1 Predefined Rules

1. If the following item has:

   see the ans:

   -

   Then probably this item has errors corrected by the ans(wer).

2 section 3.1

14 skipped

☑ 4. Largest Difference. See this

Algorithm 1: Largest Difference.

input : 232 and sdsd
procedure *largest difference*$(a_1, a_2, \ldots, a_n$: integers)
$i := 1\{i$ is the search index$\}$
$d := 0\{d$ is the target difference$\}$
while ($i \leq n - 1$)
  if $d < a_{i+1} - a_i$ **then** $d = a_{i+1} - a_i$
return d

☐ 6. Number Of Negative Integers. See this

Algorithm 2: Number Of Negative Integers.

procedure *number of negative integers*$(a_1, a_2, \ldots, a_n$: integers)
$i := 1\{i$ is the search index$\}$
$c := 0\{c$ is the number of negative integers$\}$
while ($i \leq n$)
  if $a_i < 0$ **then** $c = c + 1$
return c

see the ans:

- here the step size of i is not explicitly shown.

☐ 8. Location Of The Largest Even Integer. See this

Algorithm 3: Location Of The Largest Even Integer.

procedure *location of the largest even integer*$(a_1, a_2, \ldots, a_n$: integers)
$c := 1\{c$ is the location index$\}$
$find\_even := 0$ $\{find\_even$ tracks whether even number exists$\}$
for i := 1 **to** n
  if $(a_i\%2 == 0$ and $a_c < a_i)$ **then** $c = i$
return $find\_even$

see the ans:

- here should initialize the comparison with $-\infty$
- $find\_even$ isn't assigned value in the loop

☐10. Compute x^n. See this

Algorithm 4: Compute x^n.

procedure *compute* x^n(x: real number, n: an integer)
$c := 1$\{c is the target product\}
if $n \geq 0$ **then**
  for i := 1 **to** n
    $c = c * \cdot x$
else
  $n = -n$
  use this new n to calculate
return c

see the ans:

- use the absolute to combine the 2 cases
- the above lack the reciprocal for the negative case.

☑12. Replaces The Triple (x, y, z) With (y, z, x). See this

Here obviously at least 3, but this will lose one variable when assigning, so one extra to save this lost one.

Algorithm 5: Replaces The Triple (x, y, z) With (y, z, x).

procedure *replaces the triple* (x, y, z) *with* (y, z, x)((x, y, z): target tuple)
$c := 0$\{c is the tmp variable\}
$c = x$
$x = y$
$y = z$
$z = c$
return (x, y, z)

☐16. Smallest Integer In A Finite Sequence. See this

Algorithm 6: Smallest Integer In A Finite Sequence.

procedure *smallest integer in a finite sequence*($a_1, a_2, \ldots, a_n$: the sequence)
$c := \infty$\{c is the target number\}
for i := 1 **to** n
  if $c > a_i$ **then**
    $c = a_i$
return c

see the ans:

- $c = a_1$ is duplicate.

☑18. The Last Occurrence Of The Smallest Element . See this

This is similar to 16

Algorithm 7: The Last Occurrence Of The Smallest Element .

procedure *the last occurrence of the smallest element* ($a_1, a_2, \ldots, a_n$: the sequence)
$c := a_1\{c$ is the target number$\}$
$l := 1\{l$ is the target number index$\}$
for i := 2 **to** n
    if $c \geq a_i$ **then**
        $c = a_i$
        $l = i$
return l

☐20. The Largest And The Smallest Integers. See this

Algorithm 8: The Largest And The Smallest Integers.

procedure *the largest and the smallest integers*($a_1, a_2, \ldots, a_n$: the sequence)
$c := a_1\{c$ is the target smallest number$\}$
$l := a_1\{l$ is the target largest number$\}$
for i := 2 **to** n
    if $c \geq a_i$ **then** $c = a_i$
    if $l \leq a_i$ **then** $l = a_i$
return c, l

see the ans:

- different from 18, here we don't need to track **equal** condition.

☑22. The Longest Word In An English Sentence. See this

Algorithm 9: The Longest Word In An English Sentence.

procedure *the longest word in an English sentence*($a_1, a_2, \ldots, a_n$: symbols)
$c := 0\{c$ is the track length$\}$
$len := 0\{len$ is the max length$\}$
$word := blank\{word$ is the finded word$\}$
for i := 1 **to** n
    if a_i is a letter **then** $c = c + 1$
    else if a_i is a blank **then**
        $c = 0$
        if $len < c$ **then**
            $len = c$
            $word$ is $substr(a_{i-len} : a_{i-1})$
    else error
return $word$

Here I use *substr* which is similar to one predefined function in C, like this or others, to replace the ans $word := \lambda \ldots word := concatenation \ldots$

Also, $length := 0$ is implied in the above $c = 0$.

The ans uses λ to represent empty string as book p189 shows.

☐24. Whether A Function Is One-to-one. See this see the ans:

- lack the func parameter
- My original thoughts are that we put all values mapped from the domain in one new set, and check whether duplicate conditions occur for each mapped value by **traversing** the list for the same value with the current calculated one. Obviously, this has the much higher complexity.
  The ans just use the *hit* to make *one_one* false when accessed twice, avoiding the traverse.

Algorithm 10: Whether A Function Is One-to-one.

---

procedure *whether a function is one-to-one*($\{a_1, a_2, \ldots, a_n\}$: function domain, $\{b_1, b_2, \ldots, b_n\}$: function codomain)

☐26. Terminating Algorithm 3 If $x = a_m$.
add if $x == a_m$ then $find = 1; location = i; break$ before the 1st if block,
then if $find == 1$ then $return \ldots$ before the 2nd if block
see the ans:

- the ans modified the action when $x < a_m$, this speeds up more.

☐28. Four Sublists. See this
see the ans:

- lack the func parameter
- My original thoughts are that we put all values mapped from the domain in one new set, and check whether duplicate conditions occur for each mapped value by **traversing** the list for the same value with the current calculated one. Obviously, this has the much higher complexity.
  The ans just use the *hit* to make *one_one* false when accessed twice, avoiding the traverse.

☐29. test 3
☐31. test 5
☐33. test 5
☐35. test 5

Algorithm 11: Four Sublists.

procedure *four sublists*($\{a_1, a_2, \ldots, a_n\}$: a list, x : the target element)

$c_i := 1\{c$ is the ith split point$\}$

$i := 1, j := n\{j, i$ is the tmp index$\}$

$location := 0\{location$ is the target index, which is inited "not found"$\}$

while ($i < j$)

  $c_1 = i + \lfloor \frac{j-i}{4} \rfloor$

  $c_2 = i + \lfloor \frac{j-i}{2} \rfloor$

  $c_3 = i + \lfloor \frac{(j-i)*3}{4} \rfloor$

  if $x \leq a_{c_1}$ **then** $j = c_1$

  else if $x \leq a_{c_2}$ **then**

    $i = c_1$

    $j = c_2$

  else if $x \leq a_{c_3}$ **then**

    $i = c_2$

    $j = c_3$

  else $i = c_3$

if $x == a_j$ **then** $location = j$

return *location*