**ЛАБОРАТОРНАЯ РАБОТА № 2**

**АЛГОРИТМЫ СОРТИРОВКИ**

1. ТЕОРИЯ И ПОЯСНЕНИЯ

В работе нужно в одном проекте реализовать 3 алгоритма сортировки массивов и исследовать зависимость времени выполнения упорядочения от длины массива. Примем следующие исходные соглашения:

– задан целочисленный массив **Mass**, содержащий **n** элементов с номерами от **0** до **n–1**;

– массив заполняется случайными числами из интервала от -999 до 999;

– выполняется упорядочение по возрастанию.

***I.1. Генерация последовательности случайных чисел для заполнения массива***. В Lazarus, как и в других средах разработки, имеются процедуры и функции для получения случайных чисел:

1) **procedure Randomize;**

Randomize инициализирует встроенный генератор случайных чисел случайным значением. В программе обращение к ней делается только один раз. При каждом новом выполнении программы начальное случайное значение будет новым. Если Randomize не используется, то при каждом новом выполнении программы генератор случайных чисел будет стартовать с одного и того же значения;

2) **function Random [ ( Range: Integer) ];**

Random возвращает случайное число из интервала 0 <= X < Range. Если параметр опущен, то возвращается вещественное случайное число из интервала 0 <= X < 1.

***I.2. Идея простейшего алгоритма сортировки.*** Проверяются на упорядоченность все рядом стоящие элементы массива. Как только обнаруживается, что условие упорядоченности не выполняется, Mass[i] > Mass[i+1], пара проверяемых элементов меняется местами, и продолжение проверки производится с начала массива, т.е. текущий номер i становится равным 0.

***I.3. Идея алгоритма обменной сортировки***. Предположим, что начальные элементы массива от 0-го до i-го упорядочены. Значение следующего (i+1)-го элемента сравнивается с каждым из предыдущих, начиная с i-го, и, при необходимости, меняется с ним местами. При каждом обмене это значение перемещается на одну позицию к началу массива. Сравнения продолжаются до тех пор, пока все (i+1) элементов не окажутся упорядоченными.

***I.4. Идея алгоритма вставок***. Предположим, что начальные элементы массива от 0-го до i-го упорядочены. Для очередного (i+1)-го элемента отыскивается позиция j среди упорядоченных элементов, на которой он должен оказаться. Чтобы освободить эту позицию, следует сдвинуть все элементы от j –го до i-го на одну позицию вправо. Затем значение (i+1)-го элемента заносится на j –е место в массиве.

***I.5. Идея алгоритма выбора***. Основа – поиск максимального значения в массиве. Элемент с максимальным значением меняется местами с последним элементом массива. Затем поиск максимума проводится среди первых (n-1) элементов. Найденный элемент меняется местами с предпоследним, и т.д., пока область поиска максимума не сократится до одного элемента.

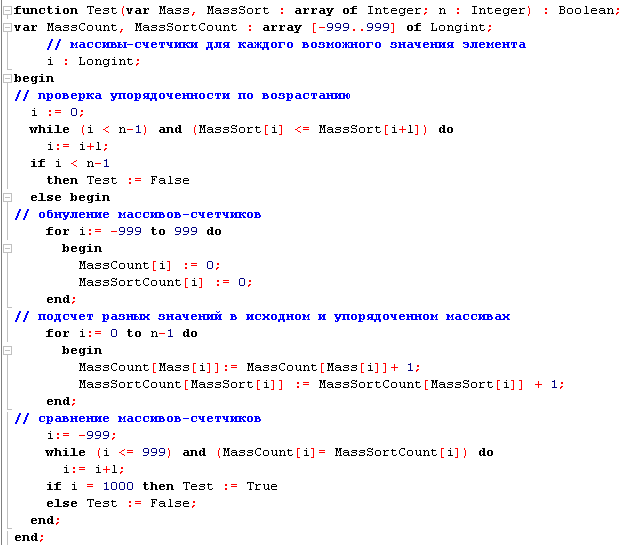
***I.6. Идея алгоритма слияния***. Сортировка слиянием основана на слиянии серий – упорядоченных последовательностей элементов массива. Для данной сортировки требуется 2 массива длины n: один (Mass) содержит набор входных серий, а во втором (Aux) из каждой пары смежных входных серий формируется одна выходная.

Далее будем считать, что при всех слияниях первую входную серию образуют элементы Mass с номерами от a до c, вторую – от c+1 до b. Выходная серия будет также занимать в массиве Aux элементы с номерами от a до b. Основой слияния является циклическая конструкция, в которой производится сравнение двух элементов массива Mass, причем один из них взят из первой серии, а другой – из второй. В выходную серию записывается меньшее или равное значение, и увеличивается индекс элемента соответствующей входной серии. Если какая-то серия к моменту сравнения уже закончилась, то выполняется переписывание элемента из оставшейся серии и увеличение индекса элемента этой серии. Необходимо, чтобы выполнялись условия a <= c <= b и 0 <= a, c, b <= n–1, т.е. переменные содержали допустимые номера элементов Mass и Aux. После слияния двух серий нужно копировать выходную серию из Aux в Mass.

Рекурсивный алгоритм сортировки слиянием имеет параметры: массив Mass, номера начального и конечного элементов той части массива, которая упорядочивается. Алгоритм выполняется следующим образом. Если упорядочиваемая часть состоит из одного элемента, то ничего не делается. В противном случае вся часть массива делится на две одинаковых доли (с точностью до одного элемента), каждая из долей рекурсивно упорядочивается этим же алгоритмом, после чего обе доли, как две упорядоченные серии, сливаются в общую выходную серию.

***I.7. Проверка результатов работы алгоритма сортировки***. Для того чтобы убедиться в правильности упорядочения, нужно сравнить все пары рядом стоящих элементов массива. Однако могут возникнуть такие алгоритмические ошибки, из-за которых некоторые исходные значения будут утрачены или заменены другими значениями.

Поскольку в данной работе используется целочисленный массив, содержащий значения только из определенного интервала, можно подсчитать, сколько раз каждое значение встречается в исходном и упорядоченном массивах. Сравнение счетчиков позволит убедиться в правильности упорядочения.



***I.8. Определение времени работы программы***. Для определения времени выполнения программы в целом или какой-либо части программного кода следует обратиться к функциям, работающим с датой и временем. Они относятся к WinAPI (Windows Application Program Interface – программный интерфейс приложений Windows) и содержатся в модуле Windows. Его следует подключить в разделе **uses** вашей программы. В данной работе рекомендуется использовать процедуру GetProcessTimes().

**GetProcessTimes()** – процедура с параметрами. С ее помощью можно узнать, сколько процессорного времени потребовалось для выполнения конкретного программного кода. Эта процедура вычисляет по отдельности время работы приложения пользователя и время выполнения ядром операционной системы так называемых системных вызовов (например, обращений к внешней памяти, устройствам ввода-вывода).

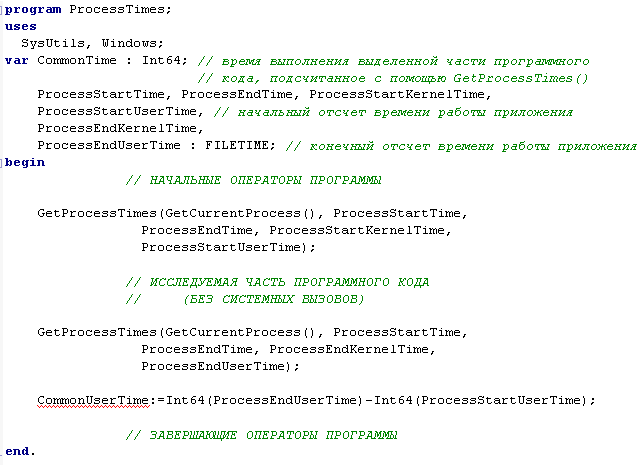
|  |  |
| --- | --- |
| **GetProcessTimes(** |  |
| **GetCurrentProcess(),** | Идентификатор текущего процесса. |
| **ProcessStartTime,** | Абсолютное время старта процесса в интервалах по 100 нс от полуночи 1 января 1601 года по Гринвичу. |
| **ProcessEndTime,** | Абсолютное текущее время завершения процесса в интервалах по 100 нс от полуночи 1 января 1601 года по Гринвичу. Если процесс еще выполняется, значение этой величины не определено. |
| **KernelTime,** | Относительное время работы ядра ОС в интервалах по 100 нс (1E-07 сек). |
| **UserTime)** | Относительное время работы приложения в интервалах по 100 нс (1E-07 сек). |

***Замечание.*** Реальная точность измерения времени с помощью этой процедуры составляет 1/64 сек. Поэтому для коротких вычислений время выполнения может быть измерено как 0.

Параметры **ProcessStartTime, ProcessEndTime, KernelTime, UserTime** имеют тип **FILETIME**. Это запись, поэтому для выполнения вычислений со значениями этих параметров их нужно преобразовать к целому типу.

Для подсчета времени выполнения программы процедуру нужно вызывать дважды, фиксируя моменты начала и завершения вычислений. Результатом является разность между двумя значениями соответствующих временных отметок.

***Пример программы*** с определением времени выполнения определенной части кода:

****

1. ЗАДАНИЯ

Все задания нужно реализовать в одном проекте.

**II.1. ПРОСТЕЙШИЙ АЛГОРИТМ УПОРЯДОЧЕНИЯ.** Написать процедуру (с параметрами) упорядочения целочисленного массива с помощью простейшего алгоритма.

**II.2.** **АЛГОРИТМ УПОРЯДОЧЕНИЯ С КВАДРАТИЧНОЙ ТРУДОЕМКОСТЬЮ**. Напишите процедуру (с параметрами) упорядочения целочисленного массива с помощью одного из алгоритмов: обмена, вставок или выбора.

**II.3**. **АЛГОРИТМ УПОРЯДОЧЕНИЯ СЛИЯНИЕМ.** Напишите процедуру (с параметрами) упорядочения целочисленного массива с помощью рекурсивного алгоритма слияния.

**II.4.** **АВТОМАТИЧЕСКАЯ ПРОВЕРКА УПОРЯДОЧЕННОСТИ**. Добавьте функцию проверки упорядоченности в проект и выполните проверку работы каждого алгоритма.

**II.5.** **ИССЛЕДОВАНИЕ ЗАВИСИМОСТИ ВРЕМЕНИ ВЫПОЛНЕНИЯ УПОРЯДОЧЕНИЯ ОТ ДЛИНЫ МАССИВА**. Для каждого из реализованных алгоритмов сортировки составьте таблицу зависимости времени выполнения CommonUserTime от длины n упорядочиваемого массива. Обязательные значения n: 10, 100, 1000, 10000, 100000, 1000000.