**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Построение и анализ алгоритмов»**

Тема: **«Сортировка массивов»**

Студент гр.5382 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Черненко А.Б.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Шолохова О.М.

Санкт-Петербург

2017

**Задание:**

9. Пирамидальная сортировка (варианты: построение пирамиды сверху вниз или снизу вверх)

То есть написать программу на языке программирования C++, реализующую метод пирамидальной сортировки, а также подсчет количества элементарных операций в ходе сортировки и время её выполнения.

**Теоретический материал:**

Назовем пирамидой(Heap) бинарное дерево высоты k, в котором

* все узлы имеют глубину k или k-1 - дерево сбалансированное.
* при этом уровень k-1 полностью заполнен, а уровень k заполнен слева направо, то есть форма пирамиды имеет приблизительно такой вид:  
  
* выполняется "свойство пирамиды": каждый элемент меньше, либо равен родителю.

Наименее хлопотно будет хранить пирамиду - поместить ее в массив.

|  |  |
| --- | --- |
| http://algolist.manual.ru/sort/gif/18.gif | Соответствие между геометрической структурой пирамиды как дерева и массивом устанавливается по следующей схеме:   * в a[0] хранится корень дерева * левый и правый сыновья элемента a[i] хранятся, соответственнно, в a[2i+1] и a[2i+2] |

Таким образом, для массива, хранящего в себе пирамиду, выполняется следующее характеристическое свойство: a[i] >= a[2i+1] и a[i] >= a[2i+2].

Плюсы такого хранения пирамиды очевидны:

* никаких дополнительных переменных, нужно лишь понимать схему.
* узлы хранятся от вершины и далее вниз, уровень за уровнем.
* узлы одного уровня хранятся в массиве слева направо.

Запишем в виде массива пирамиду, изображенную выше. Слева-направо, сверху-вниз: 94 67 18 44 55 12 06 42. На рисунке место элемента пирамиды в массиве обозначено цифрой справа-вверху от него.

Восстановить пирамиду из массива как геометрический объект легко - достаточно вспомнить схему хранения и нарисовать, начиная от корня.

**Фаза 1 сортировки: построение пирамиды**

**1 способ: построение «снизу-вверх»**

Hачать построение пирамиды можно с a[k]...a[n], k = [size/2]. Эта часть массива удовлетворяет свойству пирамиды, так как не существует индексов i,j: i = 2i+1 ( или j = 2i+2 )... Просто потому, что такие i,j находятся за границей массива.

Следует заметить, что неправильно говорить о том, что a[k]..a[n] является пирамидой как самостоятельный массив. Это, вообще говоря, не верно: его элементы могут быть любыми. Свойство пирамиды сохраняется лишь в рамках исходного, основного массива a[0]...a[n].

Далее будем расширять часть массива, обладающую столь полезным свойством, добавляя по одному элементу за шаг. Следующий элемент на каждом шаге добавления - тот, который стоит перед уже готовой частью.

Чтобы при добавлении элемента сохранялась пирамидальность, будем использовать следующую процедуру расширения пирамиды a[i+1]..a[n] на элемент a[i] влево:

1. Смотрим на сыновей слева и справа - в массиве это a[2i+1] и a[2i+2] и выбираем наибольшего из них.
2. Если этот элемент больше a[i] - меняем его с a[i] местами и идем к шагу 2, имея в виду новое положение a[i] в массиве. Иначе конец процедуры.

Новый элемент "просеивается" сквозь пирамиду.

Вот иллюстрация процесса для пирамиды из 8-и элементов:

44 55 12 42 // 94 18 06 67

44 55 12 // 67 94 18 06 42

44 55 // 18 67 94 12 06 42

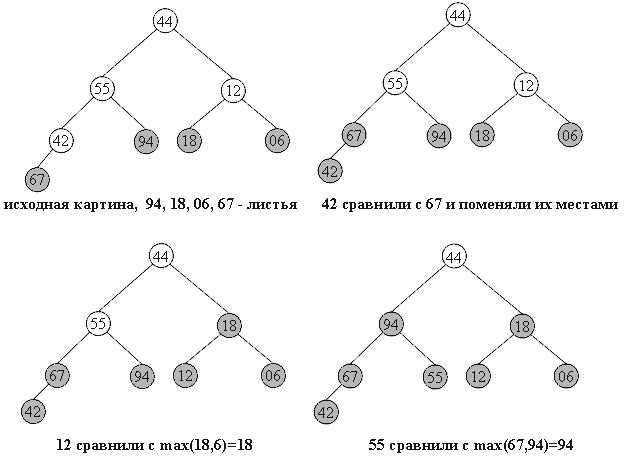
44 // 94 18 67 55 12 06 42

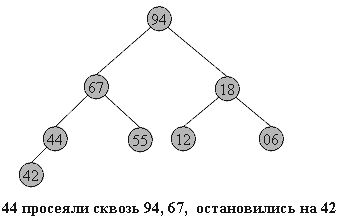
// 94 67 18 44 55 12 06 42

Справа - часть массива, удовлетворяющая свойству пирамиды, остальные элементы добавляются один за другим, справа налево.

В геометрической интерпретации ключи из начального отрезка a[size/2]...a[n] является листьями в бинарном дереве, как изображено ниже. Один за другим остальные элементы продвигаются на свои места, и так - пока не будет построена вся пирамида.

На рисунках ниже изображен процесс построения. Неготовая часть пирамиды (начало массива) окрашена в белый цвет, удовлетворяющий свойству пирамиды конец массива - в темный.





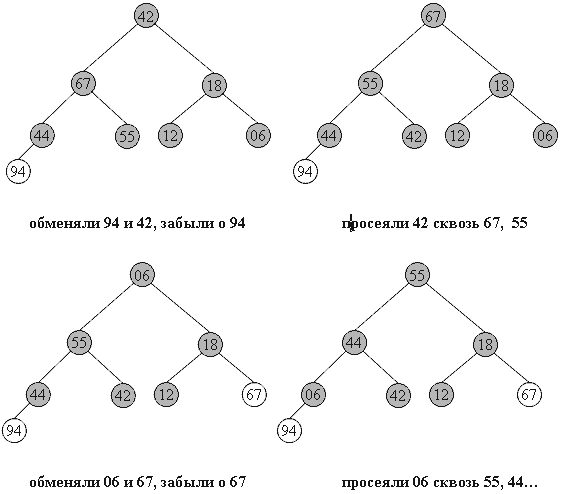
**2 способ: построение «сверху-вниз»**

Начинается построение с того, что из первых трех элементов a[0], a[1], a[2] образовываем пирамиду, и, конечно, необходимо учитывать определение «пирамиды», то есть элемент, являющийся корнем дерева, должен быть больше двух своих сыновей (a[0] > a[1] и a[0] > a[2]). Если необходимо, то меняем элементы местами, чтобы сохранить это свойство. Далее будем добавлять по одному элементу в сыновья к уже существующим элементам и на каждом шаге проверять, чтобы пирамидальность сохранялась.

**Фаза 2: собственно сортировка**

Итак, задача построения пирамиды из массива успешно решена. Как видно из свойств пирамиды, в корне всегда находится максимальный элемент. Отсюда вытекает алгоритм фазы 2:

1. Берем верхний элемент пирамиды a[0]...a[n] (первый в массиве) и меняем с последним местами. Теперь "забываем" об этом элементе и далее рассматриваем массив a[0]...a[n-1]. Для превращения его в пирамиду достаточно просеять лишь новый первый элемент.
2. Повторяем шаг 1, пока обрабатываемая часть массива не уменьшится до одного элемента.



Очевидно, в конец массива каждый раз попадает максимальный элемент из текущей пирамиды, поэтому в правой части постепенно возникает упорядоченная последовательность.

Иллюстрация 2-й фазы сортировки во внутреннем представлении пирамиды.

94 67 18 44 55 12 06 42 //

67 55 18 44 42 12 06 // 94

55 44 18 06 42 12 // 67 94

44 42 18 06 12 // 55 67 94

42 12 18 06 // 44 55 67 94

18 12 06 // 42 44 55 67 94

12 06 // 18 42 44 55 67 94

06 // 12 18 42 44 55 67 94

Построение пирамиды занимает O(n log n) операций, причем более точная оценка дает даже O(n) за счет того, что реальное время выполнения 1 фазы сортировки зависит от высоты уже созданной части пирамиды.

Вторая фаза занимает O(n log n) времени: O(n) раз берется максимум и происходит просеивание бывшего последнего элемента. Плюсом является стабильность метода: среднее число пересылок (n log n)/2, и отклонения от этого значения сравнительно малы.

Пирамидальная сортировка не использует дополнительной памяти.

Метод не является устойчивым: по ходу работы массив так "перетряхивается", что исходный порядок элементов может измениться случайным образом. Поведение неестественно: частичная упорядоченность массива никак не учитывается.

**Исследование результатов программы:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Размер массива | Количество сравнений | | Количество перестановок | | Время выполнения (мс) | |
| «снизу-вверх» | «сверху-вниз» | «снизу-вверх» | «сверху-вниз» | «снизу-вверх» | «сверху-вниз» |
| 10 | 58 | 64 | 49 | 39 | -> 0 | -> 0 |
| 50 | 508 | 660 | 360 | 353 | -> 0 | -> 0 |
| 100 | 1224 | 1605 | 823 | 812 | -> 0 | 1 |
| 500 | 8423 | 11451 | 5289 | 5197 | 1 | 1 |
| 1000 | 18834 | 25934 | 11565 | 11572 | 1 | 1 |
| 5000 | 117592 | 164741 | 69529 | 69523 | 2 | 10 |
| 10000 | 255354 | 359272 | 149174 | 148943 | 6 | 20 |
| 50000 | 1509246 | 2144404 | 861938 | 859860 | 33 | 110 |
| 100000 | 3218110 | 4590389 | 1823706 | 1820427 | 63 | 228 |

**График зависимости количества сравнений и количества перестановок от размера массива:**

**График зависимости времени выполнения сортировки от размера массива:**

**Код программы:**

#include "iostream"

#include "time.h"

using namespace std;

int start = 0, the\_end = 0, search = 0, perestanovka = 0, srav = 0;

template<class T>

void DownHeap(T m[], int k, int n, int p)

{

T new\_el;

int child;

new\_el = m[k];

while (k <= n / 2)

{

child = 2 \* k;

srav++;

if (child < n && m[child] < m[child + 1])

{

child++;

}

srav++;

if (new\_el >= m[child])

{

break;

}

m[k] = m[child];

perestanovka++;

k = child;

}

perestanovka++;

m[k] = new\_el;

}

template<class T>

void DownHeap\_v2(T m[], int kol, int n, int p)

{

int nom = 0;

while (kol >= 3)

{

if (kol % 2 == 1)

{

nom = (kol - 1) / 2;

}

else

{

nom = (kol - 2) / 2;

}

srav++;

if (m[kol] > m[nom])

{

if (p == 1)

{

cout << "Поменяем местами элементы: " << m[kol] << " и " << m[nom] << endl;

}

swap(m[kol], m[nom]);

perestanovka++;

}

kol = nom;

}

srav++;

if (m[1] > m[0] && m[1] > m[2])

{

if (p == 1)

{

cout << "Поменяем местами элементы: " << m[1] << " и " << m[0] << endl;

}

swap(m[1], m[0]);

perestanovka++;

}

srav++;

if (m[2] > m[0] && m[2] > m[1])

{

if (p == 1)

{

cout << "Поменяем местами элементы: " << m[2] << " и " << m[0] << endl;

}

swap(m[2], m[0]);

perestanovka++;

}

}

template<class T>

void HeapSort(T m[], int n, int p)

{

int i, kol = 0;

if (p == 1)

{

cout << "1 этап сортировки: Построение пирамиды 'сверху-вниз'" << endl << endl;

}

if (n >= 3)

{

if (p == 1)

{

cout << "Образуем пирамиду из трех элементов: " << m[0] << ", " << m[1] << " и " << m[2] << endl;

}

srav++;

if (m[1] > m[0] && m[1] > m[2])

{

if (p == 1)

{

cout << "Поменяем местами элементы: " << m[1] << " и " << m[0] << endl;

}

swap(m[1], m[0]);

perestanovka++;

}

srav++;

if (m[2] > m[0] && m[2] > m[1])

{

if (p == 1)

{

cout << "Поменяем местами элементы: " << m[2] << " и " << m[0] << endl;

}

swap(m[2], m[0]);

perestanovka++;

}

if (p == 1)

{

cout << "Получим массив: ";

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

{

cout << m[k] << " ";

}

cout << endl;

}

for (i = 3; i < n; i++)

{

if (p == 1)

{

cout << endl << "Добавим новый элемент в пирамиду " << m[i] << endl;

}

DownHeap\_v2(m, i, n - 1, p);

if (p == 1)

{

cout << "Получим массив: ";

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

{

cout << m[k] << " ";

}

cout << endl;

}

}

}

if (p == 1)

{

cout << endl << "2 этап сортировки: Собственно сортировка " << endl << endl;

}

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

{

if (p == 1)

{

cout << "Поменяем местами элементы: " << "корневой элемент ( " << m[0]

<< " ) и " << " самый конечный элемент, который стоит перед уже отсортированной частью ( " << m[i] << " )" << endl << endl;

}

swap(m[i], m[0]);

perestanovka++;

if (p == 1)

{

cout << "Теперь надо 'просеять' элемент, чтобы все находящиеся в нем элементы удовлетворяли определению пирамиды" << endl;

}

DownHeap(m, 0, i - 1, p);

if (p == 1)

{

cout << "Получим отсортированный массив: ";

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

{

cout << m[k] << " ";

}

cout << endl << endl;

}

}

}

template<class T>

void HeapSort\_var2(T m[], int n, int p)

{

int i, kol = 0;

if (p == 1)

{

cout << "1 этап сортировки: Построение пирамиды 'снизу-вверх'" << endl;

}

for (i = n / 2 - 1; i >= 0; --i)

{

if (p == 1)

{

cout << endl << "Добавим новый элемент в пирамиду " << m[i] << endl;

}

DownHeap(m, i, n - 1, p);

if (p == 1)

{

cout << "Получим массив: ";

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

{

cout << m[k] << " ";

}

cout << endl;

}

}

if (p == 1)

{

cout << endl << "2 этап сортировки: Собственно сортировка " << endl << endl;

}

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

{

if (p == 1)

{

cout << "Поменяем местами элементы: " << "корневой элемент ( " << m[0]

<< " ) и " << " самый конечный элемент, который стоит перед уже отсортированной частью ( " << m[i] << " )" << endl << endl;

}

swap(m[i], m[0]);

perestanovka++;

if (p == 1)

{

cout << "Теперь надо 'просеять' элемент, чтобы все находящиеся в нем элементы удовлетворяли определению пирамиды" << endl;

}

DownHeap(m, 0, i - 1, p);

if (p == 1)

{

cout << "Получим отсортированный массив: ";

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

{

cout << m[k] << " ";

}

cout << endl << endl;

}

}

}

int main()

{

setlocale(LC\_ALL, "Russian");

int n, p, var;

cout << "Введите размер массива: ";

cin >> n;

cout << endl;

cout << "Если нужно выводить промежуточные результаты, то введите '1' ";

cin >> p;

if (p != 1)

{

p = 0;

}

cout << endl;

cout << "Если нужно строить пирамиду 'сверху-вниз', то введите '0', " << endl <<

"иначе пирамида будет построена 'снизу-вверх' ";

cin >> var;

if (var != 0)

{

var = 1;

}

cout << endl;

cout << "Начальный массив: " << endl;

int \*mas = new int[n];

srand(time(NULL));

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

{

mas[i] = rand() % 2000 - 1000;

cout << mas[i] << " ";

}

cout << endl << endl;

if (var == 0)

{

start = clock();

HeapSort(mas, n, p);

the\_end = clock();

}

else

{

start = clock();

HeapSort\_var2(mas, n, p);

the\_end = clock();

}

search = the\_end - start;

cout << endl;

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

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

{

cout << mas[i] << " ";

}

cout << endl << endl;

cout << "Время затраченное на сортировку: " << search << " мс" << endl << endl;

cout << "Количество сравнений: " << srav << endl << endl;

cout << "Количество перестановок: " << perestanovka << endl << endl;

system("pause");

return 0;

}

**Тестирование программы:**

* 1 этап сортировки - построение пирамиды осуществляется способом «сверху-вниз»

1. Количество элементов в массиве: 2

*Начальный массив:* 674 -796

*Отсортированный массив:* -796 674

1. Количество элементов в массиве: 10

*Начальный массив:* -709 -963 9 161 -32 -901 245 -698 371 -372

*Отсортированный массив:* -963 -901 -709 -698 -372 -32 9 161 245 371

1. Количество элементов в массиве: 25

*Начальный массив:* -712 -882 -501 -775 -673 -712 797 30 702 -967 -362 -43 -554 106 -425 247 -757 -703 318 -994 342 725 103 147 -41

*Отсортированный массив:* -994 -967 -882 -775 -757 -712 -712 -703 -673 -554 -501 -425 -362 -43 -41 30 103 106 147 247 318 342 702 725 797

1. Количество элементов в массиве: 50

*Начальный массив:* 228 -363 -782 284 -869 -42 783 374 -162 604 -846 -547 -536 -538 431 -616 -160 -402 870 -746 -851 -690 -550 -931 -3 995 -811 306 465 464 -390 649 -757 -410 603 -148 -645 -109 -868 -239 -410 -183 567 -729 -151 865 -190 354 588 -257

*Отсортированный массив:* -931 -869 -868 -851 -846 -811 -782 -757 -746 -729 -690 -645 -616 -550 -547 -538 -536 -410 -410 -402 -390 -363 -257 -239 -190 -183 -162 -160 -151 -148 -109 -42 -3 228 284 306 354 374 431 464 465 567 588 603 604 649 783 865 870 995

1. Количество элементов в массиве: 100

*Начальный массив:* -279 459 -88 -61 299 234 -619 98 995 985 -903 -418 386 493 937 -964 431 549 309 125 -77 211 -54 -524 884 586 -901 -947 -271 327 786 -419 733 277 -447 421 560 -484 590 -886 -197 42 159 -134 -262 -140 -879 903 -402 334 608 968 860 330 -878 -79 960 558 468 159 454 -423 74 -743 650 215 800 340 350 -861 -793 -459 268 924 -979 -484 221 891 -709 799 -502 -844 -683 932 449 443 -3 -236 -326 905 633 508 -215 -449 343 534 -937 922 -562 232

*Отсортированный массив:* -979 -964 -947 -937 -903 -901 -886 -879 -878 -861 -844 -793 -743 -709 -683 -619 -562 -524 -502 -484 -484 -459 -449 -447 -423 -419 -418 -402 -326 -279 -271 -262 -236 -215 -197 -140 -134 -88 -79 -77 -61 -54 -3 42 74 98 125 159 159 211 215 221 232 234 268 277 299 309 327 330 334 340 343 350 386 421 431 443 449 454 459 468 493 508 534 549 558 560 586 590 608 633 650 733 786 799 800 860 884 891 903 905 922 924 932 937 960 968 985 995

* 1 этап сортировки - построение пирамиды осуществляется способом «снизу-вверх»

1. Количество элементов в массиве: 2

*Начальный массив:*189 -210

*Отсортированный массив:*-210 189

1. Количество элементов в массиве: 10

*Начальный массив:* 346 194 -446 955 913 -403 -577 -135 -727 -93

*Отсортированный массив:* -727 -577 -446 -403 -135 -93 194 346 913 955

1. Количество элементов в массиве: 25

*Начальный массив:* 598 -147 -393 822 704 275 -294 -160 714 -742 -136 -381 -993 46 -264 6 -646 243 -460 -785 -793 -571 -263 952 -952

*Отсортированный массив:* -993 -952 -793 -785 -742 -646 -571 -460 -393 -381 -294 -264 -263 -160 -147 -136 6 46 243 275 598 704 714 822 952

1. Количество элементов в массиве: 50

*Начальный массив:* 914 640 -272 436 -591 720 -66 807 706 719 285 -876 558 -592 -312 633 -740 18 512 116 399 -248 383 -760 -256 633 411 417 -17 -23 -643 708 -204 -264 -701 -133 234 393 -349 192 -82 374 353 -9 -860 770 627 496 -708 975

*Отсортированный массив:* -876 -860 -760 -740 -708 -701 -643 -592 -591 -349 -312 -272 -264 -256 -248 -204 -133 -82 -66 -23 -17 -9 18 116 192 234 285 353 374 383 393 399 411 417 436 496 512 558 627 633 633 640 706 708 719 720 770 807 914 975

1. Количество элементов в массиве: 100

*Начальный массив:* -664 161 -322 -357 -731 980 224 -364 781 818 -795 806 -343 -36 -339 -978 141 808 -701 -626 439 -55 -17 619 -248 -576 702 -140 -877 -557 -737 -396 659 -67 719 858 -718 -203 520 -201 -761 -764 -987 7 -769 -543 89 547 79 159 -527 38 25 586 -521 -65 -434 -66 968 999 -749 473 -72 948 -746 -558 -427 -580 -13 -903 -7 -514 -654 192 116 -499 -909 427 -935 879 -979 -76 759 165 -717 946 -228 -372 -199 75 -27 -188 389 216 -471 -355 -391 40 -858 -581

*Отсортированный массив:* -987 -979 -978 -935 -909 -903 -877 -858 -795 -769 -764 -761 -749 -746 -737 -731 -718 -717 -701 -664 -654 -626 -581 -580 -576 -558 -557 -543 -527 -521 -514 -499 -471 -434 -427 -396 -391 -372 -364 -357 -355 -343 -339 -322 -248 -228 -203 -201 -199 -188 -140 -76 -72 -67 -66 -65 -55 -36 -27 -17 -13 -7 7 25 38 40 75 79 89 116 141 159 161 165 192 216 224 389 427 439 473 520 547 586 619 659 702 719 759 781 806 808 818 858 879 946 948 968 980 999

**Вывод:**

В ходе выполнения данной лабораторной работы я изучила теоретические сведения по методу пирамидальной сортировки, а также эти навыки реализовала на языке программирования C++. Также подсчитала количество элементарных операций в ходе сортировки и время её выполнения, построила график зависимости элементарных операций в ходе сортировки и время её выполнения от размера массива.