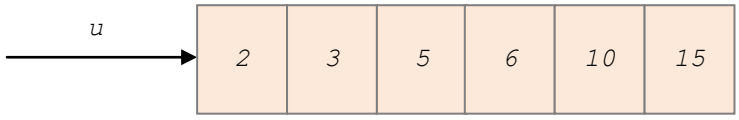
Tentamen: lösning

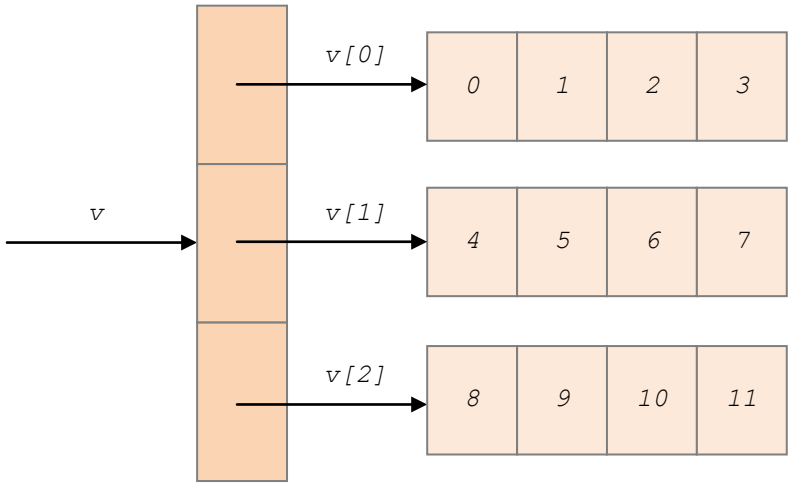
Uppgifter: lösningar

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 Car findCar (Car[] cars, String type, double maxPrice)
{
    if (cars.length == 0)
        throw new java.lang.IllegalArgumentException ("no cars");

    Car    car = null;
    for (int pos = 0; pos < cars.length; pos++)
        if (cars[pos].getType ().equals (type)  &&
            cars[pos].getPrice () <= maxPrice)
        {
            car = cars[pos];
            break;
        }

    return car;
}
```

b) (3 poäng)

```
public static Car[] selectCars (Car[] cars, String type)
{
    int    countCars = 0;
    for (Car car : cars)
        if (car.getType ().equals (type))
            countCars++;

    Car[]    selectedCars = new Car[countCars];
    int    pos = 0;
    for (Car car : cars)
        if (car.getType ().equals (type))
            selectedCars[pos++] = car;

    return selectedCars;
}
```

c) (3 poäng)

```
Car[]    cars = { new Car("Volvo", 180),
                  new Car("BMW", 220),
                  new Car("Mazda", 190),
                  new Car("Audi", 210),
                  new Car("BMW", 230),
                  new Car("Volkswagen", 180) };

Car      car = findCar (cars, "Mazda", 200);
Car[]    selectedCars = selectCars (cars, "BMW");
```

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

a) (3 poäng)

```
public String toString ()
{
    String    s = Math.abs (this.p) + "/" + Math.abs (this.q);
    if (this.p * this.q < 0)
        s = "-" + s;

    return s;
}
```

b) (3 poäng)

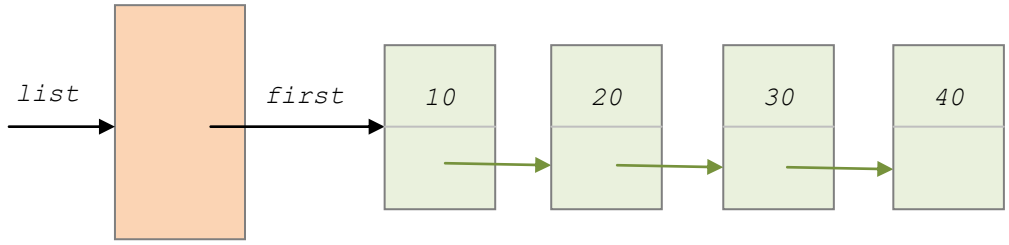
```
public boolean equals (RationalNumber r)
{
    return this.p * r.q == r.p * this.q;
}
```

c) (3 poäng)

```
public void add (RationalNumber r)
{
     this.p = this.p * r.q + r.p * this.q;
     this.q = this.q * r.q;
}
```

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

a) (3 poäng)



b) (3 poäng)

```
public int getLast () throws java.util.NoSuchElementException
{
    if (first == null)
        throw new java.util.NoSuchElementException ("empty list");

    Node    n = first;
    while (n.next != null)
        n = n.next;

    return n.number;
}
```

c) (3 poäng)

```
public void removeFirst () throws java.util.NoSuchElementException
{
    if (first == null)
        throw new java.util.NoSuchElementException ("empty list");

    first = first.next;
}
```

Uppgift 5 (5 poäng + 4 poäng)

a) (5 poäng)

```
[8, 2, 7, 1, 5, 6, 3, 4]

[2, 8, 7, 1, 5, 6, 3, 4]
[2, 7, 8, 1, 5, 6, 3, 4]
[1, 2, 7, 8, 5, 6, 3, 4]
[1, 2, 5, 7, 8, 6, 3, 4]
[1, 2, 5, 6, 7, 8, 3, 4]
[1, 2, 3, 5, 6, 7, 8, 4]
[1, 2, 3, 4, 5, 6, 7, 8]
```

b) (4 poäng)

Bästa fall uppstår när sekvensen med element är redan sorterad. I detta fall jämförs varje element, utom det första, med det element som finns framför det. Det blir precis en jämförelse per element, utom det första elementet. Totalt är det $n - 1$ jämförelser. Motsvarande komplexitetsfunktion är:

$$b(n) = n - 1$$

$$b(n) \in \Theta(n)$$

I bästa fall är algoritmen linjär när det gäller jämförelser.

Värsta fall uppstår när sekvensen med element är omvänt sorterad. I detta fall jämförs varje element, utom det första, med alla element framför det. Det totala antalet jämförelser är:

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

Det betyder att algoritmens tidskomplexitet i värsta fall, när det gäller jämförelser, kan ges med följande komplexitetsfunktion:

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

Denna funktion kan även skrivas så här:

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

Termen med n^2 dominerar för tillräckligt stora värden på n, och därför:

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

I värsta fall är algoritmen kvadratisk när det gäller jämförelser.