**Лабораторная работа 13, 14**

**Алгоритмы сортировки**

***Цель лабораторной работы:*** освоить основные алгоритмы сортировки, написать программу с использованием этих алгоритмов.

**Общие понятия**

Сортировка – это процесс упорядочения элементов массива или списка по возрастанию или убыванию.

Существует много алгоритмов сортировки, отличающихся по ряду характеристик:

· Время работы, или вычислительная сложность – количество операций, затрачиваемых алгоритмом. Обычно оценивается худший сценарий, когда исходный массив оказывается максимально неупорядочен с точки зрения алгоритма.

· Затрачиваемая память (помимо исходного массива) – некоторые алгоритмы требуют выделения дополнительной памяти для временного хранения данных или формирования нового выходного массива.

Кроме того, алгоритмы можно разделить по типу доступа к данным:

· Алгоритмы *внутренней сортировки* применяются для сортировки данных, целиком находящихся в оперативной памяти.

· Алгоритмы *внешней сортировки* оперируют данными, не помещающимися в оперативную память. Такие алгоритмы используют внешнюю память, доступ к которой требует существенно большего времени, поэтому требуются специальные алгоритмические решения, чтобы каждый элемент использовался алгоритмом минимальное количество раз.

**Алгоритмы сортировки. Метод пузырька**

Данный алгоритм является достаточно простым и поэтому получил широкое распространение. Вычислительная сложность алгоритма квадратичная – O(n2), поэтому алгоритм эффективен только на небольших массивах данных.

Алгоритм проходит все элементы массива и попарно сравнивает их друг с другом. Если порядок сравниваемых элементов неверный, алгоритм меняет элементы местами:

***// Сортировка пузырьком***

void BubbleSort(ref int[] Array)

{

// Перебираем все элементы массива (кроме последнего)

for (int i = 0; i < Array.Length - 1; i++)

// Перебираем все элементы справа от i

for (int j = i + 1; j < Array.Length; j++)

// Правильный ли порядок элементов?

if (Array[i] > Array[j])

{

// Нет - меняем порядок

int t = Array[i];

Array[i] = Array[j];

Array[j] = t;

}

}

**Сортировка выбором**

Сортировка выбором имеет квадратичную сложность O(n2) и, как и предыдущий метод пузырька, эффективен лишь на небольших объемах данных.

Алгоритм находит номер минимального значения в текущем списке, меняет этот элемент со значением первой не отсортированной позиции (если минимальный элемент не находится на данной позиции), а затем сортирует хвост списка, исключив из рассмотрения уже отсортированные элементы:

***// Сортировка выбором***

void SelectionSort(ref int[] Array)

{

// Перебираем все элементы массива (кроме последнего)

// i - позиция первого неотсортированного элемента

for (int i = 0; i < Array.Length - 1; i++)

{

// Позиция минимального элемента справа от i

int min = i;

// Перебираем все элементы справа от i

for (int j = i + 1; j < Array.Length; j++)

// Меньше ли очередной элемент минимального?

if (Array[j] < Array[min])

min = j; // Да - теперь это новый минимальный элемент

// Минимальный элемент не на первом месте? Меняем местами!

if (min != i)

{

int t = Array[i];

Array[i] = Array[min];

Array[min] = t;

}

}

}

**Быстрая сортировка**

Алгоритм быстрой сортировки является одним из самых быстрых алгоритмов сортировки: в лучшем случае он имеет логарифмическую сложность, в худшем – квадратичную. Алгоритм выполняется следующим образом:

1. Выбирается некоторый элемент, который называется *опорным*.

2. Реорганизуем массив таким образом, чтобы все элементы, меньшие или равные опорному элементу, оказались слева от него, а все элементы, большие опорного — справа от него.

3. Рекурсивно упорядочиваем массивы, лежащие слева и справа от опорного элемента.

***// Быстрая сортировка***

void QuickSort(ref int[] Array, int Left, int Right)

{

// i и j - индексы границ разделяемого массива

int i = Left;

int j = Right;

// x - индекс опорного элемента

int x = Array[(Left + Right) / 2];

do

{

// Ищем элемент слева, который больше опорного

while (Array[i] < x)

++i;

// Ищем элемент справа, который меньше опорного

while (Array[j] > x)

--j;

// Если индексы не поменялись местами, то обмениваем элементы

if (i <= j)

{

int t = Array[i];

Array[i] = Array[j];

Array[j] = t;

i++;

j--;

}

} while (i <= j);

// Рекурсивно выполняем быструю сортировку для массивов слева и справа

if (Left < j)

QuickSort(ref Array, Left, j);

if (i < Right)

QuickSort(ref Array, i, Right);

}

**Поиск элемента**

Алгоритмы поиска позволяют найти индекс элемента с требуемым значением.

Если массив не упорядочен, то возможен лишь *простой поиск*: перебор всех элементов массива до тех пор, пока не встретится элемент с нужным значением или не закончится массив. Если элемент найден, поиск должен быть прекращён, поскольку дальнейший просмотр массива не имеет смысла:

***// Простой поиск элемента в массиве***

int IndexOf(ref int[] Array, int Value)

{

// Перебираем все элементы массива

for (int i = 0; i < Array.Length; i++)

// Если нашли нужное значение, то возвращаем его индекс

if (Array[i] == Value)

return i;

// Перебор закончился безрезультатно - возвращаем -1

return -1;

}

 Если алгоритм поиска не нашёл подходящий элемент, он должен каким-то образом сигнализировать об этом вызывающей программе. Чаще всего в таком случае возвращается значение -1 – число, которое заведомо не может использоваться в качестве индекса массива.

Вычислительная сложность алгоритма простого поиска – линейная O(n).

Если же массив упорядочен по возрастанию, то возможно использование дихотомического рекурсивного алгоритма: массив каждый раз делится пополам и если искомый элемент меньше среднего, то поиск продолжается в левой его половине, иначе – в правой:

***// Дихотомический поиск элемента в массиве***

static int IndexOf(ref int[] Array, int Value, int Left, int Right)

{

// Находим середину диапазона

int x = (Left + Right) / 2;

// Если нашли значение - возвращаем его индекс

if (Array[x] == Value)

return x;

// Если середина совпадает с левой или правой границами - значение не найдено

if ((x == Left) || (x == Right))

return -1;

// Продолжаем поиск слева или справа от середины

if (Array[x] < Value)

return IndexOf(ref Array, Value, x, Right);

else

return IndexOf(ref Array, Value, Left, x);

}

Вычислительная сложность алгоритма – логарифмическая.

**6 Выполнение индивидуального задания**

Общая часть задания: сформировать массив из 100 случайных чисел. Выполнить простой поиск элемента, подсчитать количество итераций. Отсортировать массив методом, указанным в своём варианте. Выполнить поиск элемента методом дихотомии, подсчитать количество итераций. Сделать выводы.

1) Метод пузырька

2) Сортировка выбором

3) Быстрая сортировка