Create a file called `hw7.hs` that contains definitions for the following functions. You must implement each of these functions "from scratch", i.e., only using basic Haskell operations and not functions already defined in Haskell that do the same thing. Also, you should use what we have discussed in class so far as opposed to using techniques we haven’t looked at yet during lectures.

On the due date, hand in a cover sheet together with hard copy of your implementation file, print outs of tests showing your functions work properly, and a “design” document discussing challenges and/or issues you had in using Haskell for this assignment. In addition, submit your source file to the online dropoff site (https://www.cs.gonzaga.edu/dropoff/). Note that in your hard copy, you must provide a print out showing that you sufficiently tested your functions. Included with your tests, use Haskell to determine the type of each of your functions. Here is an example of what your print out should look like:

```
GHCi, version 7.6.3: http://www.haskell.org/ghc/  :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.

Prelude> :l hw7.hs
[1 of 1] Compiling Main ( hw7.hs, interpreted )
Ok, modules loaded: Main.

*Main> :t unordered3
unordered3 :: Ord a => a -> a -> a -> Bool

*Main> unordered3 1 3 2
True

*Main> unordered3 1 2 1
True

*Main> unordered3 1 2 3
False
```

1. Write a function `unordered3` that returns true if its three arguments are not in ascending or descending order (i.e., not ordered) and false otherwise. E.g., `unordered3 1 3 2` and `unordered3 1 2 1` should return true, whereas `unordered3 1 2 3`, `unordered3 3 2 1` and `unordered3 1 1 1` should all return false.

2. Write a function `distance` that returns the distance between two points. E.g., `distance (0,0) (2,0)` computes the distance between points (0, 0) and (2, 0), which is 2.0, and `distance (-2,1) (1,5)` computes the distance between points (-2,1) and (1,5), which is 5.0. Hint: the `sqrt` function included with Haskell returns the square root.
3. Write a function `midpoint` that returns the midpoint between two points. E.g., `midpoint (0,0) (2,0)` is `(1.0,0)` and `midpoint (-2,1) (1,5)` is `(-0.5,3.0)`.

4. Write a function `addToEnd` that takes a value `x` and a list of values `ys` and gives a new list with `x` added to the end of `ys`. E.g., `addToEnd 5 [3,6]` should give `[3,6,5]`.

5. Write a function `addFirstToEnd` that takes a list `xs` and returns a new list with the first value of `xs` as the last value of the new list. E.g., `addFirstToEnd [3,6,5]` should give `[6,5,3]`. Your function must call `addToEnd`. Note that `addFirstToEnd []` is undefined.

6. Write a function `addSecondToEnd` that takes a list `xs` and returns a new list with the second value of `xs` as the last value of the new list. E.g., `addSecondToEnd [3,6,5]` should give `[3,5,6]`. Your function must call `addFirstToEnd`. Note that `addSecondToEnd xs` is undefined when `xs` has fewer than 2 elements.

7. Write a function `median3` that returns the middle value of three values. For example `median3 5 2 3` should return 3. If there are not three unique values, return the value that occurs most frequently. For example, `median3 1 1 2` should return 1.

8. Write a function `quadrant` that takes a point (a pair) and returns the corresponding quadrant where the point is located (for a definition of quadrant, see http://en.wikipedia.org/wiki/Quadrant_(plane_geometry)). For example, `quadrant (3,5)` should return 1, whereas `quadrant (-3,5)` should return 2. If the x or y value of the point is zero, then `quadrant (x,y)` should return 0.

9. Write a function called `lastElem` that returns the last value of a given list. Use the functions `drop`, `head`, and `length` to implement `lastElem`. Note that `lastElem []` is undefined.

10. Assume a date is represented using a 3-tuple `(m,d,y)` where `m` is the month, `d` is the day, and `y` is the year. Write a Haskell function `earlierDate` that takes two dates and returns true if first date occurs before the second date. For example, `earlierDate (4,3,2000) (3,4,2000)` should return false (where the first date represents April 3, 2000 and the second date represents March 4, 2000).