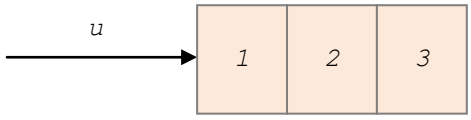
Tentamen: lösning

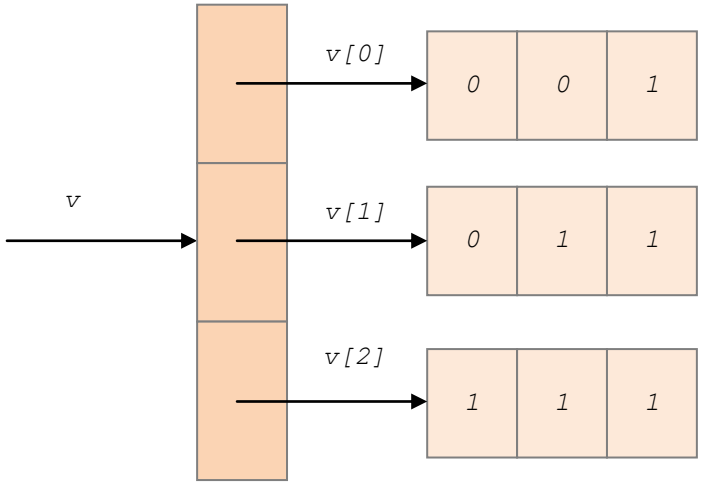
Uppgifter: lösningar

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 Point nearestPoint (Point[] points, Point point)
{
    if (points.length == 0)
        throw new java.lang.IllegalArgumentException ("no points");

    Point    nearestPoint = points[0];
    double   minDistance = nearestPoint.distance (point);
    double   distance = 0;
    for (int pos = 1; pos < points.length; pos++)
    {
        distance = points[pos].distance (point);
        if (distance < minDistance)
        {
            nearestPoint = points[pos];
            minDistance = distance;
        }
    }
```

```
    return nearestPoint;
}
```

b) (3 poäng)

```
public static Point[] internalPoints (Point[] points, double r)
{
    // antalet punkter inuti cirkeln
    int    countInternalPoints = 0;
    for (Point p : points)
        if (p.getX () * p.getX () + p.getY () * p.getY () < r * r)
            countInternalPoints++;

    // de punkter som är inuti cirkeln
    Point[]    internalPoints = new Point[countInternalPoints];
    int    pos = 0;
    for (Point p : points)
        if (p.getX () * p.getX () + p.getY () * p.getY () < r * r)
            internalPoints[pos++] = p;

    return internalPoints;
}
```

c) (3 poäng)

```
Point[]    points = { new Point (3, 4),
                      new Point (1, 2),
                      new Point (5, 6),
                      new Point (4, 5) };
Point    point = new Point (1, 1);
double    radius = 7;

Point    nearestPoint = nearestPoint (points, point);
Point[]    internalPoints = internalPoints (points, radius);
```

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

a) (3 poäng)

```
public String toString ()
{
    String    s = "";
    for (int pos = 0; pos < courseCount; pos++)
        s = s + courses[pos] + "\n";

     return s;
}
```

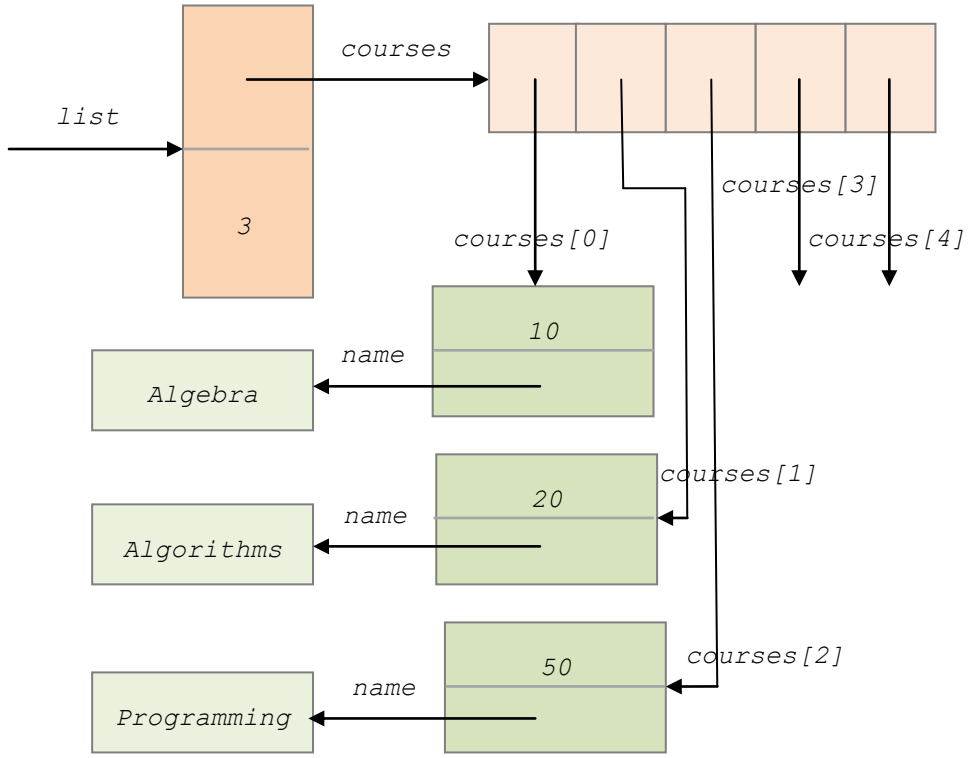
b) (3 poäng)

```
public void remove (Course course)
{
    int    courseIndex = -1;;
    for (int pos = 0; pos < courseCount; pos++)
        if (courses[pos].equals (course))
        {
            courseIndex = pos;
            break;
        }

    if (courseIndex != -1)
    {
        for (int pos = courseIndex; pos < courseCount - 1; pos++)
            courses[pos] = courses[pos + 1];
        courses[courseCount - 1] = null;
```

```
            courseCount--;
        }
    }
```

c) (3 poäng)



Uppgift 4 (4 poäng + 2 poäng + 3 poäng)

a) (4 poäng)

```
public int sizeCompare (Rectangle rec)
{
    int    compResult = 0;
    if (this.width * this.height < rec.width * rec.height)
        compResult = -1;
    else if  (this.width * this.height > rec.width * rec.height)
        compResult = 1;

    return compResult;
}

public int sizeCompare (CharSequence seq)
{
    int    compResult = 0;
    if (this.charCount < seq.charCount)
        compResult = -1;
    else if (this.charCount > seq.charCount)
        compResult = 1;

    return compResult;
}
```

b) (2 poäng)

```
Rectangle    rec1 = new Rectangle (4, 3);
```

```
Rectangle    rec2 = new Rectangle (6, 5);
Rectangle    rec = Selector.oneOfTwo (rec1, rec2);
```

c) (3 poäng)

1+2=3

Uppgift 5 (4 poäng + 5 poäng)

a) (4 poäng)

För att bestämma det element som ska ligga på den första positionen, utförs n – 1 elementjämförelser. För att bestämma det element som ska ligga på den andra positionen utförs n – 2 jämförelser. Antalet jämförelser minskar med 1 för varje ny position. Det totala antalet jämförelser är:

$$(n - 1) + (n - 2) + ... + 1 = n(n - 1)/2$$

Motsvarande komplexitetsfunktion är:

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

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

För stora n dominerar termen med n^2. Därför:

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

Algoritmen är kvadratisk när det gäller antalet elementjämförelser.

Man sätter n – 1 element på rätt plats, och därmed sorteras alla element. När ett element sätts in på rätt ställe, byter det plats med det element som redan finns där. Om ett element redan finns på rätt position, behöver det inte byta plats med något annat element. Det betyder att i värsta fall utförs n – 1 elementutbyten.

Algoritmens tidskomplexitet i värsta fall, när det gäller antalet elementutbyten, kan ges med följande komplexitetsfunktion:

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

$$w(n) \in \Theta(n)$$

Algoritmen är linjär när det gäller antalet elementutbyten i värsta fall.

c) (5 poäng)

```
public static void sort (String[] elements)
{
    int    lastPos = elements.length - 1;
    int    minPos = 0;
    String    e = "";
    for (int pos = 0; pos < lastPos; pos++)
    {
        minPos = pos;
        for (int p = pos + 1; p <= lastPos; p++)
            if (elements[p].compareTo (elements[minPos]) < 0)
                minPos = p;

        if (minPos != pos)
        {
            e = elements[pos];
            elements[pos] = elements[minPos];
            elements[minPos] = e;
        }
    }
}
```