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

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных »

Тема: Демонстрация сортировки слабой кучей

Студентка гр. 9382 _____ Голубева В.П.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студентка Голубева В.П.

Группа 9382

Тема работы: Демонстрация работы алгоритма сортировки

Исходные данные:
на вход программе подаётся цифра — количество элементов и целочисленный массив, элементы массива разделены пробелом

Содержание пояснительной записки:
«Содержание», «Введение», «Ход выполнения работы»,«Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:
Не менее 15 страниц.

Дата выдачи задания: 31.10.2020

Дата сдачи реферата: 15.11.2020

Дата защиты реферата: 15.11.2020

Студентка _____ Голубева В.П.

Преподаватель _____ Фирсов М.А.

АННОТАЦИЯ

В курсовой работе происходит сортировка массива. Программа демонстрирует процесс сортировки при помощи вывода на экран состояния элементов на каждом шаге, раскрашивая в другой цвет сравниваемые элементы. Результатом будет являться отсортированный массив.

Примеры работы реализованной программы представлены в приложении А, исходный код приведён в приложении Б.

SUMMARY

In the course work, the array is sorted. The program demonstrates the sorting process by displaying the status of elements on the screen at each step, coloring the compared elements in a different color. The result will be a sorted array.

Examples of how the implemented program works are shown in appendix A, and the source code is given in appendix B.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью работы являлось изучение сортировки методом слабой кучи. Для этого потребовалось изучить её структуру, алгоритм построения, алгоритм сортировки с помощью неё, а также придумать визуализацию работы алгоритма. Результатом является программа, которая считывает и сортирует исходный целочисленны массив, визуализируя работу алгоритма.

1. ЗАДАНИЕ

Вариант 31. Сортировка слабой кучей. Демонстрация.

2. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

2.1. Класс слабой кучи WeakHeap

Для удобной работы со слабой кучей был создан класс слабой кучи WeakHeap. Пубичными полями класса являются - std::vector <int> wheap — хранятся элементы введённого массива, это вектор, встроенная возможность языка программирования, хранится в библиотеке <vector>, int size — хранится количество элементов wheap, unsigned char* r — массив для хранения инфрмации об обмене поддеревьями слабой кучи, int s — размер массива r.

Для класса реализованы публичные методы для работы со слабой кучей.

2.1.1 Описание конструктора по умолчанию WeakHeap()

Конструктор слабой кучи по умолчанию, при помощи него можно создавать объекты слабой кучи, а потом уже вводить значения

2.1.2 Описание конструктора WeakHeap(int count, std::istream&)

Конструктор класса, получает на вход целочисленное количество элементов, которые нужно ввести и ссылку на поток. При помощи цикла for вводит значения из потока во вспомогательную переменную member, а затем вставляет в конец вектора wheap

2.1.3 Описание метода void displayArray()

Метод выводит элементы массива, который хранится в поле класса WeakHeap — std::vector <int> wheap

2.1.4 Описание метода void displayHeap(int i_1, int j_1, int col, int num)

Получает номера i_1 и j_1 элементов, которые нужно подсветить определённым цветом, int col — цвет, int num — количество отсортированных элементов в массиве. Сначала записываем элементы в кучу в порядке, удобном для вывода на экран, затем в зависимости от введённого цвета выводим первый

элемент. Затем вычисляем глубину кучи при помощи функции log(2, new_size), где new_size — количество элементов в массиве за вычетом отсортированных. Затем проходимся по элементам, выводим их в наглядном виде. У нас есть счётчик int k — количество уже выведенных элементов. Если его значение равно i_1 или j_1, то перекрашиваем выводимый элемент при помощи управляющей последовательности \x1b[<номер цвета>m. Затем выодим отсортированную часть массива, крася её в жёлтый. При помощи \x1b[33m.

2.1.5 Описание метода void displayHeap()

Метод нужен в случае, если мы просто хотим вывести кучи, без перекрашивания элементов. Точно так же сначала записываем элементы в кучу в порядке, удобном для вывода на экран, затем в зависимости от введённого цвета выводим первый элемент. Затем вычисляем глубину кучи при помощи функции log(2, size). Затем проходимся по элементам, выводим их в наглядном виде. Счётчик k нужен, если количество элементов в слабой куче не будет равно степени двойки, без него вместо отсутствующий элементов на последнем уровне выведутся нули.

2.1.6 Описание метода void weakHeapMerge(unsigned char *r, int i, int j, int num)

Если суперродитель меньше потомка, то для потомка переопределяем, порядок его потомков (кто "левый", а кто "правый"), затем меняем значения "суперродителя" и потомка при момощи std::swap(), выводим слабую кучу на экран для демонстрации, какие элементы могли поменяться.

2.1.7 Описание метода WeakHeap* fileInputHeap()

Метод возвращает указатель на объект класса слабой кучи, открывает файл для записи, создавая поток std::ifstream для чтения, вводим из файла количество элементов массива, проверяем, что оно не меньше нуля. Создаём новый объект класса слабой кучи, выводим на экран введённый массив, закрываем поток для чтения.

2.1.8 Описание метода WeakHeap* consoleInputHeap()

Метод возвращает указатель на объект класса слабой кучи, вводим с консоли количество элементов массива, проверяем, что оно не меньше нуля, иначе просим пользователя ввести количество ещё раз. Создаём новый объект класса слабой кучи, вводя элементы с консоли.

2.1.9 Описание деструктора ~WeakHeap()

Деструктор класса, очищает вектор, в wheap котором хранятся элементы массива

2.1.10 Описание деструктора void weakHeapSort()

Метод, в котором происходит сортировка. Алгоритм сортировки - сначала формируем из массива слабую кучу: перебираем элементы массива слева-направо, для текущего элемента поднимаемся вверх по родительской ветке до ближайшего «правого» родителя, сравниваем текущий элемент и ближайшего правого родителя, если ближайший правый родитель меньше текущего элемента, то: меняем местами (левый ⇔ правый) поддеревья с потомками для узла, в котором находится текущий элемент, меняем значениями ближайший «правый» родитель и узел с текущим элементом.

Затем из корня кучи текущий максимальный элемент перемещаем в конец неотсортированной части массива, после чего восстанавливаем слабую кучу: в корне кучи находится текущий максимальный элемент для неотсортированной части массива, меняем местами максимум из корня кучи и последний элемент в неотсортированной части массива. Последний элемент с максимумом перестаёт быть узлом слабой кучи. После этого обмена дерево перестало быть слабой кучей, так как в корне оказался не максимальный элемент. Поэтому делаем просейку: опускаемся из корня кучи по левым потомкам как можно ниже. Поднимаемся по левым потомкам обратно к корню кучи, сравнивая каждый левый потомок с корнем. Если элемент в корне меньше, чем очередной левый потомок, то: меняем местами (левый ⇔ правый) поддеревья с потомками для узла, в котором находится текущий левый потомок. Меняем значениями корень кучи и узел с текущим левым потомком. В корне слабой кучи снова находится максимальный элемент для оставшейся неотсортированной части массива.

Затем снова из корня кучи текущий максимальный элемент перемещаем в конец неотсортированной части массива, восстанавливаем слабую кучу, повторяем процесс, пока не будут отсортированы все элементы.

Сложность алгоритма по времени — O(n*log n).

2.2 Описание функции double log(int a, int b)

Получает на вход два целочисленных числа a — осноание логарифма и b — число, от которого надо взять логарифм., возвращает вещественное число - логарифм по основанию a от b. Для вычисления используется библиотечная функция log(a) — вычисляет логарифм по основанию 10 от преданного числа, функция входит в библиотеку <cmath>

ЗАКЛЮЧЕНИЕ

В результате выполнения работы была изучена сортировка методом слабой кучи. Была изучена структура слабой кучи, алгоритм её построения, алгоритм сортировки с помощью неё, а также визуализирована работа алгоритма. Была написана программа, которая считывает, сортирует исходный целочисленны массив и визуализирует работу алгоритма.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. The C++ Resources Network. URL: http://www.cplusplus.com (дата обращения: 15.11.2020)
2. Habr. URL: https://habr.com/en/company/edison/blog/499786/ (дата обращения: 15.11.2020)

ПРИЛОЖЕНИЕ А

ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ

Проверка на некорректных данных при вводе из файла

Содержимое входного файла:
-2
8 3 5 1 0

1)
Выберите способ ввода массива для сортировки
Введите 1 для ввода с консоли или 0 для ввода из файла
0
Вы выбрали ввод из файла

Вы ввели: Отрицательное значение эколичества элементов. Измените количество и перезапустите программу

Синим будут подсвечиваться проверяемые в куче элементы
Зелёным будут подсвечиваться элементы, которые в результате сравнения поменяли местами
Жёлтым будут подсвечиваться отсортированные элементы

Проверка на некорректных данных при вводе с консоли:
1)
Выберите способ ввода массива для сортировки
Введите 1 для ввода с консоли или 0 для ввода из файла
1
Вы выбрали ввод с консоли
Введите количество элементов в массиве
-1
Некорректное значение количества элементов, введите целое положительное число
-8
Некорректное значение количества элементов, введите целое положительное число
5
Введите элементы через пробел
1 4 2 5 3
...

Корректная обработка ввода из файла:

1)
Выберите способ ввода массива для сортировки
Введите 1 для ввода с консоли или 0 для ввода из файла
0
Вы выбрали ввод из файла

Вы ввели: 1 9 7 5 4 8 2

Синим будут подсвечиваться проверяемые в куче элементы
Зелёным будут подсвечиваться элементы, которые в результате сравнения поменяли местами
Жёлтым будут подсвечиваться отсортированные элементы

Построение первоначальной слабой кучи
1
9
7 5
4 8 2

1
9
7 5
4 8 2

Суперродитель 7 меньше потомка 8, меняем их местами
1
9
8 5
4 7 2

1
9
8 5
4 7 2

Суперродитель 1 меньше потомка 4, меняем их местами
4
9
8 5
1 7 2

4

9
8 5
1 7 2

4
9
8 5
1 7 2

Суперродитель 4 меньше потомка 8, меняем их местами
8
9
4 5
1 7 2

8
9
4 5
1 7 2

Суперродитель 8 меньше потомка 9, меняем их местами
9
8
4 5
1 7 2

Слабая куча построена
9
8
4 5
1 7 2

Переносим максимум из корня, применяем слабую просейку
9
8
4 5
1 7 2

Переместили корень 9 и элемент из конца неотсортированной части 2
2
8
4 5
1 7 9

2
8
4 5
1 7 9

Суперродитель 2 меньше потомка 5, меняем их местами
5
8
4 2
1 7 9

5
8
4 2
1 7 9

Суперродитель 5 меньше потомка 8, меняем их местами
8
5
4 2
1 7 9

Переносим максимум из корня, применяем слабую просейку
8
5
4 2
1 7

Отсортированная часть массива: 9

Переместили корень 8 и элемент из конца неотсортированной части 7
7
5
4 2
1 8

Отсортированная часть массива: 9

7
5
4 2
1 8

Отсортированная часть массива: 9

7
5
4 2
1 8

Отсортированная часть массива: 9

Переносим максимум из корня, применяем слабую просейку
7
5
4 2
1

Отсортированная часть массива: 8 9

Переместили корень 7 и элемент из конца неотсортированной части 1
1
5
4 2
7

Отсортированная часть массива: 8 9

1
5
4 2
7

Отсортированная часть массива: 8 9

Суперродитель 1 меньше потомка 4, меняем их местами
4
5
1 2
7

Отсортированная часть массива: 8 9

4
5

1 2
7

Отсортированная часть массива: 8 9

Суперродитель 4 меньше потомка 5, меняем их местами
5
4
1 2
7

Отсортированная часть массива: 8 9

Переносим максимум из корня, применяем слабую просейку
5
4
1 2

Отсортированная часть массива: 7 8 9

Переместили корень 5 и элемент из конца неотсортированной части 2
2
4
1 5

Отсортированная часть массива: 7 8 9

2
4
1 5

Отсортированная часть массива: 7 8 9

Суперродитель 2 меньше потомка 4, меняем их местами
4
2
1 5

Отсортированная часть массива: 7 8 9

Переносим максимум из корня, применяем слабую просейку
4
2
1

Отсортированная часть массива: 5 7 8 9

Переместили корень 4 и элемент из конца неотсортированной части 1
1
2
4

Отсортированная часть массива: 5 7 8 9

1
2
4

Отсортированная часть массива: 5 7 8 9

Суперродитель 1 меньше потомка 2, меняем их местами
2
1
4

Отсортированная часть массива: 5 7 8 9

2
1

Отсортированная часть массива: 4 5 7 8 9

Меняем местами корень 2 и следующий за ним элемент 1
1
2

Отсортированная часть массива: 4 5 7 8 9

Отсортированный массив: 1 2 4 5 7 8 9

2)
Выберите способ ввода массива для сортировки
Введите 1 для ввода с консоли или 0 для ввода из файла
0
Вы выбрали ввод из файла

Вы ввели: 8 10 39 25

Синим будут подсвечиваться проверяемые в куче элементы
Зелёным будут подсвечиваться элементы, которые в результате сравнения поменяли местами
Жёлтым будут подсвечиваться отсортированные элементы

Построение первоначальной слабой кучи
8
10
39 25

Суперродитель 10 меньше потомка 25, меняем их местами
8
25
39 10

8
25
39 10

Суперродитель 8 меньше потомка 39, меняем их местами
39
25
8 10

39
25
8 10

Слабая куча построена
39
25
8 10

Переносим максимум из корня, применяем слабую просейку
39
25
8 10

Переместили корень 39 и элемент из конца неотсортированной части 10
10
25
8 39

10
25
8 39

10
25
8 39

Суперродитель 10 меньше потомка 25, меняем их местами
25
10
8 39

Переносим максимум из корня, применяем слабую просейку
25
10
8

Отсортированная часть массива: 39

Переместили корень 25 и элемент из конца неотсортированной части 8
8
10
25

Отсортированная часть массива: 39

8
10
25

Отсортированная часть массива: 39

Суперродитель 8 меньше потомка 10, меняем их местами
10
8
25

Отсортированная часть массива: 39

10
8

Отсортированная часть массива: 25 39

Меняем местами корень 10 и следующий за ним элемент 8
8
10

Отсортированная часть массива: 25 39

Отсортированный массив: 8 10 25 39

Корректная обработка ввода с консоли:

1)
Выберите способ ввода массива для сортировки
Введите 1 для ввода с консоли или 0 для ввода из файла
1
Вы выбрали ввод с консоли
Введите количество элементов в массиве
4
Введите элементы через пробел
98 45 76 21
Синим будут подсвечиваться проверяемые в куче элементы
Зелёным будут подсвечиваться элементы, которые в результате сравнения поменяли местами
Жёлтым будут подсвечиваться отсортированные элементы

Построение первоначальной слабой кучи
98
45
76 21

98
45
76 21

98
45
76 21

Слабая куча построена

98
45
76 21

Переносим максимум из корня, применяем слабую просейку
98
45
76 21

Переместили корень 98 и элемент из конца неотсортированной части 21
21
45
76 98

21
45
76 98

Суперродитель 21 меньше потомка 76, меняем их местами
76
45
21 98

76
45
21 98

Переносим максимум из корня, применяем слабую просейку
76
45
21

Отсортированная часть массива: 98

Переместили корень 76 и элемент из конца неотсортированной части 21
21
45
76

Отсортированная часть массива: 98

21

45
76

Отсортированная часть массива: 98

Суперродитель 21 меньше потомка 45, меняем их местами
45
21
76

Отсортированная часть массива: 98

45
21

Отсортированная часть массива: 76 98

Меняем местами корень 45 и следующий за ним элемент 21
21
45

Отсортированная часть массива: 76 98

Отсортированный массив: 21 45 76 98

2)
Выберите способ ввода массива для сортировки
Введите 1 для ввода с консоли или 0 для ввода из файла
1
Вы выбрали ввод с консоли
Введите количество элементов в массиве
1
Введите элементы через пробел
1
Синим будут подсвечиваться проверяемые в куче элементы
Зелёным будут подсвечиваться элементы, которые в результате сравнения поменяли местами
Жёлтым будут подсвечиваться отсортированные элементы

Отсортированный массив: 1

3)
Выберите способ ввода массива для сортировки
Введите 1 для ввода с консоли или 0 для ввода из файла

1
Вы выбрали ввод с консоли
Введите количество элементов в массиве
3
Введите элементы через пробел
8 0 3
Синим будут подсвечиваться проверяемые в куче элементы
Зелёным будут подсвечиваться элементы, которые в результате сравнения поменяли местами
Жёлтым будут подсвечиваться отсортированные элементы

Построение первоначальной слабой кучи
8
0
3

8
0
3

Слабая куча построена
8
0
3

Переносим максимум из корня, применяем слабую просейку
8
0
3

Переместили корень 8 и элемент из конца неотсортированной части 3
3
0
8

3
0
8

3
0

Отсортированная часть массива: 8

Меняем местами корень 3 и следующий за ним элемент 0
0
3

Отсортированная часть массива: 8

Отсортированный массив: 0 3 8

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: WeakHeap.cpp

```
#include "Weak_heap.h"

#define GETFLAG(r, x) ((r[(x) >> 3] >> ((x) & 7)) & 1) //если в
качестве "левого" потомка родителя

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


WeakHeap::~WeakHeap(){
    wheap.clear();

}

//вычисляет логарифм от b по основанию a
double log(int a, int b)
{
    return log(b) / log(a);
}

void WeakHeap::displayHeap(int i_1, int j_1, int col, int num){

    std::vector <int> heap1;
    heap1.push_back(wheap[0]);
    heap1.push_back(wheap[1]);
    for (int i=0;i<s;i++){//записываем элементы в кучу в порядке,
удобном для вывода на экран
        heap1.push_back(wheap[2*i+r[i]]);
        heap1.push_back(wheap[2*i+1-r[i]]);

    }
    if (col==0){
      if (i_1==0 || j_1==0)
        std::cout<<"\x1b[32m"<<wheap[0]<<"\x1b[0m";
      else
        std::cout<<wheap[0];
    }
    else
    {
      if (i_1==0 || j_1==0)
        std::cout<<"\x1b[34m"<<wheap[0]<<"\x1b[0m";
      else
        std::cout<<wheap[0];
    }

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

```
    int new_size=size-num;

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

    int k=0;


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

      for (int j=0;j<pow(2,i);j++){
        if (k<new_size)
          if (col==0){//проверяем, какого цвета выводить элементы
              if (k+1==i_1 || k+1==j_1){
                std::cout<<"\x1b[32m"<<wheap[k+1]<<"\x1b[0m"<<"
";

              }

              else{
                std::cout<<wheap[k+1]<<"  ";

              }
            }
          else
          {
          if (k+1==i_1 || k+1==j_1){
              std::cout<<"\x1b[34m"<<wheap[k+1]<<"\x1b[0m"<<"  ";

            }

            else{
              std::cout<<wheap[k+1]<<"  ";

            }
          }
          k++;
        }

        std::cout<<"\n";

    }

  if (num>1){
    std::cout<<"\n";
    std::cout<<"Отсортированная часть массива: ";
    for (int i=1;i<num;i++)
        std::cout<<"\x1b[33m"<<wheap[new_size+i]<<"\x1b[0m"<<"  ";
    std::cout<<"\n";
  }

  if (col==0)
```

```
std::cout<<"-----------------------------------------------------
---------\n";
  else
    std::cout<<"\n";

}


void WeakHeap::displayHeap(){

  std::vector <int> heap1;
  heap1.push_back(wheap[0]);
  heap1.push_back(wheap[1]);
  for (int i=0;i<s;i++){//записываем элементы в кучу в порядке,
удобном для вывода на экран
      heap1.push_back(wheap[2*i+r[i]]);
      heap1.push_back(wheap[2*i+1-r[i]]);

  }
  std::cout<<wheap[0];
  std::cout<<"\n";

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

  int k=0;

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

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

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

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


    if (wheap[i] < wheap[j]) {//"Суперродитель" меньше потомка?
        //Для потомка переопределяем, порядок его потомков
        //(кто "левый", а кто "правый")

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

```
        this->displayHeap(i, j, 1, num);
        std::cout<<"Суперродитель "<<wheap[i]<<" меньше потомка
"<<wheap[j]<<", меняем их местами\n";

        std::swap(wheap[i], wheap[j]);
        this->displayHeap(i, j, 0, num);

    }
    else{
        this->displayHeap(i, j, 1, num);

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

void WeakHeap::weakHeapSort() {
    int n = size;
    int lef;
    int per;
    if(n > 1) {

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

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


        std::cout<<"Построение первоначальной слабой кучи\n";

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

    //Перенос максимума из корня в конец -->
```

```
    //слабая просейка --> и всё по новой
    std::cout<<"Слабая куча построена\n";
    this->displayHeap();

    for(i = n - 1; i >= 2; --i) {
        std::cout<<"Переносим максимум из корня, применяем слабую
просейку\n";
        //Максимум отправляем в конец неотсортированной части
массива
        //Элемент из конца неотсортированной части попадает в
корень
            this->displayHeap(0, i, 1, n-i);
        std::cout<<"Переместили корень "<<wheap[0]<<" и элемент из
конца неотсортированной части "<<wheap[i]<<"\n";
        std::swap(wheap[0], wheap[i]);
        this->displayHeap(0, i, 0, n-i);
        x = 1;
        //Опускаемся жадно вниз по "левым" веткам
        lef=GETFLAG(r, x);
        while((y = 2 * x + lef) < i) {
            x = y;
            lef=GETFLAG(r, x);
        }
        //Поднимаемся по "левой" ветке обратно до самого вверха
        //попутно по дороге делаем слияние каждого узла с корнем
        while(x > 0) {
            weakHeapMerge(r, 0, x, n-i);
            x >>= 1;
        }
    }
    //Последнее действие - меняем местами корень
    //и следующий за ним элемент
    this->displayHeap(0, 1, 1, n-1);
    std::cout<<"Меняем местами корень "<<wheap[0]<<" и следующий
за ним элемент "<<wheap[1]<<"\n";
    std::swap(wheap[0], wheap[1]);
    this->displayHeap(0, 1, 0, n-1);
    delete[] r;
  }
}
void WeakHeap::displayArray()
{
    for (int i=0;i<size;i++)
        std::cout<<wheap[i]<<" ";
    std::cout<<"\n\n";
}

WeakHeap* WeakHeap::consoleInputHeap(){
    int count;
    std::cout<<"Введите количество элементов в массиве\n";
    std::cin>>count;
    while (count<=0){
```

```
        std::cout<<"Некорректное значение количества элементов,
введите целое положительное число\n";
        std::cin>>count;
    }

    std::cout<<"Введите элементы через пробел\n";
    WeakHeap* wh=new WeakHeap(count, std::cin);
    return wh;
}

WeakHeap* WeakHeap::fileInputHeap(){
    int count;
    std::ifstream file("input.txt");//открываем файл для чтения
    file>>count;
    std::cout<<"\nВы ввели: ";
    if (count<0)
        std::cout<<"Отрицательное значение эколичества элементов.
Измените количество и перезапустите программу\n";
    WeakHeap* wh= new WeakHeap(count, file);
    wh->displayArray();
    file.close();
    return wh;
}

WeakHeap::WeakHeap(int count, std::istream& stream){

    int member;
    size=count;

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

        stream>>member;//вводим элементы массива
        wheap.push_back(member);

    }
}
```

Название файла: WeakHeap.h

```
#ifndef WEAK_HEAP_H
#define WEAK_HEAP_H

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

class WeakHeap{
```

```
public:

    std::vector <int> wheap;
    int size;
    unsigned char* r=nullptr;
    int s;
    WeakHeap(){}
    WeakHeap(int, std::istream&);

    void displayArray();
    void displayHeap(int i, int j, int col, int num);
    void displayHeap();
    void weakHeapMerge(unsigned char *r, int i, int j, int num);
    void weakHeapSort();
    WeakHeap* fileInputHeap();
    WeakHeap* consoleInputHeap();
    ~WeakHeap();

};

#endif
```

Название файла: WeakHeap.h

```
#include "Weak_heap.h"

int main()
{
    std::cout<<"Выберите способ ввода массива для сортировки\n";
    std::cout<<"Введите 1 для ввода с консоли или 0 для ввода из
файла\n";

    char s;

    std::cin>>s;
    WeakHeap* wh=nullptr;
    int flag=0;
    while (!flag){
        switch (s)
        {
        case '0':
            std::cout<<"Вы выбрали ввод из файла\n";
            wh=wh->fileInputHeap();
            flag=1;
            break;
        case '1':
            std::cout<<"Вы выбрали ввод с консоли\n";
            wh=wh->consoleInputHeap();
            flag=1;
            break;
```

```
        default:
            std::cout<<"Нет такой команды: \n"<<s<<"\n";
            std::cout<<"Введите команду\n";
            std::cin>>s;
            break;
        }
    }

    std::cout<<"\x1b[34mСиним\x1b[0m будут подсвечиваться
проверяемые в куче элементы\n";
    std::cout<<"\x1b[32mЗелёным\x1b[0m будут подсвечиваться
элементы, которые в результате сравнения поменяли местами\n";
    std::cout<<"\x1b[33mЖёлтым\x1b[0m будут подсвечиваться
отсортированные элементы\n\n";

    wh->weakHeapSort();
    if (wh->size>0){
        std::cout<<"Отсортированный массив: ";
        wh->displayArray();
    }
    wh->~WeakHeap();
    return 0;
}
```

Название файла: input.txt

```
3
9 3 1
```