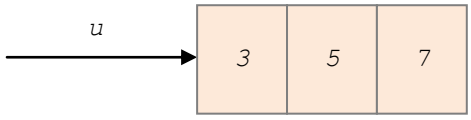
Exam: solution

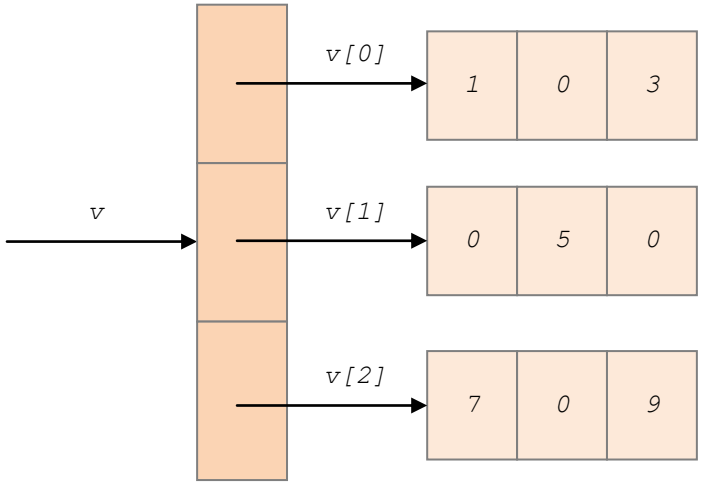
Tasks: solutions

Task 1 (3 points + 3 points)

a) (3 points)



b) (3 points)



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

a) (3 points)

```
public static RationalNumber totalSum (RationalNumber[] numbers)
{
    if (numbers.length == 0)
        throw new java.lang.IllegalArgumentException ("no numbers");

    RationalNumber    totalSum = numbers[0];
    for (int pos = 1; pos < numbers.length; pos++)
        totalSum = totalSum.sum (numbers[pos]);

    return totalSum;
}
```

b) (3 points)

```
public static RationalNumber maxNumber (RationalNumber[] numbers)
{
    if (numbers.length == 0)
        throw new java.lang.IllegalArgumentException ("no numbers");
```

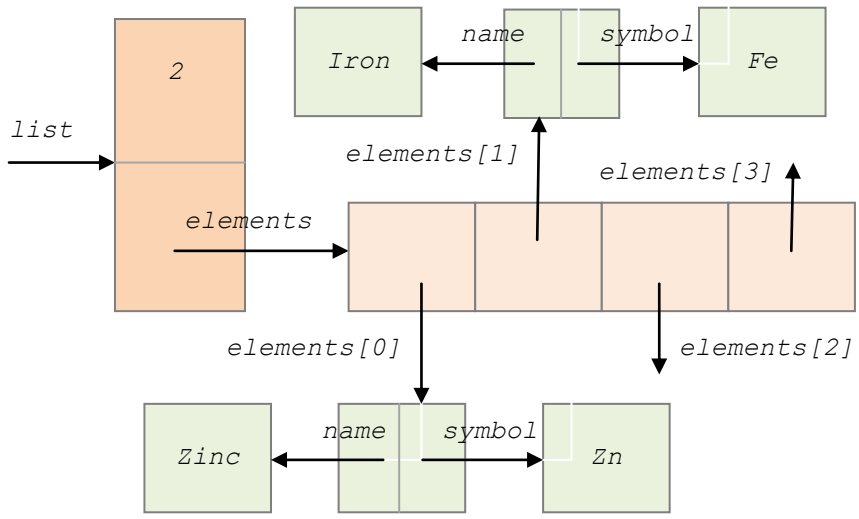
```
    RationalNumber    maxNumber = numbers[0];
    for (int pos = 1; pos < numbers.length; pos++)
        if (maxNumber.lessThan (numbers[pos]))
            maxNumber = numbers[pos];

    return maxNumber;
}
```

c) (3 points)

```
RationalNumber[]    numbers = { new RationalNumber (3, 40),
                                new RationalNumber (1, 3),
                                new RationalNumber (2, 3),
                                new RationalNumber (-1, 2) };
RationalNumber    sum = totalSum (numbers);
RationalNumber    max = maxNumber (numbers);
```

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

a) (3 points)



b) (3 points)

```
public void addFirst (Element element)
{
    if (numberOfElements == CAPACITY)
        throw new java.lang.IllegalStateException ("full list");

    for (int pos = numberOfElements; pos > 0; pos--)
        elements[pos] = elements[pos - 1];
    elements[0] = element;
    numberOfElements++;
}
```

c) (3 points)

```
public Element takeFirst ()
{
    if (numberOfElements == 0)
        throw new java.util.NoSuchElementException ("no elements");
```

```
    Element    element = elements[0];
    for (int pos = 0; pos < numberOfElements - 1; pos++)
        elements[pos] = elements[pos + 1];
    elements[numberOfElements - 1] = null;
    numberOfElements--;

    return element;
}
```

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

a) (3 points)

```
super (x, y);
this.name = name;
```

b) (3 points)

```
public String toString ()
{
    return name + " " + super.toString ();
}
```

c) (3 points)

```
class NameFormatException extends java.lang.IllegalArgumentException
{
    public NameFormatException (String message)
    {
        super (message);
    }
}
```

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

a) (3 points)

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

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

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

b) (3 points)

Precondition:

A sequence, the elements of which can be compared by the operator *greater* ($>$), is given:

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

Postcondition:

The sequence is sorted in ascending order:

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

Steps in the algorithm:

```
sort (n, x[1], x[2], …, x[n])
```

```
{
    for pos = n, n - 1, …, 3, 2
    {
        maxPos= 1;
        for p = 2, 3, …, pos
            if (x[p] > x[maxPos])
                maxPos = p;

        if maxPos != pos
            exchange x[maxPos] and x[pos]
    }
]
```

c) (3 points)

The algorithm fills position after position, from the last position to the second position, with the correct element. The greatest element in the unsorted portion of the sequence is determined, and it swaps places with the last element in that portion of the sequence. In the worst case the algorithm performs one element exchange each time a position is filled with the correct element (in the case that the right element is in another position). This means that in the worst case, there are $n - 1$ element exchanges. Thus:

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

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