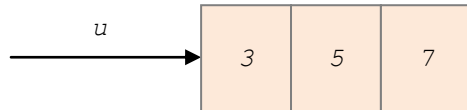


# Tentamen: lösning

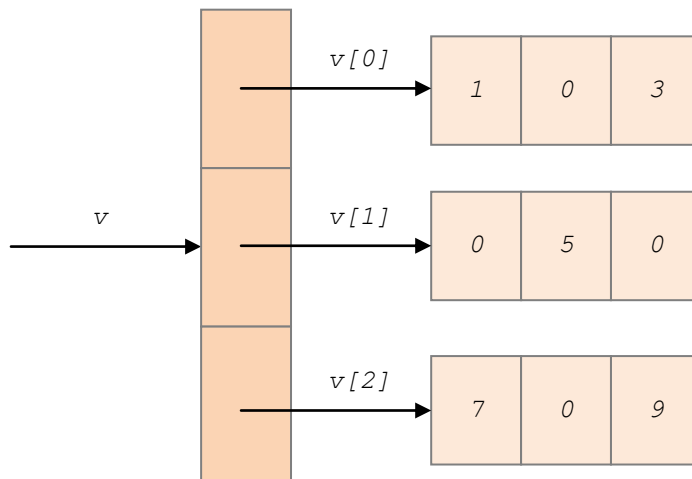
## Uppgifter: lösningar

### Uppgift 1 (3 poäng + 3 poäng)

a) (3 poäng)



b) (3 poäng)



### Uppgift 2 (3 poäng + 3 poäng + 3 poäng)

a) (3 poäng)

```
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 poäng)

```
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 poäng)

```

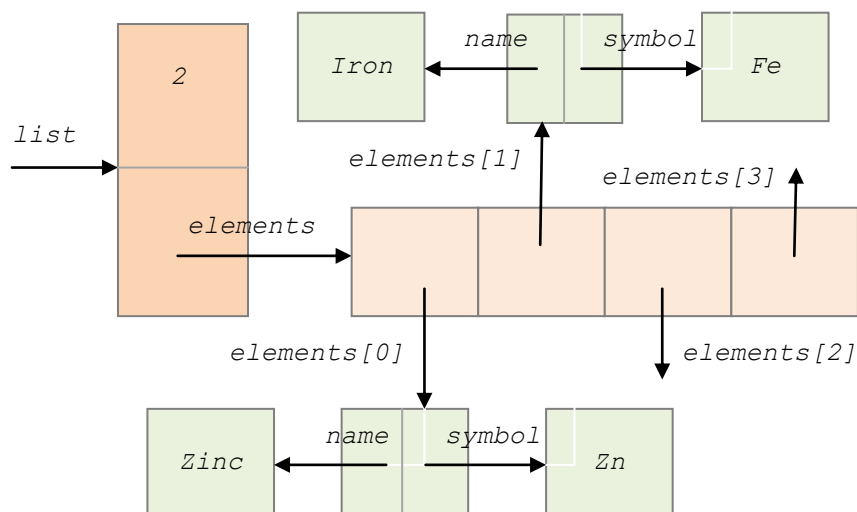
RationalNumber[]    numbers = { new RationalNumber (3, 40),
                                new RationalNumber (1, 3),
                                new RationalNumber (2, 3),
                                new RationalNumber (-1, 2) };

RationalNumber    sum = totalSum (numbers);
RationalNumber    max = maxNumber (numbers);

```

### Uppgift 3 (3 poäng + 3 poäng + 3 poäng)

a) (3 poäng)



b) (3 poäng)

```

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 poäng)

```

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;
}

```

## Uppgift 4 (3 poäng + 3 poäng + 3 poäng)

a) (3 poäng)

```

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

```

b) (3 poäng)

```

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

```

c) (3 poäng)

```

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

```

## Uppgift 5 (3 poäng + 3 poäng + 3 poäng)

a) (3 poäng)

[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 poäng)

Förvillkor:

En sekvens, vars element kan jämföras med operatören *större* (>), är given:

$x_1, x_2, x_3, \dots, x_n$ , där  $n$  är ett positivt heltal

Eftervillkor:

Sekvensen är sorterad i en stigande ordning:

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

Steg i algoritmen:

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 poäng)

Algoritmen fyller position efter position, från och med den sista positionen till och med den andra positionen, med rätt element. Det största elementet i den osorterade delen av sekvensen bestäms, och det byter plats med det sista elementet i den delen av sekvensen. I värsta fall utför algoritmen ett elementutbyte varje gång en position fylls med rätt element (i fall att rätt element ligger på en annan position). Det betyder att i värsta fall utförs  $n - 1$  elementutbyten. Alltså:

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

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