**ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

**Факультет инфокоммуникационных Сетей и систем (иксс)**

**кафедра программной инженерии и вычислительной техники (пиивт)**

**Лабораторная работа №1.**

**«Методы сортировки»**

Дисциплина: «Алгоритмы и Структуры Данных»

Выполнили: Студент группы ИКПИ-33  
 Коньков Максим

Семенихин Арсений

Принял: к.т.н., доцент кафедры ПииВТ Дагаев А.В.

### **1. Описание алгоритмов**

#### - Сортировка вставками (Insertion Sort)

Алгоритм работает по принципу вставки элементов на правильное место в уже отсортированной части массива. На каждом шаге берётся следующий элемент, сравнивается с предыдущими и вставляется в нужное место.

* **Сложность:**
  + Худший случай: O(n2) (если массив отсортирован в обратном порядке).
  + Лучший случай: O(n) (если массив уже отсортирован).
  + Средний случай: O(n2).
* **Плюсы:** Хорошо работает для почти отсортированных массивов.
* **Минусы:** Неэффективен для больших массивов.

#### - Сортировка выбором (Selection Sort)

Алгоритм проходит по массиву, выбирает минимальный элемент и меняет его местами с текущим элементом. Повторяется до тех пор, пока весь массив не будет отсортирован.

* **Сложность:**
  + Всегда O(n2), независимо от порядка элементов.
* **Плюсы:** Простая реализация, мало обменов элементов.
* **Минусы:** Неэффективен для больших массивов.

#### - Пузырьковая сортировка (Bubble Sort)

Сравнивает соседние элементы и меняет их местами, если они расположены в неправильном порядке. Повторяется, пока массив не станет отсортированным.

* **Сложность:**
  + Худший случай: O(n2).
  + Лучший случай: O(n) (если массив уже отсортирован).
* **Плюсы:** Простая реализация.
* **Минусы:** Один из самых медленных алгоритмов.

**1.1. Сортировка вставками**

#### ****Пошаговый процесс работы алгоритма:****

1. Начинаем со второго элемента (первый элемент считается уже отсортированным).
2. Берём текущий элемент (ключ) и сравниваем его с предыдущими.
3. Пока предыдущий элемент больше текущего, сдвигаем его вправо.
4. Когда найдено подходящее место, вставляем текущий элемент.
5. Повторяем процесс для всех элементов массива.

#### ****Пример работы:****

Допустим, у нас есть массив:  
[5, 3, 8, 1, 2]

**Шаги сортировки:**

1. 3 сравниваем с 5, двигаем 5 вправо → [5, 5, 8, 1, 2], вставляем 3 → [3, 5, 8, 1, 2]
2. 8 уже на своём месте → [3, 5, 8, 1, 2]
3. 1 сравниваем с 8, 5, 3, двигаем их вправо → [3, 3, 5, 8, 2], вставляем 1 → [1, 3, 5, 8, 2]
4. 2 сравниваем с 8, 5, 3, двигаем их вправо → [1, 3, 3, 5, 8], вставляем 2 → [1, 2, 3, 5, 8]

**1.2. Сортировка выбором**

#### ****Пошаговый процесс работы алгоритма:****

1. Находим наименьший элемент в массиве.
2. Меняем его местами с первым элементом.
3. Повторяем процесс для оставшейся части массива.

#### ****Пример работы:****

Допустим, у нас есть массив:  
[5, 3, 8, 1, 2]

**Шаги сортировки:**

1. Минимальный элемент — 1, меняем с 5 → [1, 3, 8, 5, 2]
2. Минимальный в оставшейся части — 2, меняем с 3 → [1, 2, 8, 5, 3]
3. Минимальный — 3, меняем с 8 → [1, 2, 3, 5, 8]
4. 5 и 8 уже на месте → [1, 2, 3, 5, 8]

**Конечный результат:** [1, 2, 3, 5, 8]

**1.3. Пузырьковая сортировка**

#### ****Пошаговый процесс работы алгоритма:****

1. Проходим по массиву и сравниваем соседние элементы.
2. Если они стоят в неправильном порядке, меняем их местами.
3. Повторяем процесс, пока массив не будет отсортирован.

#### ****Пример работы:****

Допустим, у нас есть массив:  
[5, 3, 8, 1, 2]

**Шаги сортировки:**

1. 5 > 3, меняем → [3, 5, 8, 1, 2]
2. 5 < 8, оставляем → [3, 5, 8, 1, 2]
3. 8 > 1, меняем → [3, 5, 1, 8, 2]
4. 8 > 2, меняем → [3, 5, 1, 2, 8]

(Повторяем процесс для оставшихся элементов, пока не получим [1, 2, 3, 5, 8].)

### **2. Сравнение алгоритмов**

| Алгоритм | Сложность (средний случай) | Плюсы | Минусы |
| --- | --- | --- | --- |
| Вставками | O(n2) | Быстро на почти отсортированных массивах | Медленно на больших массивах |
| Выбором | O(n2) | Минимальное количество обменов | Независимо от порядка работает за O(n2) |
| Пузырьковая | O(n2) | Прост в реализации | Один из самых медленных |

В реальных задачах предпочтение отдают более эффективным алгоритмам, таким как **быстрая сортировка (Quick Sort) или сортировка слиянием (Merge Sort)**.

### 3. Пошаговое решение

1. **Генерация случайных чисел**
   * Функция generateRandomNumbersFile() создаёт файл с 10 000 случайных чисел в диапазоне [−10000,10000].
2. **Чтение данных**
   * readNumbersFromFile() считывает данные в vector<int>.
3. **Сортировка данных разными методами**
   * insertionSort()
   * selectionSort()
   * bubbleSort()
4. **Запись результатов**
   * writeNumbersToFile() сохраняет отсортированные массивы в файлы.
   * Отсортированные массивы сохраняются в .txt файлы через writeSortedNumbersToFile().
5. **Вывод в консоль**
   * Завершение программы с сообщением о завершении.
6. **Завершение работы**
   * Файл закрывается, программа сообщает о завершении.

## ****4. Описание программы****

* **Язык:** C++
* **Среда разработки:** Vim в ОС Linux
* **Структура кода:**
  + **Функции сортировки:** insertionSort(), selectionSort(), bubbleSort()
  + **Работа с файлами:** Результаты работы сортировок записываются в CSV-файл (sorting\_results.csv), а сами отсортированные числа в файлы «сортировка\_sorted.txt» соответственно своей сортировке.
  + **Генерация случайных данных:** generateRandomNumbers()
  + **Измерение времени выполнения:** measureSortTime()
  + **Основная логика:** main(), где генерируются случайные массивы, замеряется время выполнения сортировок и сохраняются результаты.

## ****Функции программы:****

### ****1.**** void insertionSort(vector<int>& arr) ****– Сортировка вставками****

**Описание:**  
Метод сортировки вставками, в котором элементы поочерёдно вставляются на своё место.

**Аргументы:**

* vector<int>& arr – ссылка на вектор целых чисел, который нужно отсортировать.

**Алгоритм:**

1. Начинаем со второго элемента (первый считаем уже отсортированным).
2. Сравниваем его с предыдущими элементами.
3. Если текущий элемент меньше предыдущего, сдвигаем предыдущий вправо.
4. Вставляем элемент в правильную позицию.
5. Повторяем процесс для всех элементов массива.

**Пример работы:**  
Вход: [5, 3, 8, 1, 2]  
Выход: [1, 2, 3, 5, 8]

### ****2.**** void selectionSort(vector<int>& arr) ****– Сортировка выбором****

**Описание:**  
Метод сортировки выбором, находя минимальный элемент и меняя его местами с текущим.

**Аргументы:**

* vector<int>& arr – ссылка на вектор целых чисел.

**Алгоритм:**

1. Ищем минимальный элемент среди оставшихся.
2. Меняем его местами с текущим элементом.
3. Повторяем для всех элементов массива.

**Пример работы:**  
Вход: [5, 3, 8, 1, 2]  
Выход: [1, 2, 3, 5, 8]

### ****3.**** void bubbleSort(vector<int>& arr) ****– Пузырьковая сортировка****

**Описание:**  
Метод пузырьковой сортировки, который последовательно обменивает соседние элементы, если они расположены неправильно.

**Аргументы:**

* vector<int>& arr – ссылка на вектор целых чисел.

**Алгоритм:**

1. Сравниваем соседние элементы, если порядок нарушен – меняем местами.
2. Повторяем процесс, пока массив не станет отсортированным.

**Пример работы:**  
Вход: [5, 3, 8, 1, 2]  
Выход: [1, 2, 3, 5, 8]

### ****4.**** vector<int> generateRandomNumbers(int count, int minVal, int maxVal) ****– Генерация случайного массива****

**Описание:**  
Генерирует массив случайных чисел в заданном диапазоне.

**Аргументы:**

* int count – количество чисел в массиве.
* int minVal – минимальное значение чисел.
* int maxVal – максимальное значение чисел.

**Алгоритм:**

1. Создаётся пустой вектор нужного размера.
2. Используется rand() для генерации чисел в заданном диапазоне.
3. Заполненный массив возвращается.

**Пример работы:**  
Вход: generateRandomNumbers(5, 1, 10)  
Выход: [3, 8, 1, 6, 9] (результат зависит от генерации)

### ****5.**** long long measureSortTime(void (\*sortFunc)(vector<int>&), vector<int> arr) ****– Измерение времени выполнения****

**Описание:**  
Функция измеряет время выполнения переданной сортировки.

**Аргументы:**

* void (\*sortFunc)(vector<int>&) – указатель на функцию сортировки.
* vector<int> arr – массив, который нужно отсортировать.

**Алгоритм:**

1. Фиксируется начальное время (steady\_clock::now()).
2. Вызывается функция сортировки.
3. Фиксируется время завершения.
4. Разница (в миллисекундах) возвращается.

**Пример работы:**  
Вход: measureSortTime(insertionSort, [5, 3, 8, 1, 2])  
Выход: 0 (для маленького массива) или 123 мс (для большого массива)

**6. void writeSortedNumbersToFile(const vector<int>& arr, const string& filename) — Запись отсортированных массивов в файлы**

#### ****Описание:****

Функция записывает отсортированные числа из вектора в текстовый файл. Используется для сохранения результатов сортировки каждого алгоритма.

#### ****Аргументы:****

* const vector<int>& arr – ссылка на отсортированный массив целых чисел.
* const string& filename – строка с именем файла для записи.

#### ****Алгоритм работы:****

1. Открывается файл filename для записи.
2. Если файл открыть не удалось – выводится сообщение об ошибке.
3. В цикле записываются все элементы массива, каждый с новой строки.
4. Файл закрывается после записи.

#### ****Пример работы:****

Вход:

vector<int> sortedNumbers = {1, 2, 3, 5, 8};

writeSortedNumbersToFile(sortedNumbers, "sorted\_output.txt");

### ****7.**** int main() ****– Основная логика программы****

**Описание:**  
Главная функция программы. Генерирует массивы, замеряет время работы сортировок и записывает результаты в CSV-файл.

**Алгоритм:**

1. Определяется набор размеров массивов (sizes).
2. Открывается CSV-файл sorting\_results.csv для записи.
3. Для каждого размера n:
   * Генерируется случайный массив.
   * Измеряется время работы трёх алгоритмов сортировки.
   * Результаты записываются в файл.
   * Выводится информация о времени выполнения в консоль.
4. Закрывается файл, программа завершает работу.

\

**5. Результаты работы**

В ходе выполнения поставленной задачи были получены следующие результаты:



Рис. 1. Сравнение временных затрат на сортировки: вставками (Insertion Sort — IS), выбором (Selection Sort — IS), пузырьком (Bubble Sort — BS).

На приведённом графике жёлтым цветом обозначена сортировка пузырьком, оранжевым – сортировка выбором, а синим – сортировка вставками. Согласно данным, сортировка пузырьком демонстрирует наибольшее время выполнения и сильнее всех замедляется с увеличением размера входного массива. Сортировки выбором и вставками показывают схожее время выполнения, но сортировка вставками в некоторых случаях может работать быстрее за счёт адаптивности. В целом, все три алгоритма имеют квадратичную сложность O(n^2), что делает их менее эффективными для больших массивов.

**Приложение А: Листинг кода**

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <vector>  #include <algorithm>  #include <cstdlib>  #include <ctime>  #include <chrono>  using namespace std;  using namespace std::chrono;  // Функция сортировки вставками  void insertionSort(vector<int>& arr) {  int n = arr.size();  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--;  }  arr[j + 1] = key;  }  }  // Функция сортировки выбором  void selectionSort(vector<int>& arr) {  int n = arr.size();  for (int i = 0; i < n - 1; i++) {  int minIdx = i;  for (int j = i + 1; j < n; j++) {  if (arr[j] < arr[minIdx]) {  minIdx = j;  }  }  swap(arr[i], arr[minIdx]);  }  }  // Функция сортировки пузырьком  void bubbleSort(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]) {  swap(arr[j], arr[j + 1]);  }  }  }  }  // Генерация случайного массива  vector<int> generateRandomNumbers(int count, int minVal, int maxVal) {  vector<int> numbers(count);  srand(time(0));  for (int i = 0; i < count; i++) {  numbers[i] = rand() % (maxVal - minVal + 1) + minVal;  }  return numbers;  }  // Функция замера времени выполнения  long long measureSortTime(void (\*sortFunc)(vector<int>&), vector<int> arr) {  auto start = steady\_clock::now();  sortFunc(arr);  auto stop = steady\_clock::now();  return duration\_cast<milliseconds>(stop - start).count();  }  // Функция записи отсортированных данных в файл  void writeSortedNumbersToFile(const vector<int>& arr, const string& filename) {  ofstream outFile(filename);  if (!outFile) {  cerr << "Ошибка при открытии файла " << filename << endl;  return;  }  for (int num : arr) {  outFile << num << "\n";  }  outFile.close();  }  int main() {  vector<int> sizes = {1000, 2000, 5000, 10000, 15000, 20000, 50000, 100000};  ofstream results("sorting\_results.csv");  results << "Elements,Insertion Sort (ms),Selection Sort (ms),Bubble Sort (ms)\n";  for (int n : sizes) {  cout << "Тест для " << n << " элементов..." << endl;  vector<int> numbers = generateRandomNumbers(n, -10000, 10000);  // Копии массива для каждого метода сортировки  vector<int> insertionArr = numbers;  vector<int> selectionArr = numbers;  vector<int> bubbleArr = numbers;  // Измерение времени сортировки  long long insertionTime = measureSortTime(insertionSort, insertionArr);  long long selectionTime = measureSortTime(selectionSort, selectionArr);  long long bubbleTime = measureSortTime(bubbleSort, bubbleArr);  // Запись времени выполнения в CSV  results << n << "," << insertionTime << "," << selectionTime << "," << bubbleTime << "\n";  // Запись отсортированных массивов в файлы  writeSortedNumbersToFile(insertionArr, "insertion\_sorted\_" + to\_string(n) + ".txt");  writeSortedNumbersToFile(selectionArr, "selection\_sorted\_" + to\_string(n) + ".txt");  writeSortedNumbersToFile(bubbleArr, "bubble\_sorted\_" + to\_string(n) + ".txt");  cout << "Вставками: " << insertionTime << " мс | "  << "Выбором: " << selectionTime << " мс | "  << "Пузырьком: " << bubbleTime << " мс" << endl;  }  results.close();  cout << "Результаты сохранены в sorting\_results.csv" << endl;  cout << "Отсортированные данные записаны в .txt файлы" << endl;  return 0;  } |