Тема 1.6 Сортировка и поиск информации. Методы внутренней сортировки

Различают два вида сортировок:
- сортировки файлов;
- сортировки массивов.

Эти два вида часто называют внутренней и внешней сортировками, так как массивы располагаются в оперативной памяти, а файлы – во внешней.

Задана последовательность. Сортировка означает перестановку элементов последовательности в соответствии с функцией упорядочения таким образом, что каждый следующий элемент больше или меньше предыдущего в зависимости от того, возрастающая или убывающая сортировка.

Как правило, функция не вычисляется по правилу, а содержится в каждом элементе, и называется ключом.

Устойчивым называется метод сортировки, при котором относительный порядок элементов не изменяется. Устойчивая сортировка применяется в тех случаях, если элементы отсортированы по вторичным ключам (по имени, типу.)

Сортировка массивов

Основное требование к методам сортировки массивов – экономное использование памяти. Это означает, что переупорядочивание элементов необходимо выполнять на месте, т.е. не желательно использовать методы сортировок, при которых элементы из одного массива пересылаются в другой.

Существует много различных методов сортировки, которые отличаются степенью эффективности. Под степенью эффективности понимается количество сравнений и обменов, производимых в процессе сортировки, время выполнения и объем занимаемой оперативной памяти.

Среди методов выделяют три основных:
- метод включения или вставок;
- метод выбора;
- метод обмена.

Каждый из методов имеет базовый простой метод и его усовершенствованный.

Обменные сортировки – это:
- сортировка «пузырек»;
- Шейкер сортировка;
- параллельная сортировка Бетчера;
- сортировка Хоара.

Сортировки выбора:
- простым выбором;
- пирамидальная сортировка;
- сортировка с использованием дерева.

Сортировка вставками:
- простыми вставками;
- бинарными вставками;
- двухпутевыми вставками;
- метод Шелла.

Суть сортировки выбором состоит в том, что сначала в неупорядоченной последовательности выбирается минимальный элемент, далее этот элемент исключается из дальнейшей обработки, оставшаяся последовательность элементов принимается за исходную.

Процесс повторяется до тех пор, пока все элементы не будут выбраны. Таким образом, выбранные элементы образуют упорядоченную последовательность.

Выбранный в исходной последовательности минимальный элемент размещается на предназначенном ему месте несколькими способами:

- минимальный элемент 'i' – го просмотра перемещается на 'i' – е место другого массива, а в исходном массиве на его месте размещается какое-то очень большое число, превосходящее по величине любой элемент сортируемого массива. Измененный таким образом массив принимается за исходный. Дальше действия повторяются.

- минимальный элемент после 'i'-го просмотра перемещается на 'i'-е место заданного массива, а с 'i'-го места на место выбранного. После каждого просмотра упорядоченные элементы исключается из дальнейшей обработки. Этот метод сортировки применяется для массивов, не содержащих повторяющихся элементов.

Алгоритм сортировки:

1. Выбрать максимальный элемент;
2. Поменять его местами с последним элементом. После чего наибольший элемент будет стоять на своем месте.
3. Повторить п.1 и п.2 с оставшимися n-1 элементами, т.е. рассмотреть часть массива, начиная с 1-го элемента до предпоследнего, найти в ней максимальный элемент и поменять его местами с предпоследним. Так выполнять до тех пор, пока не останется один элемент.

```
void selectionSort(int data[], int lenD)
{
  int j = 0;
  int tmp = 0;
  for(int i=0; i<lenD; i++){
    j = i;
    for(int k = i; k<lenD; k++){
      if(data[j]>data[k]){
        j = k;
      }
    }
    tmp = data[i];
    data[i] = data[j];
    data[j] = tmp;
  }
}
```

Сортировка обменом – это метод, при котором все соседние элементы попарно сравниваются друг с другом и меняются местами в том случае, если предшествующий элемент больше последующего. В результате этого максимальный элемент постепенно смещается вправо и в конце концов занимает свое крайнее место в массиве, после чего исключается из обработки. Затем процесс повторяется, пока свое место не займет второй по величине элемент. Так будет продолжаться до тех пор, пока последовательность не отсортируется.

```
void bubbleSort(int data[], int lenD)
{
  int tmp = 0;
  for(int i = 0;i<lenD;i++){
    for(int j = (lenD-1);j>=(i+1);j--){
      if(data[j]<data[j-1]){
        tmp = data[j];
        data[j]=data[j-1];
        data[j-1]=tmp;
      }
    }
  }
        }
```

Сортировка вставками применяется для массивов, не содержащих повторяющихся элементов. Как и все методы производится по шагам. Суть ее в том, что на К-том шаге считается, что часть массива, содержащая К-1 элементов уже упорядочена, т.е. отсортирована по возрастанию или убыванию. Далее необходимо взять К-тый элемент и подобрать для него такое

место в отсортированной последовательности, чтобы после его вставки упорядоченность не нарушилась.

```
void insertionSort(int data[], int lenD)
{
  int key = 0;
  int i = 0;
  for(int j = 1;j<lenD;j++){
    key = data[j];
    i = j-1;
    while(i>=0 && data[i]>key){
      data[i+1] = data[i];
      i = i-1;
      data[i+1]=key;
    }
  }
       }
```

Быстрая сортировка использует алгоритм "разделяй и властвуй". Она начинается с разбиения исходного массива на две области. Эти части находятся слева и справа от отмеченного элемента, называемого опорным. В конце процесса одна часть будет содержать элементы меньшие, чем опорный, а другая часть будет содержать элементы больше опорного.

```
void quickSort(int* data, int const len)
{
  int const lenD = len;
  int pivot = 0;
  int ind = lenD/2;
  int i,j = 0,k = 0;
  if(lenD>1){
    int* L = new int[lenD];
    int* R = new int[lenD];
    pivot = data[ind];
    for(i=0;i<lenD;i++){
      if(i!=ind){
        if(data[i]<pivot){
          L[j] = data[i];
          j++;
        }
        else{
          R[k] = data[i];
          k++;
        }
      }
    }
    quickSort(L,j);
    quickSort(R,k);
    for(int cnt=0;cnt<lenD;cnt++){
      if(cnt<j){
        data[cnt] = L[cnt];;
      }
      else if(cnt==j){
        data[cnt] = pivot;
      }
      else{
        data[cnt] = R[cnt-(j+1)];
      }
    }
  }
}
```