**Лабораторная работа №13**

# Кодирование алгоритмов сортировки с оценкой их производительности

**Цель работы:** Изучить принципы сортировки массивов различными методами. Рассмотреть схемы сортировок. Разобрать приведенные примеры и выполнить самостоятельно один или более вариантов заданий.

## Теоретические сведения

Наиболее частыми операциями при работе с данными являются «поиск» и «сортировка». При этом алгоритмы решения этих задач существенно зависят от того, организованы записи в массивы или размещены на диске.

Методы сортировки классифицируются по времени их работы. Хорошей мерой эффективности может быть число сравнений ключей - $C$ и число пересылок элементов - $P$. Эти числа являются функциями $C(n)$, $P(n)$ от числа сортируемых элементов $n$. **Быстрые** (но сложные) алгоритмы сортировки требуют (при $n \to \infty$) порядка $n \log n$ сравнений, **прямые** (простые) методы -$n^2$.

Прямые методы коротки, просто программируются, быстрые, усложненные, методы требуют меньшего числа операций, но эти операции обычно сами более сложны, чем операции прямых методов, поэтому для достаточно малых $n$ $(n \leq 50)$ прямые методы работают быстрее. Значительное преимущество быстрых методов (в $n/\log(n)$ раз) начинает проявляться при $n \geq$ % 100.

Среди простых методов наиболее популярны:
1) Метод прямого обмена
2) Метод прямого выбора:

Реже используются:
3) Сортировка с помощью прямого (двоичного) включения;
4) Шейкерная сортировка (модификация пузырьковой).

Улучшенные методы сортировки:
1) Метод Д. Шелла, усовершенствование метода прямого включения.
2) Сортировка с помощью дерева, метод **HeapSort**, Д. Уильямсон.
3) Сортировка с помощью разделения, метод **QuickSort**, Ч. Хоар, улучшенная версия пузырьковой сортировки. На сегодняшний день это самый эффективный метод сортировки.

Сравнение методов сортировок показывает, что при $n > 100$ наихудшим является метод пузырька, метод QuickSort в 2-3 раза лучше, чем HeapSort, и в 3-7 раз, чем метод Шелла.

# Схемы работы некоторых видов сортировок массивов
## 1. Сортировка пузырьком

Сортировка пузырьком – простейший алгоритм сортировки, применяемый чисто для учебных целей. Практического применения этому алгоритму нет, так как он не эффективен, особенно если необходимо отсортировать массив большого размера. К плюсам сортировки пузырьком относится простота реализации алгоритма.

Алгоритм сортировки пузырьком сводится к повторению проходов по элементам сортируемого массива. Проход по элементам массива выполняет внутренний цикл. За каждый проход сравниваются два соседних элемента, и если порядок неверный элементы меняются местами. Внешний цикл будет работать до тех пор, пока массив не будет отсортирован.

Таким образом внешний цикл контролирует количество срабатываний внутреннего цикла Когда при очередном проходе по элементам массива не будет совершено ни одной перестановки, то массив будет считаться отсортированным. Чтобы хорошо понять алгоритм, отсортируем методом пузырька массив, к примеру, из 7 чисел (см. Таблица 1).

Таблица 1 - Метод пузырька

| № итерации | Элементы массива | | | | | | | Перестановки |
|---|---|---|---|---|---|---|---|---|
| исх. массив | 3 | 3 | 7 | 1 | 2 | 5 | 0 | |
| 0 | 3 | 3 | | | | | | false |
| 1 | | 3 | 7 | | | | | false |
| 2 | | | **1** | **7** | | | | 7>1, true |
| 3 | | | | **2** | **7** | | | 7>2, true |

Таблица 1 - Метод пузырька

| № итерации | Элементы массива | | | | | | | Перестановки |
|---|---|---|---|---|---|---|---|---|
| 4 | | | | | **5** | **7** | | 7>5, true |
| 5 | | | | | | **0** | **7** | 7>0, true |
| тек.массив | **3** | **3** | **1** | **2** | **5** | **0** | **7** | |
| 0 | 3 | 3 | | | | | | false |
| 1 | | **1** | **3** | | | | | 3>1, true |
| 2 | | | **2** | **3** | | | | 3>2, true |
| 3 | | | | **0** | **3** | | | 3>0, true |
| 4 | | | | | 3 | 5 | | false |
| 5 | | | | | | 5 | 7 | false |
| тек.массив | **3** | **1** | **2** | **0** | **3** | **5** | **7** | |
| 0 | **1** | **3** | | | | | | 3>1, true |
| 1 | | **2** | **3** | | | | | 3>2, true |
| 2 | | | **0** | **3** | | | | 3>0, true |
| 3 | | | | 3 | 3 | | | false |
| 4 | | | | | 3 | 5 | | false |
| 5 | | | | | | 5 | 7 | false |
| тек.массив | **1** | **2** | **0** | **3** | **3** | **5** | **7** | |
| | 1 | 2 | | | | | | false |
| | | **0** | **2** | | | | | 2>0, true |
| | | | 2 | 3 | | | | false |
| | | | | 3 | 3 | | | false |
| | | | | | 3 | 5 | | false |
| | | | | | | 5 | 7 | false |
| тек.массив | **1** | **0** | **2** | **3** | **3** | **5** | **7** | |
| | **0** | **1** | | | | | | 1>0, true |
| | | 1 | 2 | | | | | false |
| | | | 2 | 3 | | | | false |

| № итерации | Элементы массива | | | | | | | Перестановки |
|---|---|---|---|---|---|---|---|---|
| | | | | 3 | 3 | | | false |
| | | | | | 3 | 5 | | false |
| | | | | | | 5 | 7 | false |
| конечный массив | **0** | **1** | **2** | **3** | **3** | **5** | **7** | |
| Конец сортировки | | | | | | | | |

Для того чтобы отсортировать массив хватило пяти запусков внутреннего цикла, for. Запустившись, цикл for срабатывал 6 раз, так как элементов в массиве 7, то итераций (повторений) цикла for должно быть на одно меньше.

На каждой итерации сравниваются два соседних элемента массива. Если текущий элемент массива больше следующего, то меняем их местами. Таким образом, пока массив не будет отсортирован, будет запускаться внутренний цикл и выполняться операция сравнения. Обратите внимание на то, что за каждое полное выполнение цикла for как минимум один элемент массива находит своё место. В худшем случае, понадобится n-2 запуска внутреннего цикла, где n – количество элементов массива. Это говорит о том, что сортировка пузырьком крайне не эффективна для больших массивов.

## 2. Сортировка выбором

Пожалуй, самый простой алгоритм сортировок – это сортировка выбором. Судя по названию сортировки, необходимо что-то выбирать (максимальный или минимальный элементы массива). Алгоритм сортировки выбором находит в исходном массиве максимальный или минимальный элементы, в зависимости от того как необходимо сортировать массив, по возрастанию или по убыванию. Если массив должен быть отсортирован по возрастанию, то из исходного массива необходимо выбирать минимальные элементы. Если же массив необходимо отсортировать по убыванию, то выбирать следует максимальные элементы.

Допустим необходимо отсортировать массив по возрастанию. В исходном массиве находим минимальный элемент, меняем его местами с первым элементом массива. Уже, из всех элементов массива один элемент стоит на своём месте. Теперь будем рассматривать не отсортированную часть массива, то есть все элементы массива, кроме первого. В неотсортированной части массива опять ищем минимальный элемент. Найденный минимальный элемент меняем местами со вторым элементом массива и т. д.

Таким образом, суть алгоритма сортировки выбором сводится к многократному поиску минимального (максимального) элементов в неотсортированной части массива. Отсортируем массив из семи чисел согласно алгоритму «Сортировка выбором».

**Исходный массив: 3 3 7 1 2 5 0**

1)Итак, находим минимальный элемент в массиве. 0 – минимальный элемент

2)Меняем местами минимальный и первый элементы массива.

**Текущий массив: 0 3 7 1 2 5 3**

3) Находим минимальный элемент в неотсортированной части массива. 1 – минимальный элемент

4) Меняем местами минимальный и первый элементы массива.

**Текущий массив:** 0 1 7 3 2 5 3

5) min = 2

6)**Текущий массив:** 0 1 2 3 7 5 3

7)min = 3

8) **Текущий массив:** 0 1 2 3 7 5 3 в массиве ничего не поменялось, так как 3 стоит на своём месте

9) min = 3

10) **Конечный вид массива: 0 1 2 3 3 5 7** – массив отсортирован.

## 3. Сортировка вставками

Сортировка вставками — достаточно простой алгоритм. Как в и любом другом алгоритме сортировки, с увеличением размера сортируемого массива увеличивается и время сортировки.

Основным преимуществом алгоритма сортировки вставками является возможность сортировать массив по мере его получения. То есть имея часть массива, можно начинать его сортировать. В параллельном программирование такая особенность играет не маловажную роль.

Сортируемый массив можно разделить на две части — отсортированная часть и неотсортированная. В начале сортировки первый элемент массива считается отсортированным, все остальные — не отсортированные. Начиная со второго элемента массива и заканчивая последним, алгоритм вставляет неотсортированный элемент массива в нужную позицию в отсортированной части массива. Таким образом, за один шаг сортировки отсортированная часть массива увеличивается на один элемент, а неотсортированная часть массива уменьшается на один элемент. Рассмотрим пример сортировки по возрастанию массива из 7 чисел (см. Таблица 2):

**Исходный массив: 3 3 7 1 2 5 0**

| шаг | отсортированная часть массива | | | | | | | тек.элемент | вставка |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 3 | | | | | | | 3 | false |
| 2 | 3 | 3 | | | | | | 7 | false |
| 3 | 3 | 3 | 7 | | | | | 1 | true |
| 4 | 1 | 3 | 3 | 7 | | | | 2 | true |
| 5 | 1 | 2 | 3 | 3 | 7 | | | 5 | true |
| 6 | 1 | 2 | 3 | 3 | 5 | 7 | | 0 | true |
| - | 0 | 1 | 2 | 3 | 3 | 5 | 7 | - | - |

На каждом шаге сортировки сравнивается текущий элемент со всеми элементами в отсортированной части. На первом шаге сравнивается тройка с тройкой, они равны поэтому не меняем их местами. На втором шаге сравниваем 7 с двумя тройками, 7 >3, а так как сортировка по возрастанию, то опять элементы массива остаются на своих местах. На третьем шаге единица сравнивается с тремя элементами и все они больше единицы, значит единицу вставляем на первое место, в начало массива. На четвёртом шаге текущий элемент — 2 сравниваем с элементами 1, 3, 3, 7. Получается, что 1 < 2 < 3 и 7, поэтому двойку вставляем между единицей и тройкой. Пятый и шестой шаги выполняются точно также. В итоге на шестом шагу мы получаем отсортированный по возрастанию массив.

## 4. Сортировка Шелла

В 1959 году Дональд Шелл опубликовал усовершенствованный алгоритм сортировки вставками, который впоследствии получил его имя. Идея данного метода заключается в сравнение разделенных на группы элементов некоторой последовательности. Сначала, сравниваются элементы находящиеся друг от друга на расстоянии d (первое d обычно равно n/2, где n — количество всех элементов). Постепенно расстояние между элементами уменьшается, и на d=1 сортировка происходит последний раз.

## Практическая часть

1. Написать функцию поиска заданного элемента в целочисленном массиве. Отсортировать по возрастанию все элементы справа от найденного. Использовать метод выбора.

2. Имеется целочисленный массив из n элементов, среди которых есть повторяющиеся. Написать программу для ’сжатия’ этого массива путем выбрасывания из него одинаковых элементов и отсортировать его по убыванию методом Шелла. Затем дополнить массив всеми выброшенными элементами в порядке убывания.

3. Даны два массива. Массив A состоит из N элементов , массив B состоит из M элементов. Оба массива отсортированы по убыванию. Разработать программу для слияния этих массивов в отсортированный по неубыванию массив C.

4. Даны два массива. Массив A состоит из N элементов и отсортирован по возрастанию. Массив B состоит из M элементов и отсортирован по убыванию. Разработать программу для слияния этих массивов в отсортированный по убыванию массив C, не содержащий одинаковых элементов.

5. Даны два массива. Массив A состоит из N элементов и отсортирован по возрастанию. Массив B состоит из M элементов и отсортирован по убыванию. Разработать программу для слияния этих массивов в отсортированный по возрастанию массив C.

6. Дан массив вещественных чисел X=(x1,x2,..,xn). Записать элементы заданного массива X в массив Y следующим образом: в начальной части расположить положительные элементы в порядке возрастания, затем в порядке убывания отрицательные элементы, нулевые элементы не записывать.

7. Даны две последовательности a1<=a2<=...<=an и b1<=b2<=...<=bm. Образовать из них новую последовательность чисел так, чтобы она тоже была неубывающей. Дополнительный массив не использовать.

8. Дан массив целых чисел X=(x1,x2,..,xn). Записать элементы заданного массива X в массив Z следующим образом: в начальной части расположить отрицательные элементы в порядке возрастания, затем в порядке убывания положительные элементы, все нулевые элементы записать в конце массива.

9. Дана матрица размерностью M x N. Отсортировать по возрастанию все столбцы матрицы.

10. Дана матрица размерностью M x N. Отсортировать по возрастанию все строки матрицы.