 45 22 17 (16) _____ _____ Переместили корень 49 и элемент из конца неотсортированной части 16 Отсортированная часть массива: 49 50 52 59 60 86 87 91 92 Неотсортированная часть массива: 16 30 40 29 45 22 17 Текущее состояние массива бит: 0 0 0 1 0 1 1

		Текущий размер кучи: 8 (16) 30 40 29 45 22 17 (49) _____ _____ Последний элемент с максимумом перестаёт быть узлом слабой кучи. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 1 0 1 1 Текущий размер кучи: 7 (16) 30 40 29 (45) 22 17 _____ _____ Суперродитель 16 меньше потомка 45, меняем их местами. Текущее состояние массива бит: 0 0 0 0 0 1 1 Текущий размер кучи: 7 (45)

		30 40 29 (16) 22 17 _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 0 1 1 Текущий размер кучи: 7 (45) 30 (40) 29 16 22 17 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 0 1 1 Текущий размер кучи: 7 (45) (30) 40 29 16 22 17 _____

		_____ Суперродитель больше или равен потомка, оставляем всё, как есть. Переносим максимум из корня, применяем слабую просейку: Текущее состояние массива бит: 0 0 0 0 0 1 1 Текущий размер кучи: 7 (45) 30 40 29 16 22 (17) _____ _____ Переместили корень 45 и элемент из конца неотсортированной части 17 Отсортированная часть массива: 45 49 50 52 59 60 86 87 91 92 Неотсортированная часть массива: 17 30 40 29 16 22 Текущее состояние массива бит: 0 0 0 0 0 1 1 Текущий размер кучи: 7 (17) 30 40 29 16 22 (45)

		_____ _____ Последний элемент с максимумом перестаёт быть узлом слабой кучи. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 0 1 1 Текущий размер кучи: 6 (17) 30 40 29 (16) 22 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 0 1 1 Текущий размер кучи: 6 (17) 30 (40) 29 16 22 _____

		_____ Суперродитель 17 меньше потомка 40, меняем их местами. Меняем местами поддеревья с потомками для узла, в котором находится текущий левый потомок. Текущее состояние массива бит: 0 1 0 0 0 1 1 Текущий размер кучи: 6 (40) 30 (17) 29 22 16 _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 1 0 0 0 1 1 Текущий размер кучи: 6 (40) (30) 17 29 22 16 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть.

		Переносим максимум из корня, применяем слабую просейку: Текущее состояние массива бит: 0 1 0 0 0 1 1 Текущий размер кучи: 6 (40) 30 17 29 (22) 16 _____ _____ Переместили корень 40 и элемент из конца неотсортированной части 22 Отсортированная часть массива: 40 45 49 50 52 59 60 86 87 91 92 Неотсортированная часть массива: 22 30 17 29 16 Текущее состояние массива бит: 0 1 0 0 0 1 1 Текущий размер кучи: 6 (22) 30 17 29 (40) 16 _____ _____

		Последний элемент с максимумом перестаёт быть узлом слабой кучи. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 1 0 0 0 1 1 Текущий размер кучи: 5 (22) 30 (17) 29 16 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 1 0 0 0 1 1 Текущий размер кучи: 5 (22) (30) 17 29 16 _____ _____ Суперродитель 22 меньше потомка 30, меняем их местами.

		Меняем местами поддеревья с потомками для узла, в котором находится текущий левый потомок. Текущее состояние массива бит: 1 1 0 0 0 1 1 Текущий размер кучи: 5 (30) (22) 29 17 16 _____ _____ Переносим максимум из корня, применяем слабую просейку: Текущее состояние массива бит: 1 1 0 0 0 1 1 Текущий размер кучи: 5 (30) 22 29 17 (16) _____ _____ Переместили корень 30 и элемент из конца неотсортированной части 16 Отсортированная часть массива: 30 40 45 49 50 52 59 60 86 87 91 92

		Неотсортированная часть массива: 16 22 17 29 Текущее состояние массива бит: 1 1 0 0 0 1 1 Текущий размер кучи: 5 (16) 22 29 17 (30) _____ _____ Последний элемент с максимумом перестаёт быть узлом слабой кучи. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 1 0 0 0 1 1 Текущий размер кучи: 4 (16) 22 (29) 17 _____ _____ Суперродитель 16 меньше потомка 29, меняем их местами. Текущее состояние массива бит: 1 1 1 0 0 1 1

		Текущий размер кучи: 4 (29) 22 (16) 17 _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 1 1 0 0 1 1 Текущий размер кучи: 4 (29) (22) 16 17 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Переносим максимум из корня, применяем слабую просейку: Текущее состояние массива бит: 1 1 1 0 0 1 1 Текущий размер кучи: 4 (29) 22

		(16) 17 _____ _____ Переместили корень 29 и элемент из конца неотсортированной части 16 Отсортированная часть массива: 29 30 40 45 49 50 52 59 60 86 87 91 92 Неотсортированная часть массива: 16 22 17 Текущее состояние массива бит: 1 1 1 0 0 1 1 Текущий размер кучи: 4 (16) 22 (29) 17 _____ _____ Последний элемент с максимумом перестаёт быть узлом слабой кучи. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 1 1 0 0 1 1 Текущий размер кучи: 3 (16)

		(22) 17 _____ _____ Суперродитель 16 меньше потомка 22, меняем их местами. Меняем местами поддеревья с потомками для узла, в котором находится текущий левый потомок. Текущее состояние массива бит: 0 1 1 0 0 1 1 Текущий размер кучи: 3 (22) (16) 17 _____ _____ Переносим максимум из корня, применяем слабую просейку: Текущее состояние массива бит: 0 1 1 0 0 1 1 Текущий размер кучи: 3 (22) 16 (17)

		_____ _____ Переместили корень 22 и элемент из конца неотсортированной части 17 Отсортированная часть массива: 22 29 30 40 45 49 50 52 59 60 86 87 91 92 Неотсортированная часть массива: 17 16 Текущее состояние массива бит: 0 1 1 0 0 1 1 Текущий размер кучи: 3 (17) 16 (22) _____ _____ Последний элемент с максимумом перестаёт быть узлом слабой кучи. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 1 1 0 0 1 1 Текущий размер кучи: 2 (17) (16)

		_____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Текущее состояние массива бит: 0 1 1 0 0 1 1 Текущий размер кучи: 2 (17) (16) _____ _____ Меняем местами корень 17 и следующий за ним элемент 16 Текущее состояние массива бит: 0 1 1 0 0 1 1 Текущий размер кучи: 2 (16) (17) _____ _____

		В результате сортировки массив примет вид: Итоговый массив: 16 17 22 29 30 40 45 49 50 52 59 60 86 87 91 92 Список команд: 1.Ввести массив с клавиатуры 2.Завершить работу программы Введите номер команды: 2
2	10 51 89 12 70 25 41 96 22 35 76	Программа для визуализации сортировки слабой кучей. На примере данной программы, можно увидеть, как происходит сортировка слабой кучей. Список команд: 1.Ввести массив с клавиатуры 2.Завершить работу программы Введите номер команды: 1 Введите количество элементов массива: 10 Введите через пробел элементы массива: 51 89 12 70 25 41 96 22 35 76

		1 ЭТАП 1 ЭТАП 1 ЭТАП 1 ЭТАП 1 ЭТАП Построение первоначальной слабой кучи: Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 Так как бит предка 89 равен 0, его потомки должны быть на месте: Так как бит предка 12 равен 0, его потомки должны быть на месте: Так как бит предка 70 равен 0, его потомки должны быть на месте: Так как бит предка 25 равен 0, его потомки должны быть на месте: 51 89 12 70 (25) 41 96 22 35 (76) _____ _____ Суперродитель 25 меньше потомка 76, меняем их местами. Текущее состояние массива бит: 0 0 0 0 Так как бит предка 89 равен 0, его потомки должны быть на месте: Так как бит предка 12 равен 0, его потомки должны быть на месте: Так как бит предка 70 равен 0, его потомки должны быть на месте: Так как бит предка 76 равен 0, его потомки

		должны быть на месте: 51 89 12 70 (76) 41 96 22 35 (25) _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 Так как бит предка 89 равен 0, его потомки должны быть на месте: Так как бит предка 12 равен 0, его потомки должны быть на месте: Так как бит предка 70 равен 0, его потомки должны быть на месте: Так как бит предка 76 равен 0, его потомки должны быть на месте: (51) 89 12 70 76 41 96 22 (35) 25 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0

		Так как бит предка 89 равен 0, его потомки должны быть на месте: Так как бит предка 12 равен 0, его потомки должны быть на месте: Так как бит предка 70 равен 0, его потомки должны быть на месте: Так как бит предка 76 равен 0, его потомки должны быть на месте: 51 89 12 (70) 76 41 96 (22) 35 25 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 Так как бит предка 89 равен 0, его потомки должны быть на месте: Так как бит предка 12 равен 0, его потомки должны быть на месте: Так как бит предка 70 равен 0, его потомки должны быть на месте: Так как бит предка 76 равен 0, его потомки должны быть на месте: 51 (89)

		12 70 76 41 (96) 22 35 25 _____ _____ Суперродитель 89 меньше потомка 96, меняем их местами. Текущее состояние массива бит: 0 0 0 0 Так как бит предка 96 равен 0, его потомки должны быть на месте: Так как бит предка 12 равен 0, его потомки должны быть на месте: Так как бит предка 70 равен 0, его потомки должны быть на месте: Так как бит предка 76 равен 0, его потомки должны быть на месте: 51 (96) 12 70 76 41 (89) 22 35 25 _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 Так как бит предка 96 равен 0, его потомки должны быть на месте: Так как бит предка 12 равен 0, его потомки должны быть на месте:

		Так как бит предка 70 равен 0, его потомки должны быть на месте: Так как бит предка 76 равен 0, его потомки должны быть на месте: 51 96 (12) 70 76 (41) 89 22 35 25 _____ _____ Суперродитель 12 меньше потомка 41, меняем их местами. Текущее состояние массива бит: 0 0 0 0 Так как бит предка 96 равен 0, его потомки должны быть на месте: Так как бит предка 41 равен 0, его потомки должны быть на месте: Так как бит предка 70 равен 0, его потомки должны быть на месте: Так как бит предка 76 равен 0, его потомки должны быть на месте: 51 96 (41) 70 76 (12) 89 22 35 25 _____ _____

		Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 Так как бит предка 96 равен 0, его потомки должны быть на месте: Так как бит предка 41 равен 0, его потомки должны быть на месте: Так как бит предка 70 равен 0, его потомки должны быть на месте: Так как бит предка 76 равен 0, его потомки должны быть на месте: (51) 96 41 70 (76) 12 89 22 35 25 _____ _____ Суперродитель 51 меньше потомка 76, меняем их местами. Меняем местами поддеревья с потомками для узла, в котором находится текущий элемент. Текущее состояние массива бит: 0 0 0 1 Так как бит предка 96 равен 0, его потомки должны быть на месте: Так как бит предка 41 равен 0, его потомки должны быть на месте: Так как бит предка 70 равен 0, его потомки должны быть на месте:

		Так как бит предка 51 равен 1, его потомки должны быть переставлены: (76) 96 41 70 (51) 12 89 22 25 35 _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 1 Так как бит предка 96 равен 0, его потомки должны быть на месте: Так как бит предка 41 равен 0, его потомки должны быть на месте: Так как бит предка 70 равен 0, его потомки должны быть на месте: Так как бит предка 51 равен 1, его потомки должны быть переставлены: 76 (96) 41 (70) 51 12 89 22 25 35 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Выделяем рассматриваемые элементы!

		Текущее состояние массива бит: 0 0 0 1 Так как бит предка 96 равен 0, его потомки должны быть на месте: Так как бит предка 41 равен 0, его потомки должны быть на месте: Так как бит предка 70 равен 0, его потомки должны быть на месте: Так как бит предка 51 равен 1, его потомки должны быть переставлены: (76) 96 (41) 70 51 12 89 22 25 35 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 1 Так как бит предка 96 равен 0, его потомки должны быть на месте: Так как бит предка 41 равен 0, его потомки должны быть на месте: Так как бит предка 70 равен 0, его потомки должны быть на месте: Так как бит предка 51 равен 1, его потомки должны быть переставлены: (76)

		(96) 41 70 51 12 89 22 25 35 _____ _____ Суперродитель 76 меньше потомка 96, меняем их местами. Меняем местами поддеревья с потомками для узла, в котором находится текущий элемент. Текущее состояние массива бит: 1 0 0 1 Так как бит предка 76 равен 1, его потомки должны быть переставлены: Так как бит предка 70 равен 0, его потомки должны быть на месте: Так как бит предка 41 равен 0, его потомки должны быть на месте: Так как бит предка 89 равен 1, его потомки должны быть переставлены: (96) (76) 70 41 89 22 51 12 35 25 _____ _____ Теперь строим итоговую слабую кучу:

		На данном этапе имеем: 1|0|0|1| массив бит 96|76|41|70|51|12|89|22|35|25| массив чисел Вывод итоговой слабой кучи с использованием массива: Текущее состояние массива бит: 1 0 0 1 Так как бит предка 76 равен 1, его потомки должны быть переставлены: Так как бит предка 70 равен 0, его потомки должны быть на месте: Так как бит предка 41 равен 0, его потомки должны быть на месте: Так как бит предка 89 равен 1, его потомки должны быть переставлены: 96 76 70 41 89 22 51 12 35 25 _____ _____ ^^^^^^^^^^^^^^^^^^^^^^ Получившаяся слабая куча выведена на экран! 96|76|41|70|51|12|89|22|35|25| массив чисел 2 ЭТАП 2 ЭТАП 2 ЭТАП 2 ЭТАП 2 ЭТАП На 2 этапе из корня кучи текущий

		максимальный элемент перемещаем в конец неотсортированной части массива, после чего восстанавливаем слабую кучу Переносим максимум из корня, применяем слабую просейку: Текущее состояние массива бит: 1 0 0 1 Текущий размер кучи: 10 (96) 76 70 41 89 22 51 12 35 (25) _____ _____ Переместили корень 96 и элемент из конца неотсортированной части 25 Отсортированная часть массива: 96 Неотсортированная часть массива: 25 76 41 70 51 12 89 22 35 Текущее состояние массива бит: 1 0 0 1 Текущий размер кучи: 10 (25) 76 70 41 89 22 51 12 35 (96)

		_____ _____ Последний элемент с максимумом перестаёт быть узлом слабой кучи. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 0 0 1 Текущий размер кучи: 9 (25) 76 70 41 (89) 22 51 12 35 _____ _____ Суперродитель 25 меньше потомка 89, меняем их местами. Текущее состояние массива бит: 1 0 0 1 Текущий размер кучи: 9 (89) 76 70 41 (25) 22 51 12 35 _____ _____ Выделяем рассматриваемые элементы!

		Текущее состояние массива бит: 1 0 0 1 Текущий размер кучи: 9 (89) 76 (70) 41 25 22 51 12 35 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 0 0 1 Текущий размер кучи: 9 (89) (76) 70 41 25 22 51 12 35 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Переносим максимум из корня, применяем слабую просейку: Текущее состояние массива бит: 1 0 0 1 Текущий размер кучи: 9

		(89) 76 70 41 25 22 51 12 (35) _____ _____ Переместили корень 89 и элемент из конца неотсортированной части 35 Отсортированная часть массива: 89 96 Неотсортированная часть массива: 35 76 41 70 51 12 25 22 Текущее состояние массива бит: 1 0 0 1 Текущий размер кучи: 9 (35) 76 70 41 25 22 51 12 (89) _____ _____ Последний элемент с максимумом перестаёт быть узлом слабой кучи. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 0 0 1

		Текущий размер кучи: 8 (35) 76 70 41 (25) 22 51 12 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 0 0 1 Текущий размер кучи: 8 (35) 76 (70) 41 25 22 51 12 _____ _____ Суперродитель 35 меньше потомка 70, меняем их местами. Меняем местами поддеревья с потомками для узла, в котором находится текущий левый потомок. Текущее состояние массива бит: 1 0 1 1 Текущий размер кучи: 8

		(70) 76 (35) 41 22 25 51 12 _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 0 1 1 Текущий размер кучи: 8 (70) (76) 35 41 22 25 51 12 _____ _____ Суперродитель 70 меньше потомка 76, меняем их местами. Меняем местами поддеревья с потомками для узла, в котором находится текущий левый потомок. Текущее состояние массива бит: 0 0 1 1 Текущий размер кучи: 8 (76) (70)

		41 35 51 12 22 25 _____ _____ Переносим максимум из корня, применяем слабую просейку: Текущее состояние массива бит: 0 0 1 1 Текущий размер кучи: 8 (76) 70 41 35 51 12 (22) 25 _____ _____ Переместили корень 76 и элемент из конца неотсортированной части 22 Отсортированная часть массива: 76 89 96 Неотсортированная часть массива: 22 70 41 35 51 12 25 Текущее состояние массива бит: 0 0 1 1 Текущий размер кучи: 8 (22) 70 41 35

		51 12 (76) 25 _____ _____ Последний элемент с максимумом перестаёт быть узлом слабой кучи. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 1 1 Текущий размер кучи: 7 (22) 70 41 35 (51) 12 25 _____ _____ Суперродитель 22 меньше потомка 51, меняем их местами. Текущее состояние массива бит: 0 0 1 0 Текущий размер кучи: 7 (51) 70 41 35 (22) 12 25 _____

		_____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 1 0 Текущий размер кучи: 7 (51) 70 (41) 35 22 12 25 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 1 0 Текущий размер кучи: 7 (51) (70) 41 35 22 12 25 _____ _____ Суперродитель 51 меньше потомка 70, меняем их местами. Меняем местами поддеревья с потомками для узла,

		в котором находится текущий левый потомок. Текущее состояние массива бит: 1 0 1 0 Текущий размер кучи: 7 (70) (51) 35 41 25 22 12 _____ _____ Переносим максимум из корня, применяем слабую просейку: Текущее состояние массива бит: 1 0 1 0 Текущий размер кучи: 7 (70) 51 35 41 (25) 22 12 _____ _____ Переместили корень 70 и элемент из конца неотсортированной части 25 Отсортированная часть массива: 70 76 89 96 Неотсортированная часть массива: 25 51 41 35 22 12

		Текущее состояние массива бит: 1 0 1 0 Текущий размер кучи: 7 (25) 51 35 41 (70) 22 12 _____ _____ Последний элемент с максимумом перестаёт быть узлом слабой кучи. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 0 1 0 Текущий размер кучи: 6 (25) 51 (35) 41 22 12 _____ _____ Суперродитель 25 меньше потомка 35, меняем их местами. Текущее состояние массива бит: 1 0 0 0 Текущий размер кучи: 6

		(35) 51 (25) 41 22 12 _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 0 0 0 Текущий размер кучи: 6 (35) (51) 25 41 22 12 _____ _____ Суперродитель 35 меньше потомка 51, меняем их местами. Меняем местами поддеревья с потомками для узла, в котором находится текущий левый потомок. Текущее состояние массива бит: 0 0 0 0 Текущий размер кучи: 6 (51) (35) 41 25

		22 12 _____ _____ Переносим максимум из корня, применяем слабую просейку: Текущее состояние массива бит: 0 0 0 0 Текущий размер кучи: 6 (51) 35 41 25 22 (12) _____ _____ Переместили корень 51 и элемент из конца неотсортированной части 12 Отсортированная часть массива: 51 70 76 89 96 Неотсортированная часть массива: 12 35 41 25 22 Текущее состояние массива бит: 0 0 0 0 Текущий размер кучи: 6 (12) 35 41 25

		22 (51) _____ _____ Последний элемент с максимумом перестаёт быть узлом слабой кучи. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 0 Текущий размер кучи: 5 (12) 35 41 25 (22) _____ _____ Суперродитель 12 меньше потомка 22, меняем их местами. Текущее состояние массива бит: 0 0 0 1 Текущий размер кучи: 5 (22) 35 41 25 (12) _____

		_____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 1 Текущий размер кучи: 5 (22) 35 (41) 25 12 _____ _____ Суперродитель 22 меньше потомка 41, меняем их местами. Меняем местами поддеревья с потомками для узла, в котором находится текущий левый потомок. Текущее состояние массива бит: 0 1 0 1 Текущий размер кучи: 5 (41) 35 (22) 25 12 _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 1 0 1

		Текущий размер кучи: 5 (41) (35) 22 25 12 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Переносим максимум из корня, применяем слабую просейку: Текущее состояние массива бит: 0 1 0 1 Текущий размер кучи: 5 (41) 35 22 25 (12) _____ _____ Переместили корень 41 и элемент из конца неотсортированной части 12 Отсортированная часть массива: 41 51 70 76 89 96 Неотсортированная часть массива: 12 35 22 25

		Текущее состояние массива бит: 0 1 0 1 Текущий размер кучи: 5 (12) 35 22 25 (41) _____ _____ Последний элемент с максимумом перестаёт быть узлом слабой кучи. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 1 0 1 Текущий размер кучи: 4 (12) 35 (22) 25 _____ _____ Суперродитель 12 меньше потомка 22, меняем их местами. Текущее состояние массива бит: 0 0 0 1 Текущий размер кучи: 4

		(22) 35 (12) 25 _____ _____ Выделяем рассматриваемые элементы! Текущее состояние массива бит: 0 0 0 1 Текущий размер кучи: 4 (22) (35) 12 25 _____ _____ Суперродитель 22 меньше потомка 35, меняем их местами. Меняем местами поддеревья с потомками для узла, в котором находится текущий левый потомок. Текущее состояние массива бит: 1 0 0 1 Текущий размер кучи: 4 (35) (22) 25 12

		_____ _____ Переносим максимум из корня, применяем слабую просейку: Текущее состояние массива бит: 1 0 0 1 Текущий размер кучи: 4 (35) 22 (25) 12 _____ _____ Переместили корень 35 и элемент из конца неотсортированной части 25 Отсортированная часть массива: 35 41 51 70 76 89 96 Неотсортированная часть массива: 25 22 12 Текущее состояние массива бит: 1 0 0 1 Текущий размер кучи: 4 (25) 22 (35) 12

		_____ _____ Последний элемент с максимумом перестаёт быть узлом слабой кучи. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 0 0 1 Текущий размер кучи: 3 (25) (22) 12 _____ _____ Суперродитель больше или равен потомка, оставляем всё, как есть. Переносим максимум из корня, применяем слабую просейку: Текущее состояние массива бит: 1 0 0 1 Текущий размер кучи: 3 (25) 22 (12)

		_____ _____ Переместили корень 25 и элемент из конца неотсортированной части 12 Отсортированная часть массива: 25 35 41 51 70 76 89 96 Неотсортированная часть массива: 12 22 Текущее состояние массива бит: 1 0 0 1 Текущий размер кучи: 3 (12) 22 (25) _____ _____ Последний элемент с максимумом перестаёт быть узлом слабой кучи. Выделяем рассматриваемые элементы! Текущее состояние массива бит: 1 0 0 1 Текущий размер кучи: 2 (12) (22)

		_____ _____ Суперродитель 12 меньше потомка 22, меняем их местами. Текущее состояние массива бит: 0 0 0 1 Текущий размер кучи: 2 (22) (12) _____ _____ Текущее состояние массива бит: 0 0 0 1 Текущий размер кучи: 2 (22) (12) _____ _____ Меняем местами корень 22 и следующий за ним элемент 12 Текущее состояние массива бит: 0 0 0 1 Текущий размер кучи: 2

		(12) (22) _____ _____ В результате сортировки массив примет вид: Итоговый массив: 12 22 25 35 41 51 70 76 89 96 Список команд: 1.Ввести массив с клавиатуры 2.Завершить работу программы Введите номер команды: 2
3	7 5 1 3 2 7 4 6	Итоговый массив: 1 2 3 4 5 6 7
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	Итоговый массив: 120

	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;
    int size_of_array = 0;
    int nsize = 0;
    unsigned char* r = nullptr;
    int s;
    WeakHeap() {}
    void DisplayArray();
    void DisplayHeap(int i, int j, int num);
```

```
    void WeakHeapMerge(unsigned char* r, int i, int j, int num,
int* bit);
    void WeakHeapSort();
    WeakHeap* InputHeap();
    int* bit;
    int* mas;
    int flag = 0;
    int flag2 = 0;
    int flag4 = 0;
    ~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) {
    vector <int> res = wheap; //сохраняем состояние массива чисел
    int current_size = size_of_heap - num;
    cout << "Текущее состояние массива бит: ";
    for (int i = 1; i < s; i++)
        cout << bit[i] << " ";
    cout << "\n";

    for (int i = 1; i < s; i++) //сохраняем текущее занчение
масссива для дальнейшего корректного отображения
        mas[i] = bit[i];

    if (flag == 0) {
        for (int i = 1; i < s; i++) {
```

```
            if (bit[i] == 0)
                cout << "Так как бит предка " << wheap[i] << "
равен 0, его потомки должны быть на месте:\n";
            else {
                cout << "Так как бит предка " << wheap[i] << "
равен 1, его потомки должны быть переставлены:\n";
                int x = 0;
                int y;
                for (y = 1; y <= sqrt(nsize); ) {//size_of_heap
                    for (x = 0; x < y; x++) {
                        if (y * (2 * i + 1) + x < nsize) {
//size_of_heap
                            int c = wheap[y * (2 * i) + x];
                            wheap[y * (2 * i) + x] = wheap[y * (2
* i + 1) + x];
                            wheap[y * (2 * i + 1) + x] = c;
                        }
                        if (y * (2 * i + 1) + x < s) {
                            int c1 = bit[y * (2 * i) + x];
                            bit[y * (2 * i) + x] = bit[y * (2 * i
+ 1) + x];
                            bit[y * (2 * i + 1) + x] = c1;
                        }
                    }
                    y = y * 2;
                }
            }
        }

    }

    else {
        cout << "Текущий размер кучи: " << current_size + flag4 <<
"\n\n";
```

```
        for (int it = current_size + 1 - flag2; it < size_of_heap;
it++)
            wheap[it] = -2147483647;

        for (int i = 1; i < s; i++) {

            if (bit[i] == 0)
            {
                if (flag == 0)
                cout << "Так как бит предка " << wheap[i] << "
равен 0, его потомки должны быть на месте:\n";
            }
            else {
                if (flag == 0) {
                    cout << "Так как бит предка " << wheap[i] << "
равен 1, его потомки должны быть переставлены:\n";
                }
                int x = 0;
                int y;
                for (y = 1; y <= sqrt(nsize); ) { //size_of_heap
                    for (x = 0; x < y; x++) {
                        int flag3 = 0;
                        if (y * (2 * i + 1) + x < nsize) {
//size_of_heap
                            int c = wheap[y * (2 * i) + x];
                            wheap[y * (2 * i) + x] = wheap[y * (2
* i + 1) + x];
                            wheap[y * (2 * i + 1) + x] = c;

                            if (y * (2 * i) + x == j_1) {
                                flag3++;
                                j_1 = y * (2 * i + 1) + x;
                            }
```

```
                            if (y * (2 * i + 1) + x == j_1 &&
flag3 == 0)
                                j_1 = y * (2 * i) + x;

                        }
                        if (y * (2 * i + 1) + x < s) {
                            int c1 = bit[y * (2 * i) + x];
                            bit[y * (2 * i) + x] = bit[y * (2 * i
+ 1) + x];
                            bit[y * (2 * i + 1) + x] = c1;
                        }
                    }
                    y = y * 2;

                }


            }
        }

    }

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


    cout << "\n";
    int new_size;
    if (flag == 0)
        new_size = nsize - num; //size_of_heap
    else
        new_size = nsize - 1;
```

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

    int k = 0;
    int num_elem_in_str = 0; //смотрим, какой по счёту данный
элемент в строке
    double idt = depth * 3.5;
    double idt_2 = idt;

    for (int i = 0; i < depth; i++) {
        num_elem_in_str = 0;
        if (num_elem_in_str == 0) {
            for (int iter = 0; iter < idt; iter++)
                cout << " ";
            idt = idt / 2 - 1;
        }
        else
        {
            for (int iter = 0; iter < idt_2; iter++)
                cout << " ";
        }

        for (int j = 0; j < pow(2, i); j++) {
            if (k < new_size) {
                if (k + 1 == i_1 || k + 1 == j_1)
                    if (wheap[k + 1] == -2147483647)
                        cout << "  ";
                    else
                    cout << "(" << wheap[k + 1] << ")";


                else {
                    if (wheap[k + 1] == -2147483647)
                        cout << "  ";
```

```
                    else
                    cout << wheap[k + 1];
                }

            }

            for (int iter = 0; iter < idt_2; iter++)
                cout << " ";

            k++;
            num_elem_in_str++;
        }
        cout << "\n";
        if (num_elem_in_str != 1)
              idt_2 = idt_2 / 2 - 1;
    }
    for (int it = 0; it < 73; it++)
        cout << "_";
    cout << "\n";
    wheap = res;
    for (int i = 1; i < s; i++)
        bit[i] = mas[i];
}

void WeakHeap::WeakHeapMerge(unsigned char* r, int i, int j, int
num, int* bit) {
    flag4 = 0;
    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);
        this->DisplayHeap(i, j, num);

        if (j > 0 && j < size_of_heap / 2) {
```

```
            if (bit[j] == 1)
                bit[j] = 0;
            else
                bit[j] = 1;
        }

        //Меняем значения "суперродителя" и потомка
        cout << "Суперродитель " << wheap[i] << " меньше потомка "
<< wheap[j] << ", меняем их местами.\n";
        if (flag == 0) {
            if (size_of_heap / 2 > j) {
                if (wheap[2 * j] != -2147483647 && wheap[2 * j +
1] != -2147483647)
                cout << "Меняем местами поддеревья с потомками для
узла,\n в котором находится текущий элемент.\n";
            }
        }
        else {
            if ((size_of_heap - num + 1) / 2 > j) {
                if (wheap[2 * j] != -2147483647 && wheap[2 * j +
1] != -2147483647)
                cout << "Меняем местами поддеревья с потомками для
узла,\n в котором находится текущий левый потомок.\n";
            }
        }
        swap(wheap[i], wheap[j]);
        this->DisplayHeap(i, j, num);
        flag4++;
    }
    else {
        this->DisplayHeap(i, j, num);
        flag4++;
        cout << "Суперродитель больше или равен потомку, оставляем
всё, как есть.\n";
    }
```

```
}

void WeakHeap::WeakHeapSort() {
    int n = size_of_heap;
    int size_of_array = size_of_heap;
    int lef;
    if (n > 1) {
        int i, j, x, y, Gparent;
        s = (n + 7) / 8;
        r = new unsigned char[s];
        s = n / 2;
        bit = new int[s];
        mas = new int[s];
        //Массив для обозначения, какой у элемента
        //потомок "левый", а какой "правый"
        for (i = 1; i < n / 2; ++i)
            bit[i] = 0;
        //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 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, bit);
        }
        cout << "Теперь строим итоговую слабую кучу:\n\n";
        cout << "На данном этапе имеем:\n";
        cout << " ";
        for (i = 1; i < n / 2; ++i)
            cout << bit[i] << "|";
        cout << "    массив бит \n";

        for (int i = 0; i < size_of_heap; i++)
            cout << wheap[i] << "|";
        cout << "   массив чисел \n";

        cout << "\nВывод итоговой слабой кучи с использованием
массива:\n";

        this->DisplayHeap(-1, -1, 1);
        cout << "\n^^^^^^^^^^^^^^^^^^^^^^ Получившаяся слабая куча
выведена на экран!\n";

        for (int i = 0; i < size_of_heap; i++)
            cout << wheap[i] << "|";
        cout << "   массив чисел \n";

        //Перенос максимума из корня в конец -->
       //слабая просейка --> и всё по новой
        cout << "\n2 ЭТАП 2 ЭТАП 2 ЭТАП 2 ЭТАП 2 ЭТАП\n";
        cout << "На 2 этапе из корня кучи текущий максимальный
элемент перемещаем в конец\n";
```

```
        cout << "неотсортированной части массива, после чего
восстанавливаем слабую кучу\n";
        flag++;
        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); //везде i будет
последним
            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);
            cout << "Последний элемент с максимумом перестаёт быть
узлом слабой кучи.\n\n";
            flag2++;
            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, bit); // массив бит
/ i / j / количество отсортированных элементов / массив бит для
вывода
                x >>= 1;
            }
            flag2--;
        }
        //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++)
        cout << wheap[i] << " ";
    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++;
    }
    int p = 1;
    while (count > pow(2, p))
        p++;
    wh->nsize = pow(2, p);
    for (int i = count; i < wh->nsize; i++)
        wh->wheap.push_back(-2147483647);
    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;
}
```