Практическая работа№ 4.
Сортировка числовых массивов. Некоторые методы сортировки.

Задача сортировки одна из важнейших в программировании. По-существу, любые задачи, связанные с обработкой данных, связаны с алгоритмами поиска: чтобы совершить какое-либо действие над ними, нужно либо найти конкретное значение («ключ»), либо конкретное место где они хранятся (индекс, адрес) в какой-либо структуре хранения (файл, список, массив, ...). От того, как организовано такое «хранилище», зависит и скорость доступа (поиска). Если данные упорядочены, то это существенно ускоряет процессы поиска. А значит, есть смысл упорядочивать данные. Как всякий вычислительный процесс, процесс упорядочивания (сортировки по ключу) имеет и временные, и алгоритмические сложности. Поэтому важно, в зависимости от контекста задачи, подобрать такой алгоритм, который в конкретном случае будет наиболее подходящим. Сортировка - упорядочение (перестановка) элементов в подмножестве данных по какому-либо критерию. Чаще всего в качестве критерия используется некоторое числовое поле, называемое ключевым. Упорядочение элементов по ключевому полю предполагает, что ключевое поле каждого следующего элемента не больше предыдущего (сортировка по убыванию). Если ключевое поле каждого последующего элемента не меньше предыдущего, то говорят о сортировке по возрастанию. Цель сортировки — облегчить последующий поиск элементов в отсортированном множестве при обработке данных

Для того, чтобы отсортировать данные, можно вызывать стандартную функцию qsort(), входящую в библиотеку C++. Однако различные подходы к сортировке обладают разными характеристиками. Несмотря на то, что некоторые способы сортировки могут быть в среднем лучше, чем другие, ни один алгоритм не является идеальным для всех случаев.

Использование функции qsort() не является универсальным решением для всех задач сортировки. Во-первых, функцию общего назначения, такую как qsort(), невозможно применить во всех ситуациях. Например, данная функция сортирует только массивы в памяти и не может сортировать данные, хранящиеся в связанных списках. Во-вторых, *qsort()* – параметризованная функция, благодаря чему она может обрабатывать широкий набор типов данных, но вследствие этого она работает медленнее, чем эквивалентная функция, рассчитанная на какой-то один тип данных. В-третьих, алгоритм *быстрой сортировки*, примененный в функции *qsort()*, может оказаться не самым эффективным алгоритмом в некоторых конкретных ситуациях.

Существует множество различных алгоритмов сортировки. Все они имеют свои положительные и отрицательные стороны.

Алгоритмом сортировки называется алгоритм для упорядочения некоторого множества элементов

Все алгоритмы сортировки делятся на

- алгоритмы *внутренней* сортировки (сортировка массивов);
- алгоритмы *внешней* сортировки (сортировка файлов).

Уточним эти понятия. Данные хранятся в файле на внешнем носителе. Если в процессе работы программы все эти данные одновременно можно загрузить в область оперативной памяти, то говорят о *внутренней* сортировке. Не все программные алгоритмы внутренней сортировки могут быть применены к сортировке данных, хранящихся на внешних носителях.

Методы сортировки массивов (внутренняя сортировка) можно разбить на три класса:

- сортировка *выбором*;
- сортировка *включениями* (*вставками*);
- сортировка *обменом*.

Как обычно, из всех характеристик производительности алгоритмов сортировки нас в первую очередь интересует *время* их выполнения.

Один из самых простых алгоритмов сортировки работает следующим образом. Сначала просматриваются все элементы массива, находится наименьший элемент, который меняется местами с элементом, стоящим *первым* в сортируемом массиве. Потом находится второй наименьший элемент и меняется местами с элементом, стоящим *вторым* в исходном массиве. Этот процесс продолжается до тех пор, пока весь массив не будет отсортирован.

Данный метод называется сортировкой выбором (*selection sort*), поскольку он все время выбирает наименьший элемент из числа не отсортированных.

Пример.

$9,8,7,6,5,4,3,2,1,0$

$0,8,7,6,5,4,3,2,1,9$

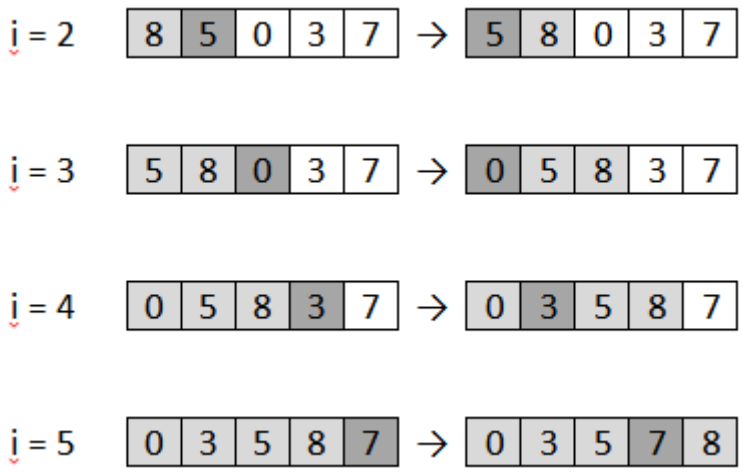
$0,1,7,6,5,4,3,2,8,9$

$0,1,2,6,5,4,3,7,8,9$

$0,1,2,3,5,4,6,7,8,9$

$0,1,2,3,4,5,6,7,8,9$

Рассмотрим на примере числовой последовательности процесс сортировки методом вставок. Клетка, выделенная темно-серым цветом – активный на данном шаге элемент, ему также соответствует i-ый номер. Светло-серые клетки это те элементы, значения которых сравниваются с i-ым элементом. Все, что закрашено белым – не затрагиваемая на шаге часть последовательности.



Последний рассматриваемый в задании метод сортировки – сортировка «пузырьком» (прямым обменом). Алгоритм сортировки прямым обменом основан на принципе сравнения и обмена пары соседних элементов до тех пор, пока не будут отсортированы все элементы.

Как и в методе прямого выбора, совершаются проходы по массиву, сдвигая каждый раз наименьший (наибольший) элемент оставшейся последовательности к началу массива.

Пример. Пусть исходный массив имеет вид: $9,8,7,6,5,4,3,2,1,0$. Отсортируем его в порядке возрастания:

$$
\begin{array}{l}
9,8,7,6,5,4,3,2,1,0 \\
8,9,7,6,5,4,3,2,1,0 \\
8,7,9,6,5,4,3,2,1,0 \\
8,7,6,9,5,4,3,2,1,0 \\
8,7,6,5,9,4,3,2,1,0 \\
\ldots\ldots\ldots\ldots\ldots\ldots \\
8,7,\ldots\ldots\ldots\ldots\ldots 1,0,9 \\
0,1,2,3,4,5,6,7,8,9
\end{array}
$$

Задание. Требуется считать из файла, находящегося на диске одномерный массив, состоящий из 30 произвольных целых чисел и отсортировать его в порядке убывания приведенными выше методами. На экран вывести количество операций сравнения и пересылки чисел и время выполнения программы для каждого метода.