Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа 15

По дисциплине «Основы алгоритмизации программирования»

На тему «Сортировка»

Выполнил:

Студент 1 курса 6 группы

Кучерук Николай Петрович

Преподаватель: асс. Андронова М.В.

2024, Минск

**Задание**

| **№ варианта** | **Условие задачи** |
| --- | --- |
| **9** | Реализовать хеш-таблицу с открытой адресацией для хранения строк. Таблица должна увеличивать свой размер втрое при достижении 70% заполнения. |

|  |
| --- |
| **Код программы** |
| #include <iostream>  #include <vector>  #include <algorithm>  #include <Windows.h>  #include <chrono>  // Функция сортировки пузырьком  void bubbleSort(std::vector<int>& arr) {  int n = arr.size();  for (int i = 0; i < n - 1; i++) {  for (int j = 0; j < n - i - 1; j++) {  if (arr[j] > arr[j + 1]) {  std::swap(arr[j], arr[j + 1]);  }  }  }  }  // Функция сортировки Хоара  void quickSort(std::vector<int>& arr, int low, int high) {  if (low < high) {  int pivot = arr[high];  int i = (low - 1);  for (int j = low; j <= high - 1; j++) {  if (arr[j] < pivot) {  i++;  std::swap(arr[i], arr[j]);  }  }  std::swap(arr[i + 1], arr[high]);  int pi = i + 1;  quickSort(arr, low, pi - 1);  quickSort(arr, pi + 1, high);  }  }  // Функция для заполнения массива случайными числами  void fillArray(std::vector<int>& arr) {  for (int& i : arr) {  i = rand() % 100; // Генерируем случайное число от 0 до 99  }  }  int main() {  system("color 70");  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  std::vector<int> sizes = { 1000, 2000, 3000, 4000, 5000 };  for (int n : sizes) {  std::vector<int> A(n), B;  fillArray(A);  for (int i = 0; i < A.size() - 1; i++) {  if (i % 2 != 0 && A[i + 1] % 2 != 0) {  B.push\_back(A[i]);  }  }  auto start = std::chrono::high\_resolution\_clock::now();  bubbleSort(B);  auto stop = std::chrono::high\_resolution\_clock::now();  auto duration = std::chrono::duration\_cast<std::chrono::microseconds>(stop - start);  std::cout << "Пузырьковая сортировка для " << n << " элементов заняла " << duration.count() << " мс.\n" << std::endl;  start = std::chrono::high\_resolution\_clock::now();  quickSort(B, 0, B.size() - 1);  stop = std::chrono::high\_resolution\_clock::now();  duration = std::chrono::duration\_cast<std::chrono::microseconds>(stop - start);  std::cout << "Сортировка Хоара для " << n << " элементов заняла " << duration.count() << " мс.\n\n" << std::endl;  }  return 0;  } |
| **Результат программы** |
|  |

Дополнительные задания

2. Разработать функцию сортировки **выбором** и добавить к списку исследуемых сортировок. Алгоритм: осуществляется поиск элемента, имеющего наименьшее значение. После того, как этот элемент найден, он меняется местами с первым элементом. Затем, начиная со второго элемента массива, осуществляется поиск следующего наименьшего значения элемента. Найденный элемент меняется местами со вторым элементом. Этот процесс продолжается до тех пор, пока все числа не будут расположены в порядке возрастания.

3. Разработать функцию сортировки методом **простой вставки** и добавить к списку исследуемых сортировок. Алгоритм: массив в процессе сортировки делится на две части: упорядоченную и неупорядоченную. Вначале весь массив неупорядочен. На каждом шаге из неупорядоченной части извлекается первый элемент, который вставляется на нужное место упорядоченной части. При этом размер упорядоченной части увеличивается на единицу. В конце весь массив окажется упорядоченным.

4. Разработать функцию сортировки методом **Шелла** и добавить к списку исследуемых сортировок. Алгоритм: метод Шелла **(**сортировка вставками с убывающим шагом) состоит в том, что упорядочиваемый массив делится на группы элементов, каждая из которых упорядочивается методом простой вставки (см. дополнительное задание 3). В процессе сортировки размеры таких групп увеличиваются до тех пор, пока все элементы не войдут в упорядоченную группу.

|  |
| --- |
| **Код программы** |
| #include <iostream>  #include <chrono> // Для измерения времени выполнения сортировок  #include <cstdlib> // Для функций rand() и srand()  #include <ctime> // Для инициализации генератора случайных чисел  using namespace std;  // Функция для сортировки массива методом выбора  void selectionSort(int arr[], int n) {  for (int i = 0; i < n - 1; ++i) {  int minIndex = i;  for (int j = i + 1; j < n; ++j) {  if (arr[j] < arr[minIndex]) {  minIndex = j;  }  }  swap(arr[i], arr[minIndex]);  }  }  // Функция для сортировки массива методом вставки  void insertionSort(int arr[], int n) {  for (int i = 1; i < n; ++i) {  int key = arr[i];  int j = i - 1;  while (j >= 0 && arr[j] > key) {  arr[j + 1] = arr[j];  j = j - 1;  }  arr[j + 1] = key;  }  }  // Функция для сортировки массива методом Шелла  void shellSort(int arr[], int n) {  for (int gap = n / 2; gap > 0; gap /= 2) {  for (int i = gap; i < n; ++i) {  int temp = arr[i];  int j;  for (j = i; j >= gap && arr[j - gap] > temp; j -= gap) {  arr[j] = arr[j - gap];  }  arr[j] = temp;  }  }  }  int main() {  setlocale(LC\_ALL, "RUS");  int n;  cout << "Введите размер массива A: ";  cin >> n;  // Генерируем массив A  int\* A = new int[n]; // Выделение памяти под массив A  srand(time(0)); // Инициализация генератора случайных чисел  cout << "Массив A:\n";  for (int i = 0; i < n; ++i) {  A[i] = rand() % 100; // Генерация случайных чисел от 0 до 99  cout << A[i] << " ";  }  cout << endl;  // Находим максимальный элемент в массиве A и его индекс  int maxElement = A[0];  int maxIndex = 0;  for (int i = 1; i < n; ++i) {  if (A[i] > maxElement) {  maxElement = A[i];  maxIndex = i;  }  }  // Создаем массив B и переносим нечетные элементы правее максимального элемента из массива A  int\* B = new int[n]; // Выделение памяти под массив B  int countB = 0; // Переменная для подсчета элементов в массиве B  for (int i = maxIndex + 1; i < n; ++i) {  if (i % 2 != 0) {  B[countB] = A[i];  countB++;  }  }  // Измеряем время выполнения сортировки выбором  auto start = chrono::high\_resolution\_clock::now(); // Начало измерения времени  selectionSort(B, countB);  auto end = chrono::high\_resolution\_clock::now(); // Конец измерения времени  chrono::microseconds duration = chrono::duration\_cast<chrono::microseconds>(end - start); // Вычисление продолжительности времени в микросекундах  cout << "Сортировка методом выбора заняла: " << duration.count() << " микросекунд" << endl;  // Выводим отсортированный массив B  cout << "Массив B (отсортированный методом выбора):\n";  for (int i = 0; i < countB; ++i) {  cout << B[i] << " ";  }  cout << endl;  // Копируем массив A в массив B и переносим нечетные элементы правее максимального элемента из массива A  countB = 0; // Сбрасываем счетчик элементов в массиве B  for (int i = maxIndex + 1; i < n; ++i) {  if (i % 2 != 0) {  B[countB] = A[i];  countB++;  }  }  // Измеряем время выполнения сортировки методом простой вставки  start = chrono::high\_resolution\_clock::now(); // Начало измерения времени  insertionSort(B, countB);  end = chrono::high\_resolution\_clock::now(); // Конец измерения времени  duration = chrono::duration\_cast<chrono::microseconds>(end - start); // Вычисление продолжительности времени в микросекундах  cout << "Сортировка методом простой вставки заняла: " << duration.count() << " микросекунд" << endl;  // Выводим отсортированный массив B  cout << "Массив B (отсортированный методом простой вставки):\n";  for (int i = 0; i < countB; ++i) {  cout << B[i] << " ";  }  cout << endl;  // Копируем массив A в массив B и переносим нечетные элементы правее максимального элемента из массива A  countB = 0; // Сбрасываем счетчик элементов в массиве B  for (int i = maxIndex + 1; i < n; ++i) {  if (i % 2 != 0) {  B[countB] = A[i];  countB++;  }  }  // Измеряем время выполнения сортировки методом Шелла  start = chrono::high\_resolution\_clock::now(); // Начало измерения времени  shellSort(B, countB);  end = chrono::high\_resolution\_clock::now(); // Конец измерения времени  duration = chrono::duration\_cast<chrono::microseconds>(end - start); // Вычисление продолжительности времени в микросекундах  cout << "Сортировка методом Шелла заняла: " << duration.count() << " микросекунд" << endl;  // Выводим отсортированный массив B  cout << "Массив B (отсортированный методом Шелла):\n";  for (int i = 0; i < countB; ++i) {  cout << B[i] << " ";  }  cout << endl;  // Освобождаем выделенную память для массивов A и B  delete[] A;  delete[] B;  return 0;  } |
| **Результат программы** |
|  |