Handin 6

Exercise (121)**.**

Implement a generic sort function that can sort any list of objects that are comparable (meaning that they inherit from Comparable). Use either merge sort from exercise 56 or quicksort from week 7. Instead of comparing elements using '<', use the compareTo method

Solution. Generalizing the sort from integers to comparable classes, is a relatively simple task. First we replace all the parameters types of the functions `split, merge, length` and `mergeSort` by the general type `[T]`. In `merge` we call `compareTo`, since this is a function defined for *Comparable* objects, we require that `T` is inherited from `Comparable[T]`.

Exercise (122)**.**

Implement the filter and zip methods of Stream. (Some design choices to consider: Do your methods compute the head or the tail first in each step? Can you avoid computing the stream elements multiple times?) Then run the code and check that it works as expected. Also, explain how primes and fibs2 work.

Solution.

```
def filter(p: T => Boolean): Stream[T] = this match {
    case SNil => SNil
    case SCons(x, xs) =>
      val f = x()
      if (p(f)) SCons(() => f, () => xs().filter(p)) else xs().filter(p)
  }
```

```
def zip[U](ys: Stream[U]): Stream[(T, U)] = (this, ys) match {
    case (SNil, SNil) => SNil
    case (_, SNil) => SNil
    case (SNil, _) => SNil
    case (SCons(x, xi), SCons(y, yi)) => SCons(() => (x(), y()), () =>
        xi().zip(yi()))
  }
```


Exercise (127)**.**

The inverse of foldRight is unfoldRight. For streams, unfoldRight looks as follows:

```
def unfoldRight[A, S](z: S, f: S => Option[(A, S)]): Stream[A] =
```

```
  f(z) match {
    case Some((h, s)) => SCons(() => h, () => unfoldRight(s, f))
    case None => SNil
  }
```

The foldRight operation traverses a given stream, whereas unfoldRight can produce streams. Use unfoldRight to re-implement the streams ones, nats, and fibs more concisely. In each case, your solution should consist of a single call to unfoldRight, which then takes care of building the stream.

Solution.

```
val ones1: Stream[Int] = unfoldRight(1, (i: Int) => Some((i, i)))

val nats1: Stream[Int] = unfoldRight(0, (i: Int) => (Some(i, i+1)))

val fibs1: Stream[Int] = unfoldRight((1,1), (i: Int, j: Int) => Some((i+
    j, (i+j, i))))
```