МИНОБРНАУКИ РОССИИ

ФГБОУ ВО Череповецкий государственный университет

Институт информационных технологий

Кафедра: МПО ЭВМ

Дисциплина: Структуры и алгоритмы обработки данных

ЛАБОРАТОРНАЯ РАБОТА № 3

Вариант №4

«Сортировка массивов»

Выполнила:

студентка гр. 1ПИб-01-11оп Пикалова А.С.

Проверил:

Пышницкий Константин Михайлович

Череповец, 2023 г.

Задание:

Составить программу, проводящую сравнительную характеристику методов сортировки массивов. Программа должна выполнять следующие действия:

1. Производить сортировку массива соответствующими методами.

2. Иллюстрировать работу каждого метода на небольших массивах (размером до 10 элементов).

3. Производить сортировку каждым из методов случайного массива, уже отсортированного массива, массива, отсортированного в обратном порядке.

Засечь время. Размер массива при этом должен выбираться пользователем. После проведения сортировки, вывести данные о скорости работы методов.

|  |
| --- |
| 4.Метод прямого выбора, пирамидальная сортировка, быстрая сортировка. |

Алгоритм работы:

Функция Selection

1.1 Функция принимает значения (int\* array, int n): указатель на массив и кол-во элементов массива;

1.2 Цикл по переменной i от 0 до n-1;

1.2.1 Присваиваем min значение array [i], k значение i;

1.2.2 Цикл по переменной j от i+1 до n;

1.2.2.1 Если значение array [j] меньше значения min, то:

1. Присваиваем min значение array [j];

2. Присваиваем k значение j;

1.2.3 Присваиваем array [index] значение array [i];

1.2.4 Присваиваем array [i] значение min;

(Среди всех элементов последовательности выбирается наименьший и меняется местами с 1, среди оставшихся ищется наименьший и меняется со следующим элементом (2) и т.д.

Т.к. он реализуется 2 вложенными циклами и бежит в 2х потоках, сложность сортировки составит О(n2). Так как сортировка реализована двумя вложенными циклами первую проходит n-1 итерацию, а внутренний (n-1)/2.)

Пример работы сортировки:

14 5 9 36 17 48 20

5 14 9 36 17 48 20

5 9 14 36 17 48 20

5 9 14 17 36 48 20

5 9 14 17 20 48 36

5 9 14 17 20 36 48

Функция heapify

которая строит дерево:

1.1 Принимает значения (int arr[], int j, int bottom): указатель на массив, корень дерева и узел;

1.2 Объявляет максимальный потомок maxChild и инициализирует метку «выполнено» done = 0;

1.3 Пока есть потомки и done = 0, выполняется:

1.3.1 Если мы в последнем ряду, то:

1.3.1.1 maxChild = j \* 2 + 1; запоминаем потомок;

1.3.2 Иначе. Если левый потомок больше правого:

1.3.2.1 Запоминаем левый потомок;

1.3.2.1 Иначе запоминаем правый потомок;

1.3.3 Если элемент вершины меньше максимального потомка, то:

1.3.3.1 Меняем местами их местами;

1.3.3.2 Присваиваем j значение maxChild;

1.3.4 Иначе пирамида сформирована, done = 1;

Основная функция сортировки пирамидальной - heapSort:

2.1 Принимает значения (int\* mas, int n): указатель на массив и кол-во элементов массива;

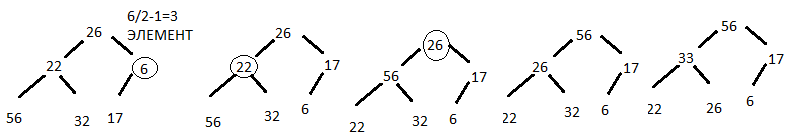
2.2 Цикл по переменной i от j / 2 - 1 до 0;

2.2.1 Передаем функции heapify(arr, i, j - 1); параметры и формируем нижний ряд пирамиды;

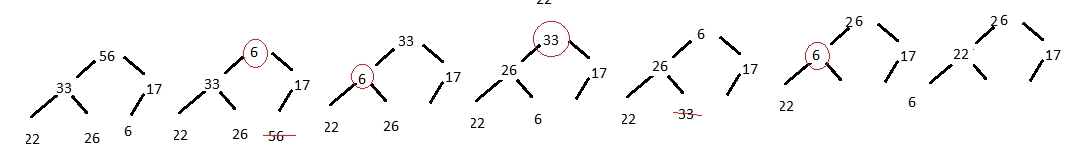
2.3 Цикл по переменной i от j - 1 до 0;

2.3.1Меняем местами и просеиваем через пирамиду остальные элементы;

(Массив представляется в виде дерева. сначала строится пирамида из элементов исходного массива, а затем осуществляется сортировка элементов. Определяем правую часть дерева, начиная с n/2-1. 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. Сложность сортировки составляет О(n\*Log2 n) )



СОРТИРУЮЩЕЕ ДЕРЕВО



Функция qSort:

1.1 Принимает значения (int\* mas, int first, int last): указатель на массив, первый и последний элемент;

1.2 Инициализация i = first, j = last, key = mas[(first + last) / 2];

1.3 Выполнять пока (i <= j):

1.3.1 Пока mas[i] < key:

1.3.1.1 i++;

1.3.2 Пока (mas[j] > key:

1.3.1.1 j++;

1.3.3 Если (i <= j), то:

1.3.3.1 меняем местами;

1.4 Если i < last, то:

1.4.1 Рекурсивно вызываем функцию и передаем ей параметры (mas, i, last);

1.5 Если (j > first), то:

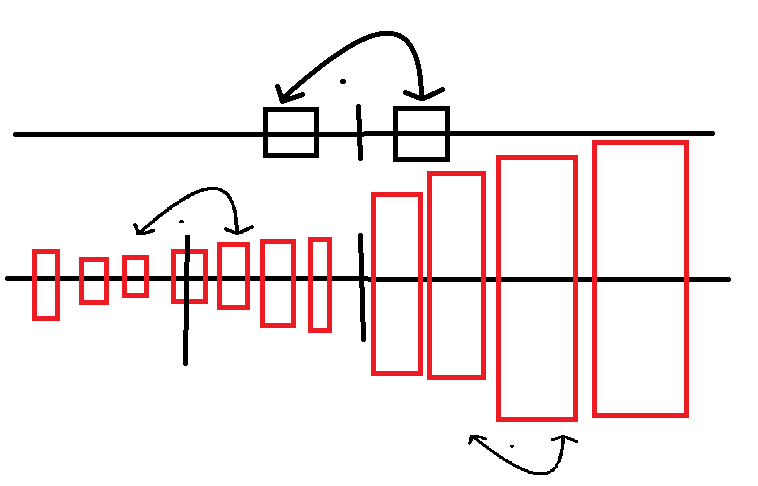
1.4.1 Рекурсивно вызываем функцию и передаем ей параметры (mas, first, j);

Основная функция быстрой сортировки quickSort:

2.1 Принимает значения (int\* mas, int n): указатель на массив и кол-во элементов массива;

2.2 Вызывает функцию и передает ей параметры qSort(mas, 0, n - 1);

(в массиве выбирается средний элемент и все эл. Упорядочатся относительно его – слева меньшие, справа большие. В наилучшем случае сложность сортировки составляет О (n\*Log2 n), а в худшем О(n2))



Главная функция main()

1. Инициализация и объявление

int n,n1=10;

int\* mas1 = new int[n1];

int\* mas2 = new int[n1];

int\* mas3 = new int[n1];

int\* array1 = new int[n];

int\* array2 = new int[n];

int\* array3 = new int[n];

2. заполнение массива:

{array1[i] = array2[i] = array3[i] = rand() % 10;

3. Показывается работа сортировок:

3.1 Создается и выводится на экран рандомный массив до 50;

3.2 Копируется рандомный массив, сортируется прямым выбором с помощью вызова функции Selection (mas2, n1); с параметрами и выводится на экран;

3.3 Копируется рандомный массив, сортируется пирамидальной сортировкой с помощью вызова функции heapSort (mas2, n1); с параметрами и выводится на экран;

3.4 Копируется рандомный массив, сортируется быстрой сортировкой и с помощью вызова функции quickSort (mas2, n1); с параметрами выводится на экран;

3.5 Вводится пользователем n;

4. Производится сортировка каждым из методов случайного массива, уже отсортированного массива, массива, отсортированного в обратном порядке, и засекается время работы каждой сортировки.

Код программы:

#include <iostream>

#include<time.h>

#include<random>

#include <chrono>

#include <algorithm>

#include <iomanip>

using namespace std;

using namespace std::chrono;

//pryamoy

void Selection(int\* array, int n) {

int min;

for (int i = 0; i < n - 1; i++) {

int k = i; min = array[i];

for (int j = i + 1; j < n; j++)

if (array[j] < min) {

min = array[j]; k = j; }

array[k] = array[i]; array[i] = min;}}

//пирамидальная сортировка

void heapify(int\* arr, int j, int bottom) {

int done = 0, maxChild, tmp;

while (j \* 2 + 1<=bottom && done == 0) {

if (j \* 2 + 1 == bottom)

maxChild = j \* 2 + 1;

else if (arr[j \* 2 + 1] > arr[j \* 2 + 2])

maxChild = j \* 2 + 1;

else maxChild = j \* 2 + 2;

if (arr[j] < arr[maxChild]) {

swap(arr[j], arr[maxChild]); j = maxChild; }

else done = 1; }}

// Основная функция, выполняющая пирамидальную сортировку

void heapSort(int \*arr, int j)

{ for (int i = j / 2 - 1; i >= 0; i--)

heapify(arr, i, j - 1);

for (int i = j - 1; i > 0; i--) {

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

heapify(arr, 0, i - 1); } }

// быстрая сортировка

void qSort(int\* mas, int first, int last) {

int i = first, j = last, key = mas[(first + last) / 2];

do { while (mas[i] < key) i++;

while (mas[j] > key) j--;

if (i <= j) {

swap(mas[i], mas[j]);

i++; j--; }

} while (i <= j);

if (i < last)qSort(mas, i, last);

if (j > first)qSort(mas, first, j); }

void quickSort(int\* mas, int n) {

qSort(mas, 0, n - 1); }

void output(int\* a, int n) {

cout.width(0);

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

cout << a[i];cout.width(2); }

cout << endl;}

void main() {

srand(time(0));

setlocale(LC\_ALL, "russian");

int n,n1=10;

int\* mas1 = new int[n1];

int\* mas2 = new int[n1];

int\* mas3 = new int[n1];

cout << "\tрандомный массив: \t";

for (int i = 0; i < n1; i++)

mas1[i] = 1 + rand() % 50;

for (int i = 0; i < n1; i++)

cout << mas1[i] << " "; cout << endl;

cout << "\tметод прямого выбора: \t";

for (int i = 0; i < n1; i++)

copy\_n(mas1, n1, mas2);

Selection(mas2, n1);

for (int i = 0; i < n1; i++)

cout << mas2[i] << " "; cout << endl;

cout << "\tпирамидальная сортировка: \t";

for (int i = 0; i < n1; i++)

copy\_n(mas1, n1, mas2);

heapSort(mas2, n1);

for (int i = 0; i < n1; i++)

cout << mas2[i] << " "; cout << endl;

cout << "\tбыстрая сортировка: \t";

for (int i = 0; i < n1; i++)

copy\_n(mas1, n1, mas3);

quickSort(mas2, n1);

for (int i = 0; i < n1; i++)

cout << mas2[i] << " "; cout << endl; cout << endl;

cout << " Задание массива" << endl;

cout << "Задайте размер массива: ";

cin >> n;

cout << endl;

int\* array1 = new int[n];

int\* array2 = new int[n];

int\* array3 = new int[n];

for (int i = 0; i < n; i++) {array1[i] = array2[i] = array3[i] = rand() % 10; 1;}

cout << "---Время работы в миллисекундах---" << endl << endl;

high\_resolution\_clock::time\_point start = high\_resolution\_clock::now();

Selection(array1, n);

high\_resolution\_clock::time\_point end = high\_resolution\_clock::now();

duration<double, milli> duration1 = end - start;

start = high\_resolution\_clock::now();

Selection(array1, n);

end = high\_resolution\_clock::now();

duration<double, milli> duration2 = end - start;

reverse(&array1[0], &array1[n]);

start = high\_resolution\_clock::now();

Selection(array1, n);

end = high\_resolution\_clock::now();

duration<double, milli> duration3 = end - start;

start = high\_resolution\_clock::now();

heapSort(array2, n);

end = high\_resolution\_clock::now();

duration<double, milli> duration4 = end - start;

start = high\_resolution\_clock::now();

heapSort(array2, n);

end = high\_resolution\_clock::now();

duration<double, milli> duration5 = end - start;

reverse(&array2[0], &array2[n]);

start = high\_resolution\_clock::now();

heapSort(array2, n);

end = high\_resolution\_clock::now();

duration<double, milli> duration6 = end - start;

start = high\_resolution\_clock::now();

quickSort(array3, n);

end = high\_resolution\_clock::now();

duration<double, milli> duration7 = end - start;

start = high\_resolution\_clock::now();

quickSort(array3, n);

end = high\_resolution\_clock::now();

duration<double, milli> duration8 = end - start;

reverse(&array3[0], &array3[n]);

start = high\_resolution\_clock::now();

quickSort(array3, n);

end = high\_resolution\_clock::now();

duration<double, milli> duration9 = end - start;

cout.width(20);

cout << endl << fixed << setprecision(5);//точность

cout << " \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "| | Рандомный | Прямой | Обратный |" << endl;

cout << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|" << endl;

cout << "| Прямого выбора | " << duration1.count() << " | " << duration2.count() << " | " << duration3.count() << " |" << endl;

cout << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|" << endl;

cout << "| Пирамидальная | " << duration4.count() << " | " << duration5.count() << " | " << duration6.count() << " |" << endl;

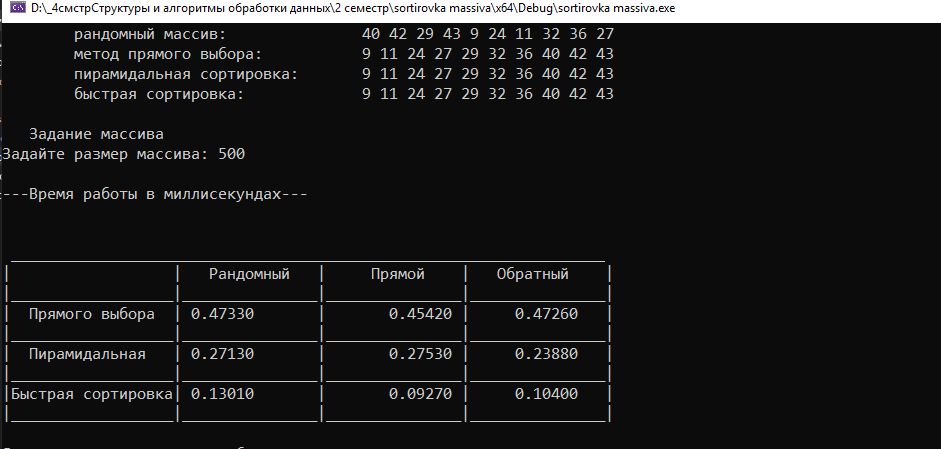
cout << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|" << endl;

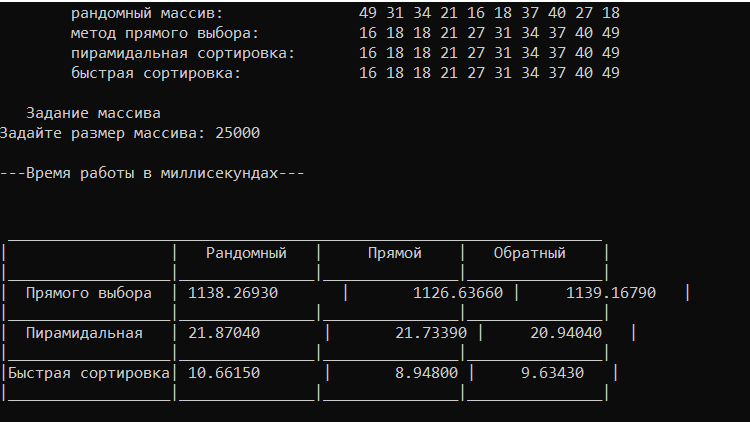
cout << "|Быстрая сортировка| " << duration7.count() << " | " << duration8.count() << " | " << duration9.count() << " |" << endl;

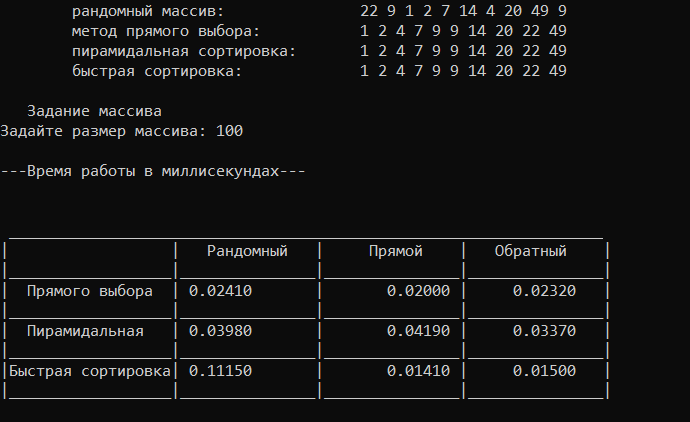
cout << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|" << endl << endl;

system("pause");}

Результат работы:







Вывод: В данной работе я воспроизвёла три вида сортировок, а именно: Метод прямого выбора, пирамидальная сортировка, быстрая сортировка. Сравнила скорость работы этих методов на массиве с большим количеством элементов.