**МИНОБРНАУКИ РОССИИ  
Санкт-Петербургский государственный  
электротехнический университет  
«ЛЭТИ» им. В.И. Ульянова (Ленина)  
Кафедра МО ЭВМ**

**ОТЧЕТ**по лабораторной работе № 1

«Пирамидальная сортировка»

по дисциплине «Построение и анализ алгоритмов»

Выполнила: Коппель Т.С.

Факультет: КТИ

Группа №: 5382

Преподаватель: Шолохова О. М.

Санкт-Петербург

2017 г.

**Цель работы:**

Изучение и реализация алгоритма пирамидальной сортировки на языке C++

**Теоретические сведения:**

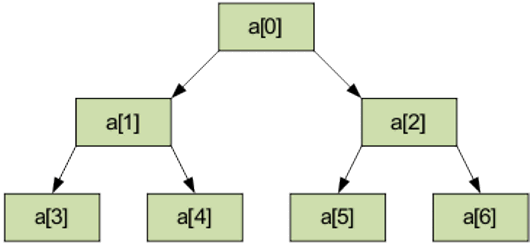
**Пирамидальная сортировка** (англ. *Heapsort*, «Сортировка кучей»[[1]](https://ru.wikipedia.org/wiki/%D0%9F%D0%B8%D1%80%D0%B0%D0%BC%D0%B8%D0%B4%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0#cite_note-Lecture5-1)) — алгоритм сортировки, работающий в худшем, в среднем и в лучшем случае (то есть гарантированно) за *Θ(n* log *n)* операций при сортировке *n* элементов.[[2]](https://ru.wikipedia.org/wiki/%D0%9F%D0%B8%D1%80%D0%B0%D0%BC%D0%B8%D0%B4%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0#cite_note-2) Количество применяемой служебной памяти не зависит от размера массива (то есть, O(1)).

Может рассматриваться как усовершенствованная сортировка пузырьком, в которой элемент всплывает (min-heap) / тонет (max-heap) по многим путям.

***Пирамидой*** (***кучей***) называется двоичное дерево такое, что

**a[i] ≤ a[2i+1];**

**a[i] ≤ a[2i+2].**

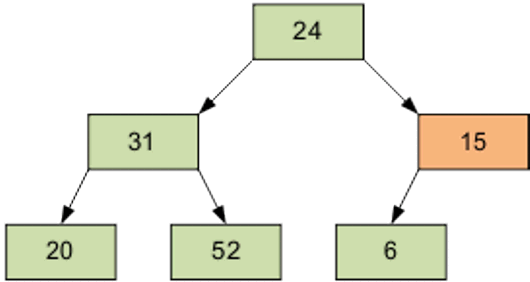
****

**a[0]** — минимальный элемент пирамиды.

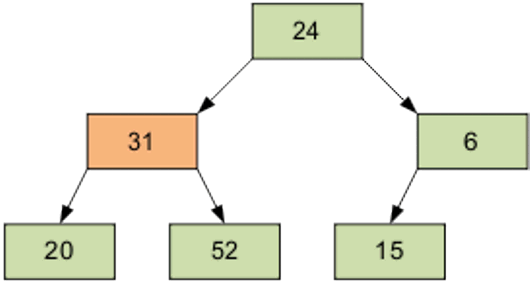
Общая идея пирамидальной сортировки заключается в том, что сначала строится пирамида из элементов исходного массива, а затем осуществляется сортировка элементов.  
  
Выполнение алгоритма разбивается на два этапа.  
  
***1 этап.*** Построение пирамиды. Определяем правую часть дерева, начиная с **n/2-1** (нижний уровень дерева). Берем элемент левее этой части массива и просеиваем его сквозь пирамиду по пути, где находятся меньшие его элементы, которые одновременно поднимаются вверх; из двух возможных путей выбираете путь через меньший элемент.  
  
Например, массив для сортировки

**24,   31,  15,   20,   52,   6**

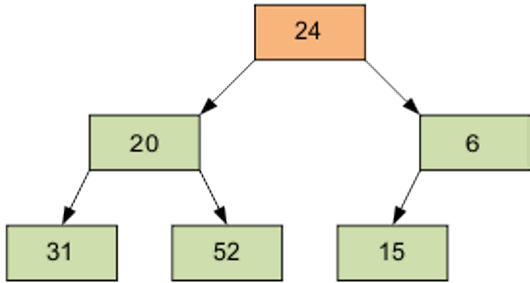
Расположим элементы в виде исходной пирамиды; номер элемента правой части (6/2-1)=2 - это элемент 15.

******

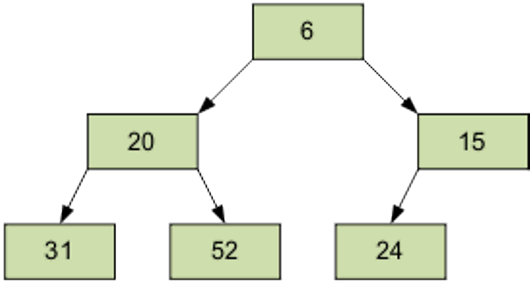
Результат просеивания элемента 15 через пирамиду.



Следующий просеиваемый элемент – 1,  равный 31.

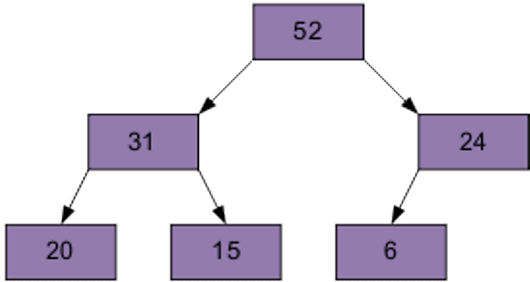


Затем – элемент 0, равный 24.



Разумеется, полученный массив еще не упорядочен. Однако процедура просеивания является основой для пирамидальной сортировки. В итоге просеивания наименьший элемент оказывается на вершине пирамиды.  
  
***2 этап*** Сортировка на построенной пирамиде. Берем последний элемент массива в качестве текущего. Меняем верхний (наименьший) элемент массива и текущий местами. Текущий элемент (он теперь верхний) просеиваем сквозь n-1 элементную пирамиду. Затем берем предпоследний элемент и т.д.

В итоге массив будет отсортирован по убыванию.



**Достоинства и недостатки**

*Достоинства*

* Сортирует на месте, то есть требует всего *O*(1) дополнительной памяти (если дерево организовывать так, как показано выше)
* Имеет доказанную оценку худшего случая {\displaystyle O(n\cdot \log n)}

*Недостатки*

* Сложен в реализации.
* Неустойчив — для обеспечения устойчивости нужно расширять ключ.
* На почти отсортированных массивах работает столь же долго, как и на хаотических данных.
* Методу требуется «мгновенный» прямой доступ; не работает на связанных списках и других структурах памяти последовательного доступа.

Сортировка слиянием при расходе памяти *O*(*n*) быстрее ({\displaystyle O(n\cdot \log n)}с меньшей константой) и не подвержена деградации на неудачных данных.

Из-за сложности алгоритма выигрыш получается только на больших *n*. На небольших *n* (до нескольких тысяч) быстрее сортировка Шелла.

**Реализация**:

Алгоритм пирамидальной сортировки реализован с помощью 3 функций:

1. void heapify(int \*arr, int pos, int n);//нормализация кучи

heapify принимает на вход позицию корня кучи в массиве, указатель на первый элемент массива и количество элементов в массиве

1. void heap\_make(int \*arr, int n);//функция построения кучи

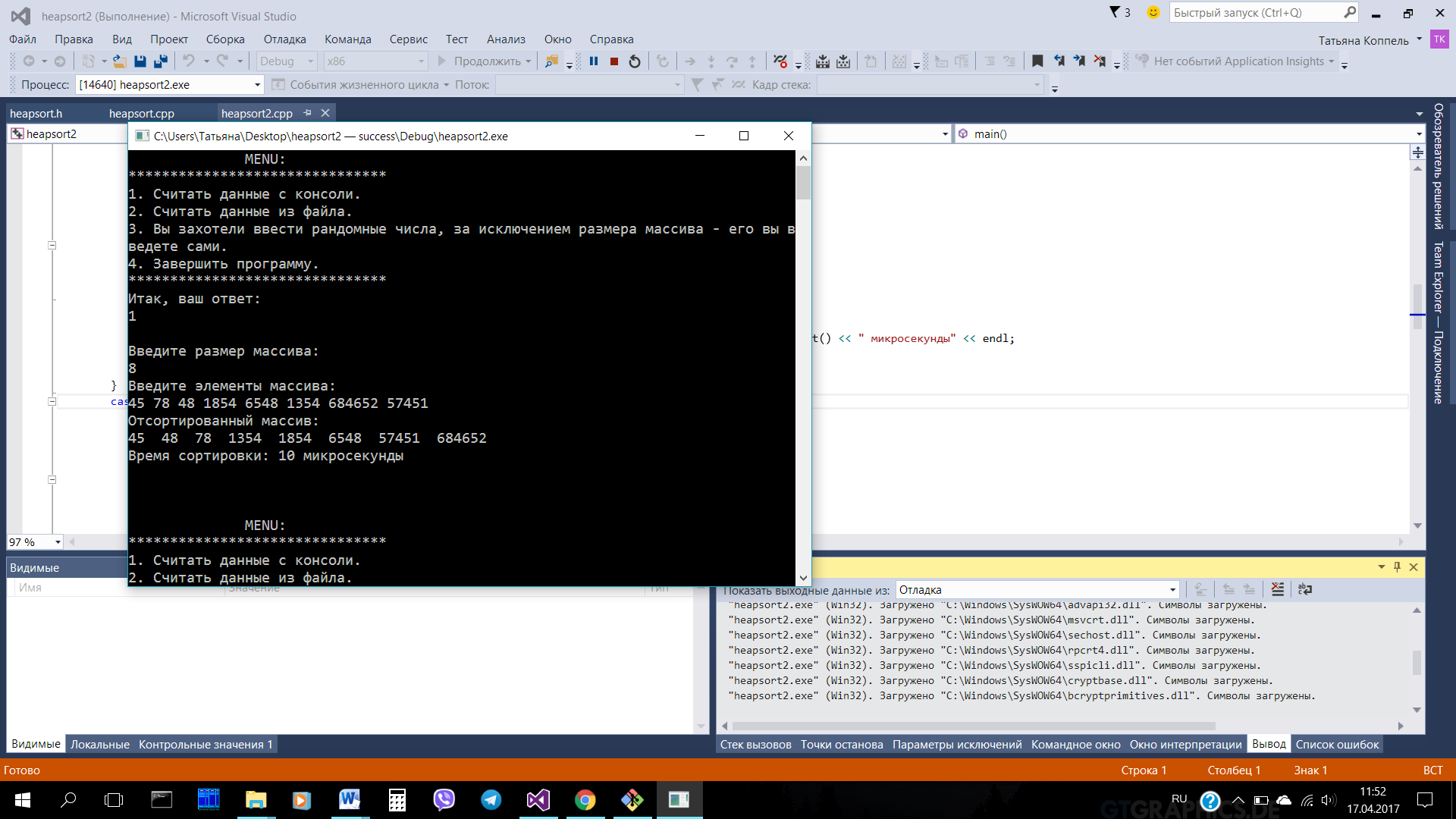
Функция, строящая пирамиду. На вход принимает указатель на начало массива и размер массива. Результатом является правильно построенная куча.

1. void heap\_sort(int \*arr, int n);//сортировка

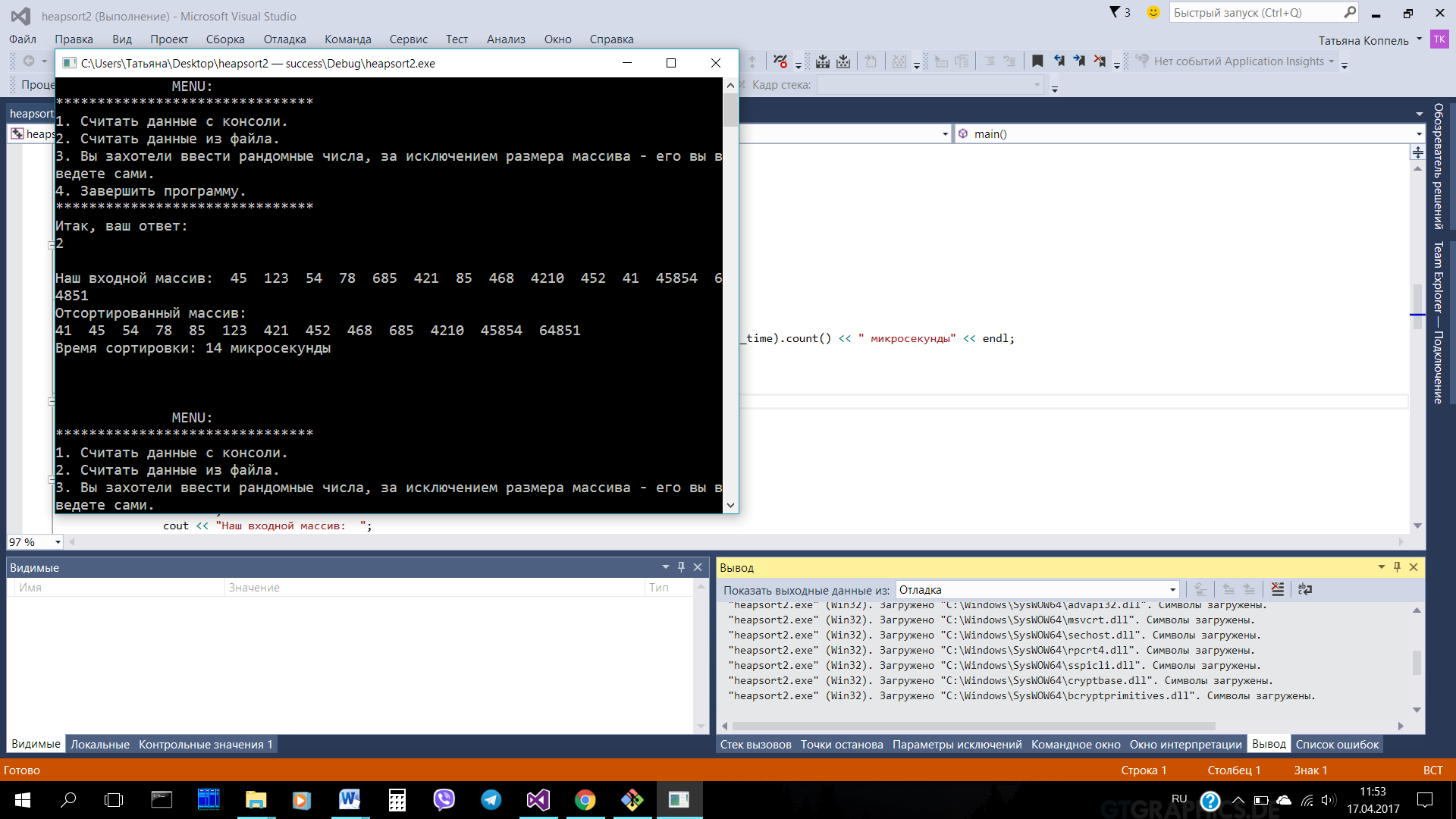
И наконец, чтобы отсортировать массив, т.е из кучи получить массив необходимо выполнить следующую функцию heap\_sort. Массив arr является глобальным для всей программы.

**Тестирование:**

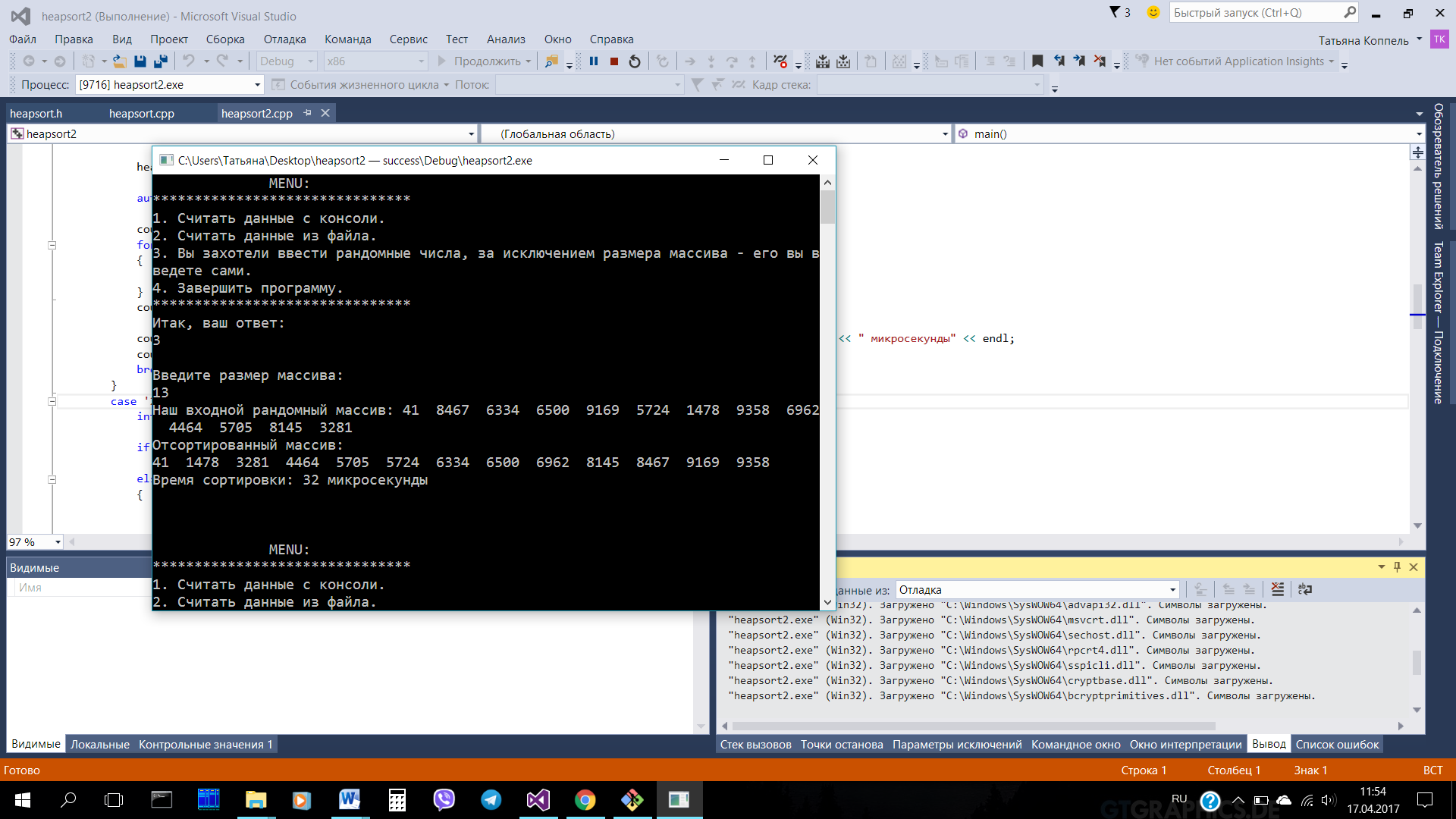
1. Введем данные с консоли:



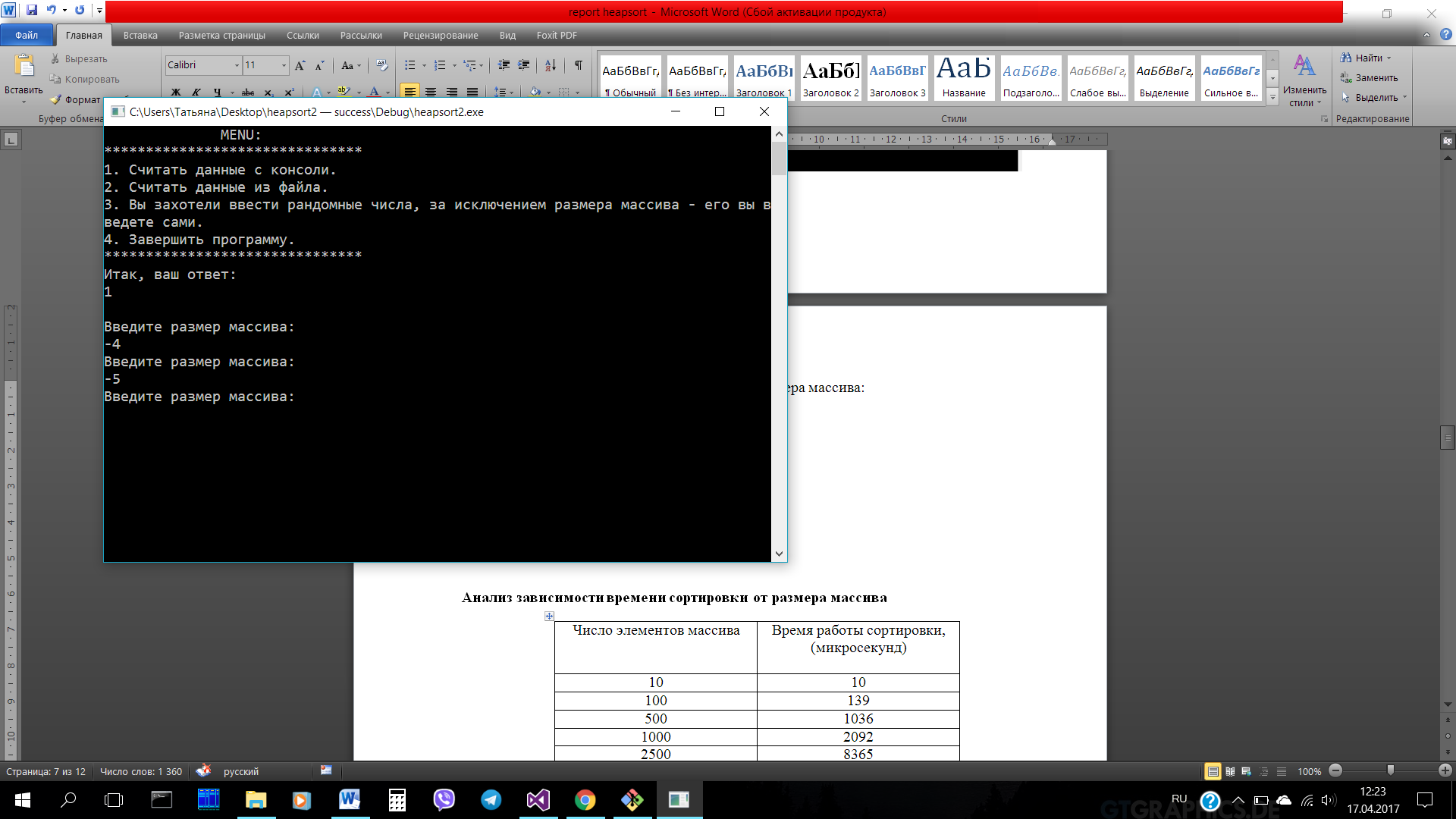
1. Введем данные из файла:



1. Запросим рандомный массив с определенным размером:



1. Проверка на введение неправильного размера массива:



**Анализ зависимости времени сортировки от размера массива**

|  |  |
| --- | --- |
| Число элементов массива | Время работы сортировки, (микросекунд) |
| 10 | 10 |
| 100 | 139 |
| 500 | 1036 |
| 1000 | 2092 |
| 2500 | 8365 |
| 5000 | 13737 |
| 7500 | 26524 |
| 10000 | 35095 |

*Таблица 1. Рандомизированный массив*

**Вывод:**

В процессе выполнения лабораторной работы был изучен и проанализирован алгоритм быстрой сортировки: была выполнена реализация на языке C++, изучена зависимость времени сортировки и числа операций от размера массива.

Из полученных результатов видно, что время сортировки растет пропорционально размеру массиву (график имеет линейную зависимость).

**Приложение А. Исходный код.**

**Файл heapsort2.cpp:**

// heapsort2.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include <iostream>

#include <algorithm>

#include "heapsort.h"

#include <fstream>

#include <cstdlib>

#include <time.h> /\* clock\_t, clock, CLOCKS\_PER\_SEC \*/

#include <ctime>

#include <windows.h>

#include <chrono>

using namespace std;

int main()

{

SetConsoleCP(1251); // для вывода кирилицы

SetConsoleOutputCP(1251); // для вывода кирилицы

bool exit = false;

char your\_answer;

ifstream fin("data.txt");

while (!exit)

{

cout << " MENU:" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "1. Считать данные с консоли." << endl;

cout << "2. Считать данные из файла." << endl;

cout << "3. Вы захотели ввести рандомные числа, за исключением размера массива - его вы введете сами." << endl;

cout << "4. Завершить программу." << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "Итак, ваш ответ:" << endl;

cin >> your\_answer;

cout << " " << endl;

switch (your\_answer)

{

case '1': {

int n = -1;

while (n <= 0)

{

cout << "Введите размер массива: " << endl;

cin >> n;

}

cout << "Введите элементы массива: " << endl;

for (int i = 0; i < n; i++)

{

cin >> arr[i];

}

auto start\_time = chrono::steady\_clock::now();

heap\_sort(arr, n);

auto end\_time = chrono::steady\_clock::now();

cout << "Отсортированный массив: " << endl;

for (int i = 0; i < n; i++)

{

cout << arr[i] << " ";

}

cout << "\n";

cout << "Время сортировки: " << chrono::duration\_cast<chrono::microseconds>(end\_time - start\_time).count() << " микросекунды" << endl;

cout << "\n\n\n";

break;

}

case '2': {

int n;

if (!fin.is\_open()) // если файл не открыт

cout << "Файл не может быть открыт!\n"; // сообщить об этом

else

{

fin >> n;

cout << "Наш входной массив: ";

for (int i = 0; i < n; i++)

{

fin >> arr[i];

cout << arr[i] << " ";

}

cout << endl;

auto start\_time1 = chrono::steady\_clock::now();

heap\_sort(arr, n);

auto end\_time1 = chrono::steady\_clock::now();

cout << "Отсортированный массив: " << endl;

for (int i = 0; i < n; i++)

{

cout << arr[i] << " ";

}

cout << "\n";

cout << "Время сортировки: " << chrono::duration\_cast<chrono::microseconds>(end\_time1 - start\_time1).count() << " микросекунды" << endl;

cout << "\n\n\n";

fin.close(); // закрываем файл

}

break;

}

case '3': {

int n;

cout << "Введите размер массива: " << endl;

cin >> n;

int random\_arr[NMAX + 1];

cout << "Наш входной рандомный массив: ";

for (int i = 0; i < n; i++)

{

random\_arr[i] = rand() % 10000; //запись случайного числа

cout << random\_arr[i] << " ";

}

cout << endl;

auto start\_time2 = chrono::steady\_clock::now();

heap\_sort(random\_arr, n);

auto end\_time2 = chrono::steady\_clock::now();

cout << "Отсортированный массив: " << endl;

for (int i = 0; i < n; i++)

{

cout << random\_arr[i] << " ";

}

cout << "\n";

cout << "Время сортировки: " << chrono::duration\_cast<chrono::microseconds>(end\_time2 - start\_time2).count() << " микросекунды" << endl;

cout << "\n\n\n";

break;

}

case '4':

exit = true;

break;

default:

cout << " Error \n";

system("cls");

}

}

//\_getch();

return 0;

}

**Файл heapsort.cpp:**

#include "stdafx.h"

#include <iostream>

#include <cstdlib>

extern const int NMAX = 10000;

extern int arr[NMAX + 1];

using namespace std;

void heapify(int \*arr, int pos, int n) {//pos - позиция корня в куче, n - количество элементов в массиве

while (2 \* pos + 1 < n) {

//пока не вышли за границу массива

int t = 2 \* pos + 1; //находим первого сына

if (2 \* pos + 2 < n && arr[2 \* pos + 2] >= arr[t])

//если второй сын существет и больше чем первый

{

t = 2 \* pos + 2;//берем его

}

if (arr[pos] < arr[t])

{

//если наибольший из сыновей больше

//чем предок, то меняем их местами

swap(arr[pos], arr[t]);

pos = t;//теперь предком является следующий элемент

}

else break;

//иначе куча с головой в pos построена

}

}

void heap\_make(int \*arr, int n)//функция построения кучи

{

for (int i = n - 1; i >= 0; i--)

{

heapify(arr, i, n);// функция добавления нового элемента в кучу

}

}

void heap\_sort(int \*arr, int n)//собственно, сама сортировка

{

heap\_make(arr, n);//делаем кучу(пирамиду)

while (n > 0)

{

swap(arr[0], arr[n - 1]);

n--;

heapify(arr, 0, n);

}

}

**Файл heapsort.h:**

//функции

const int NMAX = 100000;

int arr[NMAX + 1];

void heapify(int \*arr, int pos, int n);//нормализация кучи

void heap\_make(int \*arr, int n);//функция построения кучи

void heap\_sort(int \*arr, int n);//сортировка