Лабораторная работа №3

Сортировка массивов

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.

Вариант 6:

Метод прямого обмена, метод Шелла, пирамидальная сортировка.

Результат работы программы представлен в соответствии с рис.1-4.

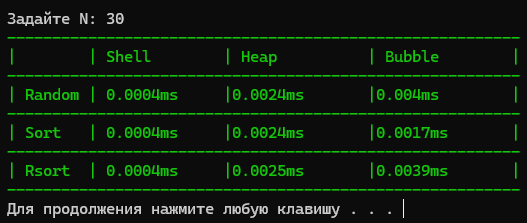
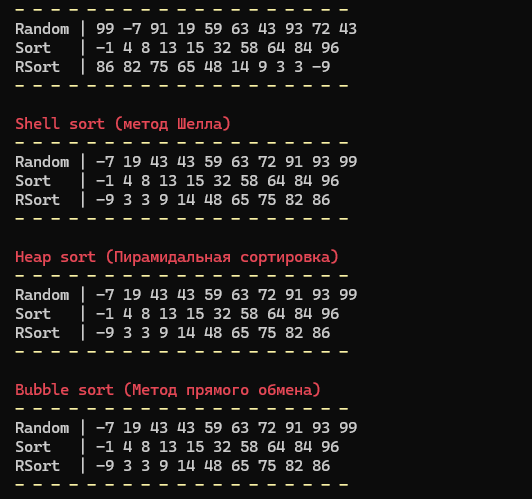


Рис. 1.

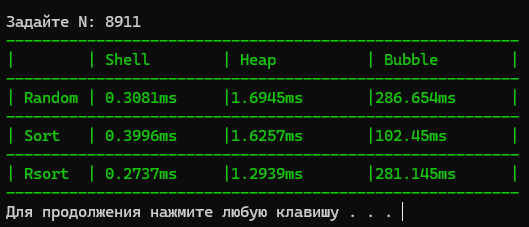


Рис. 2.

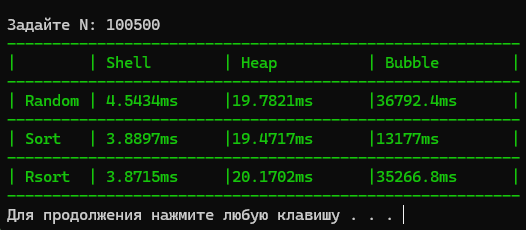


Рис. 3.

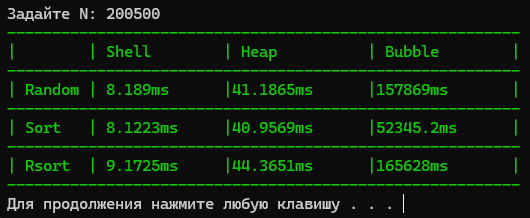


Рис. 4.

Алгоритм работы программы:

Описание функций:

1. Функция shellSort – сортировка по методу Шелла.

Проходим по всему массиву, и сортируем элементы находящиеся на расстоянии n/2, где n – длинна массива.

Уменьшаем расстояние сортировки между элементами в 2 раза, и получаем n/4.

Повторяем шаги 1 и 2, пока не дойдем до обычной одинарной сортировки.

1. Функция heapSort – пирамидальная сортировка.

Массив считаем деревом, где левый и правый помок i-того элемента находится на места 2\*i+1 и 2\*i+2 соответственно.

Дерево сортируется, тем самым максимальный элемент и пошагово поднимается в корень.

Корень меняется местами с последним элементом.

Повторяется построение, но в дерево последний элемент не входит.

Повторять, пока массив не кончится.

1. Функция bubbleSort – быстрая сортировка.

Реализует сортировку пузырьком, один из простейших алгоритмов сортировки. Алгоритм проходит по массиву множество раз, сравнивая соседние элементы и меняя их местами, если они находятся в неправильном порядке (текущий элемент меньше предыдущего). Каждый проход по массиву сортирует хотя бы один элемент на своем месте, так что после n проходов весь массив становится отсортированным.

1. Функция WhithTime.

Сохранение времени старта.

Запуск функции сортировки переданной ей.

После окончания сортировки, сохранение времени окончания.

Вычисление и возвращение результата времени работы сортировочной функции.

1. Функция printArrays – выводит переданные ей массивы.
2. Функция initArrays – заполняет массивы случайными числами в диапазоне от -9 до 99.

Запрашиваем число N.

Создание 3 массивов размером N, 1 – заполнен случайными числами, 2 - отсортирован по возрастанию, 3 – отсортирован по убыванию. Заполнение массивов происходит в функции createmas.

Запуск функции Whith\_Time для каждого массива и вида сортировки.

Вывод данных о времени работы функции в виде таблицы.

Код программы:

#include <iostream>

#include <string>

#include <stdio.h>

#include <stdlib.h>

#include <algorithm>

#include <time.h>

#include <iomanip>

#include <chrono>

using namespace std;

// Метод Шелла

void shellSort(int\* m, int n)

{

int step = n / 2;

while (step > 0) {

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

int j = i;

while (j >= 0 && m[j] > m[j + step]) {

swap(m[j], m[j + step]);

j -= step;

}

}

step /= 2;

}

}

void pushDown(int\* m, int root, int bottom)

{

int done = 0, maxChild;

while ((root \* 2 + 1 <= bottom) && !done) {

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

maxChild = root \* 2 + 1;

//следний элемент последний

else if (m[2 \* root + 1] > m[2 \* root + 2])

maxChild = root \* 2 + 1;

// левый > правый

else

maxChild = root \* 2 + 2;

// правый > левый

if (m[root] < m[maxChild]) {

swap(m[root], m[maxChild]);

root = maxChild;

}

else done = 1;

}

}

// Пирамидальная сортировка

void heapSort(int\* m, int n)

{

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

pushDown(m, i, n - 1);

}

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

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

pushDown(m, 0, i - 1);

}

}

// Метод прямого обмена

void bubbleSort(int\* m, int n)

{

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

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

{

if (m[j] < m[j - 1])

swap(m[j], m[j - 1]);

}

}

}

// Заполнение массива [случайными] числами

void initArrays(int\* r, int\* s, int\* u, int n)

{

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

r[i] = -9 + rand() % 109;

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

s[i] = -9 + rand() % 109;

sort(s, s + n);

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

u[i] = -9 + rand() % 109;

sort(u, u + n);

reverse(u, u + n);

}

void printArrays(int\* r, int\* s, int\* u, int n)

{

// Вывод верхней границы

printf("\033[1;33m");

cout << " - - - - - - - - - ";

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

cout << "- ";

cout << endl;

printf("\033[0m");

// Вывод массивов

cout << " Random | ";

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

cout << r[i] << " ";

cout << endl << " Sort | ";

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

cout << s[i] << " ";

cout << endl << " RSort | ";

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

cout << u[i] << " ";

// Вывод нижней границы

printf("\033[1;33m");

cout << endl << " - - - - - - - - - ";

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

cout << "- ";

printf("\033[0m");

cout << endl << endl;

}

double WhithTime(void (\*func)(int\*, int), int\* m, int n)

{

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

func(m, n);

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

chrono::duration<double, milli> duration = end - start;

return duration.count();

}

int main()

{

srand(time(0));

setlocale(LC\_ALL, "ru");

int n = 10;

int\* randArray = new int[n];

int\* sortedArray = new int[n];

int\* rSortedArray = new int[n];

initArrays(randArray, sortedArray, rSortedArray, n);

printArrays(randArray, sortedArray, rSortedArray, n);

printf("\033[1;31m Shell sort (метод Шелла)\033[0m\n");

shellSort(randArray, n);

shellSort(sortedArray, n);

shellSort(rSortedArray, n);

printArrays(randArray, sortedArray, rSortedArray, n);

printf("\033[1;31m Heap sort (Пирамидальная сортировка)\033[0m\n");

heapSort(randArray, n);

heapSort(sortedArray, n);

heapSort(rSortedArray, n);

printArrays(randArray, sortedArray, rSortedArray, n);

printf("\033[1;31m Bubble sort (Метод прямого обмена)\033[0m\n");

bubbleSort(randArray, n);

bubbleSort(sortedArray, n);

bubbleSort(rSortedArray, n);

printArrays(randArray, sortedArray, rSortedArray, n);

int N;

cout << "Задайте N: ";

cin >> N;

int\* randArrayN = new int[N];

int\* sortArrayN = new int[N];

int\* rSortArrayN = new int[N];

initArrays(randArrayN, sortArrayN, rSortArrayN, N);

printf("\033[1;32m");

cout << "---------------------------------------------------------" << endl;

cout << "| | Shell \t| Heap \t| Bubble \t|" << endl;

cout << "---------------------------------------------------------" << endl;

cout << "| Random | " << WhithTime(shellSort, randArrayN, N) << "ms \t|" << WhithTime(heapSort, randArrayN, N) << "ms \t|" << WhithTime(bubbleSort, randArrayN, N) << "ms \t|" << endl;

cout << "---------------------------------------------------------" << endl;

cout << "| Sort | " << WhithTime(shellSort, sortArrayN, N) << "ms \t|" << WhithTime(heapSort, sortArrayN, N) << "ms \t|" << WhithTime(bubbleSort, sortArrayN, N) << "ms \t|" << endl;

cout << "---------------------------------------------------------" << endl;

cout << "| Rsort | " << WhithTime(shellSort, rSortArrayN, N) << "ms \t|" << WhithTime(heapSort, rSortArrayN, N) << "ms \t|" << WhithTime(bubbleSort, rSortArrayN, N) << "ms \t|" << endl;

cout << "---------------------------------------------------------" << endl;

printf("\033[0m");

system("pause");

return 0;

}

Вывод

Мною была написана программа, выполняющая сортировку массивов методом прямого обмена, пирамидальной сортировкой и методом Шелла. Также программа измеряет время выполнения всех сортировок.

По результатам измерения можно сделать следующие выводы:

* На массивах, заполненных рандомными элементами, эффективнее всего себя показала сортировка Шелла, хуже всего – метод прямого обмена.
* На уже отсортированном массиве эффективнее всего была сортировка Шелла, а наименее эффективной – метод прямого обмена.
* На отсортированном в обратном порядке наиболее эффективной была сортировка Шелла, а наименее эффективным – метод прямого обмена.

Метод Шелла:

296, -192, -229, -187, -305, 242, -329, -243, -293, 98

242, -329, -243, -293, -305, 296, -192, -229, -187, 98

242, -329, -243, -293, -305, 296, -192, -229, -187, 98

-305, -329, -243, -293, -192, -229, -187, 98, 242, 296

-305, -329, -243, -293, -192, -229, -187, 98, 242, 296

-329, -305, -293, -243, -229, -192, -187, 98, 242, 296

