# Tutorial 10

## Example 1

- **f(S:string):int** =
  $res := 0$
  **for** $i := 1$ to $n$ **do**
  **case** $S[i]$ **of**
  ”a”: $res := res + 1$
  ”b”: $res := res + 2$
  ”c”: $res := res + 3$
  **end case**
  **end for**
  **return** $res$

- f(”a”) = 1
  f(”b”) = 2
  f(”ab”) = 3
  f(”aab”) = 4
  f(”cab”) = 6
  f(”c”) = 3

- 

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| | a | b | ab | aab | c | cab |

- 

| 0 | 1 | 2 |
|---|---|---|
| ab | a | b |
| cab | aab | |
| c | | |
| | | |
| | | |

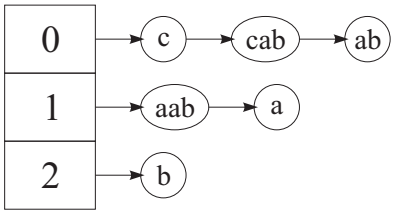


Figure 1: Chaining.

## Example 2

- **bubble_sort(A[1..n])** =
  $sorted := false$
  **while** NOT $sorted$ **do**
    $sorted := true$
    **for** $i := 1$ to $n - 1$ **do**
      **if** $A[i] > A[i + 1]$ **then**
        $temp := A[i + 1]$
        $A[i + 1] := A[i]$
        $A[i] := temp$
        $sorted := false$
      **end if**
    **end for**
  **end while**

- In the worst-case the smallest element is in the last position of the input array. An example of such an array is e.g. $A = [n, n - 1, n - 2, ..., 2, 1]$.
  Worst-case time complexity is O($n^2$).

- The algorithm performs fast when the array is almost sorted apart from some local changes in the order. If this is not the case the algorithm performs very bad and it is not very useful in general.

## Example 3

- **equal(A[1..n], B[1..n]):boolean** =
  $merge\_sort(A)$
  $merge\_sort(B)$
  $res := true$
  **for** $i := 1$ to $n$ **do**
    **if** NOT $A[i] = B[i]$ **then**
      $res := false$
    **end if**
  **end for**
  **return** $res$

- Two calls of merge_sort take $O(n \log n)$ + $O(n \log n)$ which is $O(n \log n)$. The FOR loop has $n$ iterations and takes $O(n)$. Hence the worst-case time complexity is $O(n \log n)$ + $O(n)$ which is $O(n \log n)$.