```
;;
;; ***************************************************
;; Sample Solutions
;; CS 135 Fall 2019
;; Assignment 03, Problem 2
;; ***************************************************
;;


;;(pnormp p v) produces the p-norm-p of v
;; pnormp: Num (listof Num) -> Num
;; requires: v has at most 3 elements
;;           p > 0
;; Examples:
(check-expect (pnormp 1 empty) 0)
(check-expect (pnormp 3 (cons 3 (cons -4 (cons 5 empty)))) 216)

(define (pnormp p v)
  (cond
    [(empty? v) 0]
    [(empty? (rest v)) (expt (abs (first v)) p)]
    [(empty? (rest (rest v))) (+ (expt (abs (first v)) p) (expt (abs (second v)) p))]
    [(empty? (rest (rest (rest v))))
     (+ (expt (abs (first v)) p) (expt (abs (second v)) p) (expt (abs (third v)) p))]))

;; Tests:
(check-expect (pnormp 1 (cons -2 empty)) 2)
(check-expect (pnormp 2 (cons 2 (cons -3 empty))) 13)



;;
;; ***************************************************
;; Sample Solutions
;; CS 135 Fall 2019
;; Assignment 03, Problem 3
;; ***************************************************
;;


;; 3a)

;;(in-order? str1 str2 str3) determines if (str1, str2, str3) is lexicographically sorted
;; in-order?: Str Str Str -> Bool
;; Examples:
(check-expect (in-order? "a" "b" "c") true)
(check-expect (in-order? "b" "a" "b") false)

(define (in-order? str1 str2 str3)
  (and (string<=? str1 str2) (string<=? str2 str3)))

;; Tests:
(check-expect (in-order? "a" "a" "c") true)
(check-expect (in-order? "a" "c" "c") true)
(check-expect (in-order? "a" "c" "b") false)
(check-expect (in-order? "a" "a" "a") true)
```

```
;; 3b)

;;(sort3 los) produces a lexicographically sorted list of the elements in los
;; sort3: (listof Str) -> (listof Str)
;; requires: los contains exactly 3 strings
;; Examples:
(check-expect (sort3 (cons "a" (cons "b" (cons "c" empty))))
              (cons "a" (cons "b" (cons "c" empty))))
(check-expect (sort3 (cons "c" (cons "b" (cons "a" empty))))
              (cons "a" (cons "b" (cons "c" empty))))

(define (sort3 los)
  (cond
    [(in-order? (first los) (second los) (third los)) los]
    [(in-order? (first los) (third los) (second los))
     (cons (first los) (cons (third los) (cons (second los) empty)))]
    [(in-order? (second los) (first los) (third los))
     (cons (second los) (cons (first los) (cons (third los) empty)))]
    [(in-order? (second los) (third los) (first los))
     (cons (second los) (cons (third los) (cons (first los) empty)))]
    [(in-order? (third los) (first los) (second los))
     (cons (third los) (cons (first los) (cons (second los) empty)))]
    [(in-order? (third los) (second los) (first los))
     (cons (third los) (cons (second los) (cons (first los) empty)))]))

;; Tests:
(check-expect (sort3 (cons "a" (cons "c" (cons "b" empty))))
              (cons "a" (cons "b" (cons "c" empty))))
(check-expect (sort3 (cons "c" (cons "a" (cons "b" empty))))
              (cons "a" (cons "b" (cons "c" empty))))
(check-expect (sort3 (cons "b" (cons "c" (cons "a" empty))))
              cons "a" (cons "b" (cons "c" empty))))
(check-expect (sort3 (cons "b" (cons "a" (cons "c" empty))))
              (cons "a" (cons "b" (cons "c" empty))))
(check-expect (sort3 (cons "a" (cons "a" (cons "b" empty))))
              (cons "a" (cons "a" (cons "b" empty))))
(check-expect (sort3 (cons "b" (cons "a" (cons "a" empty))))
              (cons "a" (cons "a" (cons "b" empty))))
(check-expect (sort3 (cons "a" (cons "b" (cons "a" empty))))
              (cons "a" (cons "a" (cons "b" empty))))
(check-expect (sort3 (cons "a" (cons "a" (cons "a" empty))))
              (cons "a" (cons "a" (cons "a" empty))))


;; 3c)

;;(find-second los) produces the second lexicographically largest string found in los
;;   or empty if all strings are equal
;; find-second: (listof Str) -> Str
;; requires: los contains exactly 3 strings
;; Examples:
(check-expect (find-second (cons "c" (cons "a" (cons "b" empty)))) "b")
(check-expect (find-second (cons "a" (cons "a" (cons "a" empty)))) empty)

(define (find-second los)
  (cond
    [(and (string=? (first los) (second los)) (string=? (second los) (third los))) empty]
    [(string=? (second (sort3 los)) (third (sort3 los))) (first (sort3 los))]
    [(string=? (first (sort3 los)) (second (sort3 los))) (first (sort3 los))]
    [else (second (sort3 los))]))

;; Tests:
(check-expect (find-second (cons "a" (cons "b" (cons "c" empty)))) "b")
(check-expect (find-second (cons "c" (cons "a" (cons "a" empty)))) "a")
(check-expect (find-second (cons "c" (cons "c" (cons "a" empty)))) "a")



;;
;; ***************************************************
;; Sample Solutions
;; CS 135 Fall 2019
;; Assignment 03, Problem 4
;; ***************************************************
;;


;; 4a)

;;(make-point x y) produces list of the coordinates (x, y)
;; make-point: Num Num -> (listof Num)
;; Examples:
(check-expect (make-point 0 0) (cons 0 (cons 0 empty)))
(check-expect (make-point 3/4 4/3) (cons 3/4 (cons 4/3 empty)))

(define (make-point x y) (cons x (cons y empty)))

;; Test:
(check-expect (make-point -1 0.5) (cons -1 (cons 0.5 empty)))


;;(x-coord point) produces the x coordinate of point
;; x-coord: (listof Num) -> Num
;; requires: point contains exactly 2 numbers
;; Examples:
(check-expect (x-coord (cons 0 (cons 0 empty))) 0)
(check-expect (x-coord (cons 3 (cons 7 empty))) 3)

(define (x-coord point) (first point))

;; Test:
(check-expect (x-coord (make-point -1 0.5)) -1)


;;(y-coord point) produces the y coordinate of point
;; y-coord: (listof Num) -> Num
;; requires: point contains exactly 2 numbers
;; Examples:
(check-expect (y-coord (cons 0 (cons 0 empty))) 0)
(check-expect (y-coord (cons 3 (cons 8 empty))) 8)

(define (y-coord point) (second point))

;; Test:
(check-expect (y-coord (make-point -1 0.5)) 0.5)
```

```racket
(define p (make-point 3 2))
(define q (make-point 7 3))
(define r (make-point 5 1))
(define s (make-point 8 6))


;; 4b)

;;(on-line-between p1 p2 x) determines the y coordinate of the point (x, y)
;;  on the line passing through p1 and p2
;; on-line-between: (listof Num) (listof Num) Num -> Num
;; requires: p1 and p2 have exactly 2 elements
;; Examples:
(check-expect (on-line-between (make-point 0 0) (make-point 3 2) 1.5) 1)
(check-expect (on-line-between (make-point 0 5) (make-point 10 5) 3) 5)

(define (on-line-between p1 p2 x)
  (+ (y-coord p1)
     (* (/ (- (y-coord p2) (y-coord p1))
           (- (x-coord p2) (x-coord p1)))
        (- x (x-coord p1)))))


(define spline-start (make-point 0 0))
(define spline-end (make-point 10 0))


;;(below-spline? p1 p2 point) determines if point lies between
;;   the spline passing through p1 and p2, and the x-axis
;; below-spline?: (listof Num) (listof Num) (listof Num) -> Bool
;; requires: p1, p2, and point each contain exactly 2 numbers
;;           (x-coord spline-start) < (x-coord p1) < (x-coord p2) < (x-coord spline-end)
;;           (y-coord p1), (y-coord p2) > (y-coord spline-start)
;; Examples:
(check-expect (below-spline? p q (make-point -1 2)) false)
(check-expect (below-spline? p q (make-point 1 1)) false)

(define (below-spline? p1 p2 point)
  (cond
    [(or (< (x-coord point) (x-coord spline-start))
         (> (x-coord point) (x-coord spline-end))
         (< (y-coord point) (y-coord spline-start))) false]
    [(< (x-coord point) (x-coord p1)) (<= (y-coord point)
                                          (on-line-between spline-start p1 (x-coord point)))]
    [(< (x-coord point) (x-coord p2)) (<= (y-coord point)
                                          (on-line-between p1 p2 (x-coord point)))]
    [else (<= (y-coord point) (on-line-between p2 spline-end (x-coord point)))]))

;; Tests:
(check-expect (below-spline? r s (make-point -10 5)) false)
(check-expect (below-spline? r s (make-point 20 5)) false)
(check-expect (below-spline? r s spline-start) true)
(check-expect (below-spline? r s r) true)
(check-expect (below-spline? r s s) true)
(check-expect (below-spline? r s spline-end) true)
(check-expect (below-spline? r s (make-point 4 -10)) false)
(check-expect (below-spline? r s (make-point 4 0)) true)
```

```
(check-expect (below-spline? r s (make-point 1 0.1)) true)
(check-expect (below-spline? r s (make-point 1 0.2)) true)
(check-expect (below-spline? r s (make-point 1 0.3)) false)
(check-expect (below-spline? r s (make-point 6 2)) true)
(check-expect (below-spline? r s (make-point 6 8/3)) true)
(check-expect (below-spline? r s (make-point 6 3)) false)
(check-expect (below-spline? r s (make-point 9 1)) true)
(check-expect (below-spline? r s (make-point 9 3)) true)
(check-expect (below-spline? r s (make-point 9 4)) false)



;;
;; ***************************************************
;; Sample Solutions
;; CS 135 Fall 2019
;; Assignment 03, Problem 5
;; ***************************************************
;;

;; 5a)

;; *************** start of copied functions from 4a) ****************

;;(make-point x y) produces list of the coordinates (x, y)
;; make-point: Num Num -> (listof Num)
;; Examples:
(check-expect (make-point 0 0) (cons 0 (cons 0 empty)))
(check-expect (make-point 3/4 4/3) (cons 3/4 (cons 4/3 empty)))

(define (make-point x y) (cons x (cons y empty)))

;; Test:
(check-expect (make-point -1 0.5) (cons -1 (cons 0.5 empty)))


;;(x-coord point) produces the x coordinate of point
;; x-coord: (listof Num) -> Num
;; requires: point contains exactly 2 numbers
;; Examples:
(check-expect (x-coord (cons 0 (cons 0 empty))) 0)
(check-expect (x-coord (cons 3 (cons 7 empty))) 3)

(define (x-coord point) (first point))

;; Test:
(check-expect (x-coord (make-point -1 0.5)) -1)


;;(y-coord point) produces the y coordinate of point
;; y-coord: (listof Num) -> Num
;; requires: point contains exactly 2 numbers
;; Examples:
(check-expect (y-coord (cons 0 (cons 0 empty))) 0)
(check-expect (y-coord (cons 3 (cons 8 empty))) 8)

(define (y-coord point) (second point))
```

```
;; Test:
(check-expect (y-coord (make-point -1 0.5)) 0.5)

;; *************** end of copied functions from 4a) ****************


;;(make-step start dir dist) produces the point resulting from
;;  moving dist steps away from start in dir
;; make-step: (listof Num) (anyof 'N 'E 'S 'W) Num -> (listof Num)
;; requires: list p to be a list of two numbers
;;           dist > 0
;; Examples:
(check-expect (make-step (make-point 1 2) 'N 3) (make-point 1 5))
(check-expect (make-step (make-point 1 2) 'S 3) (make-point 1 -1))

(define (make-step start dir dist)
  (cond
    [(symbol=? dir 'N) (make-point (x-coord start) (+ (y-coord start) dist))]
    [(symbol=? dir 'S) (make-point (x-coord start) (- (y-coord start) dist))]
    [(symbol=? dir 'E) (make-point (+ (x-coord start) dist) (y-coord start))]
    [(symbol=? dir 'W) (make-point (- (x-coord start) dist) (y-coord start))]))

;; Tests:
(check-expect (make-step (make-point 1 2) 'E 3) (make-point 4 2))
(check-expect (make-step (make-point 1 2) 'W 3) (make-point -2 2))


;; 5b)

;;(two-steps start directions distances) produces the point resulting from
;;  moving a series of distances steps away from start in the corresponding directions
;; two-steps: (listof Num) (listof (anyof 'N 'E 'S 'W)) (listof Num) -> (listof Num)
;; requires: lists pos, directions and distances to have length 2
;;           each num in distances > 0
;; Examples:
(check-expect (two-steps (make-point 1 -1)
                         (cons 'N (cons 'E empty))
                         (cons 2 (cons 3 empty))) (make-point 4 1))
(check-expect (two-steps (make-point 3 3)
                         (cons 'E (cons 'S empty))
                         (cons 4 (cons 1 empty))) (make-point 7 2))

(define (two-steps start directions distances)
  (make-step (make-step start (first directions) (first distances))
             (second directions) (second distances)))

;; Tests:
(check-expect (two-steps (make-point 3 3)
                         (cons 'S (cons 'S empty))
                         (cons 4 (cons 1 empty))) (make-point 3 -2))
(check-expect (two-steps (make-point 3 3)
                         (cons 'E (cons 'W empty))
                         (cons 4 (cons 4 empty))) (make-point 3 3))
```