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

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: сортировка n-арной кучей

Студент гр. 9381	____________________	Любимов В.А.
Преподаватель	____________________	Фирсов М.А.

Санкт-Петербург

2021

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Любимов В.А.

Группа 9381

Тема работы: Сортировка n-арной кучей (n=1, 2, 3, …), 2 варианта просеивания (сверху-вниз и снизу-вверх). Демонстрация.

Исходные данные:

На вход программе подаётся размерность кучи n 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: 29.03.2021

Дата защиты реферата: 05.04.2021

Студент	____________________	Любимов В.А.
Преподаватель	____________________	Фирсов М.А.

АННОТАЦИЯ

В курсовой работе происходит сортировка массива. Програмна демонстрирует процесс сортировки массива n-арной кучей при помощи вывода на экран информации, иллюстрирующей процесс работы программы. Результатом работы программы является массив, отсортированный с помощью n-арной кучи. Для создания программы потребовалось изучить структуру n-арной кучи, алгоритм построения n-арной кучи, алгоритм сортировки n-арной кучи, и разработать визуализацию работы этих алгоритмов. Результатом является программа, которая из файла считывает размерность кучи и исходный целочисленный массив, создаёт из него кучу указанной размерности, при этом визуализируя этот процесс.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

n-арная куча представляет собой сортирующее дерево, в котором любой родитель больше (или равен) любого его потомка, а разница в глубине любых двух листьев не превышает единицы. Так же у каждого элемента ровно n потомков. Это правило не относиться к листьям и к самому правому не листу на предпоследнем уровне. Последний уровень заполняется слева направо без «дырок». Главной особенностью сортировки n-арной кучей является независимость временной сложности сортировки от сложности сортируемого массива, равная O(NlogN), где N – размер массива. Из этого следуют основные преимущество и недостаток сортировки n-арной кучей. Преимущество – гарантированная временная эффективность даже для хаотичного входного массива. Недостаток – не лучшая скорость работы для почти отсортированных массивов.

Цель работы

Целью работы является изучение сортировки при помощи n-арной кучи и разработка программы, которая будет сортировать входной массив.

Основные теоретические положения.

n-арной куча – структура данных, представляемая в виде n-арного дерева и имеющая следующие свойства:

1. Значение каждого узла не меньше любого его потомка.
2. Глубина всех листьев отличается не более, чем на 1 слой.
3. Слои заполняются слева направо и без «дырок»

Из этих свойств следует, что реализацией кучи является массив A, в котором в A[0] хранится корень кучи, а сыновьями элемента A[i] являются элементы A[i*n+1], A[i*n+2], … , A[i*n+n]. Очевидно, что при n = 1 куча превращается в массив, отсортированный по убыванию. n – натуральное число.

Так как потомками A[i] являются элементы A[i*n+1], A[i*n+2], … , A[i*n+n], то если обозначить индекс j-го потомка, как x, то x = n*i+j. Из этого получаем, что i = x-j/n. Это выражение принимает нецелое значение при любом j отличном от нуля и меньшим n. В этом случае его необходимо округлить вниз до ближайшего целого числа, иначе получим, что x = x+α, где α – добавка при округлении. Такое значение x невозможно. Значит нужно округлять вниз. Тогда можно заменить j на 1 для любого допустимого j. Итого индекс предка любого потомка можно выразить, как i = floor((x-1)/n), где x – индекс рассматриваемого потомка, i – индекс предка, floor(число) – округление вниз до ближайшего целого числа.

Итоговый алгоритм сортировки n-арной кучей.

1. Формируем из массива n-арную кучу:
   - Выполняем просейку сверху-вниз от элементов массива с индексами от $\frac{N}{n}$, где N – количество элементов в массиве, а n – размерность кучи, до 0 включительно. Из свойств n-арной кучи элементы с большими индексами являются листьями, а значит удовлетворяют определению кучи. Просейкой сверху-вниз называется процесс восстановления свойств кучи следующим образом: если значение текущего узла меньше значений его потомков, то оно меняется с наибольшим из значений потомков, а текущим узлом становится узел, бывший наибольшим потомком. Если значение текущего узла больше значения любого потомка или текущий узел – лист, то просейка завершается.
2. Выполняем сортировку с помощью полученной n-арной кучи:
   - Из определения n-арной кучи следует, что корнем является наибольший элемент в массиве. Сохраняем это значение в буферной переменной.

- Присваиваем корню значения последнего элемента в кучи.
- Записываем в последний элемент кучи сохранённое в буферной переменной значение корня. Уменьшаем размер кучи на один. При этом размер массива, содержащего кучу остаётся неизменным. То есть в нём сначала располагается куча, а затем бывшие корни кучи в обратном порядке исключения, то есть отсортированными по возрастанию
- Теперь необходимо восстановить кучу, если в ней больше одного элемента, иначе она пуста или в ней один элемент, который, очевидно, больше своих не существующих потомков. Для этого существует два способа:
    1. Вызвать просейку сверху-вниз, алгоритм которой был описан выше, от корня кучи.
    2. Вызвать восходящую просейку от корня кучи. Она работает следующим образом: от текущего узла находится путь до листа, проходящий через наибольшие узлы. Когда лист достигнут начинается подъём по найденному пути, пока элементы этого пути меньше корня. Как только найден первый элемент на пути больший корня, его значение сохраняется в буферную переменную, а элемент заменяется на корень. Далее оставшийся элементы пути, кроме корня, сдвигаются на один уровень вверх. Сохранённой в буферной переменной становиться отцом элемента, которому присвоили значение корня. Отцом элемента, сохранённого в буферной переменной, становиться элемент, который он заменил. Это продолжается, пока корнем не станет первый после корня элемент в пути.
- Все эти действия повторяются, пока куча не является пустой.

3. Как только куча опустеет, то вместо кучи в массиве, который её хранил, окажется отсортированный по возрастанию входной массив.

1. ЗАДАНИЕ

Вариант № 29. Сортировка n-арной кучей (n=1, 2, 3, …), 2 варианта просеивания (сверху-вниз и снизу-вверх). Демонстрация.

2. ОПИСАНИЕ КЛАССОВ, СТРУКТУР, ФУНКЦИЙ

Класс class Dheap реализует n-арную кучу и методы её обработки.

Объекты класса имеют следующие приватные поля:

- m_arr - массив, содержащей кучу
- m_root - корень кучи
- m_size - размер кучи
- m_arr_size - размер массива
- m_mem_size - размер массива в памяти
- m_d - порядок кучи, то есть n

Были реализованы следующие методы:

- Dheap(int* arr = nullptr, int root = 0, int size = 0, int d = 2) - конструктор, из полученного массива arr копирует элементы в массив m_arr, предварительно выделив под него необходимый объём памяти.
- bool readHeapFromFile(ifstream &fin) - ссылку на файл fin, содержащий входные данные. Записывает первое число из файла в поле m_d, а остальные числа - в полученный массив m_arr. Если заполнен весь массив, то увеличивает количество выделенной под него памяти. Если размер считанного массива нуль или m_d меньше единицы, то возвращает false. Иначе возвращает true.
- int calcHeight() - вычисляет и возвращает высоту дерева, спускаясь по левым сыновьям пока они существуют.
- int findMaxLeaf(int root) - находит индекс максимального потомка полученного узла root. Последовательного перебирает значения всех потомков узла root, сравнивает их с текущим максимальным значением

и запоминает индекс потомка с максимальным значением, который и возвращает.

- int findMax(int root) - находит индекс максимального потомка полученного узла root. Последовательного перебирает значения всех потомков узла root и самого узла, сравнивает их с текущим максимальным значением и запоминает индекс элемента с максимальным значением, который и возвращает.
- void printNode(int node_value, int step, bool is_col, char side)- выводит узел с полученным значением node_value в консоль на определённом месте задающимся отступом step. В зависимости от значения is_col выделяет узел цветом. Сначала выводит step пробелов или знаков «_», включает выделение цветом при помощи управляющей эскейп-последовательности, если is_col == true. Выводит node_value в поле шириной четыре символа и выравниванием по левому краю. В завершение выводит ещё step пробелов или знаков «_». Пробелы выводятся слева от левого сына и справа от самого правого сына.
- void printAsArr(bool is_col_first) - выводит кучу как массив. Ту часть массива, которая является кучей выделяет зелёным, отсортированную часть – белым, а первый элемент отсортированной части – голубым, если is_col_first == true.
- void printHeap(int* color_nodes, int col_size) - выводит кучу, как дерево. Если размер кучи равен нуль, то выводит сообщение о том, что куча пуста. Принимает на вход массив color_nodes, содержащий индексы узлов, которые необходимо раскрасить, и их количество col_size. Высчитывает высоту кучи методом calcHeight(). Для каждого узла высчитывает отступ по описанной далее формуле. После чего выводит этот узел при помощи метода printNode(). Если индекс выводимого узла совпадает с текущим элементом в массиве узлов для раскраски, то раскрашивает его и переходит к следующему элементу в

этом массиве. Если индекс текущего элемента равен максимальному индексу, допустимому на этом уровне, то увеличивает уровень на единицу и переносит строку. Вывод формулы отступа таков. Чтобы при выводе кучи, как дерева, гарантировать корректность вывода, то есть все сыновья умещаются и не накладываются друг на друга, необходимо рассчитать расстояние между началом строки и первым потомком в этом слое и между двумя соседними потомками. Расстояние будет измеряться в символах. Во-первых, узел должен находится ровно посередине между крайними потомками. Во-вторых, на любом уровне между потомками должен быть хотя бы один символ. В-третьих, первые два пункта должны выполняться при любой высоте дерева и размерности кучи. Итак, каждый уровень дерева будет занимать одинаковое количество символов, и каждый узел – тоже. Для узла количество символов, отображающих его значение равно 4 (если вывод значения узла занимает менее 4 символов, то лишние символы заполняются пробелами). Для n-арного дерева количество узлов на i-ом ($0 \leq i <$ количество уровней) уровне равно n^i. Для вывода каждого узла потребуется 2*step + 4, где step – размер отступа от поля вывода значения узла, то есть между двумя узлами 2*step пробелов. Значит, ширина уровня равна S = (2*step + 4)*n^i и такова для любого i. Найдём минимальный $step_i$, то есть step для i-го уровня. $step_i = \frac{S}{2 \cdot n^i} - 2$. Так как необходимо минимальное S и step больший нуля, то на самом нижнем уровне, то есть tree_height – 1, step будет равен 1. Из этого следует, что $S = 3 \cdot 2 \cdot n^{tree\_height-1}$. Тогда на i-ом ($0 \leq i <$ количество уровней) уровне $step_i = \frac{3 \cdot 2 \cdot n^{tree\_height-1}}{2 \cdot n^i} - 2$. Также из этой формулы видно, что ширина вывода бинарной кучи из пяти уровней равна 96 символов, для 3-арной кучи из 4 уровней – 162 символа, а для 8-арной кучи из 3 уровней - 384

- void siftDown(int root) - выполняет просейку сверху-вниз для узла root. Если в куче 1 элемент, то просейка не выполняется, так как один элемент уже является кучей. Для выполнения просейки среди текущего узла и его потомков находится элемент с наибольшим значением. Если этот элемент является текущим, то просейка останавливается, ибо куча востановлена. В противном случае делает новым текущим элементом этот наибольший элемент и меняет его значение со значением его отца (старого текущего элемента). После чего просейка сверху-вниз начинает выполняться для нового текущего элемента. Смыслом этого алгоритма является нахождения места под элемент, нарушающий определение кучи (больше отца), спуская его вниз по уровням, пока не найдётся подходящее место.
- void makeHeap() - строит кучу при помощи просейки сверху-вниз. Выполняется просейка сверху-вниз от элементов массива с индексами от $\frac{N}{n}$, где N – количество элементов в массиве, а n – размерность кучи, до 0 включительно. Из свойств n-арной кучи элементы с большими индексами являются листьями, так как сыновья для элемента с индексом $\frac{N}{n}$ начинаются с $\frac{N}{n} \cdot n + 1$, что больше чем количество элементов в куче. После выполнения просейки для указанных элементов получаем n-арную кучу.
- void dragMax() - удаляет максимальный элемент из кучи, то есть корень. Значение корня записывается в буферную переменную. После чего корню присваивается значение последнего элемента в кучи, а значение последнего элемента становится равным сохранённому значению старого корня. Размер кучи уменьшается на единицу. После этих операций кучу необходимо восстановить, так как новый корень, бывший последним элементом в куче, вероятно, меньше, чем хоть кто-то из его потомков.

- void upwardSift() - выполняет восходящую просейку для корня. Спускаемся вниз от корня по наибольшим вершинам, поднимаемся по полученному пути до первой вершины больше корня, сохраняем её, заменяем её корнем, сдвигаем ветку на один уровень вверх через буфферную переменную. После этого получаем восстановленную кучу.
- void upwardSiftSort() - выполняет сортировку при помощи n-арной кучи и восходящей просейки. Пока в кучи есть элементы вызывает метод dragMax(). Если после этого в куче ещё есть элементы, то восстанавливает кучу восходящей просейкой. В результате в массиве, где раньше располагалась куча, находится отсортированный по возрастанию входной массив.
- void siftDownSort() - выполняет сортировку при помощи n-арной кучи и просейки сверху-вниз. Пока в кучи есть элементы вызывает метод dragMax(). Если после этого в куче ещё есть элементы, то восстанавливает кучу просейкой сверху-вниз. В результате в массиве, где раньше располагалась куча, находится отсортированный по возрастанию входной массив.
- void printArr() - просто последовательно выводит все элементы массива, хранящего кучу.

Функция int main() реализует простейший функционал взаимодействия с пользователем. Пользователю предлагают выбор из нескольких опций: завершить выполнение программы или начать работу. В последнем случае предлагается ввести путь до файла с входными данными. Создаётся объект класса Dheap и методами readHeapFromFile() и makeHeap() создаётся куча. После этого пользователю предлагают выбрать вид просейки, который будет использоваться при сортировке. В зависимости от выбора пользователя вызывается соответствующий метод, выполняющий сортировку. Отсортированный массив выводится на экран. Выделенная память освобождается. После чего пользователю снова предлагается выбор из двух выше указанных опций.

3. ОПИСАНИЕ АЛГОРИТМА СОРТИРОВКИ

Сначала формируем из массива n-арную кучу. Для этого выполняется просейка сверху-вниз от элементов массива с индексами от $\frac{N}{n}$, где N – количество элементов в массиве, а n – размерность кучи, до 0 включительно. Из свойств n-арной кучи элементы с большими индексами являются листьями, так как сыновья для элемента с индексом $\frac{N}{n}$ начинаются с $\frac{N}{n} \cdot n + 1$, что больше чем количество элементов в дереве. Алгоритм просейки сверху-вниз был описан в описании метода siftDown().

Затем приступаем к сортировке. Обмениваем корень кучи (наибольший элемент) с последним элементом в куче (он гарантировано не больше корня) и уменьшаем размер кучи на единицу. При этом куча, вероятно, повреждена. Но прежде заметим, что в массиве, содержащем кучу, после оставшихся в куче элементов располагаются бывшие корни кучи, добавленные туда в порядке их уменьшения, то есть эти элементы отсортированы по возрастанию. Значит, после того, как размер кучу станет нуль, мы получим отсортированный исходный массив. Но если размер кучи не нуль, её необходимо восстановить после удаления корня. Для этого можно использовать восходящую просейку или просейку сверху-вниз. Их алгоритмы были уже были описаны выше. После восстановления кучи выше описанные действия повторяются, пока не будет получен полностью отсортированный массив.

Между просейкой сверху-вниз и восходящей просейкой есть разница в эффективности особенно заметная при малых n. В обоих просейках количество обменов будет одинаковым, но в просейки сверху-вниз потомки сравниваются и с родителем, и с друг другом, в то время как в восходящий просейки потомки в основном сравниваются только с друг другом. Так для бинарной кучи при восходящей просейки нужно будет сделать почти вдвое меньше сравнений (на 1 родитель - 2 потомка; сравнивая только потомков получим одно сравнения), чем при просейки сверху-вниз (одно сравнение для узла и левого потомка и второе – для результата первого и правого узла). С увеличением n количество «выигранных» сравнений будет играть всё меньше роли.

Так же заметим, что с увеличением количества n количество обменов за одну просейку уменьшается, но увеличивается количество сравнений. Оценим временную сложность алгоритма для кучи порядка n, состоящей из N элементов. Не умоляя общности будем использовать просейку сверху-вниз для восстановления кучи после изъятия корня. Будет проведено N изъятий. Для каждого изъятия будет произведена просейка. Сложность просейки зависит от количества узлов (обменов) за проход и количество сравнений для каждого узла (n штук). Рассмотрим наихудший случай, где необходимо дойти до листа на последнем уровне. Высота дерева (количество уровней) $\log_n N + 1$. Тогда сложность просейки O((n+1)* ($\log_n N + 1$)) = O($\log_n N$). Тогда сложность всей сортировки равна O(N*$\log_n N$). Для больших n, константный множитель в виде количества сравнений будет оказывать существенное влияние.

4. ТЕСТИРОВАНИЕ

Номер теста	Входные данные	Выходные данные
1	2 13 67 5 -9 67 3456 67 0 0 1 2 9 Выбрана просейка сверху-вниз	Input array: 13 67 5 -9 67 3456 67 0 0 1 2 9 Fistly we need to make heap from starting array. -- --------------------- Let's make heap! The current state of heap is:

13 |||||

67_______________________5 |||||

-9__________67 3456________67 |||||

0_____0 1_____2 9

It is the sifting down.

No son of this node exist, so it is already subheap.

No sifting needed.

Sifting down has ended.

It is the sifting down.

Let's find the maximal elememt in this root or its sons!

The value of root is 3456

13 |||||

67_______________________5 |||||

-9__________67 3456________67 |||||

0_____0 1_____2 9

9 is less or equal than current maximum value, which is 3456

13 |||||

67_______________________5 |||||

-9__________67 3456________67 |||||

0_____0 1_____2 9

Summary, the value of maximal element of root and its leaf is 3456

13 |||||

		67________________________5 ||||| -9___________67 3456_________67 ||||| 0_____0 1_____2 9 ---------------- The root, which value is 3456 is more than his sons, so this subtree is heap! 13 ||||| 67________________________5 ||||| -9___________67 3456_________67 ||||| 0_____0 1_____2 9 -- It is the sifting down. ---------------- Let's find the maximal elememt in this root or its sons! The value of root is 67 13 ||||| 67________________________5 ||||| -9___________67 3456_________67 ||||| 0_____0 1_____2 9 1 is less or equal than current maximum value, which is 67 13 ||||| 67________________________5 ||||| -9___________67 3456_________67 ||||| 0_____0 1_____2 9 2 is less or equal than current maximum value, which is 67

13 |||||

67________________________5 |||||

-9___________67 3456_________67 |||||

0_____0 1_____2 9

Summary, the value of maximal element of root and its leaf is 67

13 |||||

67________________________5 |||||

-9___________67 3456_________67 |||||

0_____0 1_____2 9

The root, which value is 67 is more than his sons, so this subtree is heap!

13 |||||

67________________________5 |||||

-9___________67 3456_________67 |||||

0_____0 1_____2 9

It is the sifting down.

Let's find the maximal elememt in this root or its sons!

The value of root is -9

13 |||||

67________________________5 |||||

-9___________67 3456_________67 |||||

0_____0 1_____2 9

0 is more than current maximum value, which is -

9

13 |||||

67_________________________5 |||||

-9___________67 3456_________67 |||||

0______0 1______2 9

0 is less or equal than current maximum value, which is 0

13 |||||

67_________________________5 |||||

-9___________67 3456_________67 |||||

0______0 1______2 9

Summary, the value of maximal element of root and its leaf is 0

13 |||||

67_________________________5 |||||

-9___________67 3456_________67 |||||

0______0 1______2 9

The son, which value is 0, is bigger than father, which value is -9! So let's change their value.

13 |||||

67_________________________5 |||||

-9___________67 3456_________67 |||||

0______0 1______2 9

Values successfully changed! One more step to make heap has been done!

13 |||||

67_________________________5 |||||

0_____________67 3456_________67 |||||

		-9____0 1_____2 9 Sifting down has ended. --- --- It is the sifting down. --------------- Let's find the maximal elememt in this root or its sons! The value of root is 5 13 ||||| 67______________________5 ||||| 0___________67 3456________67 ||||| -9____0 1_____2 9 3456 is more than current maximum value, which is 5 13 ||||| 67______________________5 ||||| 0___________67 3456________67 ||||| -9____0 1_____2 9 67 is less or equal than current maximum value, which is 3456 13 ||||| 67______________________5 ||||| 0___________67 3456________67 ||||| -9____0 1_____2 9 Summary, the value of maximal element of root and its leaf is 3456 13 |||||

67_________________________5 |||||

0_____________67 3456__________67 |||||

-9_____0 1______2 9

The son, which value is 3456, is bigger than father, which value is 5! So let's change their value.

13 |||||

67_________________________5 |||||

0_____________67 3456__________67 |||||

-9_____0 1______2 9

Values successfully changed! One more step to make heap has been done!

13 |||||

67_________________________3456 |||||

0_____________67 5_____________67 |||||

-9_____0 1______2 9

Let's find the maximal elememt in this root or its sons!

The value of root is 5

13 |||||

67_________________________3456 |||||

0_____________67 5_____________67 |||||

-9_____0 1______2 9

9 is more than current maximum value, which is 5

13 |||||

67_________________________3456 |||||

0____________67 5____________67 |||||

-9____0 1_____2 9

Summary, the value of maximal element of root and its leaf is 9

13 |||||

67________________________3456 |||||

0____________67 5____________67 |||||

-9____0 1_____2 9

The son, which value is 9, is bigger than father, which value is 5! So let's change their value.

13 |||||

67________________________3456 |||||

0____________67 5____________67 |||||

-9____0 1_____2 9

Values successfully changed! One more step to make heap has been done!

13 |||||

67________________________3456 |||||

0____________67 9____________67 |||||

-9____0 1_____2 5

Sifting down has ended.

It is the sifting down.

Let's find the maximal elememt in this root or its sons!

The value of root is 67

13 |||||

67______________________3456 |||||

0___________67 9___________67 |||||

-9____0 1_____2 5

0 is less or equal than current maximum value, which is 67

13 |||||

67______________________3456 |||||

0___________67 9___________67 |||||

-9____0 1_____2 5

67 is less or equal than current maximum value, which is 67

13 |||||

67______________________3456 |||||

0___________67 9___________67 |||||

-9____0 1_____2 5

Summary, the value of maximal element of root and its leaf is 67

13 |||||

67______________________3456 |||||

0___________67 9___________67 |||||

-9____0 1_____2 5

The root, which value is 67 is more than his sons, so this subtree is heap!

13 |||||

67______________________3456 |||||

0___________67 9___________67 |||||

<table>
<tr>
<td></td>
<td></td>
<td>-9____0 1_____2 5

It is the sifting down.

Let's find the maximal elememt in this root or its sons!
The value of root is 13
13 |||||
67______________________3456 |||||
0___________67 9___________67 |||||
-9____0 1_____2 5
67 is more than current maximum value, which is 13
13 |||||
67______________________3456 |||||
0___________67 9___________67 |||||
-9____0 1_____2 5
3456 is more than current maximum value, which is 67
13 |||||
67______________________3456 |||||
0___________67 9___________67 |||||
-9____0 1_____2 5
Summary, the value of maximal element of root and its leaf is 3456
13 |||||
67______________________3456 |||||
0___________67 9___________67 |||||</td>
</tr>
</table>

-9____0   1_____2   5

The son, which value is 3456, is bigger than father, which value is 13! So let's change their value.

13 |||||

67________________________3456 |||||

0____________67   9____________67 |||||

-9____0   1_____2   5

Values successfully changed! One more step to make heap has been done!

3456 |||||

67________________________13 |||||

0____________67   9____________67 |||||

-9____0   1_____2   5

Let's find the maximal elememt in this root or its sons!

The value of root is 13

3456 |||||

67________________________13 |||||

0____________67   9____________67 |||||

-9____0   1_____2   5

9 is less or equal than current maximum value, which is 13

3456 |||||

67________________________13 |||||

0____________67   9____________67 |||||

```
 -9____0   1_____2   5
```

67 is more than current maximum value, which is 13

```
             3456              |||||
     67______________________13       |||||
  0___________67     9___________67   |||||
 -9____0   1_____2   5
```

Summary, the value of maximal element of root and its leaf is 67

```
             3456              |||||
     67______________________13       |||||
  0___________67     9___________67   |||||
 -9____0   1_____2   5
```

The son, which value is 67, is bigger than father, which value is 13! So let's change their value.

```
             3456              |||||
     67______________________13       |||||
  0___________67     9___________67   |||||
 -9____0   1_____2   5
```

Values successfully changed! One more step to make heap has been done!

```
             3456              |||||
     67______________________67       |||||
  0___________67     9___________13   |||||
 -9____0   1_____2   5
```

Sifting down has ended.

Building heap has been ended!

The current state of heap is:

```
                  3456                 |||||
         67_______________________67           |||||
    0____________67          9____________13      |||||
 -9_____0     1______2     5
```

Choose sifting type. 'd' for sifting down and 'u' for upward sifting:

d

You choose sifting down sort.

Heapsort with sifting down.

This sort using the sifting down to restore heap after draging max element.

Exclude the node with biggest value. It is the root because we are working with max-heap. Save the root value in buffer variable

```
                  3456                 |||||
         67_______________________67           |||||
    0____________67          9____________13      |||||
 -9_____0     1______2     5
```

It is heap as array. The green part is actually the

<table>
<tr>
<td></td>
<td></td>
<td>heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: 3456 67 67 0 67 9 13 -9 0 1 2 5
Make the value of the last element in heap the root value. Eventually, put the old root value into the last position in heap and decrease the size of the heap. So we have already sorted elements after the heap in the array that is storing our elements as heap and sorted sequence.
Current state of heap as tree:
5 |||||
67________________________67 |||||
0____________67 9____________13 |||||
-9____0 1_____2
Current state of heap as array:
It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: 5 67 67 0 67 9 13 -9 0 1 2 3456

Heap is corrupted after draging maximal element!
Let's restore our heap with sifting it down.

It is the sifting down.

Let's find the maximal elememt in this root or its sons!
The value of root is 5
5 |||||</td>
</tr>
</table>

67_________________________67 |||||

0____________67 9____________13 |||||

-9____0 1_____2

67 is more than current maximum value, which is 5

5 |||||

67_________________________67 |||||

0____________67 9____________13 |||||

-9____0 1_____2

67 is less or equal than current maximum value, which is 67

5 |||||

67_________________________67 |||||

0____________67 9____________13 |||||

-9____0 1_____2

Summary, the value of maximal element of root and its leaf is 67

5 |||||

67_________________________67 |||||

0____________67 9____________13 |||||

-9____0 1_____2

The son, which value is 67, is bigger than father, which value is 5! So let's change their value.

5 |||||

67_________________________67 |||||

0____________67 9____________13 |||||

-9____0 1_____2

Values successfully changed! One more step to make heap has been done!

67 |||||

5______________________67 |||||

0___________67 9___________13 |||||

-9____0 1_____2

Let's find the maximal elememt in this root or its sons!

The value of root is 5

67 |||||

5______________________67 |||||

0___________67 9___________13 |||||

-9____0 1_____2

0 is less or equal than current maximum value, which is 5

67 |||||

5______________________67 |||||

0___________67 9___________13 |||||

-9____0 1_____2

67 is more than current maximum value, which is 5

67 |||||

5______________________67 |||||

0___________67 9___________13 |||||

-9____0 1_____2

Summary, the value of maximal element of root and its leaf is 67

67 |||||

5_________________________67 |||||

0____________67 9____________13 |||||

-9____0 1_____2

The son, which value is 67, is bigger than father, which value is 5! So let's change their value.

67 |||||

5_________________________67 |||||

0____________67 9____________13 |||||

-9____0 1_____2

Values successfully changed! One more step to make heap has been done!

67 |||||

67_________________________67 |||||

0____________5 9____________13 |||||

-9____0 1_____2

Let's find the maximal elememt in this root or its sons!

The value of root is 5

67 |||||

67_________________________67 |||||

0____________5 9____________13 |||||

-9____0 1_____2

1 is less or equal than current maximum value, which is 5

67 |||||

67_________________________67 |||||

		0____________5 9____________13 ||||| -9____0 1_____2 2 is less or equal than current maximum value, which is 5 67 ||||| 67________________________67 ||||| 0____________5 9____________13 ||||| -9____0 1_____2 Summary, the value of maximal element of root and its leaf is 5 67 ||||| 67________________________67 ||||| 0____________5 9____________13 ||||| -9____0 1_____2 ---------------- The root, which value is 5 is more than his sons, so this subtree is heap! 67 ||||| 67________________________67 ||||| 0____________5 9____________13 ||||| -9____0 1_____2 The heap restored! Current state of the heap: 67 ||||| 67________________________67 ||||| 0____________5 9____________13 ||||| -9____0 1_____2

Exclude the node with biggest value. It is the root because we are working with max-heap. Save the root value in buffer variable

```
                 67                  |||||
         67___________________________67          |||||
    0_____________5          9_____________13      |||||
 -9_____0     1______2
```

It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: 67 67 67 0 5 9 13 -9 0 1 2 3456

Make the value of the last element in heap the root value. Eventually, put the old root value into the last position in heap and decrease the size of the heap. So we have already sorted elements after the heap in the array that is storing our elements as heap and sorted sequence.

Current state of heap as tree:

```
                 2                   |||||
         67___________________________67          |||||
    0_____________5          9_____________13      |||||
 -9_____0     1
```

Current state of heap as array:

It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old

		root, which just has been added to the sorted sequence: 2 67 67 0 5 9 13 -9 0 1 67 3456 -- Heap is corrupted after draging maximal element! Let's restore our heap with sifting it down. -- It is the sifting down. ---------------- Let's find the maximal elememt in this root or its sons! The value of root is 2 2 ||||| 67________________________67 ||||| 0____________5 9____________13 ||||| -9____0 1 67 is more than current maximum value, which is 2 2 ||||| 67________________________67 ||||| 0____________5 9____________13 ||||| -9____0 1 67 is less or equal than current maximum value, which is 67 2 ||||| 67________________________67 ||||| 0____________5 9____________13 ||||| -9____0 1 Summary, the value of maximal element of root and its leaf is 67

2 |||||

67________________________67 |||||

0____________5 9____________13 |||||

-9____0 1

The son, which value is 67, is bigger than father, which value is 2! So let's change their value.

2 |||||

67________________________67 |||||

0____________5 9____________13 |||||

-9____0 1

Values successfully changed! One more step to make heap has been done!

67 |||||

2_________________________67 |||||

0____________5 9____________13 |||||

-9____0 1

Let's find the maximal elememt in this root or its sons!

The value of root is 2

67 |||||

2_________________________67 |||||

0____________5 9____________13 |||||

-9____0 1

0 is less or equal than current maximum value, which is 2

67 |||||

2__________________________67 |||||

0____________5 9____________13 |||||

-9____0 1

5 is more than current maximum value, which is 2

67 |||||

2__________________________67 |||||

0____________5 9____________13 |||||

-9____0 1

Summary, the value of maximal element of root and its leaf is 5

67 |||||

2__________________________67 |||||

0____________5 9____________13 |||||

-9____0 1

The son, which value is 5, is bigger than father, which value is 2! So let's change their value.

67 |||||

2__________________________67 |||||

0____________5 9____________13 |||||

-9____0 1

Values successfully changed! One more step to make heap has been done!

67 |||||

5__________________________67 |||||

0____________2 9____________13 |||||

-9____0 1

Let's find the maximal elememt in this root or its sons!

The value of root is 2

```
                 67                    |||||
         5___________________________67            |||||
    0_____________2          9_____________13      |||||
 -9_____0    1
```

1 is less or equal than current maximum value, which is 2

```
                 67                    |||||
         5___________________________67            |||||
    0_____________2          9_____________13      |||||
 -9_____0    1
```

Summary, the value of maximal element of root and its leaf is 2

```
                 67                    |||||
         5___________________________67            |||||
    0_____________2          9_____________13      |||||
 -9_____0    1
```

The root, which value is 2 is more than his sons, so this subtree is heap!

```
                 67                    |||||
         5___________________________67            |||||
    0_____________2          9_____________13      |||||
 -9_____0    1
```

The heap restored!

Current state of the heap:

67 |||||

5__________________________67 |||||

0____________2 9____________13 |||||

-9____0 1

Exclude the node with biggest value. It is the root because we are working with max-heap. Save the root value in buffer variable

67 |||||

5__________________________67 |||||

0____________2 9____________13 |||||

-9____0 1

It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: 67 5 67 0 2 9 13 -9 0 1 67 3456

Make the value of the last element in heap the root value. Eventually, put the old root value into the last position in heap and decrease the size of the heap. So we have already sorted elements after the heap in the array that is storing our elements as heap and sorted sequence.

Current state of heap as tree:

1 |||||

5__________________________67 |||||

		0____________2 9____________13 ||||| -9____0 Current state of heap as array: It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: 1 5 67 0 2 9 13 -9 0 67 67 3456 --- Heap is corrupted after draging maximal element! Let's restore our heap with sifting it down. --- It is the sifting down. --------------- Let's find the maximal elememt in this root or its sons! The value of root is 1 1 ||||| 5_________________________67 ||||| 0____________2 9____________13 ||||| -9____0 5 is more than current maximum value, which is 1 1 ||||| 5_________________________67 ||||| 0____________2 9____________13 ||||| -9____0 67 is more than current maximum value, which is 5 1 ||||| 5_________________________67 |||||

		0____________2 9____________13 ||||| -9____0 Summary, the value of maximal element of root and its leaf is 67 1 ||||| 5_________________________67 ||||| 0____________2 9____________13 ||||| -9____0 ---------------- The son, which value is 67, is bigger than father, which value is 1! So let's change their value. 1 ||||| 5_________________________67 ||||| 0____________2 9____________13 ||||| -9____0 Values successfully changed! One more step to make heap has been done! 67 ||||| 5_________________________1 ||||| 0____________2 9____________13 ||||| -9____0 ---------------- Let's find the maximal elememt in this root or its sons! The value of root is 1 67 ||||| 5_________________________1 ||||| 0____________2 9____________13 |||||

<table>
<tr>
<td></td>
<td></td>
<td>-9____0
9 is more than current maximum value, which is 1
67 |||||
5_______________________1 |||||
0___________2 9___________13 |||||
-9____0
13 is more than current maximum value, which is 9
67 |||||
5_______________________1 |||||
0___________2 9___________13 |||||
-9____0
Summary, the value of maximal element of root and its leaf is 13
67 |||||
5_______________________1 |||||
0___________2 9___________13 |||||
-9____0

The son, which value is 13, is bigger than father, which value is 1! So let's change their value.
67 |||||
5_______________________1 |||||
0___________2 9___________13 |||||
-9____0
Values successfully changed! One more step to make heap has been done!
67 |||||
5_______________________13 |||||</td>
</tr>
</table>

0____________2   9____________1 |||||

-9____0

Sifting down has ended.

The heap restored!

Current state of the heap:

67 |||||

5_________________________13 |||||

0____________2   9____________1 |||||

-9____0

Exclude the node with biggest value. It is the root because we are working with max-heap. Save the root value in buffer variable

67 |||||

5_________________________13 |||||

0____________2   9____________1 |||||

-9____0

It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: 67 5 13 0 2 9 1 -9 0 67 67 3456

Make the value of the last element in heap the root value. Eventually, put the old root value into the last position in heap and decrease the size of the heap. So we have already sorted elements after the heap in the array that is storing our elements as heap and sorted sequence.

Current state of heap as tree:

0 |||||

5________________________13 |||||

0____________2 9____________1 |||||

-9

Current state of heap as array:

It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: 0 5 13 0 2 9 1 -9 67 67 67 3456

Heap is corrupted after draging maximal element! Let's restore our heap with sifting it down.

It is the sifting down.

Let's find the maximal elememt in this root or its sons!

The value of root is 0

0 |||||

5________________________13 |||||

0____________2 9____________1 |||||

-9

5 is more than current maximum value, which is 0

0 |||||

<table>
<tr>
<td></td>
<td></td>
<td>5__________________________13 |||||
0____________2 9____________1 |||||
-9
13 is more than current maximum value, which is 5
0 |||||
5__________________________13 |||||
0____________2 9____________1 |||||
-9
Summary, the value of maximal element of root and its leaf is 13
0 |||||
5__________________________13 |||||
0____________2 9____________1 |||||
-9

The son, which value is 13, is bigger than father, which value is 0! So let's change their value.
0 |||||
5__________________________13 |||||
0____________2 9____________1 |||||
-9
Values successfully changed! One more step to make heap has been done!
13 |||||
5__________________________0 |||||
0____________2 9____________1 |||||
-9</td>
</tr>
</table>

Let's find the maximal elememt in this root or its sons!

The value of root is 0

```
                    13                    |||||
         5___________________________0           |||||
   0_____________2          9_____________1       |||||
 -9
```

9 is more than current maximum value, which is 0

```
                    13                    |||||
         5___________________________0           |||||
   0_____________2          9_____________1       |||||
 -9
```

1 is less or equal than current maximum value, which is 9

```
                    13                    |||||
         5___________________________0           |||||
   0_____________2          9_____________1       |||||
 -9
```

Summary, the value of maximal element of root and its leaf is 9

```
                    13                    |||||
         5___________________________0           |||||
   0_____________2          9_____________1       |||||
 -9
```

The son, which value is 9, is bigger than father, which value is 0! So let's change their value.

```
                    13                    |||||
```

		5_______________________________0 ||||| 0_______________2 9_______________1 ||||| -9 Values successfully changed! One more step to make heap has been done! 13 ||||| 5_______________________________9 ||||| 0_______________2 0_______________1 ||||| -9 Sifting down has ended. --- The heap restored! Current state of the heap: 13 ||||| 5_______________________________9 ||||| 0_______________2 0_______________1 ||||| -9 --- Exclude the node with biggest value. It is the root because we are working with max-heap. Save the root value in buffer variable 13 ||||| 5_______________________________9 ||||| 0_______________2 0_______________1 |||||

		-9 It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: 13 5 9 0 2 0 1 -9 67 67 67 3456 Make the value of the last element in heap the root value. Eventually, put the old root value into the last position in heap and decrease the size of the heap. So we have already sorted elements after the heap in the array that is storing our elements as heap and sorted sequence. Current state of heap as tree: -9 ||||| 5____________9 ||||| 0______2 0______1 ||||| Current state of heap as array: It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: -9 5 9 0 2 0 1 13 67 67 67 3456 -- Heap is corrupted after draging maximal element! Let's restore our heap with sifting it down. -- It is the sifting down. ---------------- Let's find the maximal elememt in this root or its sons!

The value of root is -9

```
         -9          |||||
    5____________9      |||||
 0_____2    0_____1    |||||
```

5 is more than current maximum value, which is -9

```
         -9          |||||
    5____________9      |||||
 0_____2    0_____1    |||||
```

9 is more than current maximum value, which is 5

```
         -9          |||||
    5____________9      |||||
 0_____2    0_____1    |||||
```

Summary, the value of maximal element of root and its leaf is 9

```
         -9          |||||
    5____________9      |||||
 0_____2    0_____1    |||||
```

The son, which value is 9, is bigger than father, which value is -9! So let's change their value.

```
         -9          |||||
    5____________9      |||||
 0_____2    0_____1    |||||
```

Values successfully changed! One more step to make heap has been done!

```
          9            |||||
    5____________-9      |||||
 0_____2     0_____1    |||||
```

Let's find the maximal elememt in this root or its sons!

The value of root is -9

```
          9            |||||
    5____________-9      |||||
 0_____2     0_____1    |||||
```

0 is more than current maximum value, which is -9

```
          9            |||||
    5____________-9      |||||
 0_____2     0_____1    |||||
```

1 is more than current maximum value, which is 0

```
          9            |||||
    5____________-9      |||||
 0_____2     0_____1    |||||
```

Summary, the value of maximal element of root and its leaf is 1

```
          9            |||||
```

5____________-9 |||||
0______2 0______1 |||||

The son, which value is 1, is bigger than father, which value is -9! So let's change their value.

9 |||||
5____________-9 |||||
0______2 0______1 |||||

Values successfully changed! One more step to make heap has been done!

9 |||||
5____________1 |||||
0______2 0______-9 |||||

Sifting down has ended.

The heap restored!

Current state of the heap:

9 |||||
5____________1 |||||
0______2 0______-9 |||||

Exclude the node with biggest value. It is the root because we are working with max-heap. Save the root value in buffer variable

```
          9              |||||
    5____________1       |||||
 0_____2     0_____-9   |||||
```

It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: 9 5 1 0 2 0 -9 13 67 67 67 3456

Make the value of the last element in heap the root value. Eventually, put the old root value into the last position in heap and decrease the size of the heap. So we have already sorted elements after the heap in the array that is storing our elements as heap and sorted sequence.

Current state of heap as tree:

```
          -9             |||||
    5____________1       |||||
 0_____2     0
```

Current state of heap as array:

It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: -9 5 1 0 2 0 9 13 67 67 67 3456

Heap is corrupted after draging maximal element!

		Let's restore our heap with sifting it down. -- It is the sifting down. ---------------- Let's find the maximal elememt in this root or its sons! The value of root is -9 -9 ||||| 5____________1 ||||| 0_____2 0 5 is more than current maximum value, which is -9 -9 ||||| 5____________1 ||||| 0_____2 0 1 is less or equal than current maximum value, which is 5 -9 ||||| 5____________1 ||||| 0_____2 0 Summary, the value of maximal element of root and its leaf is 5 -9 ||||| 5____________1 ||||| 0_____2 0 ---------------- The son, which value is 5, is bigger than father, which value is -9! So let's change their value. -9 |||||

		5____________1 ||||| 0______2 0 Values successfully changed! One more step to make heap has been done! 5 ||||| -9___________1 ||||| 0______2 0 ---------------- Let's find the maximal elememt in this root or its sons! The value of root is -9 5 ||||| -9___________1 ||||| 0______2 0 0 is more than current maximum value, which is -9 5 ||||| -9___________1 ||||| 0______2 0 2 is more than current maximum value, which is 0 5 ||||| -9___________1 ||||| 0______2 0 Summary, the value of maximal element of root and its leaf is 2 5 ||||| -9___________1 ||||| 0______2 0

The son, which value is 2, is bigger than father, which value is -9! So let's change their value.

```
          5            |||||
    -9____________1       |||||
 0______2     0
```

Values successfully changed! One more step to make heap has been done!

```
          5            |||||
    2_____________1       |||||
 0______-9    0
```

Sifting down has ended.

The heap restored!

Current state of the heap:

```
          5            |||||
    2_____________1       |||||
 0______-9    0
```

Exclude the node with biggest value. It is the root because we are working with max-heap. Save the root value in buffer variable

```
          5            |||||
    2_____________1       |||||
```

		0_____-9 0 It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: 5 2 1 0 -9 0 9 13 67 67 67 3456 Make the value of the last element in heap the root value. Eventually, put the old root value into the last position in heap and decrease the size of the heap. So we have already sorted elements after the heap in the array that is storing our elements as heap and sorted sequence. Current state of heap as tree: 0 ||||| 2___________1 ||||| 0_____-9 Current state of heap as array: It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: 0 2 1 0 -9 5 9 13 67 67 67 3456 -- Heap is corrupted after draging maximal element! Let's restore our heap with sifting it down. -- It is the sifting down. --------------- Let's find the maximal elememt in this root or its sons! The value of root is 0

<table>
<tr>
<td></td>
<td></td>
<td>0 |||||
2____________1 |||||
0______-9
2 is more than current maximum value, which is 0
0 |||||
2____________1 |||||
0______-9
1 is less or equal than current maximum value, which is 2
0 |||||
2____________1 |||||
0______-9
Summary, the value of maximal element of root and its leaf is 2
0 |||||
2____________1 |||||
0______-9

The son, which value is 2, is bigger than father, which value is 0! So let's change their value.
0 |||||
2____________1 |||||
0______-9
Values successfully changed! One more step to make heap has been done!
2 |||||
0____________1 |||||
0______-9</td>
</tr>
</table>

		---------------- Let's find the maximal elememt in this root or its sons! The value of root is 0 2 ||||| 0____________1 ||||| 0_____-9 0 is less or equal than current maximum value, which is 0 2 ||||| 0____________1 ||||| 0_____-9 -9 is less or equal than current maximum value, which is 0 2 ||||| 0____________1 ||||| 0_____-9 Summary, the value of maximal element of root and its leaf is 0 2 ||||| 0____________1 ||||| 0_____-9 ---------------- The root, which value is 0 is more than his sons, so this subtree is heap! 2 ||||| 0____________1 ||||| 0_____-9

		The heap restored! Current state of the heap: 2 ||||| 0____________1 ||||| 0_____-9 --- Exclude the node with biggest value. It is the root because we are working with max-heap. Save the root value in buffer variable 2 ||||| 0____________1 ||||| 0_____-9 It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: 2 0 1 0 -9 5 9 13 67 67 67 3456 Make the value of the last element in heap the root value. Eventually, put the old root value into the last position in heap and decrease the size of the heap. So we have already sorted elements after the heap in the array that is storing our elements as heap and sorted sequence. Current state of heap as tree: -9 ||||| 0____________1 |||||

		0 Current state of heap as array: It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: -9 0 1 0 2 5 9 13 67 67 67 3456 --- Heap is corrupted after draging maximal element! Let's restore our heap with sifting it down. --- It is the sifting down. ---------------- Let's find the maximal elememt in this root or its sons! The value of root is -9 -9 ||||| 0____________1 ||||| 0 0 is more than current maximum value, which is -9 -9 ||||| 0____________1 ||||| 0 1 is more than current maximum value, which is 0 -9 ||||| 0____________1 ||||| 0 Summary, the value of maximal element of root and its leaf is 1

-9 |||||
0____________1 |||||
0

The son, which value is 1, is bigger than father, which value is -9! So let's change their value.
-9 |||||
0____________1 |||||
0
Values successfully changed! One more step to make heap has been done!
1 |||||
0____________-9 |||||
0

Sifting down has ended.

The heap restored!
Current state of the heap:
1 |||||
0____________-9 |||||
0

Exclude the node with biggest value. It is the root because we are working with max-heap. Save the

		root value in buffer variable 1 ||||| 0____________-9 ||||| 0 It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: 1 0 -9 0 2 5 9 13 67 67 67 3456 Make the value of the last element in heap the root value. Eventually, put the old root value into the last position in heap and decrease the size of the heap. So we have already sorted elements after the heap in the array that is storing our elements as heap and sorted sequence. Current state of heap as tree: 0 ||||| 0_____-9 ||||| Current state of heap as array: It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: 0 0 -9 1 2 5 9 13 67 67 67 3456 --- Heap is corrupted after draging maximal element! Let's restore our heap with sifting it down. --- It is the sifting down. ----------------

<table>
<tr>
<td></td>
<td></td>
<td>Let's find the maximal elememt in this root or its sons!
The value of root is 0
    0 |||||
 0______-9 |||||

0 is less or equal than current maximum value, which is 0
    0 |||||
 0______-9 |||||

-9 is less or equal than current maximum value, which is 0
    0 |||||
 0______-9 |||||

Summary, the value of maximal element of root and its leaf is 0
    0 |||||
 0______-9 |||||

The root, which value is 0 is more than his sons, so this subtree is heap!
    0 |||||
 0______-9 |||||

The heap restored!</td>
</tr>
</table>

		Current state of the heap: 0 ||||| 0_____-9 ||||| -- Exclude the node with biggest value. It is the root because we are working with max-heap. Save the root value in buffer variable 0 ||||| 0_____-9 ||||| It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: 0 0 -9 1 2 5 9 13 67 67 67 3456 Make the value of the last element in heap the root value. Eventually, put the old root value into the last position in heap and decrease the size of the heap. So we have already sorted elements after the heap in the array that is storing our elements as heap and sorted sequence. Current state of heap as tree: -9 ||||| 0 Current state of heap as array:

		It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: -9 0 0 1 2 5 9 13 67 67 67 3456 -- Heap is corrupted after draging maximal element! Let's restore our heap with sifting it down. -- It is the sifting down. ---------------- Let's find the maximal elememt in this root or its sons! The value of root is -9 -9 ||||| 0 0 is more than current maximum value, which is -9 -9 ||||| 0 Summary, the value of maximal element of root and its leaf is 0 -9 ||||| 0 ---------------- The son, which value is 0, is bigger than father, which value is -9! So let's change their value. -9 ||||| 0 Values successfully changed! One more step to

		make heap has been done! 0 ||||| -9 Sifting down has ended. -- The heap restored! Current state of the heap: 0 ||||| -9 -- Exclude the node with biggest value. It is the root because we are working with max-heap. Save the root value in buffer variable 0 ||||| -9 It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: 0 -9 0 1 2 5 9 13 67 67 67 3456 Make the value of the last element in heap the root value. Eventually, put the old root value into the last position in heap and decrease the size of the heap. So we have already sorted elements after the heap in the array that is storing our elements as

		heap and sorted sequence. Current state of heap as tree: -9 ||||| Current state of heap as array: It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: -9 0 0 1 2 5 9 13 67 67 67 3456 -- Heap is corrupted after draging maximal element! Let's restore our heap with sifting it down. -- It is the sifting down. Only root is reamining in the heap, so it is already heap. No sifting needed. Sifting down has ended. -- The heap restored! Current state of the heap: -9 ||||| -- Exclude the node with biggest value. It is the root because we are working with max-heap. Save the

		root value in buffer variable -9 ||||| It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: -9 0 0 1 2 5 9 13 67 67 67 3456 Make the value of the last element in heap the root value. Eventually, put the old root value into the last position in heap and decrease the size of the heap. So we have already sorted elements after the heap in the array that is storing our elements as heap and sorted sequence. Current state of heap as tree: Empty heap! Current state of heap as array: It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: -9 0 0 1 2 5 9 13 67 67 67 3456 -- It was the last node in the heap! -- Sort has successfully ended! It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: -9 0 0 1 2 5 9 13 67 67 67 3456

		Sorted array: -9 0 0 1 2 5 9 13 67 67 67 3456
2	3 123 67 5 -9 67 3456 67 0 10 -1 -2 -9 Восходящая просейка	Input array: 123 67 5 -9 67 0 10 -1 -2 -9 Fistly we need to make heap from starting array. -- --------------------- Let's make heap! The current state of heap is: 123 ||||| 67_________________5__________________- 9 ||||| 67____0_____10 -1____-2____-9 -- It is the sifting down. No son of this node exist, so it is already subheap. No sifting needed. Sifting down has ended. -- -- It is the sifting down. ---------------- Let's find the maximal elememt in this root or its sons! The value of root is 5

123 |||||

67_________________5__________________-

9 |||||

67____0_____10 -1____-2____-9

-1 is less or equal than current maximum value, which is 5

123 |||||

67_________________5__________________-

9 |||||

67____0_____10 -1____-2____-9

-2 is less or equal than current maximum value, which is 5

123 |||||

67_________________5__________________-

9 |||||

67____0_____10 -1____-2____-9

-9 is less or equal than current maximum value, which is 5

123 |||||

67_________________5__________________-

9 |||||

67____0_____10 -1____-2____-9

Summary, the value of maximal element of root and its leaf is 5

123 |||||

67_________________5__________________-

9 |||||

67____0_____10 -1____-2____-9

The root, which value is 5 is more than his sons, so this subtree is heap!

123 |||||
67_________________5__________________-9 |||||
67____0_____10 -1____-2____-9

It is the sifting down.

Let's find the maximal elememt in this root or its sons!

The value of root is 67

123 |||||
67_________________5__________________-9 |||||
67____0_____10 -1____-2____-9

67 is less or equal than current maximum value, which is 67

123 |||||
67_________________5__________________-9 |||||
67____0_____10 -1____-2____-9

0 is less or equal than current maximum value, which is 67

123 |||||
67_________________5__________________-9 |||||
67____0_____10 -1____-2____-9

10 is less or equal than current maximum value, which is 67

123 |||||

67_________________5__________________-

9 |||||

67____0_____10 -1____-2____-9

Summary, the value of maximal element of root and its leaf is 67

123 |||||

67_________________5__________________-

9 |||||

67____0_____10 -1____-2____-9

The root, which value is 67 is more than his sons, so this subtree is heap!

123 |||||

67_________________5__________________-

9 |||||

67____0_____10 -1____-2____-9

It is the sifting down.

Let's find the maximal elememt in this root or its sons!

The value of root is 123

123 |||||

67_________________5__________________-

9 |||||

67____0_____10 -1____-2____-9

67 is less or equal than current maximum value, which is 123

123 |||||

67________________5_________________-9 |||||

67____0_____10 -1____-2____-9

5 is less or equal than current maximum value, which is 123

123 |||||

67________________5_________________-9 |||||

67____0_____10 -1____-2____-9

-9 is less or equal than current maximum value, which is 123

123 |||||

67________________5_________________-9 |||||

67____0_____10 -1____-2____-9

Summary, the value of maximal element of root and its leaf is 123

123 |||||

67________________5_________________-9 |||||

67____0_____10 -1____-2____-9

The root, which value is 123 is more than his sons, so this subtree is heap!

123 |||||

		67__________________5___________________- 9 ||||| 67____0_____10 -1____-2____-9 Building heap has been ended! The current state of heap is: 123 ||||| 67__________________5___________________- 9 ||||| 67____0_____10 -1____-2____-9 -- -------------------- Choose sifting type. 'd' for sifting down and 'u' for upward sifting: u You choose upward sifting sort. Heapsort with upward sifting. This sort using the upward sifting to restore heap after draging max element. -- -------------------- --- Exclude the node with biggest value. It is the root because we are working with max-heap. Save the root value in buffer variable 123 |||||

		67__________________5___________________-9 ||||| 67____0_____10 -1____-2____-9 It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: 123 67 5 -9 67 0 10 -1 -2 -9 Make the value of the last element in heap the root value. Eventually, put the old root value into the last position in heap and decrease the size of the heap. So we have already sorted elements after the heap in the array that is storing our elements as heap and sorted sequence. Current state of heap as tree: -9 ||||| 67__________________5___________________-9 ||||| 67____0_____10 -1____-2 Current state of heap as array: It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: -9 67 5 -9 67 0 10 -1 -2 123 --- Heap is corrupted after draging maximal element! Let's restore our heap with sifting it down. --- Starting upward sifting. Firstly, we need to find a route from heap root to

the leaf which consist of the biggest sons.

Let's find the maximal son of this root!

The value of root is -9

-9 |||||

67__________________5___________________-

9 |||||

67____0______10 -1____-2

67 is first son, so it is new maximum value.

-9 |||||

67__________________5___________________-

9 |||||

67____0______10 -1____-2

5 is less or equal than current maximum value,

which is 67

-9 |||||

67__________________5___________________-

9 |||||

67____0______10 -1____-2

-9 is less or equal than current maximum value,

which is 67

-9 |||||

67__________________5___________________-

9 |||||

67____0______10 -1____-2

Summary, the value of maximal leaf is 67

-9 |||||

67__________________5___________________-

9 |||||

		67____0_____10 -1____-2 ---------------- ---------------- Let's find the maximal son of this root! The value of root is 67 -9 ||||| 67________________5_________________- 9 ||||| 67____0_____10 -1____-2 67 is first son, so it is new maximum value. -9 ||||| 67________________5_________________- 9 ||||| 67____0_____10 -1____-2 0 is less or equal than current maximum value, which is 67 -9 ||||| 67________________5_________________- 9 ||||| 67____0_____10 -1____-2 10 is less or equal than current maximum value, which is 67 -9 ||||| 67________________5_________________- 9 ||||| 67____0_____10 -1____-2 Summary, the value of maximal leaf is 67 -9 ||||| 67________________5_________________-

9 |||||

67____0_____10 -1____-2

So we have got the next route:

-9 |||||

67________________5_________________-

9 |||||

67____0_____10 -1____-2

Let's find the first element on this route which is bigger than root.

Current node, which value is 67 is more than root, which value is -9

-9 |||||

67________________5_________________-

9 |||||

67____0_____10 -1____-2

Save current node in buffer variable, replace the value of current node with the value of the root and exclude this node from the route.

-9 |||||

67________________5_________________-

9 |||||

-9____0_____10 -1____-2

Now we are going to shift all remaining nodes in the route, which we have got early, to the one level upper. The nearest node which will be replaced with previous saved in buffer variable node value, which is 67

		Remaining route: -9 ||||| 67_________________5__________________- 9 ||||| -9____0_____10 -1____-2 The value of current will be saved in buffer variable and then will be replaced with the value of its biggest son, which is 67 and has been saved in the another buffer variable. -9 ||||| 67_________________5__________________- 9 ||||| -9____0_____10 -1____-2 -9 ||||| 67_________________5__________________- 9 ||||| -9____0_____10 -1____-2 The value of current will be saved in buffer variable and then will be replaced with the value of its biggest son, which is 67 and has been saved in the another buffer variable. -9 ||||| 67_________________5__________________- 9 ||||| -9____0_____10 -1____-2 67 ||||| 67_________________5__________________- 9 ||||| -9____0_____10 -1____-2

		The heap root has been reached. Shifting nodes to upper level has successfully ended. Upward sifting has ended. -- The heap restored! Current state of the heap: 67 ||||| 67_________________5__________________- 9 ||||| -9____0_____10 -1____-2 -- Exclude the node with biggest value. It is the root because we are working with max-heap. Save the root value in buffer variable 67 ||||| 67_________________5__________________- 9 ||||| -9____0_____10 -1____-2 It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: 67 67 5 -9 -9 0 10 -1 -2 123 Make the value of the last element in heap the root value. Eventually, put the old root value into the last position in heap and decrease the size of the

		heap. So we have already sorted elements after the heap in the array that is storing our elements as heap and sorted sequence. Current state of heap as tree: -2 ||||| 67_________________5__________________-9 ||||| -9____0_____10 -1 Current state of heap as array: It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: -2 67 5 -9 -9 0 10 -1 67 123 --- Heap is corrupted after draging maximal element! Let's restore our heap with sifting it down. --- Starting upward sifting. Firstly, we need to find a route from heap root to the leaf which consist of the biggest sons. --------------- Let's find the maximal son of this root! The value of root is -2 -2 ||||| 67_________________5__________________-9 ||||| -9____0_____10 -1 67 is first son, so it is new maximum value. -2 |||||

		67____________________5_____________________- 9 ||||| -9_____0______10 -1 5 is less or equal than current maximum value, which is 67 -2 ||||| 67____________________5_____________________- 9 ||||| -9_____0______10 -1 -9 is less or equal than current maximum value, which is 67 -2 ||||| 67____________________5_____________________- 9 ||||| -9_____0______10 -1 Summary, the value of maximal leaf is 67 -2 ||||| 67____________________5_____________________- 9 ||||| -9_____0______10 -1 ---------------- ---------------- Let's find the maximal son of this root! The value of root is 67 -2 ||||| 67____________________5_____________________- 9 ||||| -9_____0______10 -1 -9 is first son, so it is new maximum value.

-2 |||||

67_________________5__________________-

9 |||||

-9____0_____10 -1

0 is more than current maximum value, which is -

9

-2 |||||

67_________________5__________________-

9 |||||

-9____0_____10 -1

10 is more than current maximum value, which is

0

-2 |||||

67_________________5__________________-

9 |||||

-9____0_____10 -1

Summary, the value of maximal leaf is 10

-2 |||||

67_________________5__________________-

9 |||||

-9____0_____10 -1

So we have got the next route:

-2 |||||

67_________________5__________________-

9 |||||

-9____0_____10 -1

Let's find the first element on this route which is bigger than root.

Current node, which value is 10 is more than root, which value is -2

-2 |||||

67__________________5___________________-

9 |||||

-9____0______10 -1

Save current node in buffer variable, replace the value of current node with the value of the root and exclude this node from the route.

-2 |||||

67__________________5___________________-

9 |||||

-9____0______-2 -1

Now we are going to shift all remaining nodes in the route, which we have got early, to the one level upper. The nearest node which will be replaced with previous saved in buffer variable node value, which is 10

Remaining route:

-2 |||||

67__________________5___________________-

9 |||||

-9____0______-2 -1

The value of current will be saved in buffer variable and then will be replaced with the value of its biggest son, which is 10 and has been saved in the another buffer variable.

-2 |||||

		67_________________5__________________- 9 ||||| -9____0_____-2 -1 -2 ||||| 10_________________5__________________- 9 ||||| -9____0_____-2 -1 The value of current will be saved in buffer variable and then will be replaced with the value of its biggest son, which is 67 and has been saved in the another buffer variable. -2 ||||| 10_________________5__________________- 9 ||||| -9____0_____-2 -1 67 ||||| 10_________________5__________________- 9 ||||| -9____0_____-2 -1 The heap root has been reached. Shifting nodes to upper level has successfully ended. Upward sifting has ended. --- The heap restored! Current state of the heap: 67 ||||| 10_________________5__________________- 9 ||||| -9____0_____-2 -1

		-- Exclude the node with biggest value. It is the root because we are working with max-heap. Save the root value in buffer variable 67 ||||| 10________________5_________________-9 ||||| -9____0_____-2 -1 It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: 67 10 5 -9 -9 0 -2 -1 67 123 Make the value of the last element in heap the root value. Eventually, put the old root value into the last position in heap and decrease the size of the heap. So we have already sorted elements after the heap in the array that is storing our elements as heap and sorted sequence. Current state of heap as tree: -1 ||||| 10________________5_________________-9 ||||| -9____0_____-2 Current state of heap as array: It is heap as array. The green part is actually the

heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: -1 10 5 -9 -9 0 -2 67 67 123

Heap is corrupted after draging maximal element!

Let's restore our heap with sifting it down.

Starting upward sifting.

Firstly, we need to find a route from heap root to the leaf which consist of the biggest sons.

Let's find the maximal son of this root!

The value of root is -1

-1 |||||

10__________________5___________________-9 |||||

-9____0_____-2

10 is first son, so it is new maximum value.

-1 |||||

10__________________5___________________-9 |||||

-9____0_____-2

5 is less or equal than current maximum value, which is 10

-1 |||||

10__________________5___________________-9 |||||

-9____0_____-2

-9 is less or equal than current maximum value, which is 10

-1 |||||

10_________________5__________________-

9 |||||

-9____0_____-2

Summary, the value of maximal leaf is 10

-1 |||||

10_________________5__________________-

9 |||||

-9____0_____-2

Let's find the maximal son of this root!

The value of root is 10

-1 |||||

10_________________5__________________-

9 |||||

-9____0_____-2

-9 is first son, so it is new maximum value.

-1 |||||

10_________________5__________________-

9 |||||

-9____0_____-2

0 is more than current maximum value, which is -

9

-1 |||||

10_________________5__________________-

9 |||||

-9____0_____-2

-2 is less or equal than current maximum value, which is 0

-1 |||||

10_________________5__________________-

9 |||||

-9____0_____-2

Summary, the value of maximal leaf is 0

-1 |||||

10_________________5__________________-

9 |||||

-9____0_____-2

So we have got the next route:

-1 |||||

10_________________5__________________-

9 |||||

-9____0_____-2

Let's find the first element on this route which is bigger than root.

Current node, which value is 0 is more than root, which value is -1

-1 |||||

10_________________5__________________-

9 |||||

-9____0_____-2

Save current node in buffer variable, replace the value of current node with the value of the root and exclude this node from the route.

-1 |||||

10__________________5____________________-

9 |||||

-9____-1____-2

Now we are going to shift all remaining nodes in the route, which we have got early, to the one level upper. The nearest node which will be replaced with previous saved in buffer variable node value, which is 0

Remaining route:

-1 |||||

10__________________5____________________-

9 |||||

-9____-1____-2

The value of current will be saved in buffer variable and then will be replaced with the value of its biggest son, which is 0 and has been saved in the another buffer variable.

-1 |||||

10__________________5____________________-

9 |||||

-9____-1____-2

-1 |||||

0___________________5____________________-

9 |||||

-9____-1____-2

The value of current will be saved in buffer variable and then will be replaced with the value of its biggest son, which is 10 and has been saved

		in the another buffer variable. -1 ||||| 0__________________5__________________-9 ||||| -9____-1____-2 10 ||||| 0__________________5__________________-9 ||||| -9____-1____-2 The heap root has been reached. Shifting nodes to upper level has successfully ended. Upward sifting has ended. --- The heap restored! Current state of the heap: 10 ||||| 0__________________5__________________-9 ||||| -9____-1____-2 --- Exclude the node with biggest value. It is the root because we are working with max-heap. Save the root value in buffer variable 10 ||||| 0__________________5__________________-

		9 ||||| -9____-1____-2 It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: 10 0 5 -9 -9 -1 -2 67 67 123 Make the value of the last element in heap the root value. Eventually, put the old root value into the last position in heap and decrease the size of the heap. So we have already sorted elements after the heap in the array that is storing our elements as heap and sorted sequence. Current state of heap as tree: -2 ||||| 0__________________5__________________-9 ||||| -9____-1 Current state of heap as array: It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: -2 0 5 -9 -9 -1 10 67 67 123 -- Heap is corrupted after draging maximal element! Let's restore our heap with sifting it down. -- Starting upward sifting. Firstly, we need to find a route from heap root to the leaf which consist of the biggest sons.

		---------------- Let's find the maximal son of this root! The value of root is -2 -2 ||||| 0___________________5___________________- 9 ||||| -9____-1 0 is first son, so it is new maximum value. -2 ||||| 0___________________5___________________- 9 ||||| -9____-1 5 is more than current maximum value, which is 0 -2 ||||| 0___________________5___________________- 9 ||||| -9____-1 -9 is less or equal than current maximum value, which is 5 -2 ||||| 0___________________5___________________- 9 ||||| -9____-1 Summary, the value of maximal leaf is 5 -2 ||||| 0___________________5___________________- 9 ||||| -9____-1 ----------------

		So we have got the next route: -2 ||||| 0__________________5__________________- 9 ||||| -9____-1 Let's find the first element on this route which is bigger than root. Current node, which value is 5 is more than root, which value is -2 -2 ||||| 0__________________5__________________- 9 ||||| -9____-1 Save current node in buffer variable, replace the value of current node with the value of the root and exclude this node from the route. -2 ||||| 0__________________-2_________________- 9 ||||| -9____-1 Now we are going to shift all remaining nodes in the route, which we have got early, to the one level upper. The nearest node which will be replaced with previous saved in buffer variable node value, which is 5 Remaining route: -2 ||||| 0__________________-2_________________-

		9 ||||| -9____-1 The value of current will be saved in buffer variable and then will be replaced with the value of its biggest son, which is 5 and has been saved in the another buffer variable. -2 ||||| 0_________________-2________________- 9 ||||| -9____-1 5 ||||| 0_________________-2________________- 9 ||||| -9____-1 The heap root has been reached. Shifting nodes to upper level has successfully ended. Upward sifting has ended. --- The heap restored! Current state of the heap: 5 ||||| 0_________________-2________________- 9 ||||| -9____-1 ---

		Exclude the node with biggest value. It is the root because we are working with max-heap. Save the root value in buffer variable 5 ||||| 0__________________-2_________________-9 ||||| -9____-1 It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: 5 0 -2 -9 -9 -1 10 67 67 123 Make the value of the last element in heap the root value. Eventually, put the old root value into the last position in heap and decrease the size of the heap. So we have already sorted elements after the heap in the array that is storing our elements as heap and sorted sequence. Current state of heap as tree: -1 ||||| 0__________________-2_________________-9 ||||| -9 Current state of heap as array: It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: -1 0 -2 -9 -9 5 10 67 67 123 --- Heap is corrupted after draging maximal element!

		Let's restore our heap with sifting it down. --- Starting upward sifting. Firstly, we need to find a route from heap root to the leaf which consist of the biggest sons. ---------------- Let's find the maximal son of this root! The value of root is -1 -1 ||||| 0__________________-2_________________- 9 ||||| -9 0 is first son, so it is new maximum value. -1 ||||| 0__________________-2_________________- 9 ||||| -9 -2 is less or equal than current maximum value, which is 0 -1 ||||| 0__________________-2_________________- 9 ||||| -9 -9 is less or equal than current maximum value, which is 0 -1 ||||| 0__________________-2_________________- 9 ||||| -9

		Summary, the value of maximal leaf is 0 -1 ||||| 0__________________-2_________________- 9 ||||| -9 ---------------- ---------------- Let's find the maximal son of this root! The value of root is 0 -1 ||||| 0__________________-2_________________- 9 ||||| -9 -9 is first son, so it is new maximum value. -1 ||||| 0__________________-2_________________- 9 ||||| -9 Summary, the value of maximal leaf is -9 -1 ||||| 0__________________-2_________________- 9 ||||| -9 ---------------- So we have got the next route: -1 ||||| 0__________________-2_________________- 9 ||||| -9

Let's find the first element on this route which is bigger than root.

Current node, which value is -9 is less than root, which value is -1. So we exclude it from the route.

-1 |||||

0__________________-2_________________-

9 |||||

-9

Current node, which value is -9 is less than root, which value is -1. So we exclude it from the route.

-1 |||||

0__________________-2_________________-

9 |||||

-9

Current node, which value is 0 is more than root, which value is -1

-1 |||||

0__________________-2_________________-

9 |||||

-9

Save current node in buffer variable, replace the value of current node with the value of the root and exclude this node from the route.

-1 |||||

-1_________________-2_________________-9

|||||

-9

Now we are going to shift all remaining nodes in the route, which we have got early, to the one level upper. The nearest node which will be replaced with previous saved in buffer variable node value, which is 0

Remaining route:

```
                 -1                    |||||
     -1_________________-2_________________-9
|||||
 -9
```

The value of current will be saved in buffer variable and then will be replaced with the value of its biggest son, which is 0 and has been saved in the another buffer variable.

```
                 -1                    |||||
     -1_________________-2_________________-9
|||||
 -9
                 0                     |||||
     -1_________________-2_________________-9
|||||
 -9
```

The heap root has been reached. Shifting nodes to upper level has successfully ended.

Upward sifting has ended.

The heap restored!

Current state of the heap:

```
                 0                     |||||
     -1_________________-2_________________-9
```

		||||| -9 -- Exclude the node with biggest value. It is the root because we are working with max-heap. Save the root value in buffer variable 0 ||||| -1_________________-2_________________-9 ||||| -9 It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: 0 -1 -2 -9 -9 5 10 67 67 123 Make the value of the last element in heap the root value. Eventually, put the old root value into the last position in heap and decrease the size of the heap. So we have already sorted elements after the heap in the array that is storing our elements as heap and sorted sequence. Current state of heap as tree: -9 ||||| -1____-2____-9 ||||| Current state of heap as array:

		It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: -9 -1 -2 -9 0 5 10 67 67 123 --- Heap is corrupted after draging maximal element! Let's restore our heap with sifting it down. --- Starting upward sifting. Firstly, we need to find a route from heap root to the leaf which consist of the biggest sons. ---------------- Let's find the maximal son of this root! The value of root is -9 -9 ||||| -1_____-2_____-9 ||||| -1 is first son, so it is new maximum value. -9 ||||| -1_____-2_____-9 ||||| -2 is less or equal than current maximum value, which is -1 -9 ||||| -1_____-2_____-9 ||||| -9 is less or equal than current maximum value, which is -1 -9 |||||

-1____-2____-9 |||||

Summary, the value of maximal leaf is -1

-9 |||||

-1____-2____-9 |||||

So we have got the next route:

-9 |||||

-1____-2____-9 |||||

Let's find the first element on this route which is bigger than root.

Current node, which value is -1 is more than root, which value is -9

-9 |||||

-1____-2____-9 |||||

Save current node in buffer variable, replace the value of current node with the value of the root and exclude this node from the route.

-9 |||||

-9____-2____-9 |||||

Now we are going to shift all remaining nodes in the route, which we have got early, to the one level upper. The nearest node which will be replaced with previous saved in buffer variable node value, which is -1

Remaining route:

```
      -9        |||||
 -9____-2____-9   |||||
```

The value of current will be saved in buffer variable and then will be replaced with the value of its biggest son, which is -1 and has been saved in the another buffer variable.

```
      -9        |||||
 -9____-2____-9   |||||

      -1        |||||
 -9____-2____-9   |||||
```

The heap root has been reached. Shifting nodes to upper level has successfully ended.

Upward sifting has ended.

The heap restored!

Current state of the heap:

```
      -1        |||||
 -9____-2____-9   |||||
```

		Exclude the node with biggest value. It is the root because we are working with max-heap. Save the root value in buffer variable
		      -1 |||||
		 -9____-2____-9 |||||
		
		It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: -1 -9 -2 -9 0 5 10 67 67 123
		Make the value of the last element in heap the root value. Eventually, put the old root value into the last position in heap and decrease the size of the heap. So we have already sorted elements after the heap in the array that is storing our elements as heap and sorted sequence.
		Current state of heap as tree:
		      -9 |||||
		 -9____-2
		Current state of heap as array:
		It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: -9 -9 -2 -1 0 5 10 67 67 123
		-- 	
		Heap is corrupted after draging maximal element! Let's restore our heap with sifting it down.
		-- 	
		Starting upward sifting.

<table>
<tr><td></td><td></td><td>Firstly, we need to find a route from heap root to the leaf which consist of the biggest sons.

Let's find the maximal son of this root!
The value of root is -9
-9 |||||
-9____-2
-9 is first son, so it is new maximum value.
-9 |||||
-9____-2
-2 is more than current maximum value, which is -9
-9 |||||
-9____-2
Summary, the value of maximal leaf is -2
-9 |||||
-9____-2

So we have got the next route:
-9 |||||
-9____-2

Let's find the first element on this route which is bigger than root.
Current node, which value is -2 is more than root, which value is -9
-9 |||||
-9____-2
Save current node in buffer variable, replace the</td></tr>
</table>

value of current node with the value of the root and exclude this node from the route.

    -9 |||||
 -9____-9

Now we are going to shift all remaining nodes in the route, which we have got early, to the one level upper. The nearest node which will be replaced with previous saved in buffer variable node value, which is -2

Remaining route:

    -9 |||||
 -9____-9

The value of current will be saved in buffer variable and then will be replaced with the value of its biggest son, which is -2 and has been saved in the another buffer variable.

    -9 |||||
 -9____-9
    -2 |||||
 -9____-9

The heap root has been reached. Shifting nodes to upper level has successfully ended.

Upward sifting has ended.

The heap restored!

Current state of the heap:

    -2 |||||
 -9____-9

		--- Exclude the node with biggest value. It is the root because we are working with max-heap. Save the root value in buffer variable -2 ||||| -9____-9 It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: -2 -9 -9 -1 0 5 10 67 67 123 Make the value of the last element in heap the root value. Eventually, put the old root value into the last position in heap and decrease the size of the heap. So we have already sorted elements after the heap in the array that is storing our elements as heap and sorted sequence. Current state of heap as tree: -9 ||||| -9 Current state of heap as array: It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: -9 -9 -2 -1 0 5 10 67 67 123 --- Heap is corrupted after draging maximal element!

		Let's restore our heap with sifting it down. --- Starting upward sifting. Firstly, we need to find a route from heap root to the leaf which consist of the biggest sons. --------------- Let's find the maximal son of this root! The value of root is -9 -9 ||||| -9 -9 is first son, so it is new maximum value. -9 ||||| -9 Summary, the value of maximal leaf is -9 -9 ||||| -9 --------------- So we have got the next route: -9 ||||| -9 Let's find the first element on this route which is bigger than root. Current node, which value is -9 is more than root, which value is -9 -9 ||||| -9 Save current node in buffer variable, replace the value of current node with the value of the root

		and exclude this node from the route. -9 ||||| -9 Now we are going to shift all remaining nodes in the route, which we have got early, to the one level upper. The nearest node which will be replaced with previous saved in buffer variable node value, which is -9 Remaining route: -9 ||||| -9 The value of current will be saved in buffer variable and then will be replaced with the value of its biggest son, which is -9 and has been saved in the another buffer variable. -9 ||||| -9 -9 ||||| -9 The heap root has been reached. Shifting nodes to upper level has successfully ended. Upward sifting has ended. -- The heap restored! Current state of the heap: -9 ||||| -9

		-- Exclude the node with biggest value. It is the root because we are working with max-heap. Save the root value in buffer variable -9 ||||| -9 It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: -9 -9 -2 -1 0 5 10 67 67 123 Make the value of the last element in heap the root value. Eventually, put the old root value into the last position in heap and decrease the size of the heap. So we have already sorted elements after the heap in the array that is storing our elements as heap and sorted sequence. Current state of heap as tree: -9 ||||| Current state of heap as array: It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: -9 -9 -2 -1 0 5 10 67 67 123 -- Heap is corrupted after draging maximal element! Let's restore our heap with sifting it down.

		-- Starting upward sifting. Only root is reamining in the heap, so it is already heap. No sifting needed. Upward sifting has ended. -- The heap restored! Current state of the heap: -9 ||||| -- Exclude the node with biggest value. It is the root because we are working with max-heap. Save the root value in buffer variable -9 ||||| It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: -9 -9 -2 -1 0 5 10 67 67 123 Make the value of the last element in heap the root value. Eventually, put the old root value into the last position in heap and decrease the size of the heap. So we have already sorted elements after the heap in the array that is storing our elements as

		heap and sorted sequence. Current state of heap as tree: Empty heap! Current state of heap as array: It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: -9 -9 -2 -1 0 5 10 67 67 123 -- It was the last node in the heap! --- Sort has successfully ended! It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: -9 -9 -2 -1 0 5 10 67 67 123 Sorted array: -9 -9 -2 -1 0 5 10 67 67 123
3	4 123 34 56 90 78	123 ||||| 34____56____90____78 ||||| Sorted array: 34 56 78 90 123
4	1 0 1 2 3 4 5 6 7 8 9	9 ||||| 8 |||||

		7 ||||| 6 ||||| 5 ||||| 4 ||||| 3 ||||| 2 ||||| 1 ||||| 0 ||||| Sorted array: 0 1 2 3 4 5 6 7 8 9
5	3 19 121 12 13 1 2 3 4 5 86 7 8 9	Sorted array: 1 2 3 4 5 7 8 9 12 13 19 86 121
6	8 18 145 54 -985 554 32 8 1 2 3 6 5 4 9 8 7 1 23 6448 35 185 188 61 -8 -9 3 -5 0 1452	Sorted array: -985 -9 -8 -5 0 1 1 2 3 3 4 5 6 7 8 8 9 18 23 32 35 54 61 145 185 188 554 1452 6448
7	4 89 81 42 43 61 2 37 14 -1 -2 -3 -4 -5 -6 -7 -8 -9 0	Sorted array: -9 -8 -7 -6 -5 -4 -3 -2 -1 0 2 14 37 42 43 61 81 89

5. ДЕМОНСТРАЦИЯ

После запуска программы пользователю будет предложено выбрать из двух команд: запуск или завершение программы. Им соответствуют значения ‘s’ и 'q':

's' – Начать работу программы.

'q' - Завершить работу программы

Такая реализация позволяет обработать множество различных входных данных, не прерывая выполнения программы.

После выбора команды ‘s’, пользователю предлагается ввести путь до файла с входными данными. После успешного чтения входных данных и построения кучи, сопровождающегося соответствующими комментариями, выводимыми в стандартный поток вывода, пользователю предложат выбрать тип просейки, который будет использован при сортировке.

Далее будет выполнена соответствующая сортировка, сопровождающиеся необходимыми комментариями, выводимыми в стандартный поток вывода

Для завершения программы необходимо ввести команду ‘q’. Это можно сделать в стартовом меню, которое выводится автоматически после успешного выполнения сортировки или после введения пользователем ошибочной команды или неверных входных данных.

ЗАКЛЮЧЕНИЕ

В результате выполнения работы была изучена сортировка при помощи n-арной кучи. Была изучена структура n-арной кучи, а также алгоритм её построения. На языке программирования C++ реализован алгоритм сортировки с помощью n-арной кучи, а работа программы сопровождается визуализацией выполняемых операций.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Habr. URL: https://habr.com/ru/company/edison/blog/495420/
2. Habr. URL: https://habr.com/ru/company/edison/blog/509330/
3.
4. https://en.wikipedia.org/wiki/Binary_heap

ПРИЛОЖЕНИЕ

ИСХОДНЫЙ КОД ПРОГРАММЫ.

Название файла main.cpp

```
#include <iostream>
#include <string>
#include <fstream>
#include <cstdlib>
#include <cmath>

using namespace std;

class Dheap{
    private:
        int* m_arr = nullptr;//массив хранящий кучу
        int m_root = 0;//корень кучи
        int m_size = 0;//размер кучи
        int m_arr_size = 0;//размер массива
        int m_mem_size = 0;//размер кучи в памяти
        int m_d = 2;//порядок кучи; по умолчанию куча бинарная

    public:
        Dheap(int* arr = nullptr, int root = 0, int size = 0, int d = 2):
m_root(root), m_size(size), m_d(d){//конструктор копирует полученный массив в
массив вершин; пока в это ещё не d-арное дерево
            if(arr){
                m_arr = new int[size];
                m_arr_size = size;
                for(int i = 0; i < size; i++){
                    m_arr[i] = arr[i];
                }
            }
        }

        bool readHeapFromFile(ifstream &fin){//метод считывающий массив из входного
файла
            m_size = 0;
            m_mem_size = 0;
            m_root = 0;
            m_d = 0;
            fin >> m_d;

            if(m_d <= 0){
                cout << "Non natural value of amount of node sons!\n";
                return false;
            }
```

```
    while(1){
        if(m_size == m_mem_size){
            m_mem_size += 10;
            int* new_arr = new int[m_mem_size];
            for(int i = 0; i < m_size; i++){
                new_arr[i] = m_arr[i];
            }
            delete[] m_arr;
            m_arr = new_arr;
        }

        if(fin.eof()){
            break;
        }

        fin >> m_arr[m_size];
        m_size += 1;
    }

    m_arr_size = m_size;
    if(m_size == 0){
        cout << "Error! Empty heap has inputed!\n";
        return false;
    }
    return true;
}

int calcHeight(){//высчитывает количество уровней в дереве
    int i = m_root;
    int height = 0;
    while(i < m_size){
        height += 1;
        i = i*m_d + 1;
    }
    return height;
}

int findMaxLeaf(int root){//поиск индекса максимального элемента среди
потомков вершины
    cout << "----------------\n";
    cout << "Let's find the maximal son of this root!\n";
    int max = -1;
    int j = 0;
    int nodes[m_d+1];
    cout << "The value of root is " << m_arr[root] << "\n";
    nodes[j] = root;
    j += 1;
    printHeap(nodes, j);
    for(int i = root*m_d+1; i <= root*m_d + m_d && i < m_size; i++){
        if(max == -1){
```

```
                max = i;
                cout << m_arr[i] << " is first son, so it is new maximum
value.\n";
                nodes[0] = i;
                //j += 1;
                printHeap(nodes, j);
                continue;
            }

            if(m_arr[i] > m_arr[max]){
                cout << m_arr[i] << " is more than current maximum value, which
is " << m_arr[max] << '\n';
                nodes[0] = max;
                max = i;
            }
            else{
                cout << m_arr[i] << " is less or equal than current maximum
value, which is " << m_arr[max] << '\n';
                nodes[0] = max;
            }
            nodes[j] = i;
            //j += 1;
            printHeap(nodes, j+1);
        }
        cout << "Summary, the value of maximal leaf is " << m_arr[max] << "\n";
        nodes[0] = max;
        printHeap(nodes, 1);
        cout << "----------------\n";
        return max;
    }

    int findMax(int root){//поиск индекса максимального элемента среди вершины
и потомков
        cout << "----------------\n";
        cout << "Let's find the maximal elememt in this root or its sons!\n";
        int max = root;
        int j = 0;
        int nodes[m_d+1];
        cout << "The value of root is " << m_arr[root] << "\n";
        nodes[j] = root;
        j += 1;
        printHeap(nodes, j);
        for(int i = root*m_d+1; i <= root*m_d + m_d && i < m_size; i++){
            if(m_arr[i] > m_arr[max]){
                cout << m_arr[i] << " is more than current maximum value, which
is " << m_arr[max] << '\n';
                nodes[0] = max;
                max = i;
            }
            else{
```

```
                    cout << m_arr[i] << " is less or equal than current maximum 
value, which is " << m_arr[max] << '\n';
                    nodes[0] = max;
                }
                nodes[j] = i;
                //j += 1;
                printHeap(nodes, j+1);
            }
            cout << "Summary, the value of maximal element of root and its leaf is 
" << m_arr[max] << "\n";
            nodes[0] = max;
            printHeap(nodes, 1);
            cout << "----------------\n";
            return max;
        }

        void siftDown(int root){//обыкновенная просейка сверху-вниз
            cout << "-----------------------------------------------\n";
            cout << "It is the sifting down.\n";
            if(m_size < 2){
                cout << "Only root is reamining in the heap, so it is already heap. 
No sifting needed.\n";
                cout << "Sifting down has ended.\n";
                cout << "-----------------------------------------------\n";
                return;
            }

            if(!(root * m_d + 1 < m_size)){
                cout << "No son of this node exist, so it is already subheap. No 
sifting needed.\n";
                cout << "Sifting down has ended.\n";
                cout << "-----------------------------------------------\n";
                return;
            }

            while(root * m_d + 1 < m_size){
                int n_root = findMax(root);

                int nodes[2] {root, n_root};
                if(n_root == root){
                    cout << "The root, which value is " << m_arr[root] << " is more 
than his sons, so this subtree is heap!\n";
                    printHeap(nodes, 2);
                    cout << "\n";
                    return;
                }

                cout << "The son, which value is " << m_arr[n_root] << ", is bigger 
than father, which value is " << m_arr[root] << "! So let's change their value.\n";
                printHeap(nodes, 2);
```

```
                int c = m_arr[root];
                m_arr[root] = m_arr[n_root];
                m_arr[n_root] = c;
                root = n_root;
                cout << "Values successfully changed! One more step to make heap
has been done!\n";
                printHeap(nodes, 2);
                cout << "\n";
            }
            cout << "Sifting down has ended.\n";
            cout << "-----------------------------------------------\n";
        }

        void makeHeap(){//получение кучи из массива за O(n) времени, где n -
количетсво элементов в массиве
            cout << "-------------------------------------------------------------
--------------------\n";
            cout << "Let's make heap!\nThe current state of heap is:\n";
            printHeap(nullptr, -1);
            int i = m_size/m_d;//элементы с большими индексами не имеют потомков,
то есть они уже являются кучами
            while(i >= 0){
                siftDown(i);
                i -= 1;
            }
            m_arr_size = m_size;
            cout << "Building heap has been ended!\nThe current state of heap
is:\n";
            printHeap(nullptr, -1);
            cout << "-------------------------------------------------------------
--------------------\n";
        }

        void dragMax(){//удаляет вершину из кучи перенося её в конец массива
предварительно заменив его последним элементом
            cout << "-----------------------------------------------\n";
            cout << "Exclude the node with biggest value. It is the root because we
are working with max-heap. Save the root value in buffer variable\n";
            int nodes[2] {m_root, m_size-1};
            printHeap(nodes, 2);
            printAsArr(false);
            int max = m_arr[m_root];
            m_arr[m_root] = m_arr[m_size-1];
            m_arr[m_size-1] = max;
            m_size -= 1;
            cout << "Make the value of the last element in heap the root value.
Eventually, put the old root value into the last position in heap and decrease the
size of the heap. So we have already sorted elements after the heap in the array
that is storing our elements as heap and sorted sequence.\n";
            cout << "Current state of heap as tree:\n";
```

```
        nodes[0] = m_root;
        printHeap(nodes, 1);
        cout << "Current state of heap as array:\n";
        printAsArr(true);
        cout << "-----------------------------------------------\n";
    }

    void upwardSift(){//восходящая просейка (модифицированная просейка снизу-вверх); спускаемся вниз по наибольшим вершинам, поднимаемся по этой ветке до первой вершины больше корня, сохраняем её, заменяем её корнем, сдвигаем ветку на один уровень вверх через буфферную переменную
        cout << "-----------------------------------------------\n";
        cout << "Starting upward sifting.\n";
        if(m_size > 1){
            int buf;
            int cur = m_root;
            int way[calcHeight()];
            int i = 0;
            way[i] = m_root;
            i += 1;

            cout << "Firstly, we need to find a route from heap root to the leaf which consist of the biggest sons.\n";
            while(cur*m_d+1 < m_size){
                cur = findMaxLeaf(cur);
                way[i] = cur;
                i += 1;
            }
            cout << "So we have got the next route:\n";
            printHeap(way, i);

            cout << "\nLet's find the first element on this route which is bigger than root.\n";
            int nodes[2] = {m_root, cur};
            if(m_arr[m_root] > m_arr[cur]){
                while(m_arr[m_root] > m_arr[cur]){
                    cout << "Current node, which value is " << m_arr[cur] << " is less than root, which value is " << m_arr[m_root] <<". So we exclude it from the route.\n";
                    printHeap(nodes, 2);
                    cur = way[i-1];
                    nodes[1] = cur;
                    i -= 1;
                }
            }
            else{
                i -= 1;
                cur = way[i];
            }
```

```
                nodes[1] = cur;
                cout << "Current node, which value is " << m_arr[cur] << " is more than root, which value is " << m_arr[m_root] <<"\n";
                printHeap(nodes, 2);

                cout << "Save current node in buffer variable, replace the value of current node with the value of the root and exclude this node from the route.\n";
                buf = m_arr[cur];
                m_arr[cur] = m_arr[m_root];
                printHeap(nodes, 2);
                i -= 1;
                int add_buf = buf;

                cout << "Now we are going to shift all remaining nodes in the route, which we have got early, to the one level upper. The nearest node which will be replaced with previous saved in buffer variable node value, which is " << buf << "\n";
                cout << "Remaining route:\n";
                printHeap(way, i+1);

                while(cur > m_root){
                    cout << "The value of current will be saved in buffer variable and then will be replaced with the value of its biggest son, which is " << add_buf << " and has been saved in the another buffer variable.\n";
                    cur = way[i];
                    nodes[0] = cur;
                    printHeap(nodes, 1);
                    buf = m_arr[cur];
                    m_arr[cur] = add_buf;
                    add_buf = buf;
                    i -= 1;
                    nodes[0] = cur;
                    printHeap(nodes, 1);
                }
                cout << "The heap root has been reached. Shifting nodes to upper level has successfully ended.\n";
            }
            else{
                cout << "Only root is reamining in the heap, so it is already heap. No sifting needed.\n";
            }

            cout << "Upward sifting has ended.\n";
            cout << "-----------------------------------------------\n";
        }

        void upwardSiftSort(){//сортировка с использованием восходящей просейки
            cout << "Heapsort with upward sifting.\n This sort using the upward sifting to restore heap after draging max element.\n";
```

```
            cout << "-----------------------------------------------------------------
--------------------\n";
            while(m_size){;
                dragMax();
                if(!m_size){
                    cout << "It was the last node in the heap!\n";
                    break;
                }
                cout << "\033[1;30;43mHeap is corrupted after draging maximal
element!\033[0m Let's restore our heap with sifting it down.\n";
                upwardSift();
                cout << "\033[1;30;42mThe heap restored!\033[0m\nCurrent state of
the heap:\n";
                printHeap(nullptr, -1);
                cout << "\n\n\n\n";
            }
            cout << "-----------------------------------------------------------------
--------------------\n";
            cout << "Sort has successfully ended!\n";
            printAsArr(false);
        }

        void siftDownSort(){//сортировка с использованием просейки сверху-вниз
            cout << "Heapsort with sifting down.\n This sort using the sifting down
to restore heap after draging max element.\n";
            cout << "-----------------------------------------------------------------
--------------------\n";
            while(m_size){;
                dragMax();
                if(!m_size){
                    cout << "It was the last node in the heap!\n";
                    break;
                }
                cout << "\033[1;30;43mHeap is corrupted after draging maximal
element!\033[0m Let's restore our heap with sifting it down.\n";
                siftDown(m_root);
                cout << "\033[1;30;42mThe heap restored!\033[0m\nCurrent state of
the heap:\n";
                printHeap(nullptr, -1);
                cout << "\n\n\n\n";
            }
            cout << "-----------------------------------------------------------------
--------------------\n";
            cout << "Sort has successfully ended!\n";
            printAsArr(false);
        }

        void printArr(){//выводит массив
            for(int i = 0; i < m_arr_size; i++){
                cout << m_arr[i] << ' ';
```

```
            }
            cout << '\n';
        }

        void printAsArr(bool is_col_first){//выводит кучу как массив
            cout << "It is heap as array. The green part is actually the heap, white is sorted sequence and cyan is the old root, which just has been added to the sorted sequence: ";
            for(int i = 0; i < m_arr_size; i++){
                cout << "\033[1;30;42m";
                if(i >= m_size){
                    if(i == m_size && is_col_first){
                        cout << "\033[1;30;46m";
                    }
                    else{
                        cout << "\033[1;30;47m";
                    }
                }
                cout << m_arr[i] << ' ';
                cout << "\033[0m";
            }
            cout << '\n';
        }

        void printNode(int node_value, int step, bool is_col, char side){//выводит узел в консоль
            if(side == 'l' || side == 't'){
                for(int i = 0; i < step; i++){
                    cout << ' ';
                }
            }
            else{
                for(int i = 0; i < step; i++){
                    cout << '_';
                }
            }

            if(is_col){
                cout << "\033[1;30;47m";
            }
            cout.setf(ios::left);
            cout.width(4);
            if(side != 'r' && side != 't'){
                cout.fill('_');
            }
            cout << node_value;
            cout.unsetf(ios::left);
            cout.fill(' ');
            cout << "\033[0m";
```

```
            if(side == 'r' || side == 't'){
                for(int i = 0; i < step; i++){
                    cout << ' ';
                }
            }
            else{
                for(int i = 0; i < step; i++){
                    cout << '_';
                }
            }
        }

        void printHeap(int* color_nodes, int col_size){//выводит кучу в консоль, 
как дерево
            if(m_size == 0){
                cout << "Empty heap!\n";
                return;
            }
            int lev = 0;
            int height = calcHeight();
            bool is_col = false;
            int j = 0;
            char side = 0;

            int step = 0;
            for(int i = 0; i < m_size; i++){
                step = int(3*2*int(pow(double(m_d),double(height-
1)))/(2*int(pow(double(m_d),double(lev))))-2);

                is_col = false;
                if(j < col_size){
                    if(i == color_nodes[j]){
                        is_col = true;
                        j += 1;
                    }
                }

                if(lev == 0){
                    printNode(m_arr[i], step, is_col, 't');
                    lev += 1;
                    cout << "|||||";
                    cout << '\n';
                    continue;
                }

                if(i%m_d == 1){
                    side = 'l';
                }
                if(i%m_d == 0 || i == m_size-1){
                    side = 'r';
```

```
                }

                if(m_d == 1 || (i == m_size-1 && i%m_d == 1)){
                    side = 't';
                }
                printNode(m_arr[i], step, is_col, side);
                side = 0;

                if(i%((int)(double(1.0-pow(double(m_d), double(lev+1)))/double(1-
m_d))-1) == 0 || m_d == 1){
                    lev += 1;
                    cout << "|||||";
                    cout << '\n';
                }
            }
            cout << "\n";
        }

        int getHeight(){//возвращает высоту дерева
            return calcHeight();
        }

        ~Dheap(){//деструктор; очищает память выделенную под массив-кучу
            delete[] m_arr;
        }
};


int main(){
    ifstream fin;
    char command;
    string fname;
    Dheap* heap = nullptr;
    bool isD = true;

    while(1){
        cout << "Input 's' to start the program or input 'q' to stop the
program:\n";
            cin >> command;
            switch (command){
                case 'q':
                    cout << "You choose to end the programm!\n";
                    return 0;

                case 's':
                    cout << "Input the path to data file:\n";
                    cin >> fname;
                    fin.open(fname, ifstream::in);
                    if(!fin.is_open()){
                        cout << "Opening file with test data failed! Try again!\n";
```

```
                    break;
                }

                heap = new Dheap;
                isD = heap->readHeapFromFile(fin);
                fin.close();
                if(!isD){
                    cout << "Error in input data";
                    delete heap;
                    break;
                }

                cout << "Input array:\n";
                heap->printArr();
                cout << "Fistly we need to make heap from starting array.\n";
                heap->makeHeap();
                cout << "\n\n\n";

                cout << "Choose sifting type. 'd' for sifting down and 'u' for
upward sifting:\n";
                cin >> command;
                if(command == 'd'){
                    cout << "You choose sifting down sort.\n";
                    heap->siftDownSort();
                }
                else{
                    if(command == 'u'){
                        cout << "You choose upward sifting sort.\n";
                        heap->upwardSiftSort();
                    }
                    else{
                        cout << "Error command!\n";
                        delete heap;
                        break;
                    }
                }

                cout << "\n\n\n\n";
                cout << "Sorted array:\n";
                heap->printArr();
                cout << "\n\n\n\n";
                delete heap;
                break;

            default:
                cout << "Error command! Try again!\n";
                break;

        }
    }
```

```
    return 0;
}
```