**Отчёт по лабораторной работе №3**

**Сортировка массивов**

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.

|  |
| --- |
| 2.Метод прямого включения, метод Шелла, пирамидальная сортировка. |

Код программы:

|  |
| --- |
| Source.cpp |
| #include <iostream>  #include <iomanip>  #include <cstdarg>  #include <sstream>  #include <random>  #include <chrono>  #include <string>  #include <vector>  class Sorting {  private:  std::vector<int> random;  std::vector<int> sorted;  std::vector<int> reverse;  void pushDown(std::vector<int>& array, int root, int bottom) {  int done = 0, maxChild;  while ((root \* 2 + 1 <= bottom) && (!done)) {  if (root \* 2 + 1 == bottom) maxChild = root \* 2 + 1;  else if (array[root \* 2 + 1] > array[root \* 2 + 2]) maxChild = root \* 2 + 1;  else maxChild = root \* 2 + 2;  if (array[root] < array[maxChild]) {  std::swap(array[root], array[maxChild]);  root = maxChild;  }  else done = 1;  }  }  public:  Sorting(int size) {  random = {};  for (int i = 0; i < size; i++)  random.push\_back(rand() % 10);  sorted = {};  for (int i = 0; i < size; i++)  sorted.push\_back(i);  reverse = {};  for (int i = 1; i < size; i++)  reverse.push\_back(size - i);  }  // Функции для получения копий массивов  std::vector<int> get\_random() { return random; }  std::vector<int> get\_sorted() { return sorted; }  std::vector<int> get\_reverse() { return reverse; }  // Вывод массива  void print(std::vector<int>& array, int iter = -1) {  if (iter != -1) {  std::cout.width(4);  std::cout << std::to\_string(iter) + ". ";  }  std::cout << "[ ";  for (int el : array)  std::cout << std::to\_string(el) + " ";  std::cout << ']' << std::endl;  }  double direct(std::vector<int>& array, bool demo = false) {  auto t1 = std::chrono::steady\_clock::now();  int iter = 0;  for (int i = 1; i < array.size(); i++) {  int x = array[i], j = i;  while (j != 0 && x < array[j - 1]) {  // Demo print  if (demo) this->print(array, iter++);  array[j] = array[j - 1];  j--;  }  array[j] = x;  }  // Demo print  if (demo) this->print(array, iter++);  auto t2 = std::chrono::steady\_clock::now();  std::chrono::duration<double, std::milli> dur = t2 - t1;  return dur.count();  }  double shell(std::vector<int>& array, bool demo = false) {  auto t1 = std::chrono::steady\_clock::now();  int iter = 0;  int step = array.size() / 2;  while (step > 0) {  for (int i = 0; i < array.size() - step; i++) {  int j = i;  while (j >= 0 && array[j] > array[j + step]) {  // Demo print  if (demo) this->print(array, iter++);  std::swap(array[j], array[j + step]);  j -= step;  }  }  step /= 2;  }  // Demo print  if (demo) this->print(array, iter++);  auto t2 = std::chrono::steady\_clock::now();  std::chrono::duration<double, std::milli> dur = t2 - t1;  return dur.count();  }  double pyramid(std::vector<int>& array, bool demo = false) {  auto t1 = std::chrono::steady\_clock::now();  int iter = 0;  for (int i = array.size() / 2 - 1; i >= 0; i--) {  // Demo print  if (demo) this->print(array, iter++);  pushDown(array, i, array.size() - 1);  }  for (int i = array.size() - 1; i >= 0; i--) {  // Demo print  if (demo) this->print(array, iter++);  std::swap(array[0], array[i]);  pushDown(array, 0, i - 1);  }  // Demo print  if (demo) this->print(array, iter++);  auto t2 = std::chrono::steady\_clock::now();  std::chrono::duration<double, std::milli> dur = t2 - t1;  return dur.count();  }  };  class Table {  private:  char cell\_ceil = 196;  char cell\_wall = 179;  char cell\_cross = 197;  int cell\_size;  int width;  public:  Table(int cell\_size, int width) {  this->cell\_size = cell\_size;  this->width = width;  }  void print\_ceiling() {  std::cout << cell\_cross;  for (int i = 0; i < width; i++)  std::cout << std::string(cell\_size, cell\_ceil) << cell\_cross;  std::cout << std::endl;  }  void print\_header(std::vector<std::string> values) {  std::cout << cell\_wall;  for (int i = 0; i < width; ++i) {  std::cout.width(cell\_size);  std::cout << values[i];  std::cout << cell\_wall;  }  std::cout << std::endl;  }  void print\_cell(std::string name, std::vector<double> values) {  std::cout << cell\_wall;  std::cout.width(cell\_size);  std::cout << name << cell\_wall;  for (int i = 0; i < width-1; ++i) {  std::ostringstream stream;  stream.precision(3);  stream << std::fixed << values[i];  std::cout.width(cell\_size);  std::cout << stream.str() + "ms";  std::cout << cell\_wall;  }  std::cout << std::endl;  }  };  int main() {  srand(time(0));  int d\_size = 10;  Sorting d\_sorting(d\_size);  std::vector<int> d\_input;  std::cout << std::endl << "Direct sorting demo:" << std::endl;  d\_input = d\_sorting.get\_random();  d\_sorting.direct(d\_input, true);  std::cout << std::endl << "Shell sorting demo:" << std::endl;  d\_input = d\_sorting.get\_random();  d\_sorting.shell(d\_input, true);  std::cout << std::endl << "Pyramid sorting demo:" << std::endl;  d\_input = d\_sorting.get\_random();  d\_sorting.pyramid(d\_input, true);    int size = 0;  std::cout << std::endl << "Input arrays size: "; std::cin >> size;  Sorting sorting(size);  std::vector<int> input;  input = sorting.get\_random();  double r\_direct = sorting.direct(input);  input = sorting.get\_sorted();  double s\_direct = sorting.direct(input);  input = sorting.get\_reverse();  double v\_direct = sorting.direct(input);  input = sorting.get\_random();  double r\_shell = sorting.shell(input);  input = sorting.get\_sorted();  double s\_shell = sorting.shell(input);  input = sorting.get\_reverse();  double v\_shell = sorting.shell(input);  input = sorting.get\_random();  double r\_pyramid = sorting.pyramid(input);  input = sorting.get\_sorted();  double s\_pyramid = sorting.pyramid(input);  input = sorting.get\_reverse();  double v\_pyramid = sorting.pyramid(input);  std::cout << "Result table:" << std::endl;  std::ostringstream stream;  stream.precision(3);  stream << std::fixed << std::max(  { r\_direct, s\_direct, v\_direct,  r\_shell, s\_shell, v\_shell,  r\_pyramid, s\_pyramid, v\_pyramid  });  int cell\_size = stream.str().size();  Table table(cell\_size + 2, 4);  table.print\_ceiling();  table.print\_header({ "", "Random", "Sorted", "Reverse" });  table.print\_ceiling();  table.print\_cell("Direct", { r\_direct, s\_direct, v\_direct });  table.print\_ceiling();  table.print\_cell("Shell", { r\_shell, s\_shell, v\_shell });  table.print\_ceiling();  table.print\_cell("Pyramid", { r\_pyramid, s\_pyramid, v\_pyramid });  table.print\_ceiling();    return 0;  } |

Алгоритм работы:

По началу программа создаёт демо элемент сортировки, который внутри содержит 3 массива (std::array) (случайный, отсортированный, обратно сортированный) размером 10 элементов. Программа вызывает 3 метода сортировки (прямого включения (direct()), Шелла (shell()), пирамидальный (pyramid())) из этого массива с параметром demo, который означает что необходимо пошагово вывести как происходит сортировка.

В методы массив передаётся по значению, что означает, что будет передана копия исходного массива.

После того как отработала демонстрационная сортировка, происходит запрос размера массива от пользователя для создания объекта сортировки. Как и в случае с демонстрационной сортировкой вызываются методы сортировки, в которые передаются копии исходных массивов. В свою очередь методы возвращают время работы самих себя в миллисекундах.

После того как отработали все вызванные методы в консоли строится сравнительная таблица времени, которые вернули каждый из методов.

Результат работы:





