Лектор: А.Д.Хапугин

Основы программного конструирования

Лекция 8-1. Сортировки

Материалы доступны в Интернете по адресу: http://www.excelsior.ru/afti/

Простой пример — сортировка колоды карт

1. На карты можно посмотреть на все сразу, их можно брать в руку по нескольку штук — и так далее. Таких возможностей у компьютера меньше.

2. Если присмотреться как работает наше мышение при сортировке карт, то можно заметить, что есть основа этого процесса, а есть куча оптимизаций (например, брать по нескольку штук, пропускать отсортированные подпоследовательности и т.п.)

Алгоритм

Для карт:
Считаем, что часть карт слева отсортирована
Берем самую левую карту из неотсортированной части
находим для нее место в отсортированной части
вставляем эту карту в это место
продолжаем этот процесс пока не кончится неотсортированная часть

Для массива:
1. Позиция в массиве = положение элемента в последовательности
2. Граница отсортированной части = переменная i (индекс)

Переформулируем для массива

Пусть слева часть массива до какого-то i отсортирована.
Берем x[i]
Ищем для x[i] место, куда его нужно вставить - n
Вставляем x[i] на место x[n], сдвигая вправо элементы от n до i
после этого граница отсортированной части массива сдвигается вправо на 1
повторяем этот процесс пока i не дойдет до конца массива

Первая реализация

```
PROCEDURE sort(VAR x: ARRAY OF CARDINAL);
  VAR i,j,n: CARDINAL; e: CARDINAL;
BEGIN
  FOR i:=1 TO HIGH(x) DO (* основной цикл *)
    j:=i-1;
    (* находим место для x[i] *)
    WHILE (j>0) & (x[j]>x[i]) DO
      DEC(j)
    END;
    IF x[j]<=x[i] THEN INC(j) END; (* техническая проверка *)
    (* найденное место = j *)
    IF j<i THEN
      e:=x[i]; n:=i;
      WHILE n>j DO (* сдвигаем элементы *)
        x[n]:=x[n-1];
        DEC(n);
      END;
      x[n]:=e; (* вставляем новый элемент на место *)
    END;
  END;
END sort;
```

Улучшение

Объединяем циклы поиска места и сдвига элементов, потому что оба эти цикла «ходят» по одному и тому же месту.

```
PROCEDURE sort(VAR x: ARRAY OF CARDINAL);
  VAR i,j,n: CARDINAL; e: CARDINAL;
BEGIN
  FOR i:=1 TO HIGH(x) DO (* главный цикл *)
(* ищем место для x[i] одновременно сдвигая вправо элементы *)
    j:=i-1; e:=x[i];
    WHILE (j>0) & (x[j]>e) DO
      x[j+1]:=x[j];
      DEC(j);
    END;
    IF x[j]<=e THEN INC(j) END; (* техническая проверка *)
    x[j]:=e; (* вставляем новый элемент на место *)
  END;
END sort;
```

Теперь некрасиво выглядит «техническая проверка». Улучшим еще.

Полировка

```
PROCEDURE sort(VAR x: ARRAY OF CARDINAL);
  VAR i,j,n: INTEGER; e: CARDINAL;
BEGIN
  n:=HIGH(x); (* преобразование типов, больше ничего *)
  FOR i:=1 TO n DO
    e:=x[i]; j:=i-1;
    WHILE (j>=0) & (x[j]>=e) DO
      x[j+1]:=x[j];
      DEC(j);
    END;
    x[j+1]:=e;
  END;
END sort;
```

Сложность

Сколько времени будет работать алгоритм?

Такой вопрос неправомерен, потому что время зависит от данных.

Вид такой зависимости и есть характеристика сложности алгоритма.

Для данного алгоритма сложность выражается формулой:
$C(n) = A*n^2 + B*n + C$
где A,B,C — какие-то коэффициенты.

Оценить приблизительные значения коэффициентов оставляем на самостоятельную работу читателя.
Для практики точное значение коэффициентов имеет ценность, только если мы сравниваем несколько алгоритмов для относительно небольших объемов данных. Для больших объемов данных значение B и C пропадает

Конец лекции.