**Методы сортировки**

**Описание сортировки**

Пусть задано некоторое множество x, на элементах которого установлено отношение порядка. Пусть есть некоторая последовательность x1, x2, …, xn (1), элементы которой принадлежат множеству x.

Пусть есть перестановка чисел i1, i2, …, in такая, что последовательность xi1, xi2,…, xin (2) обладает следующим свойством: xi1 <= xi2 <= …<= xin.

Тогда последовательность (2) называется отсортированной последовательностью, а задачу получения последовательности (2) по последовательности (1) называют задачей сортировки.

Алгоритмы, с помощью которых решаются задачи сортировки, называются алгоритмами сортировки. Алгоритмов сортировки существует большое множество, однако их все можно разбить на два больших класса:

* алгоритмы внутренней сортировки;
* алгоритмы внешней сортировки.

Алгоритмы внутренней сортировки применяются в тех случаях, когда сортируемые данные целиком помещаются в оперативной памяти.

Алгоритмы внешней сортировки применяются тогда, когда сортируемые данные целиком не помещаются в оперативной памяти и хранятся на внешних устройствах.

Обычно алгоритмы сортировки применяются для некоторых массивов. Структура элементов такого массива может быть сложной, и тогда для сортировки выбирается некоторое поле в структуре элемента массива, после чего сортировка ведется по данному полю.

Главным условием является то, что такое поле должно иметь упорядоченный тип.

Важным показателем эффективности работы любого алгоритма являются время работы и объем памяти.

Эти параметры необходимо оценивать еще до написания кода программы, так как не каждый алгоритм может эффективно работать на представленных данных. Указывать точные параметры оценки алгоритмов нет необходимости. Достаточно лишь указать вид зависимости от объема исходных данных.

Существуют следующие виды зависимости:

* O (n)-порядка n — линейная зависимость;
* O (n2) — квадратичная зависимость;
* O (log) — логарифмическая зависимость.

**Пузырьковая сортировка**

Является самым простым для понятия и реализации алгоритмом сортировки, но малоэффективным на больших объемах данных.

Идея метода заключается в том, что элементы массива просматриваются слева на право и проверяются парой соседних элементов. Если элементы некоторой пары находятся в неупорядоченном отношении, то они обмениваются местами.

Таким образом, на каждом шаге самый маленький элемент массива из оставшихся вытягивается в начало массива.

Пример.

const int n=10;

int a[n];

for (int i=0; i<n; i++)

cin>>a[i];

for (int i=0; i<n; i++)

for (int j=n-1; j>i; j--)

if (a[j-1]>a[j])

{

int t=a[j-1];

a[j-1]=a[j];

a[j]=t;

}

for (int i=0; i<n; i++)

cout<<a[i]<< « »;

Память — O (n) c1\*n+c2.

Время — O (n2).

**Метод линейного поиска с обменом**

Идея данного метода заключается в следующем:

* на первом шаге отыскивается минимальный элемент среди элементов массива с первого по последний, затем найденный элемент обменивается местами с первым элементом массива;
* на втором шаге отыскивается минимальный элемент массива среди элементов со 2-го по последний, затем найденное значение обменивается местами со вторым элементом массива и так далее.

Таким образом, за (n-1) шаг массив окажется отсортированным.

Для реализации линейного поиска с обменом объявляют вспомогательную функцию, которая возвращает номер минимального элемента массива в заданном диапазоне.

Пример.

int minin\_d (int a[], int n, int f) // f — с какой позиции ищем минимум

{int min =a[f];

int in\_d=f;

for (int i=f+1; i<n; i++)

if (a[i]<min)

{

Min=a[i];

in\_d=i;

}

return in\_d;

}

int main()

{

const int n=10;

int a[n];

//необходимо заполнить массив

for (int i=0; i<n-1; i++)

{

int p=minin\_d(a,n,i);

int t=a[i];

a[i]=a[p];

a[p]=t;

}

//вывод a

Время работы квадратичное и не зависит от перестановки элементов.

**Метод челночной сортировки**

Элементы массива просматриваются последовательно слева направо и при необходимости обмениваются местами. Если произошел обмен, тогда меньший элемент протаскивается по возможности в начало массива.

int main()

{

const int n=10;

int a[n];

//ввод массива

for (int i=1; i<n; i++)

if (a[i-1]>a[i])

{

int t=a[i];

a[i] =a[i-1];

a[i-1]=t;

int k=i-1;

while (k>=1)

{

if (a[k-1]>a[k])

{

int p=a[k-1];

a[k-1]=a[k];

a[k]=t;

k--;

}

else

k=0;

}

}

// вывод а

Оценка по времени в общем случае квадратичная и зависит от начальной расстановки элементов.

**Метод подсчета**

Используется в тех случаях, когда исходные и отсортированные данные должны находиться в разных ячейках памяти.

Идею алгоритма можно проиллюстрировать следующим примером:

int c=0;

for (int i=1; i<n; i++)

if (a[i] < a[0])

c++;

В отсортированном массиве элемент a[0] должен занять позицию в «с», так как в исходном массиве меньше с-элементов.

Пример.

int main()

{

const int n=10;

int a[n], b[n];

int c[n]={};

for (int i=0; i<n-1; i++)

for (int j=i+1;j<n;j++)

if (a[i] <a[j])

c[i]++;

else

c[j]++;

for (int i=0; i<n; i++)

b[c[i]]=a[i];

}

Оценка по времени — квадратичная.

**Быстрая сортировка**

Идея алгоритма заключается в следующем: если в массиве имеется такой элемент, что все элементы, расположенные левее, меньше него, а все элементы, расположенные правее, — больше, то такой элемент уже занимает свою позицию в массиве. Тогда для сортировки массива достаточно независимо друг от друга отсортировать левую и правую части относительно данного элемента.

Такой элемент массива называется опорным элементом, и в общем случае опорного элемента может и не быть в исходном массиве, поэтому основная сложность алгоритма быстрой сортировки заключается в получении опорного элемента. Вследствие этого возникает высокая вероятность необходимости многократной перестановки элементов в массиве.

Пример.

void quick\_sort (int a[], int left, int right)

{

int i=left();

int j=right;

int f=a[(left+right/2)]; // элемент, который мы взяли в качестве опорного

while (i<j)

{

while (a[i]<f)

i++;

while (a[j]>f)

j--;

if(i<=j)

{

int t=a[i];

a[i]=a[j];

a[j]=t;

j++;

j--;

}

}

if (left<j)

quick\_sort (a, left, j);

if (i<right)

quick\_sort (a, I, right);

}

int main()

{

const int n=10;

int a[n];//ввод массива

quick\_sort (a, 0, n-1);

//вывод а

}

Оценка по времени — логарифмическая.