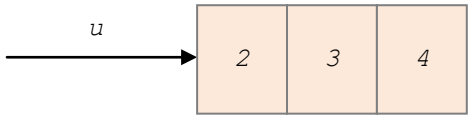
Exam: solution

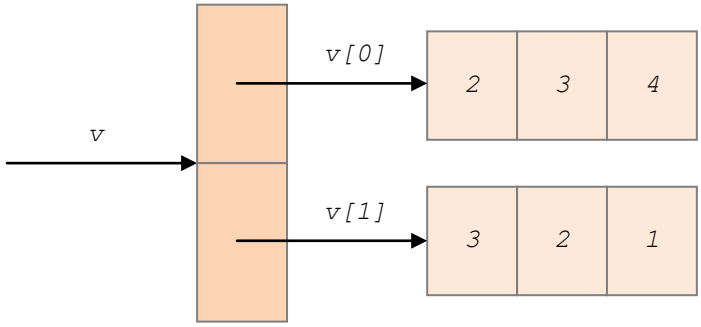
Tasks: solutions

Task 1 (3 points + 3 points)

a) (3 points)



b) (3points)



Task 2 (3 points + 3 points + 3 points)

a) (3 points)

```
public static Triangle maxTriangle (Triangle[] triangles)
{
    if (triangles.length == 0)
        throw new java.lang.IllegalArgumentException ("empty array");

    Triangle    maxTr = triangles[0];
    double       maxPerimeter = maxTr.perimeter ();
    for (int pos = 1; pos < triangles.length; pos++)
        if (triangles[pos].perimeter () > maxPerimeter)
        {
            maxTr = triangles[pos];
            maxPerimeter = maxTr.perimeter ();
        }

    return maxTr;
}
```

b) (3 points)

```
public static double totalArea (Triangle[] triangles)
{
    double    totalAr = 0;
    for (Triangle t : triangles)
```

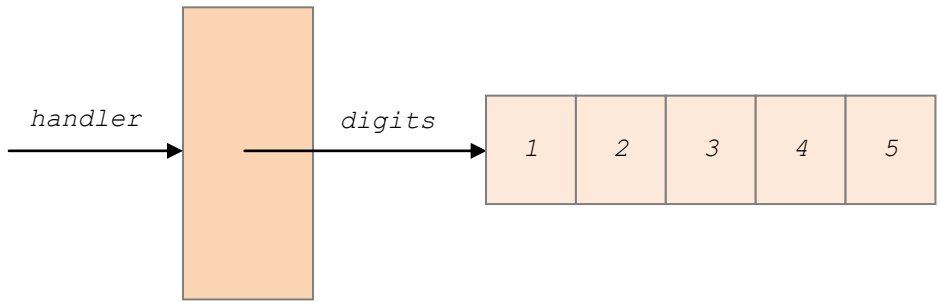
```
        totalAr = totalAr + t.area ();

     return totalAr;
}
```

c) (3 points)

```
Triangle[]    triangles = { new Triangle (3, 4, 5),
                           new Triangle (4, 4, 4),
                           new Triangle (4, 3, 4),
                           new Triangle (4, 5, 6) };
Triangle    maxTr = maxTriangle (triangles);
double    totalAr = totalArea (triangles);
```

Task 3 (3 points + 3 points + 3 points)

a) (3 points)



b) (3 points)

```
public void circularShiftLeft ()
{
    if (digits.length > 0)
    {
        byte    b = digits[0];
        for (int pos = 0; pos < digits.length - 1; pos++)
           digits[pos] = digits[pos + 1];
        digits[digits.length - 1] = b;
     }
}
```

c) (3 points)

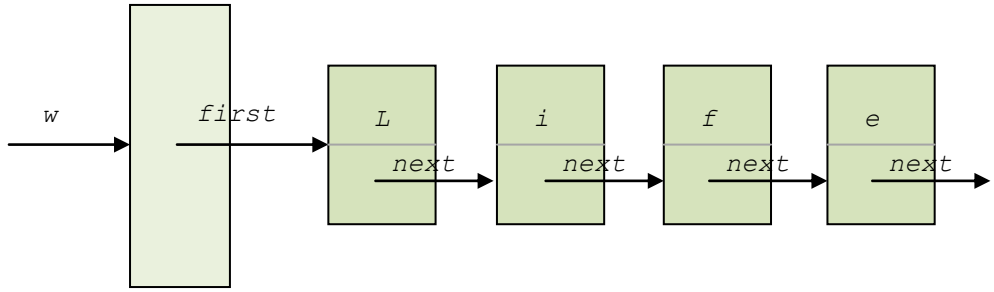
```
public void circularShiftRight ()
{
    if (digits.length > 0)
    {
        byte    b = digits[digits.length - 1];
        for (int pos = digits.length - 1; pos > 0; pos--)
            digits[pos] = digits[pos - 1];
        digits[0] = b;
    }
}
```

Task 4 (3 points + 3 points + 3 points)

a) (3points)

```
public Word (char[] v)
{
    if (v.length != 0)
    {
        first = new Node (v[0]);
        Node    n = first;
        for (int pos = 1; pos < v.length; pos++)
        {
            n.next = new Node (v[pos]);
            n = n.next;
        }
    }
}
```

b) (3 points)



c) (3 points)

[LightLove]

Task 5 (3 points + 3 points + 3 points)

a) (3points)

[5, 4, 3, 2, 1]

[1, 5, 4, 3, 2]
[1, 2, 5, 4, 3]
[1, 2, 3, 5, 4]
[1, 2, 3, 4, 5]

[1, 2, 3, 4, 5]

b) (3 points)

Preconditions:

A sequence, whose elements can be compared with the operator less ($<$), is given:

$x_1, x_2, x_3, \cdots, x_n$, where n is a positive integer

Postconditions:

The sequence is sorted in increasing order:

$x_{i1} < x_{i2} < x_{i3} < \cdots < x_{in}$

The steps in the algorithm:

```
sort (n, x[1], x[2], …, x[n])
{
```

```
    for pos = 1, 2, …, n - 1
    {
        for p = pos + 1, pos + 2, …, n
            if (x[p] < x[pos])
                exchange x[p] and x[pos]
    }
]
```

c) (3 points)

To fill the first position with the proper element, n – 1 element comparations are done. To fill the second position with the proper element, n – 2 element comparations are done. The number of comparations decreases with 1 for each new position. The total number of comparations is:

$$(n - 1) + (n - 2) + \ldots + 1 = n(n - 1)/2$$

The corresponding complexity function is:

$$t(n) = n(n - 1)/2$$

$$t(n) = n^2/2 - n/2$$

For large n the term with n^2 dominates. Therefore:

$$t(n) \in \theta(n^2)$$

The algorithm is quadratic in terms of element comparations.

When regarding the number of element exchanges, the worst case for the algorithm arises when the sequence is reversed sorted in the beginning. In such a case element exchange is done each time two elements are compared. There are as many element exchanges as element comparations, hence:

$$w(n) = n(n - 1)/2$$

$$w(n) \in \theta(n^2)$$