Алгоритм описан в процедуре ShellSort [2.9] для T = 4:

```
PROCEDURE ShellSort;                                    (* ADruS2_Sorts *)
    CONST T = 4;
    VAR i, j, k, m, s: INTEGER;
        x: Item;
        h: ARRAY T OF INTEGER;
BEGIN
    h[0] := 9; h[1] := 5; h[2] := 3; h[3] := 1;
    FOR m := 0 TO T-1 DO
        k := h[m];
        FOR i := k TO n-1 DO
            x := a[i]; j := i-k;
            WHILE (j >= k) & (x < a[j]) DO
                a[j+k] := a[j]; j := j-k
            END;
            IF (j >= k) OR (x >= a[j]) THEN
                a[j+k] := x
            ELSE
                a[j+k] := a[j];
                a[j] := x
            END
        END
    END
END ShellSort
```

Анализ сортировки Шелла. Анализ этого алгоритма – очень сложная математическая проблема, полностью еще не решенная. В частности, неизвестно, какая последовательность расстояний дает наилучший результат. Но удивительно то, что расстояния не должны быть кратными друг другу. Тогда исчезнет явление, наблюдаемое в приведенном примере, когда каждый проход сортировки объединяет две цепочки, до того никак не «общавшиеся». На самом деле желательно, чтобы различные цепочки «общались» как можно чаще, причем выполняется следующая теорема: если последовательность после k-сортировки подвергается i-сортировке, то она остается k-отсортированной. Кнут в [2.7] (см. с. 105–115 перевода) приводит аргументы в пользу того, что неплохим выбором расстояний является последовательность (записанная здесь в обратном порядке)

1, 4, 13, 40, 121, ...

где $h_{k-1} = 3h_k+1$, $h_T = 1$, и $T = k \times \lfloor \log_3(n) \rfloor - 1$. Он также рекомендует последовательность

1, 3, 7, 15, 31, ...

где $h_{k-1} = 2h_k+1$, $h_T = 1$, и $T = k \times \lfloor \log_2(n) \rfloor - 1$. В последнем случае математическое исследование показывает, что при сортировке n элементов алгоритмом Шелла затраты пропорциональны $n^{1.2}$. Хотя это гораздо лучше, чем n^2, мы не будем углубляться в этот метод, так как есть алгоритмы еще лучше.