UNIVERSITATEA DE STAT DIN REPUBLICA MOLDOVA

СПЕЦИАЛЬНОСТЬ: “Informatica aplicată”

**Индивидуальная работа №1**

По предмету: Алгоритмы и структуры данных

Тема: Методы и процедуры поиска

Преподаватель: Выполнил:

M.Malai Rusanov Nichita

Кишинёв, 2023

Условие

Создать класс с пятью полями разных типов, включая уникальный идентификатор (id), а также массива из 50 экземпляров этого класса с заполненными полями.

Требуется реализовать программу на любом удобном вам языке программирования, в которой реализованы методы сортировки массивов данных. Чем больше методов, тем выше оценка. Для каждого метода сортировки проанализировать теоретическую и практическую сложности.

В конечной программе необходимо включить комментарии, объясняющие ваш подход и использованные алгоритмы. Программу следует прикрепить к заданию на moodle и продемонстрировать преподавателю на лабораторном занятии.

Для каждого алгоритма вывести следующую информацию:

* теоретическая оценка сложности
* количество сравнений
* количество перестановок
* время выполнения алгоритма

Методы сортировки:

* Сортировка пузырьком (Bubble Sort)
* Быстрая сортировка (Quick Sort)
* Сортировка вставками (Insertion Sort)
* Сортировка слиянием (Merge Sort)
* Сортировка выбором (Selection Sort)

Решение

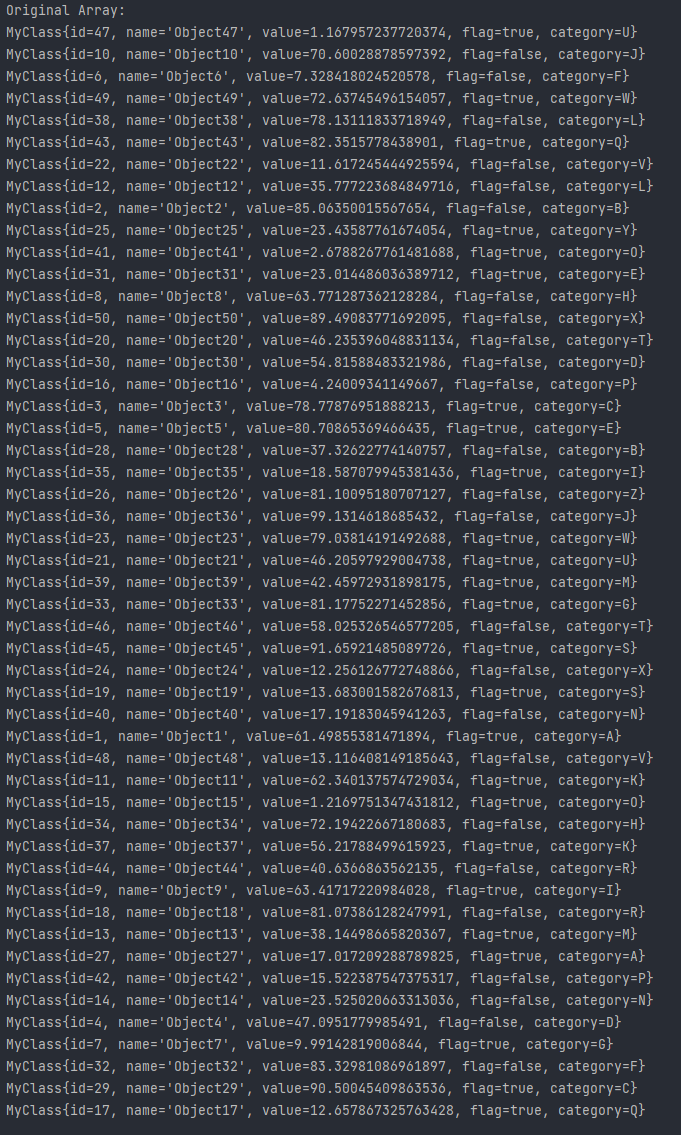
MyClass.java:

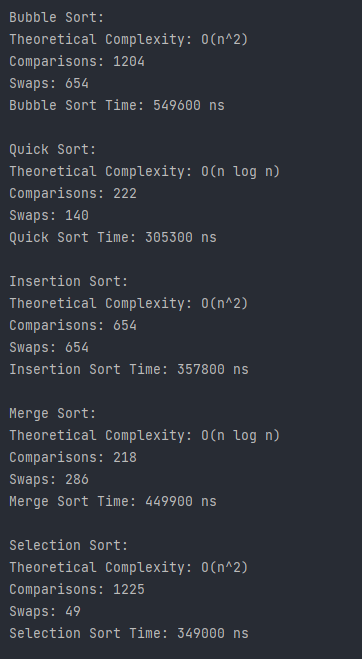
*import* java.util.Arrays;  
  
*class* MyClass {  
 *// Поля разных типов  
 private int* id;  
 *private* String name;  
 *private double* value;  
 *private boolean* flag;  
 *private char* category;  
  
 *// Конструктор класса  
 public* MyClass(*int* id, String name, *double* value, *boolean* flag, *char* category) {  
 *this*.id = id;  
 *this*.name = name;  
 *this*.value = value;  
 *this*.flag = flag;  
 *this*.category = category;  
 }  
  
 *// Геттер для уникального идентификатора  
 public int* getId() {  
 *return* id;  
 }  
  
 *// Метод для поиска объекта по id в неотсортированном массиве  
 @Override  
 public* String toString() {  
 *return* "MyClass{" +  
 "id=" + id +  
 ", name='" + name + '\'' +  
 ", value=" + value +  
 ", flag=" + flag +  
 ", category=" + category +  
 '}';  
 }  
}

SortingExample.java:

*import* java.util.ArrayList;  
*import* java.util.Arrays;  
*import* java.util.Collections;  
*import* java.util.List;  
  
*public class* SortingExample {  
  
 *private static int* mergeComparisons = 0;  
 *private static int* mergeSwaps = 0;  
  
 *private static int* quickComparisons = 0;  
 *private static int* quickSwaps = 0;  
  
 *// Метод для генерации массива экземпляров класса  
 private static* MyClass[] generateArray() {  
 MyClass[] array = *new* MyClass[50];  
 *for* (*int* i = 0; i < 50; i++) {  
 array[i] = *new* MyClass(i + 1, "Object" + (i + 1), Math.random() \* 100, i % 2 == 0, (*char*) ('A' + i % 26));  
 }  
  
 List<MyClass> list = *new* ArrayList<>(Arrays.asList(array));  
 Collections.shuffle(list);  
 list.toArray(array);  
  
 *return* array;  
 }  
  
 *// Метод для вывода массива  
 private static void* printArray(MyClass[] array) {  
 Arrays.stream(array).forEach(System.out::println);  
 }  
  
 *private static void* bubbleSort(MyClass[] array) {  
 *int* n = array.length;  
 *boolean* swapped;  
 *int* comparisons = 0;  
 *int* swaps = 0;  
  
 *for* (*int* i = 0; i < n - 1; i++) {  
 swapped = *false*;  
 *for* (*int* j = 0; j < n - i - 1; j++) {  
 comparisons++;  
 *if* (array[j].getId() > array[j + 1].getId()) {  
 *// Обмен элементов* MyClass temp = array[j];  
 array[j] = array[j + 1];  
 array[j + 1] = temp;  
 swapped = *true*;  
 swaps++;  
 }  
 }  
  
 *// Если не было обменов на данной итерации, массив уже отсортирован  
 if* (!swapped) {  
 *break*;  
 }  
 }  
  
 System.out.println("\nBubble Sort:");  
 System.out.println("Theoretical Complexity: O(n^2)");  
 System.out.println("Comparisons: " + comparisons);  
 System.out.println("Swaps: " + swaps);  
 }  
  
 *// Метод для быстрой сортировки  
 // Метод для быстрой сортировки  
 private static void* quickSort(MyClass[] array) {  
 quickComparisons = 0;  
 quickSwaps = 0;  
 *// Реализация быстрой сортировки (Quicksort)* quickSortHelper(array, 0, array.length - 1);  
  
 System.out.println("Quick Sort:");  
 System.out.println("Theoretical Complexity: O(n log n)");  
 System.out.println("Comparisons: " + quickComparisons);  
 System.out.println("Swaps: " + quickSwaps);  
 }  
  
 *private static void* quickSortHelper(MyClass[] array, *int* low, *int* high) {  
 *if* (low < high) {  
 *int* partitionIndex = partition(array, low, high);  
  
 *// Рекурсивно сортируем элементы перед и после partition* quickSortHelper(array, low, partitionIndex - 1);  
 quickSortHelper(array, partitionIndex + 1, high);  
 }  
 }  
  
 *private static int* partition(MyClass[] array, *int* low, *int* high) {  
 *int* pivot = array[high].getId();  
 *int* i = low - 1;  
  
 *for* (*int* j = low; j < high; j++) {  
 quickComparisons++;  
 *if* (array[j].getId() < pivot) {  
 i++;  
  
 *// Обмен элементов* MyClass temp = array[i];  
 array[i] = array[j];  
 array[j] = temp;  
 quickSwaps++;  
 }  
 }  
  
 *// Обмен pivot и элемента на позиции i+1* MyClass temp = array[i + 1];  
 array[i + 1] = array[high];  
 array[high] = temp;  
 quickSwaps++;  
  
 *return* i + 1;  
 }  
  
 *// Метод для сортировки вставками  
 private static void* insertionSort(MyClass[] array) {  
 *int* n = array.length;  
 *int* comparisons = 0;  
 *int* swaps = 0;  
  
 *for* (*int* i = 1; i < n; i++) {  
 MyClass key = array[i];  
 *int* j = i - 1;  
  
 *while* (j >= 0 && array[j].getId() > key.getId()) {  
 comparisons++;  
 *// Перестановка элементов* array[j + 1] = array[j];  
 j--;  
 swaps++;  
 }  
 array[j + 1] = key;  
 }  
  
 System.out.println("Insertion Sort:");  
 System.out.println("Theoretical Complexity: O(n^2)");  
 System.out.println("Comparisons: " + comparisons);  
 System.out.println("Swaps: " + swaps);  
 }  
  
 *// Метод для сортировки слиянием  
 private static void* mergeSort(MyClass[] array) {  
 mergeComparisons = 0;  
 mergeSwaps = 0;  
 *// Реализация сортировки слиянием (Merge Sort)* mergeSortHelper(array, 0, array.length - 1);  
  
 System.out.println("Merge Sort:");  
 System.out.println("Theoretical Complexity: O(n log n)");  
 System.out.println("Comparisons: " + mergeComparisons);  
 System.out.println("Swaps: " + mergeSwaps);  
 }  
  
 *private static void* mergeSortHelper(MyClass[] array, *int* left, *int* right) {  
 *if* (left < right) {  
 *int* middle = left + (right - left) / 2;  
  
 *// Рекурсивно сортируем две половины* mergeSortHelper(array, left, middle);  
 mergeSortHelper(array, middle + 1, right);  
  
 *// Объединяем отсортированные половины* merge(array, left, middle, right);  
 }  
 }  
  
 *private static void* merge(MyClass[] array, *int* left, *int* middle, *int* right) {  
 *int* n1 = middle - left + 1;  
 *int* n2 = right - middle;  
  
 *// Создаем временные массивы* MyClass[] leftArray = Arrays.copyOfRange(array, left, left + n1);  
 MyClass[] rightArray = Arrays.copyOfRange(array, middle + 1, middle + 1 + n2);  
  
 *int* i = 0, j = 0, k = left;  
 *while* (i < n1 && j < n2) {  
 mergeComparisons++;  
 *if* (leftArray[i].getId() <= rightArray[j].getId()) {  
 array[k] = leftArray[i];  
 i++;  
 } *else* {  
 array[k] = rightArray[j];  
 j++;  
 }  
 k++;  
 mergeSwaps++;  
 }  
  
 *// Копируем оставшиеся элементы из leftArray  
 while* (i < n1) {  
 array[k] = leftArray[i];  
 i++;  
 k++;  
 mergeSwaps++;  
 }  
  
 *// Копируем оставшиеся элементы из rightArray  
 while* (j < n2) {  
 array[k] = rightArray[j];  
 j++;  
 k++;  
 mergeSwaps++;  
 }  
 }  
  
 *// Метод для сортировки выбором  
 private static void* selectionSort(MyClass[] array) {  
 *int* n = array.length;  
 *int* comparisons = 0;  
 *int* swaps = 0;  
  
 *for* (*int* i = 0; i < n - 1; i++) {  
 *int* minIndex = i;  
 *for* (*int* j = i + 1; j < n; j++) {  
 comparisons++;  
 *if* (array[j].getId() < array[minIndex].getId()) {  
 minIndex = j;  
 }  
 }  
  
 *// Обмен элементов* MyClass temp = array[minIndex];  
 array[minIndex] = array[i];  
 array[i] = temp;  
 swaps++;  
 }  
  
 System.out.println("Selection Sort:");  
 System.out.println("Theoretical Complexity: O(n^2)");  
 System.out.println("Comparisons: " + comparisons);  
 System.out.println("Swaps: " + swaps);  
 }  
  
 *public static void* main(String[] args) {  
 MyClass[] array = generateArray();  
  
 *// Вывод исходного массива* System.out.println("Original Array:");  
 printArray(array);  
  
 *// Сортировка пузырьком  
 long* startTime = System.nanoTime();  
 bubbleSort(array.clone());  
 *long* endTime = System.nanoTime();  
 System.out.println("Bubble Sort Time: " + (endTime - startTime) + " ns\n");  
  
 *// Быстрая сортировка* startTime = System.nanoTime();  
 quickSort(array.clone());  
 endTime = System.nanoTime();  
 System.out.println("Quick Sort Time: " + (endTime - startTime) + " ns\n");  
  
 *// Сортировка вставками* startTime = System.nanoTime();  
 insertionSort(array.clone());  
 endTime = System.nanoTime();  
 System.out.println("Insertion Sort Time: " + (endTime - startTime) + " ns\n");  
  
 *// Сортировка слиянием* startTime = System.nanoTime();  
 mergeSort(array.clone());  
 endTime = System.nanoTime();  
 System.out.println("Merge Sort Time: " + (endTime - startTime) + " ns\n");  
  
 *// Сортировка выбором* startTime = System.nanoTime();  
 selectionSort(array.clone());  
 endTime = System.nanoTime();  
 System.out.println("Selection Sort Time: " + (endTime - startTime) + " ns");  
 }  
}

Результаты





Вывод

В ходе выполнения данной индивидуальной работы были реализованы пять разных методов сортировки массивов данных на языке программирования Java.

Для каждого из методов была проведена аналитическая оценка теоретической сложности, а также подсчитано количество сравнений, количество перестановок и измерено время выполнения алгоритма.

Программа также содержит комментарии, поясняющие подход и используемые алгоритмы. Исходные массивы генерируются с использованием класса MyClass и предоставляют разнообразные данные для тестирования сортировок. Каждый метод сортировки вызывается с клонированным массивом для изоляции результатов и измерения времени выполнения.

Данная программа дает возможность сравнить эффективность различных сортировок на практике и выбрать наиболее подходящий метод в зависимости от конкретных требований и характеристик входных данных.