# Лабораторная работа № 2
# Алгоритмы сортировки

**Цель работы** является изучение алгоритмов сортировки, исследование сложности алгоритмов сортировки при различных исходных данных, закрепление навыков алгоритмизации.

## Общие сведения

В настоящее время известно большое количество алгоритмов сортировки, отличающихся трудоемкостью реализации, временной и пространственной сложностью, зависимостью от вспомогательных структур данных, наличием или отсутствием рекурсии и т.п. Но наиболее часто разработчика интересует именно временная сложность алгоритма сортировки. Эта характеристика алгоритма зависит от следующих особенностей алгоритма и природы сортируемых объектов: способ хранения элементов сортируемой последовательности, суммарное количество операций сравнения и длительность выполнения этой операции, суммарное количество операций присвоения и длительность выполнения этой операции. Если для простых типов данных длительность выполнения операций сравнения и присвоения относительно невелика, то для сложных типов данных она может быть весьма существенной. С другой стороны, при хранении элементов сложных типов данных, на практике часто используют указатели на элементы, что позволяет существенно снизить время, расходуемое на операции присваивания (и связанные с ней операции более высокого уровня – обмен, вставка, сдвиг), но указатели не позволяют уменьшить время выполнения операции сравнения. Поэтому в данной работе в качестве исследуемых свойств алгоритмов сортировки выбраны временная сложность и сложность в смысле суммарного количества операций сравнения ключей.

Многие библиотеки предоставляют разработанные и реализованные функции сортировки. Стандартная библиотека C также предоставляет подобную универсальную функцию сортировки:

```
void qsort(void *base, size_t nelem, size_t width, int (_USERENTRY
*fcmp)(const void *, const void *));
```

Как следует из сигнатуры функции, в качестве фактических параметров этой функции необходимо передать указатель на сортируемую последовательность (массив), размер одного элемента, количество элементов и указатель на функцию, которая «умеет» сравнивать элементы пользовательского типа. Например

```
int SortFunction(const void *a, const void *b)
{
return (*(int*)a) < (*(int*)b) ? -1 :
       ((*(int*)a) == (*(int*)b) ? 0 : 1);
}

int main(void)
{
  int x[5] = {7,3,6,5,1};
  qsort((void *)x, 5, sizeof(int), SortFunction);
…
```

Для использования данной функции необходимо подключить заголовочный файл stdlib.h.

**Задание**

1. В таблице № 1 представлены алгоритмы сортировки (и в некоторых случаях их параметры), которые необходимо реализовать в соответствии с вариантом (таблица № 2). Для проверки реализованных функций сортировки разработать функцию, проверяющую упорядоченность элементов в последовательности.
2. Разработать функции, позволяющие производить оценку временной сложности алгоритма $T(n)$ ($n$ – количество элементов сортируемой последовательности) и суммарного количества операций сравнения $S(n)$[1], выполняемых в ходе его работы.
3. На основе функций формирования последовательностей, реализованных в ходе предыдущей лабораторной работы, и результатов выполнения пп. 1 и 2, разработать программу, с помощью которой по экспериментальным данным построить зависимости $T(n)$ и $S(n)$ для заданных алгоритмов сортировки по четырем последовательностям (упорядоченная, случайная, упорядоченная в обратном порядке и указанная в варианте). Размеры сортируемых последовательностей следует выбирать самостоятельно в соответствии с характеристиками исследуемого алгоритма (например, $5 \cdot 10^3$, $10 \cdot 10^3$, …, $50 \cdot 10^3$). Кроме того, необходимо оценить зависимости $T(n)$ и $S(n)$ для функции `qsort` из стандартной библиотеки C.
4. Составить отчет, в котором привести графики зависимостей $T(n)$ и $S(n)$, результаты анализа полученных экспериментальных данных и теоретических оценок сложности алгоритмов, сравнительную оценку реализованных алгоритмов (по сложности, устойчивости, естественности, требованиям к памяти), выводы по работе.

[1] Для тех алгоритмов, которые используют операции сравнения.

Таблица №1

## Алгоритмы сортировки

| № | Наименование алгоритма (условное) | Примечание |
|---|---|---|
| 1 | Метод «пузырька» | |
| 2 | Модифицированный метод «пузырька» I | С использованием флага – признака обмена |
| 3 | Модифицированный метод «пузырька» II | С использованием индекса последнего обмена |
| 4 | Двунаправленный метод обмена | «Шейкер»-сортировка |
| 5 | Сортировка обменом на расстоянии | Применение идеи Шелла, окончательная сортировка вставками |
| 6 | Быстрая сортировка | Сортировка Хоара |
| 7 | Модифицированная быстрая сортировка I | Почти упорядоченная последовательность сортируется вставками (при размере участков $M = 10$) |
| 8 | Модифицированная быстрая сортировка I | Почти упорядоченная последовательность сортируется вставками (при размере участков $M \in [5; 20]$, найти оптимум) |
| 9 | Модифицированная быстрая сортировка II | Выбор разделяющего элемента с помощью медианы |
| 10 | Модифицированная быстрая сортировка III | Оптимизация для совпадающих ключей |
| 11 | Сортировка вставками | |
| 12 | Модифицированная сортировка вставками I | С использованием сигнального ключа |
| 13 | Модифицированная сортировка вставками II | С использованием бинарного поиска для вставки |
| 14 | Сортировка Шелла | Последовательность: S1: $h_i = 3*h_{i-1} + 1$, |
| 15 | Сортировка Шелла | Последовательности: S1: $h_i = 3*h_{i-1} + 1$, $i \geq 0$, S2: $h_i = 2^{i+1} - 1$, $i \geq 0$, S3: $h_i = 1$, $i = 0$ и $h_i = 4^i + 3*2^{i-1} + 1$, $i \geq 1$, S4: $h_i = 1, 2, 3, 4, 6, 8 \ldots$ |
| 16 | Сортировка выбором | |
| 17 | Пирамидальная S-арная сортировка | $S = 2$ |
| 18 | Пирамидальная S-арная сортировка | $S \in [2; 7]$ |
| 19 | Бинарная поразрядная сортировка (MSD) | |
| 20 | Поразрядная сортировка (MSD) | Разряд: 1, 2, 4, 8 двоичных разрядов |
| 21 | Сортировка слиянием | |
| 22 | Сортировка подсчетом | |
| 23 | Бинарная поразрядная сортировка подсчетом (LSD) | |
| 24 | Поразрядная сортировка подсчетом (LSD) | Разряд: 1, 2, 4, 8 двоичных разрядов |
| 25 | FlashSort | |
| 26 | Fastest Sorting Algorithm | |
| 27 | Adaptive Left Radix (ARL) | |
| 28 | Stable ARL | |

Таблица № 2

Варианты заданий

| № | Алгоритмы сортировки | № | Алгоритмы сортировки |
|---|---|---|---|
| 1 | 9, 11, 12, 22 {ступенчатая}, int | 13 | 1, 5, 15 {квазиупорядоченная}, float |
| 2 | 2, 3, 17, 19 {пилообразная}, int | 14 | 2, 7, 18 {ступенчатая}, double |
| 3 | 5, 10, 13, 21 {синусоидальная}, float | 15 | 8, 11, 22 {квазиупорядоченная}, float |
| 4 | 1, 2, 8 {квазиупорядоченная}, double | 16 | 12, 14, 20 {пилообразная}, int |
| 5 | 3, 4, 15 {ступенчатая}, float | 17 | 4, 9, 19, 23 {ступенчатая}, double |
| 6 | 5, 11, 18 {квазиупорядоченная}, double | 18 | 6, 17, 24 {пилообразная}, int |
| 7 | 12, 13, 20 {ступенчатая}, int | 19 | 7, 8, 15 {квазиупорядоченная}, float |
| 8 | 4, 10, 16, 21 {синусоидальная}, float | 20 | 9, 10, 16, 21 {синусоидальная}, double |
| 9 | 14, 16, 24 {ступенчатая}, int | 21 | 17, 19, 22, 23 {ступенчатая}, int |
| 10 | 1, 3, 6, 7 {пилообразная}, double | 22 | 18, 20, 24, {ступенчатая}, int |
| 11 | 6, 13, 14, 23 {ступенчатая}, int | 23 | 1, 3, 9 {квазиупорядоченная}, float |
| 12 | 1, 8, 18 { квазиупорядоченная }, double | 24 | 3, 20, 21 {ступенчатая}, int |