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 Люоимов 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 п 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

АННОТАЦИЯ

В курсовой работе происходит сортировка массива. Программа 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	5
1.	Задание	9
2.	Описание классов, структур, функций	9
3.	Описание алгоритма сортировки	14
4.	Тестирование	15
5.	Демонстрация	113
	Заключение	115
	Список использованных источников	116
	Приложение. Исходный код программы.	117

ВВЕДЕНИЕ

п-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 п потомков. Это правило не относиться к листьям и к самому правому не листу на предпоследнем уровне. Последний уровень заполняется слева направо без Главной особенностью сортировки n-арной кучей «дырок». является независимость временной сложности сортировки от сложности сортируемого массива, равная O(NlogN), где N – размер массива. Из этого следуют основные преимущество и недостаток сортировки п-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 п-арной кучей.

- 1. Формируем из массива п-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 п-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: OT текущего 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 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 п-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 эскейппоследовательности, 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 п-арного дерева количество узлов на і-ом $(0 \le i < \text{количество уровней})$ 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 і. Найдём минимальный step_i, то есть step для i-го уровня. $step_i = \frac{S}{2\pi n^i} - 2$. 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 $(0\leq \mathrm{i}<$ 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 ктото 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 п-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 п-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 п. 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 п количество «выигранных» сравнений будет играть всё меньше роли.

Так же заметим, что с увеличением количества п количество обменов за одну просейку уменьшается, но увеличивается количество сравнений. Оценим временную сложность алгоритма для кучи порядка п, 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 (п 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	Входные данные	Выходные данные
теста		
1	2 13 67 5 -9 67 3456	Input array:
	67 0 0 1 2 9	13 67 5 -9 67 3456 67 0 0 1 2 9
	Выбрана просейка	Fistly we need to make heap from starting array.
	сверху-вниз	
		Let's make heap!
		The current state of heap is:

13
675
-967 345667
00 12 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 element in this root or its
sons!
The value of root is 3456
13
675
-967 345667
0
9 is less or equal than current maximum value,
which is 3456
13
675
-967 345667 00 12 9
Summary, the value of maximal element of root
and its leaf is 3456
13

675	
-967 345667	
00 12 9	
The root, which value is 3456 is more than his	S
sons, so this subtree is heap!	
13	
00 12 9	
It is the sifting down.	
· 	
Let's find the maximal element in this root or	its
	113
sons!	
The value of root is 67	
13	
675	
-967 345667	
00 12 9	
1 is less or equal than current maximum value	·,
which is 67	
13	
	Ш
U/	
0 67 2456	
-967 345667	
-967 345667 00 12 9	
-967 345667 00 12 9 2 is less or equal than current maximum value	

-967 345667 D0 12 9 Summary, the value of maximal element of room of the state of the sta	-967 345667 00 12 9 Summary, the value of maximal element of rocand its leaf is 67 13 6767 34565 -967 34565 The root, which value is 67 is more than his socator is subtree is heap! 13 675 -967 34565	-967 345667 D0 12 9 Summary, the value of maximal element of room on dits leaf is 67 13 675 -967 345667 D0 12 9 The root, which value is 67 is more than his so to this subtree is heap! 13 675 -967 345667 D0 12 9 The sifting down. The value of root is -9 The value of root is -9					
00 12 9 Summary, the value of maximal element of roomed its leaf is 67 13 675 -967 345667 2 9 The root, which value is 67 is more than his so this subtree is heap! 13 675 -967 34565 -967 345667	00 12 9 Summary, the value of maximal element of rocand its leaf is 67 13 675 -967 345667 5 6767 345667 067 345667 067 345667 0	D	67_			_5	
Summary, the value of maximal element of room of the leaf is 67 13 13 13 13 13 14 67 -9 67 14 67 15 17 18 18 19 19 19 19 19 19 19 19	Summary, the value of maximal element of rocand its leaf is 67 13 67	13	-9	67	3456		_67
13 675 -967 345667 e root, which value is 67 is more than his so this subtree is heap! 13 675 -967 34565	13 67	13 67	0	12	9		
13 675 -967 345667 e root, which value is 67 is more than his so this subtree is heap! 13 675 -967 345667	13 67	13 67	mmary,	the value of	maximal ele	ement	t of 1
675 -967	67	67	nd its leaf	f is 67			
-967 345667 00 12 9 The root, which value is 67 is more than his so this subtree is heap! 13 675 -967 345667	-967 345667 00 12 9	-967		13			
00 12 9 the root, which value is 67 is more than his so this subtree is heap! 13 675 -967 345667	00 12 9	0 1 2 9	67_			_5	
The root, which value is 67 is more than his so this subtree is heap! 13 675 -967 345667	The root, which value is 67 is more than his so this subtree is heap! 13 675 -967 345667 00 12 9	The root, which value is 67 is more than his so this subtree is heap! 13 67	-9	67	3456		_67
o this subtree is heap! 13 675 -967 345667	o this subtree is heap! 13 675 -967 345667 00 12 9	13 675 -967 345667 00 12 9 t is the sifting down. Let's find the maximal element in this root or sons! The value of root is -9)0	12	9		
o this subtree is heap! 13 675 -967 345667	so this subtree is heap! 13 675 -967 345667 00 12 9	13 675 -967 345667 00 12 9 t is the sifting down. Let's find the maximal element in this root or sons! The value of root is -9					
13 675 -967 345667	13 675 -967 345667 00 12 9	13 67	The root, v	which value	is 67 is more	than	his
675 -967 345667	675 -967 345667 00 12 9	675 -967 345667 00 12 9 t is the sifting down. Let's find the maximal element in this root or sons! The value of root is -9	o this sub	tree is heap!			
-967 345667	-967 345667 00 12 9	-967 345667 D0 12 9 It is the sifting down. Let's find the maximal element in this root or sons! The value of root is -9		13			
	00 12 9	o 12 9 is the sifting down. et's find the maximal element in this root or sons! the value of root is -9	67_			_5	
	00 12 9	t is the sifting down. Let's find the maximal element in this root or sons! The value of root is -9	-9	67	3456		_67
<u> </u>		t is the sifting down. Let's find the maximal element in this root or sons! The value of root is -9					
	It is the sifting down.	Let's find the maximal element in this root or it ons! The value of root is -9	0 0	12	9		
	It is the sifting down.	Let's find the maximal element in this root or ons! The value of root is -9					
		ons! The value of root is -9					
t is the sifting down.		ons! The value of root is -9		ting down.			
		ne value of root is -9	is the sif		elememt in	 this ro	oot (
et's find the maximal element in this root or			is the sif		elememt in	 this re	oot o
et's find the maximal element in this root or one!	ons!		is the sif	the maximal	elememt in	 this re	oot o
et's find the maximal element in this root or sons! he value of root is -9 13	ons! he value of root is -9 13		is the sif et's find tons! he value	the maximal of root is -9			oot (
et's find the maximal element in this root or sons! The value of root is -9	ons! The value of root is -9 13	675	et's find tons!	the maximal of root is -9			oot (
et's find the maximal element in this root or sins! The value of root is -9	ns! ne value of root is -9 13 675		is the sif et's find t ns! ne value	the maximal of root is -9		_5	
t's find the maximal element in this root or sas! e value of root is -9 13 675	13 675 -967 345667	-967 345667	s the sif t's find the s! e value	the maximal of root is -9 1367	3456	_5	

9				
	13			
67			_5	
-9	67	3456	67	
00	12	9		
0 is less or	equal than o	current maxi	mum val	ue,
which is 0				
	13			
67			_5	
-9	67	3456	67	
00	12	9		
Summary,	the value of	maximal ele	ement of	root
and its leaf	is 0			
	13			
67			_5	
-9	67	3456	67	
00	12	9		
The son, w	hich value i	s 0, is bigger	r than fat	her,
which valu	e is -9! So l	et's change t	heir valu	e.
	13			
-9	67	3456	67	
00	12	9		
Values suc	cessfully ch	anged! One	more ste	p to
make heap	has been do	one!		
	13			
67			_5	

	-9	_0	12	2 9				
			n has en					
			ng dowr	1.				
	Let's fi			nal ele	memt in	thic 1	ont o	ar ita
	sons!	ina tii	C maxim	iai Cic		unsı	.001 0	1 103
		ilue o	f root is	5				
	1110 70	iiuc o	13	J				
		67				5		
	0_				3456		67	
	-9		12					
	3456 is	s mor	e than cu	urrent	maximu	m va	lue, w	vhich
	is 5							
			13					
	(67				_5		
	0		67		3456		_67	
	-9	_0	12	2 9				
	67 is le	ess or	equal th	an cu	rrent max	ximu	m val	ue,
	which	is 34:	56					
			13					
	(
	0		67		3456		_67	
			12					
				of ma	iximal el	emen	t of r	oot
	and its	leaf	is 3456					
			13					

67			_5	
0	67	3456	67	
-90	12	9		
The son, wh	nich value is	3456, is big	gger than	
father, whic	h value is 5	! So let's cha	ange their	
value.				
	13			
67			_5	
0	67	3456	67	
-90	12	9		
Values succ	essfully cha	inged! One	more step	to
make heap l	has been do	ne!		
	13			
67			_3456	
0	67	5	67	
-90	12	9		
Let's find th	e maximal	elememt in t	his root o	r its
sons!				
The value of	f root is 5			
	13			
67			_3456	
0	67	5	67	
-90	12	9		
9 is more th	an current r	naximum va	lue, whic	h is 5
	13			
67			_3456	

0		67	5		67	
-9	_0	12	9			
Summ	ary, 1	the value	of maxin	nal ele	ement of ro	oot
and its	s leaf	is 9				
		13				
	67				_3456	
0		67	5		67	
-9	_0	12	9			
The so	on, w	hich value	e is 9, is	bigger	than fathe	er,
which	valu	e is 5! So	let's cha	nge th	eir value.	
		13				
	67				_3456	
0		67	5		67	
-9	_0	12	9			
Values	s succ	cessfully	changed!	One	more step	to
make l	heap	has been	done!			
		13				
	67				_3456	
0		67	9		67	
-9	_0	12	5			
Sifting	g dow	n has end	led.			
It is th	e sift	ing down	•			
	ind th	ne maxim	al eleme	mt in	this root or	its
sons!						

The v	aiue (of roc	ot is o	/			
		13					
	67					3456	
0_			_67	9		67	
-9	0	1	2	5			
0 is 16	ess or	equa	l than	curre	nt max	imum val	ue,
which	is 67	7					
		13					
	67					3456	
0_			_67	9		67	
-9	0	1	2	5			
67 is	less o	or equ	al thai	n curr	ent ma	ximum va	ılue,
which	n is 67	7					
		13					
	67					3456	
0_			_67	9		67	
-9	0	1	2	5			
Sumn	nary,	the v	alue o	f max	imal e	lement of	root
and it	s leaf	f is 67	•				
		13					
	67					3456	
0_						67	
-9	0	1	2	5			
				_			
 The r	oot, v	 vhich	value		is mor	e than his	sons,
				is 67	is mor	e than his	sons,
			s heap	is 67			sons,
	s subi	tree is	s heap	is 67 !			sons,

-9	0	1	2	5			
It is the	he sift	ting d	own.				
Let's	find t	he ma	aximal	eleme	mt in	this root or	rits
sons!							
The v	alue o	of roc	ot is 13	}			
		13					
	67					_3456	
0_			_67	9_		67	
-9	0	1	2	5			
67 is	more	than	curren	t maxii	mum י	value, which	ch is
13							
		13					
	67					_3456	
0_			_67	9_		67	
-9	0	1	2	5			
3456	is mo	re tha	an curr	ent ma	ximuı	m value, w	hich
is 67							
		13					
	67					_3456	
0_			_67	9		67	
-9		1	2	5			
Sumr	nary,	the va	alue of	f maxin	nal ele	ement of ro	oot
and it	s leaf	is 34	56				
		13					
	67					_3456	
0_			_67	9		67	

-9	0	1	2	5				
The se	on, w	hich v	alue i	s 3456,	is big	ger t	han	
father	, whi	ch val	ue is 1	3! So 1	et's ch	ange	thei	r
value								
		13						
	67					3450	5	
0			_67	9			_67	
			2					
Value	es suc	cessfu	lly ch	anged!	One n	nore	step	to
make	heap	has be	een do	ne!				
		345	56					
	67					13		
0			_67	9			_67	
-9	0	1	2	5				
Let's	find t	he ma	ximal	elemer	nt in tl	nis ro	oot o	r its
sons!								
The v	alue	of root	t is 13					
		345	56					
	67					13		
0			_67	9			_67	
-9	0	1	2	5				
9 is le	ess or	equal	than c	urrent	maxin	num	value	e,
which	is 13	3						
		345	56					
	67					13		
0			_67	9			_67	

-9	0	12	2 5			
67 is	more	than curr	ent maxi	mum valu	e, whi	ich is
13						
		3456		IIIII		
	67			13		IIIII
0_		67	9_		67	
-9	0	12	2 5			
Sumr	nary,	the value	of maxin	mal eleme	nt of 1	oot
and it	ts leaf	is 67				
		3456		IIIII		
	67			13		
0_						
-9		12				
The s	son, w	hich valu	e is 67, i	s bigger th	nan fat	ther,
whicl	h valu	e is 13! S	o let's ch	nange their	value	e.
		3456		IIIII		
	67			13		IIIII
0_		67	9_		67	
-9	0	12	2 5			
Value	es suc	cessfully	changed	! One mor	e step	to
make	heap	has been	done!			
		3456		IIIII		
	67			67		IIIII
0_		67	9_		13	
-9	0	12	2 5			
Siftin	ıg dov	vn has en	ded.			
					-	
J						

Building heap has been ended!
The current state of heap is:
3456
6767
067 913
-90 12 5
Choose sifting type. 'd' for sifting down and 'u' fo
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 roo
because we are working with max-heap. Save the
root value in buffer variable
3456
6767
067 913
-90 12 5
It is heap as array. The green part is actually the

heap, white	e is sorted se	quence and c	yan is the old
root, which	n just has bee	en added to th	e sorted
sequence:	3456 67 67 0	67 9 13 -9 0	1 2 5
Make the v	alue of the l	ast element in	heap the root
value. Eve	ntually, put t	he old root va	alue into the
last positio	n in heap an	d decrease the	e size of the
heap. So w	e have alread	dy sorted elen	nents after the
heap in the	array that is	storing our e	lements as
heap and s	orted sequen	ce.	
Current sta	ate of heap as	s tree:	
	5		
67		(67
0	67	9	13
-90	12		
Current sta	ate of heap as	s array:	
It is heap a	s array. The	green part is	actually the
heap, white	e is sorted se	quence and cy	yan is the old
root, which	n just has bee	en added to th	e sorted
sequence:	5 67 67 0 67	9 13 -9 0 1 2	3456
Heap is co	rrupted after	draging maxi	mal element!
Let's restor	re our heap w	vith sifting it o	down.
It is the sif	ting down.		
Let's find t	he maximal	elememt in th	is root or its
sons!			
The value	of root is 5		
	5		

57_					67		
		67	9			13	
_0	1	2					
ore	than	currer	nt max	imum	ı valu	e, wh	ich i
	5						
57_					67		
		67	9.			13	
_0	1	2					
ss o	or equ	ial tha	n curr	ent ma	aximı	ım va	lue,
is 67	7						
	5						
57_					67		
		67	9			13	
_0	1	2					
ary,	the v	alue o	f max	imal e	eleme	nt of 1	roo
leaf	f is 6	7					
	5						
57					67		
		67	9.			13	
_0	1	2					
n, w	hich	value	is 67,	is big	ger th	nan fa	theı
	5						
57					67		
57 <u> </u>		67				13	
		_0 1	67676767676767676767676767676767	67 9			

make h	neap l	nas been d	done!				
		67					
4	5				_67		
0		67	9_			_13	
-9	_0	12					
Let's fi	nd th	e maxima	al eleme	mt in	this r	oot o	r its
sons!							
The va	lue o	f root is 5					
		67					
4	5				_67		
0		67	9_			_13	
-9	_0	12					
0 is les	s or e	equal than	current	maxi	mum	valu	e,
which	is 5						
		67					
4	5				_67		
0		67	9			_13	
-9	_0	12					
67 is m	nore t	han curre	nt maxii	num י	value	, whi	ch is
5							
		67					
4	5				_67		
0		67	9_			_13	
-9	_0	12					
Summa	ary, tl	he value o	of maxin	nal ele	emen	t of r	oot
and its	leaf i	is 67					
		67					

067		5_			67	
The son, which value is 67, is bigger than father, which value is 5! So let's change their value. 67 5		0	67	9	13	3
which value is 5! So let's change their value. 67 5		-90	12			
which value is 5! So let's change their value. 67 5						
67 567 067 913 -90 12 Values successfully changed! One more step to make heap has been done! 67 6767 05 913 -90 12 Let's find the maximal element in this root or its sons! The value of root is 5 67 6767 05 913 -90 12 I is less or equal than current maximum value,		The son,	which value	is 67, is big	ger than fa	ather,
5		which va	lue is 5! So le	et's change	their value	e.
067			67			
-90 12 Values successfully changed! One more step to make heap has been done! 67 6767 05 913 -90 12 Let's find the maximal element in this root or its sons! The value of root is 5 67 6767 05 913 -90 12 1 is less or equal than current maximum value,		5_			67	
Values successfully changed! One more step to make heap has been done! 67 67		0	67	9	13	3
make heap has been done! 67 67		-90	12			
67 67		Values su	accessfully cl	nanged! On	e more ste	p to
6767 05 913 -90 12 Let's find the maximal element in this root or its sons! The value of root is 5 67 6767 05 913 -90 12 1 is less or equal than current maximum value,		make hea	ap has been d	one!		
05 913 -90 12 Let's find the maximal element in this root or its sons! The value of root is 5 67 6767 05 913 -90 12 1 is less or equal than current maximum value,			67			
-90 12 Let's find the maximal element in this root or its sons! The value of root is 5 67 6767 05 913 -90 12 1 is less or equal than current maximum value,		67_			67	
Let's find the maximal element in this root or its sons! The value of root is 5 67 6767 05 913 -90 12 1 is less or equal than current maximum value,		0	5	9	13	
sons! The value of root is 5 67 6767 05 913 -90 12 1 is less or equal than current maximum value,		-90	12			
sons! The value of root is 5 67 6767 05 913 -90 12 1 is less or equal than current maximum value,						
sons! The value of root is 5 67 6767 05 913 -90 12 1 is less or equal than current maximum value,						
The value of root is 5 67 6767 05 913 -90 12 1 is less or equal than current maximum value,		Let's find	the maximal	l elememt ii	n this root	or its
67 6767 05 913 -90 12 1 is less or equal than current maximum value,		sons!				
6767 05 913 -90 12 1 is less or equal than current maximum value,		The value	e of root is 5			
05 913 -90 12 1 is less or equal than current maximum value,			67			
-90 12 1 is less or equal than current maximum value,		67_			67	
1 is less or equal than current maximum value,		0	5	9	13	
		-90	12			
which is 5		1 is less of	or equal than	current max	ximum val	lue,
		which is	5			
67			67			
6767		67_			67	

	0		5	9		13	
	-9	_0	12				
	2 is le	ss or e	equal than co	ırrent	maxim	ıum valu	ie,
	which	is 5					
			67				
		67				67	
	0		5	9		13	IIIII
	-9	_0	12				
	Summ	nary, t	he value of	maxin	nal eler	nent of 1	root
	and its	s leaf	is 5				
			67				
		67				67	
	0		5	9		13	IIIII
	-9	_0	12				
	The ro	oot, w	hich value is	s 5 is r	nore th	an his s	ons,
	so this	subtr	ree is heap!				
			67				
		67				67	
	0		5	9		13	
	-9	_0	12				
	The he	eap re	stored!				
	Curre	nt stat	e of the heap	o :			
			67				
		67				67	
	0		5	9		13	IIIII
	-9	_0	12				
L							

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: 67_____67 -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	n added to	the s	orted	
sequence: 2 67 67 0 5 9	13 -9 0 1	67 34	56	
Heap is corrupted after of Let's restore our heap with	lraging m	axima it dov		ment!
It is the sifting down.				
Let's find the maximal e sons!	lememt in	this 1	root o	r its
The value of root is 2	11111			
2		<i>(</i> 7		11111
67				
05	9		_13	
-90 1		1	1.	1 .
67 is more than current in	maximum	varue	e, wnı	cn 1s
2				
2		6 7		
67		67	1.0	
05	9		_13	
-90 1				
67 is less or equal than of	current ma	ximu	m val	ue,
which is 67				
2				
67		67		
05	9		_13	
-90 1				
Summary, the value of r	naximal e	lemer	nt of r	oot
and its leaf is 67				

67	2
-9	6767
The son, which value is 67, is bigger than father, which value is 2! So let's change their value. 2 67	05 913
which value is 2! So let's change their value. 2 67	-90 1
which value is 2! So let's change their value. 2 67	
2 67	The son, which value is 67, is bigger than father,
67	which value is 2! So let's change their value.
05 913 -90 1 Values successfully changed! One more step to make heap has been done! 67 267 -90 1 Let's find the maximal element in this root or its sons! The value of root is 2 67 267 05 913 -90 1 0 is less or equal than current maximum value, which is 2	2
-90 1 Values successfully changed! One more step to make heap has been done! 67 267 -90 1	6767
Values successfully changed! One more step to make heap has been done! 67 2	05 913
make heap has been done! 67 2	-90 1
2	Values successfully changed! One more step to
2	make heap has been done!
0	67
-90 1 Let's find the maximal element in this root or its sons! The value of root is 2 67 267 05 913 -90 1 0 is less or equal than current maximum value, which is 2	267
Let's find the maximal element in this root or its sons! The value of root is 2 67 2	05 913
sons! The value of root is 2 67 2	-90 1
sons! The value of root is 2 67 2	
sons! The value of root is 2 67 2	
The value of root is 2 67 267 05 913 -90 1 0 is less or equal than current maximum value, which is 2	Let's find the maximal element in this root or its
67 267 05 913 -90 1 0 is less or equal than current maximum value, which is 2	sons!
2	The value of root is 2
05 913 -90 1 0 is less or equal than current maximum value, which is 2	67
-90 1 0 is less or equal than current maximum value, which is 2	267
0 is less or equal than current maximum value, which is 2	05 913
which is 2	-90 1
	0 is less or equal than current maximum value,
67	which is 2
	67

	2					_67		
0_			_5	9			13	
-9	0	1						
5 is n	nore tl	han cu	rrent n	naxim	um va	alue, v	whic	h is 2
		67						
	2					_67		
0_			_5	9			13	
-9	0	1						
Sumi	mary,	the va	lue of	maxin	nal ele	ement	of r	oot
and i	ts leaf	is 5						
		67						
	2					_67		
0				0			10	11111
0_			_5	9			13	
-9	0	1	_5	9			13	IIIII
-9 	0		_5	9			13	IIIII
		1	_5 ralue is					
The s	son, w	1 hich v		5, is 1	oigger	than	fath	
The s	son, w	1 hich v	alue is	5, is 1	oigger	than	fath	
The s	son, w	1 hich v e is 2!	alue is	5, is 1	oigger	than	fath	
The s	son, w	1 hich v e is 2!	alue is So let	5, is l	oigger	than eir va _67	fath	er,
The s	son, which was a second	1 hich v e is 2! 67	alue is So let	5, is l	bigger nge th	than eir va _67	fath lue.	er,
The s which	son, when the second se	1 hich v e is 2! 67	alue is So let	5, is because 5, is character 5, is character 5, is because 5, is becaus	oigger nge th	than eir va _67	fath lue.	er,
The s which	20 es suc	hich ve is 2! 67 1 cessfu	So let	9	oigger nge th	than eir va _67	fath lue.	er,
The s which	20 es suc	hich ve is 2! 67 1 cessfu	So let 5	9	oigger nge th	than eir va _67	fath lue.	er,
The s which	20 es suc	hich ve is 2! 67 1 cessfue has been	So let 5	9	oigger nge th	than eir va _67	fath lue.	er,
The s which	son, who walus 20 es suce heap	hich ve is 2! 67 cessfue has be 67	So let 5	9	oiggeringe th	than eir va _67 more	fath lue.	er, to
The swhich	son, who walus 20 es suce heap	hich ve is 2! 67 cessfue has be 67	So let So let Ily cha	9	oiggeringe th	than eir va _67 more	fath lue.	er,
The swhich	son, who walue 20 es such the heap	hich ve is 2! 67 cessfue has be 67	So let So let Ily cha	9	oiggeringe th	than eir va _67 more	fath lue.	er, to
The swhick	son, who walue 20 es such the heap	hich ve is 2! 67 cessfue has be 67	So let So let Ily cha	9	oiggeringe th	than eir va _67 more	fath lue.	er, to

Let's	find the	he maxima	ıl elemei	mt in tl	nis root	or its
sons	!					
The	value (of root is 2				
		67				
	5				67	
0_		2	9		13	
-9_	0	1				
1 is 1	less or	equal than	current	maxin	num val	ue,
	ch is 2					
		67				
	5				67	
0_		2	9		13	
-9_	_0	1				
		the value o	of maxin	nal elei	ment of	root
	its leaf					
		67				
	5				67	
0		2	9		13	
-9	0	1				11111
The	root, w	which value	e is 2 is 1	more th	nan his s	ons,
		tree is heap				
		67				
	5				67	
0_		2	9		13	
-9_	0	1				
The	heap re	estored!				
		te of the he	eap:			
	37					

67
567
02 913
-90 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
567
02 913
-90 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
567

0	2	0		11111
0	2	9	13	
-90				
Current stat	e of heap as	array:		
It is heap as	array. The	green part i	s actually	the
heap, white	is sorted sec	quence and	cyan is th	e old
root, which	just has bee	n added to	the sorted	
sequence: 1	5 67 0 2 9 1	3 -9 0 67 6	7 3456	
Heap is cor	rupted after	draging ma	ximal eler	ment!
Let's restore	e our heap w	ith sifting i	t down.	
It is the sift	ing down.			
Let's find th	ne maximal e	elememt in	this root o	r its
sons!				
The value of	f root is 1			
	1			
5			67	
0	2	9	13	
-90				******
5 is more th	an current n	naximum v	alue, whic	h is 1
	1			
5			67	
	2	9		
-9 0				11111
	han current	maximum	value, whi	ch is
5			- , ···	
-	1			
5	1	11111	67	
5			_0,	11111

0		2	9		13	
-9	0					
Summ	nary, th	e value of	maxin	nal elem	nent of r	root
and its	s leaf is	s 67				
		1				
	5			6	57	
0		2	9		13	IIIII
-9	0					
		_				
The so	on, whi	ich value i	s 67, is	bigger	than fat	her,
which	value	is 1! So le	t's chai	nge thei	r value.	
		1				
	5			6	57	
0		2	9		13	
-9	0					
Value	s succe	essfully ch	anged!	One me	ore step	to
make	heap h	as been do	one!			
		67				
	5			1		
0		2	9		13	
-9	0					
		-				
Let's f	find the	e maximal	elemei	mt in thi	is root c	or its
sons!						
The va	alue of	root is 1				
		67				
	5			1		
0		2	9		13	

-90				
9 is more	than current	maximum va	alue, whi	ch is 1
	67			
5_			_1	
0	2	9	13	
-90				
13 is mor	re than current	maximum v	value, wh	nich is
9				
	67			
5_			_1	
0	2	9	13	
-90				
Summary	y, the value of	maximal ele	ement of	root
and its le	af is 13			
	67			
5_			_1	
0	2	9	13	
-90				
The son,	which value is	s 13, is bigge	er than fa	ther,
which va	lue is 1! So le	t's change th	eir value	•
	67			
5_			_1	
0	2	9	13	
-90				
Values su	accessfully ch	anged! One	more ste	p to
make hea	ap has been do	one!		
	67			
5_			_13	IIIII

	0	2	9	1	
	-90				
	_	n has ended			
	The heap re				
	Current stat	e of the hear	p:		
		67			
		2	9	1	
	-90				
	Exclude the	node with b	oiggest value	. It is the	root
	because we	are working	g with max-h	eap. Save	e the
	root value i	n buffer vari	able		
		67			
	5			.13	
	0	2	9	1	
	-90				
	It is heap as	array. The	green part is	actually	the
	heap, white	is sorted see	quence and c	yan is the	e old
	root, which	just has bee	n added to th	ne sorted	
	sequence: 6	7 5 13 0 2 9	1 -9 0 67 67	3456	
	Make the va	alue of the la	ast element in	n heap th	e root
	value. Even	tually, put tl	ne old root v	alue 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
513
01
-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 element in this root or its
sons!
The value of root is 0
0
513
02 91
-9
5 is more than current maximum value, which is 0
0

513
02 91
-9
13 is more than current maximum value, which is
5
0
513
02 91
-9
Summary, the value of maximal element of root
and its leaf is 13
0
513
02 91
-9
The son, which value is 13, is bigger than father,
which value is 0! So let's change their value.
0
513
02 91
-9
Values successfully changed! One more step to
make heap has been done!
13
50
02 91
-9

Let's find the maximal element in this root	or its
sons!	01 165
The value of root is 0	
13	
50	
02 91	
-9	
9 is more than current maximum value, wh	ich is 0
13	
50	
02 91	
-9	
1 is less or equal than current maximum va	lue,
which is 9	
13	
5 0	
0 2 9 1	
-9	11111
	front
Summary, the value of maximal element of	1001
and its leaf is 9	
13	
50	
02 91	
-9	
The son, which value is 9, is bigger than fa	ther,
which value is 0! So let's change their value	e.
13	

		5			()	
	0_		2	9		1	
	-9						
7	Value	es succ	essfully c	hanged	! One m	ore ste	ep to
			nas been o				
		-	13				
		5			9)	
	0		2	0		1	
	-9						
9	Siftin	g dow	n has end	ed.			
-	The h	iean re	stored!				
			e of the h	ean.			
	Curre	ni stati	13	cap.	IIIII		
		5	13)	11111
	0	5				1	
	0_		2	0		1	
	-9						
-							
			node wit				
l	becau	ise we	are work	ing with	max-he	ap. Sa	ve the
1	root v	alue ir	n buffer v	ariable			
			13				
		5			9)	
1	0		2	0		4	

-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:

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 element in this root or its sons!

The value of root is -9 -9 0____1 5 is more than current maximum value, which is -9 -9 0_____1 ||||| 9 is more than current maximum value, which is 5 -9 5____9 0_____1 ||||| Summary, the value of maximal element of root and its leaf is 9 -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
59
01
Let's find the maximal element in this root or its
sons!
The value of root is -9
9
59
01
0 is more than current maximum value, which is -
9
9
59
01
1 is more than current maximum value, which is 0
9
59
01
Commons the velve of meximal element of next
Summary, the value of maximal element of root
and its leaf is 1
9
40

	59
	02 01
	The son, which value is 1, is bigger than father,
	which value is -9! So let's change their value.
	9
	59
	02 01
	Values successfully changed! One more step to
	make heap has been done!
	9
	51
	02 09
	<u> </u>
	Sifting down has ended.
	The heap restored!
	Current state of the heap:
	9
	51
	0
	02 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

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:

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
It is the sifting down.
Let's find the maximal element in this root or its
sons!
The value of root is -9
-9
51
02 0
5 is more than current maximum value, which is -
9
-9
51
02 0
1 is less or equal than current maximum value,
which is 5
-9
51
02 0
Summary, the value of maximal element of root
and its leaf is 5
-9
51
02 0
The son, which value is 5, is bigger than father,
which value is -9! So let's change their value.
-9

51
02 0
Values successfully changed! One more step to
make heap has been done!
5
-91
02 0
Let's find the maximal element in this root or its
sons!
The value of root is -9
5
-91
02 0
0 is more than current maximum value, which is -
9
5
-91
02 0
2 is more than current maximum value, which is 0
5
-91
02 0
Summary, the value of maximal element of root
and its leaf is 2
5
-91
02 0

T	
	The son, which value is 2, is bigger than father,
	which value is -9! So let's change their value.
	5
	-91
	02 0
	Values successfully changed! One more step to
	make heap has been done!
	5
	21
	09 0
	Sifting down has ended.
	The heap restored!
	Current state of the heap:
	5
	21
	09 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
	21

09 0 It is heap as array. The green part is actually heap, white is sorted sequence and cyan is th root, which just has been added to the sorted sequence: 5 2 1 0 -9 0 9 13 67 67 67 3456	e old
heap, white is sorted sequence and cyan is the root, which just has been added to the sorted	e old
root, which just has been added to the sorted	
	e root
sequence: 5 2 1 0 -9 0 9 13 67 67 67 3456	e root
	e root
Make the value of the last element in heap th	3 0 0
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 aft	er the
heap in the array that is storing our elements	as
heap and sorted sequence.	
Current state of heap as tree:	
0	
21	
09	
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 th	e 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 eler	nent!
Let's restore our heap with sifting it down.	
It is the sifting down.	
Let's find the maximal element in this root of	r its
sons!	
The value of root is 0	

	0
	21
	09
	2 is more than current maximum value, which is 0
	0
	21
	09
	1 is less or equal than current maximum value,
	which is 2
	0
	21
	09
	Summary, the value of maximal element of root
	and its leaf is 2
	0
	21
	09
	The son, which value is 2, is bigger than father,
	which value is 0! So let's change their value.
	0
	21
	09
	Values successfully changed! One more step to
	make heap has been done!
	2
	01
	09

Let's find the maximal element in this root or its
sons!
The value of root is 0
2
01
09
0 is less or equal than current maximum value,
which is 0
2
01
09
-9 is less or equal than current maximum value,
which is 0
2
01
09
Summary, the value of maximal element of root
and its leaf is 0
2
01
09
The root, which value is 0 is more than his sons,
so this subtree is heap!
2
01
09

The heap restored!
Current state of the heap:
2
01
09
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
01
09
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
01

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 element in this root or its
sons!
The value of root is -9
-9
01
0
0 is more than current maximum value, which is -
9
-9
01
0
1 is more than current maximum value, which is 0
-9
01
0
Summary, the value of maximal element of root
and its leaf is 1

-9
01
0

The son, which value is 1, is bigger than father,
which value is -9! So let's change their value.
-9
01
0
Values successfully changed! One more step to
make heap has been done!
1
09
0
6'6' 1 1 1 . 1
Sifting down has ended.
The heap restored!
Current state of the heap:
1
09
0
O .
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
09
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
09
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.

	Let's find the maximal element in this root or its
	sons!
	The value of root is 0
	0
	09
	0 is less or equal than current maximum value,
	which is 0
	0
	09
	-9 is less or equal than current maximum value,
	which is 0
	0
	09
	Summary, the value of maximal element of root
	and its leaf is 0
	0
	09
	The root, which value is 0 is more than his sons,
	so this subtree is heap!
	0
	09
	The heap restored!

Current state of the heap:
0
09
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
09
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 element 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
1	

make neap 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.
Curre	ent state of heap as tree:
-9	
Curre	ent state of heap as array:
It is l	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
	ence: -9 0 0 1 2 5 9 13 67 67 67 3456
Неар	is corrupted after draging maximal element!
Let's	restore our heap with sifting it down.
It is t	the sifting down.
Only	root is reamining in the heap, so it is already
heap	. No sifting needed.
Siftir	ng down has ended.
The l	heap restored!
Curre	ent state of the heap:
-9	-
	ude the node with biggest value. It is the root
becau	use 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	Input array:					
	67 0 10 -1 -2 -9	123 67 5 -9 67 0 10 -1 -2 -9					
	Восходящая просейка	Fistly we need to make heap from starting array.					
		Let's make heap!					
		The current state of heap is:					
		123					
		67					
		9					
		67010 -129					
		It is the sifting down					
		It is the sifting down.					
		No son of this node exist, so it is already subheap.					
		No sifting needed. Sifting down has ended.					
		Sitting down has ended.					
		It is the sifting down.					
		Let's find the maximal element in this root or its					
		sons!					
		The value of root is 5					

123
67
9
67010 -129
-1 is less or equal than current maximum value,
which is 5
123
67
9
67010 -129
-2 is less or equal than current maximum value,
which is 5
123
67
9
67010 -129
-9 is less or equal than current maximum value,
which is 5
123
67
9
67010 -129
Summary, the value of maximal element of root
and its leaf is 5
123
67
9
67010 -129

The root, which value is 5 is more than his sons,
so this subtree is heap!
123
67
9
67010 -129
To be about 100 and 10
It is the sifting down.
Let's find the maximal element in this root or its
sons!
The value of root is 67
123
675
9
67010 -129
67 is less or equal than current maximum value,
which is 67
123
675
9
67010 -129
0 is less or equal than current maximum value,
which is 67
123
67
9
67010 -129

which is 67	10 is less or equal than current maximum value,			
67	which is 67			
9 67010129 Summary, the value of maximal element of root and its leaf is 67 123 67		123		
67010 -129 Summary, the value of maximal element of root and its leaf is 67	67		_5	
Summary, the value of maximal element of root and its leaf is 67 123 67	9			
and its leaf is 67 123 67	670	_10 -1	2	9
123 67	Summary, the	value of ma	aximal e	lement of root
67	and its leaf is	67		
9 67010 -129		123		
67010 -129	67		_5	
The root, which value is 67 is more than his sons, so this subtree is heap! 123 67	9			
123	670	_10 -1	2	9
123				
123 67	The root, which	ch value is 6	57 is moi	re than his sons,
67	so this subtree	e is heap!		
9 67010 -129		123		IIIII
67010 -129	67		_5	
It is the sifting down. Let's find the maximal element in this root or its sons! The value of root is 123 123 67	9			
Let's find the maximal element in this root or its sons! The value of root is 123 123 67	670	_10 -1	2	9
Let's find the maximal element in this root or its sons! The value of root is 123 123 67				
Let's find the maximal element in this root or its sons! The value of root is 123 123 67				
sons! The value of root is 123 123 675	It is the sifting	g down.		
sons! The value of root is 123 123 675				
The value of root is 123 123 67	Let's find the	maximal ele	ememt in	this root or its
123 67	sons!			
67	The value of r	root is 123		
		123		
	67		_5	
9	9			

670	10 -1	2	9
67 is less or	equal than cu	arrent m	aximum value,
which is 123	;		
	123		
67		5	
9			
670	10 -1	2	9
5 is less or e	qual than cur	rent ma	ximum value,
which is 123	}		
	123		
67		5	
9			
670	10 -1	2	9
-9 is less or	equal than cu	rrent ma	aximum value,
which is 123	}		
	123		
67		5	
9			
670	10 -1	2	9
Summary, th	ne value of m	aximal e	element of root
and its leaf i	s 123		
	123		
67		5	-
9			
670	10 -1	2	9
	-		
The root, wh	nich value is	123 is m	ore than his sons,
so this subtro	ee is heap!		
	123		IIIII
		·	

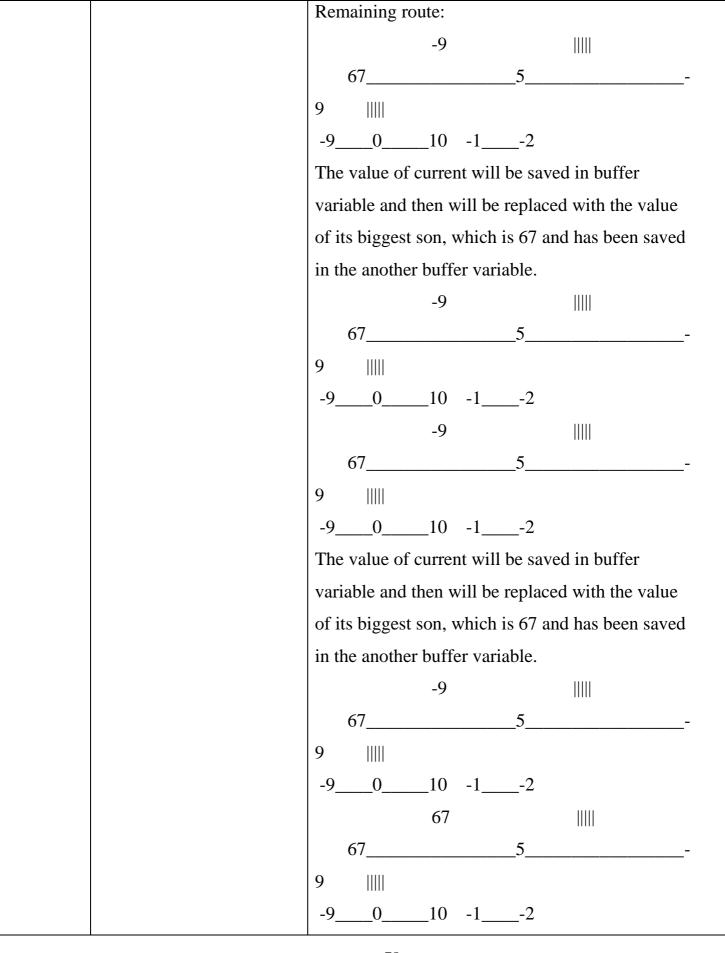
9	
67010 -129	
Building heap has been ended!	
The current state of heap is:	
123	
675	
9	
67010 -129	
Choose sifting type. 'd' for sifting down and 'u'	for
upward sifting:	
u	
You choose upward sifting sort.	
Heapsort with upward sifting.	
This sort using the upward sifting to restore he	eap
after draging max element.	
Exclude the node with biggest value. It is the re	oot
because we are working with max-heap. Save to	he
root value in buffer variable	
123	

	67
	9
	67010 -129
	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
	675
	9
	67010 -12
	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 lea	af which	consis	t of the bigg	gest sons.
Let's f	and the n	naxima	al son of this	s root!
The va	alue of ro	oot is -	9	
		-9		
6	7		5	
9				
67	0	_10 -	12	
67 is f	irst son,	so it is	new maxin	num value.
		-9		
6	7		5	-
9				
67	0	_10 -	12	
5 is le	ss or equ	al than	current ma	ximum value,
which	is 67			
		-9		
67	7		5	
9				
67	0	_10 -	12	
-9 is le	ess or equ	ual tha	n current m	aximum value,
which	is 67			
		-9		
67	7		5	
9				
67	0	_10 -	12	
Summ	nary, the	value o	of maximal	leaf is 67
		-9		
67	7		5	
9				

1					
	67_	0	_10 -1	-2	
	L ot's	find the n	novimol	son of this	s root!
					8 1001!
	The	value of ro	oot 18 67	1	
			-9		
		67		5	
	9				
	67_	0	_10 -1	2	
	67 is	s first son,	so it is	new maxin	num value.
			-9		
		67		5	-
	9			3	
			10 1	2	
		0			
	0 is	less or equ	al than	current ma	ximum value,
	whic	ch is 67			
			-9		
		67		5	
	9				
	67	0	10 -1	-2	
					aximum value,
			luai tiiai	i cuirciit iii	aximum varue,
	WIIIC	ch is 67	0		
			-9		
		67		5	-
	9				
	67_	0	_10 -1	2	
	Sum	mary, the	value of	f maximal	leaf is 67
		-	-9		
		67	-	5	
		01		J	

	9
	67010 -12
	So we have got the next route:
	-9
	67
	9
	67010 -12
	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
	9
	67010 -12
	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
	9
	-9010 -12
	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
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
67
9
-9010 -12
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
9
-9010 -12
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
The 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
9
-9010 -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
9
-9010 -1
67 is first son, so it is new maximum value.
-2

67
9
-9010 -1
5 is less or equal than current maximum value,
which is 67
-2
67
9
-9010 -1
-9 is less or equal than current maximum value,
which is 67
-2
67
9
-9010 -1
Summary, the value of maximal leaf is 67
-2
67
9
-9010 -1
Let's find the maximal son of this root!
The value of root is 67
-2
67
9
-9010 -1
-9 is first son, so it is new maximum value.

			-2			
		67			5	-
	9					
	-9_	0	10	-1		
	0 is	more tha	ın curr	ent ma	aximum	value, which is -
	9					
			-2			
		67			5	-
	9					
	-9_	0	10	-1		
	10 is	s more th	ian cui	rrent n	naximu	m value, which is
	0					
			-2			
		67			5	
	9					
	-9_	0	10	-1		
	Sum	ımary, th	e valu	e of m	aximal	leaf is 10
			-2			
		67			5	-
	9					
	-9_	0	10	-1		
			-			
	So v	ve have g	got the	next 1	oute:	
			-2			
		67			5	
	9					
	-9_	0	10	-1		
	Let's	s find the	e first e	elemen	nt on thi	is route which is

1.
bigger than root.
Current node, which value is 10 is more than root,
which value is -2
-2
67
9
-9010 -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
9
-902 -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
9
-902 -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
9
-902 -1
-2
10
9
-902 -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
9
-902 -1
67
10
9
-902 -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
9
-902 -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
	9
	-902 -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
	9
	-902
	Current state of heap as array:
	It is heap as array. The green part is actually the

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	Heap is co	orrupted aft	er draging	maximal elemer
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		_		_
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				
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 -1 -1 -9 -1 5 is less or equal than current maximum value, which is 10 -1	Firstly, we	e need to fin	nd a route	from heap root to
The value of root is -1 -1 10	the leaf w	hich consis	t of the bi	ggest sons.
The value of root is -1 -1 10				
-1 10	Let's find	the maxima	al son of the	his root!
105	The value	of root is -	1	
9 -902 10 is first son, so it is new maximum value. -1 105 9 -902 5 is less or equal than current maximum value, which is 10 -1		-1		
-902 10 is first son, so it is new maximum value. -1 105 9 -902 5 is less or equal than current maximum value, which is 10 -1	10		5	
10 is first son, so it is new maximum value. -1 10	9			
-1 105	-90_	2		
105 9 -902 5 is less or equal than current maximum value, which is 10 -1	10 is first	son, so it is	new max	imum value.
9 -902 5 is less or equal than current maximum value, which is 10 -1		-1		IIIII
-902 5 is less or equal than current maximum value, which is 10 -1	10		5	
-902 5 is less or equal than current maximum value, which is 10 -1	9			
5 is less or equal than current maximum value, which is 10 -1		2		
which is 10 -1			current n	naximum value,
-1				,
11111				IIIII
	10		5	
3 11111		2		
9 -902	-9U_			

which is 10				
	-1			
10		5		
9				
-90_	2			
Summary, t	the value of	maximal	leaf is 10	
	-1		IIII	
10		5		
9				
-90_	2			
Let's find th	ne maximal	son of th	is root!	
The value of	of root is 10			
	-1		IIII	
10		5	-	
9				
-90_	2			
-9 is first so	on, so it is no	ew maxir	num value.	
	-1			
10		5		
9				
-90_	2			
0 is more th	nan current i	naximun	n value, which is -	
9				
	-1			
10		5	 	
9		5		
		5		

-2 is less or equal than current maximum value,
which is 0
-1
10
9
-902
Summary, the value of maximal leaf is 0
-1
10
9
-902
So we have got the next route:
-1
10
9
-902
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
9
-902
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
9
-912
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
9
-912
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
9
-912
-1
0
9
-912
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
9
-912
10
0
9
-912
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
9
-912
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

	9			
	-912			
	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			
	9			
	-91			
	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				root!
			-2		
	0_			_5	
	9				
	-9	1			
	0 is fir	st son, sc	it is new	maximu	ım value.
			-2		
	0_			_5	
	9				
	-9	1			
	5 is m	ore than o	current ma	ximum	value, which is 0
			-2		
	0_			_5	
	9				
	-9	1			
	-9 is le	ess or equ	ıal than cuı	rent ma	aximum value,
	which	is 5			
			-2		
	0_			_5	
	9				
	-9	1			
	Summ	ary, the v	alue of ma	aximal l	eaf is 5
			-2		
	0_			_5	
	9				
	-9	1			
1	Ī				

So we have got the next route:
-2
0
9
-91
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
9
-91
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
02
9
-91
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
02

	9				
	-91				
	The value of	current will b	e 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 anothe	er buffer varial	ble.		
		-2	IIIII		
	0		-2		
	9				
	-91				
		5	IIIII		
	0		-2		
	9				
	-91				
	The heap roo	ot has been rea	ached. Shifting no	des to	
	upper level h	nas successfull	y ended.		
	Upward sifti	ng has ended.			
	The heap res	stored!			
	Current state	of the heap:			
		5	IIIII		
	0		2		
	9				
	-91				

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			
02			
9			
-91			
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			
02			
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
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
02
9
-9

Summary,	the value of 1	naximal	leaf is 0
	-1		
0		2	
9			
-9			
Let's find	the maximal s	on of this	s root!
The value	of root is 0		
	-1		
0		2	-
9			
-9			
-9 is first s	son, so it is ne	w maxim	num value.
	-1		
0		2	-
9			
-9			
Summary,	the value of 1	maximal	leaf is -9
	-1		
0		2	-
9			
-9			
So we hav	e got the next	route:	
	-1		
0		2	
9			
-9			

Let	's fin	d the	first el	ement	on thi	s route	which is
big	ger tl	han r	oot.				
Cu	rrent	node	, which	ı value	e is -9	is less th	an root,
wh	ich v	alue	is -1. S	o we e	exclude	e it from	the route.
			-1				
	0				2		-
9							
-9							
Cu	rrent	node	, which	ı value	e is -9	is less th	an root,
wh	ich v	alue	is -1. S	o we e	exclude	e it from	the route.
			-1				
	0				2		
9							
-9							
Cu	rrent	node	, which	ı value	e is 0 is	s more tl	nan root,
wh	ich v	alue	is -1				
			-1				
	0				2		-
9							
-9							
Sav	ve cu	rrent	node in	ı buffe	er varia	able, rep	lace the
val	ue of	curr	ent nod	le with	the va	alue of tl	he root
anc	l excl	lude 1	this noo	de fror	n the r	oute.	
			-1				
	-1_				2		9
-9							
No	w we	e are g	going to	o shift	all re	naining	nodes in

	the rout	e, which	we have	got ear	ly, to the on	e
	level up	per. The	e nearest i	node wh	nich will be	
	replaced	d with p	revious sa	aved in 1	ouffer variab	ole
	node va	lue, whi	ch is 0			
	Remain	ing rout	e:			
			-1			
	-1_			2		9
	-9					
	The val	ue of cu	rrent will	be save	d in buffer	
	variable	and the	en will be	replace	d with the va	alue
	of its bi	ggest so	n, which	is 0 and	has been sa	ved
	in the a	nother b	uffer vari	able.		
			-1			
	-1_			2		9
	-9					
		(0			
	-1_			2		9
	-9					
	The hea	p root h	as been re	eached.	Shifting noc	les to
	upper le	evel has	successfu	ılly ende	ed.	
	Upward	sifting	has endec	1.		
	The hea	p restor	ed!			
	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
-19
-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
-19
Current state of heap as array:

It	is heap as array. The green part is actually the
h	eap, white is sorted sequence and cyan is the old
ro	oot, which just has been added to the sorted
Se	equence: -9 -1 -2 -9 0 5 10 67 67 123
Н	leap is corrupted after draging maximal element!
L	et's restore our heap with sifting it down.
S	tarting upward sifting.
F	irstly, we need to find a route from heap root to
th	ne leaf which consist of the biggest sons.
L	et's find the maximal son of this root!
Т	The value of root is -9
	-9
-	19
-1	l is first son, so it is new maximum value.
	-9
-	19
-2	2 is less or equal than current maximum value,
W	which is -1
	-9
-	19
-9	9 is less or equal than current maximum value,
W	which is -1
	-9
	101

Summary, the value of maximal leaf is -1

So we have got the next route:

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

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.

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___-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___-9____ -1 ||||| -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: -1 -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

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:

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
	-92
	-9 is first son, so it is new maximum value.
	-9
	-92
	-2 is more than current maximum value, which is -
	9
	-9
	-92
	Summary, the value of maximal leaf is -2
	-9
	-92
	So we have got the next route:
	-9
	-92
	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
	-92
	Save current node in buffer variable, replace the
L	1

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

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:

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.

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:

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

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

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 in the array that is storing our elements as

heap. So we have already sorted elements after the

		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
		34569078
		Sorted array:
	101001555	34 56 78 90 123
4	10123456789	9
		8

1		
		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	Sorted array:
	5 86 7 8 9	1 2 3 4 5 7 8 9 12 13 19 86 121
6	8 18 145 54 -985 554	Sorted array:
	3281236549871	-985 -9 -8 -5 0 1 1 2 3 3 4 5 6 7 8 8 9 18 23 32 35
	23 6448 35 185 188 61	54 61 145 185 188 554 1452 6448
	-8 -9 3 -5 0 1452	
7	4 89 81 42 43 61 2 37	Sorted array:
	14 -1 -2 -3 -4 -5 -6 -7 -	-9 -8 -7 -6 -5 -4 -3 -2 -1 0 2 14 37 42 43 61 81 89
	8 -9 0	
1	1	I I

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 С++ 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++){</pre>
                    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";</pre>
                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++){</pre>
                         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";</pre>
                return false;
            return true;
        }
        int calcHeight(){//высчитывает количество уровней в дереве
            int i = m_root;
            int height = 0;
            while(i < m_size){</pre>
                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";</pre>
            int max = -1;
            int j = 0;
            int nodes[m_d+1];
            cout << "The value of root is " << m_arr[root] << "\n";</pre>
            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++){</pre>
                if(max == -1){
```

```
max = i;
                     cout << m_arr[i] << " is first son, so it is new maximum</pre>
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</pre>
is " << m_arr[max] << '\n';
                     nodes[0] = max;
                     max = i;
                }
                else{
                     cout << m_arr[i] << " is less or equal than current maximum</pre>
value, which is " << m_arr[max] << '\n';</pre>
                     nodes[0] = max;
                nodes[j] = i;
                //j += 1;
                printHeap(nodes, j+1);
            cout << "Summary, the value of maximal leaf is " << m_arr[max] << "\n";</pre>
            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 element in this root or its sons!\n";</pre>
            int max = root;
            int j = 0;
            int nodes[m_d+1];
            cout << "The value of root is " << m_arr[root] << "\n";</pre>
            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++){</pre>
                 if(m_arr[i] > m_arr[max]){
                     cout << m_arr[i] << " is more than current maximum value, which</pre>
is " << m arr[max] << '\n';</pre>
                     nodes[0] = max;
                     max = i;
                }
                else{
```

```
cout << m_arr[i] << " is less or equal than current maximum</pre>
value, which is " << m_arr[max] << '\n';</pre>
                   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</pre>
" << m_arr[max] << "\n";</pre>
           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";</pre>
           if(m_size < 2){</pre>
               cout << "Only root is reamining in the heap, so it is already heap.</pre>
No sifting needed.\n";
               cout << "Sifting down has ended.\n";</pre>
               cout << "----\n":
               return;
           }
           if(!(root * m_d + 1 < m_size)){</pre>
               cout << "No son of this node exist, so it is already subheap. No</pre>
sifting needed.\n";
               cout << "Sifting down has ended.\n";</pre>
               cout << "-----\n";
               return;
           }
           while(root * m d + 1 < m size){</pre>
               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</pre>
than his sons, so this subtree is heap!\n";
                   printHeap(nodes, 2);
                   cout << "\n";</pre>
                   return;
               }
               cout << "The son, which value is " << m_arr[n_root] << ", is bigger</pre>
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</pre>
has been done!\n";
              printHeap(nodes, 2);
              cout << "\n";</pre>
          cout << "Sifting down has ended.\n";</pre>
           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";</pre>
          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</pre>
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.</pre>
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";</pre>
```

```
nodes[0] = m_root;
           printHeap(nodes, 1);
           cout << "Current state of heap as array:\n";</pre>
           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";</pre>
           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</pre>
leaf which consist of the biggest sons.\n";
               while(cur*m d+1 < m size){</pre>
                   cur = findMaxLeaf(cur);
                   way[i] = cur;
                   i += 1;
               }
               cout << "So we have got the next route:\n";</pre>
               printHeap(way, i);
               cout << "\nLet's find the first element on this route which is</pre>
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] << "</pre>
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</pre>
than root, which value is " << m_arr[m_root] <<"\n";</pre>
                printHeap(nodes, 2);
                cout << "Save current node in buffer variable, replace the value of</pre>
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</pre>
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";</pre>
                printHeap(way, i+1);
                while(cur > m_root){
                    cout << "The value of current will be saved in buffer variable</pre>
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</pre>
level has successfully ended.\n";
            }
            else{
                cout << "Only root is reamining in the heap, so it is already heap.</pre>
No sifting needed.\n";
            }
            cout << "Upward sifting has ended.\n";</pre>
            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";</pre>
                 break;
              }
              cout << "\033[1;30;43mHeap is corrupted after draging maximal</pre>
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</pre>
the heap:\n";
              printHeap(nullptr, -1);
              cout << "\n\n\n\n";</pre>
          }
          cout << "-----
-----\n";
          cout << "Sort has successfully ended!\n";</pre>
          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";</pre>
                 break;
              }
              cout << "\033[1;30;43mHeap is corrupted after draging maximal</pre>
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</pre>
the heap:\n";
              printHeap(nullptr, -1);
              cout << "\n\n\n\n";</pre>
          }
          cout << "-----
-----\n";
          cout << "Sort has successfully ended!\n";</pre>
          printAsArr(false);
       }
       void printArr(){//выводит массив
          for(int i = 0; i < m_arr_size; i++){</pre>
              cout << m arr[i] << ' ';</pre>
```

```
}
             cout << '\n';</pre>
         }
         void printAsArr(bool is_col_first){//выводит кучу как массив
             cout << "It is heap as array. The green part is actually the heap,</pre>
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++){</pre>
                 cout << "\033[1;30;42m";</pre>
                 if(i >= m_size){
                      if(i == m_size && is_col_first){
                          cout << "\033[1;30;46m";</pre>
                      }
                      else{
                          cout << "\033[1;30;47m";</pre>
                      }
                 }
                 cout << m_arr[i] << ' ';</pre>
                 cout << "\033[0m";</pre>
             }
             cout << '\n';</pre>
         }
         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++){</pre>
                      cout << ' ';
                 }
             }
             else{
                 for(int i = 0; i < step; i++){</pre>
                      cout << ' ';
                 }
             }
             if(is_col){
                 cout << "\033[1;30;47m";</pre>
             }
             cout.setf(ios::left);
             cout.width(4);
             if(side != 'r' && side != 't'){
                 cout.fill('_');
             }
             cout << node value;</pre>
             cout.unsetf(ios::left);
             cout.fill(' ');
             cout << "\033[0m";
```

```
if(side == 'r' || side == 't'){
                 for(int i = 0; i < step; i++){</pre>
                     cout << ' ';
                 }
            }
            else{
                 for(int i = 0; i < step; i++){</pre>
                     cout << '_';
                 }
            }
        }
        void printHeap(int* color_nodes, int col_size){//выводит кучу в консоль,
как дерево
            if(m_size == 0){
                 cout << "Empty heap!\n";</pre>
                 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++){</pre>
                 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){</pre>
                     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';</pre>
                     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';</pre>
                 }
            }
            cout << "\n";</pre>
        }
        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</pre>
program:\n";
            cin >> command;
             switch (command){
                 case 'q':
                     cout << "You choose to end the programm!\n";</pre>
                     return 0;
                 case 's':
                     cout << "Input the path to data file:\n";</pre>
                     cin >> fname;
                     fin.open(fname, ifstream::in);
                     if(!fin.is_open()){
                         cout << "Opening file with test data failed! Try again!\n";</pre>
```

```
break;
                      }
                      heap = new Dheap;
                      isD = heap->readHeapFromFile(fin);
                      fin.close();
                      if(!isD){
                           cout << "Error in input data";</pre>
                           delete heap;
                          break;
                      }
                      cout << "Input array:\n";</pre>
                      heap->printArr();
                      cout << "Fistly we need to make heap from starting array.\n";</pre>
                      heap->makeHeap();
                      cout << "\n\n\n";</pre>
                      cout << "Choose sifting type. 'd' for sifting down and 'u' for</pre>
upward sifting:\n";
                      cin >> command;
                      if(command == 'd'){
                           cout << "You choose sifting down sort.\n";</pre>
                          heap->siftDownSort();
                      }
                      else{
                           if(command == 'u'){
                               cout << "You choose upward sifting sort.\n";</pre>
                               heap->upwardSiftSort();
                           }
                           else{
                               cout << "Error command!\n";</pre>
                               delete heap;
                               break;
                           }
                      }
                      cout << "\n\n\n\n";</pre>
                      cout << "Sorted array:\n";</pre>
                      heap->printArr();
                      cout << "\n\n\n\n";</pre>
                      delete heap;
                      break;
                  default:
                      cout << "Error command! Try again!\n";</pre>
                      break;
             }
    }
```

```
return 0;
}
```