CMSC 15100 Introduction to Computer Science I

Homework 8

Your answers should be saved in a .rkt file and submitted via Canvas before the start of the next lecture. At the top of each file for this assignment and all future assignments include a comment,

```
;; Homework 8
;;
;; your name
;; your CNET id
;; your collaborators
```

For this homework, some parts require you to explain things in English. For these questions, write your answers in a Racket comment box, which you can create by clicking on the "Insert" tab at the top of DrRacket.

If your code doesn't compile, it may not be graded. Please double check to make sure your code runs before submitting!

Problems

1. (8 points) The `Grocery` type is defined by

   ```
   (define-struct Grocery
     ([name : String]
      [days-old : Integer]))
   ```

   (a) Unfortunately, groceries do start to go bad and get smelly after some time. Here is a function `count-old` that counts how many grocery items are older than the given number of days `date`,

   ```
   (: count-old : (Listof Grocery) Integer -> Integer)
   (define (count-old items date)
     (cond
       [(empty? items) 0]
       [else (+
              (count-old (rest items) date)
              (if (> (Grocery-days-old (first items)) date) 1 0))]))
   ```

   Write a runtime recurrence for the runtime $T(n)$ of `count-old` on a list of size n, and explain it. Then solve the recurrence. Use big O notation.

   (b) Rewrite the function `count-old` without recursion by using a lambda function.

   Analyze the runtime of your `count-old` function using big O notation. Explain your answer.

2. (17 points) There are tons of different algorithms for sorting. A few you may encounter in the future: bubble sort, quicksort, bogo sort, heap sort, radix sort. Here you will implement another natural recursive sorting strategy, known as selection sort. The underlying idea for

selection sort is simple: to sort a list, pull out the smallest element, then recursively sort the remaining part, and finally put the smallest element in front.

(a) Make a helper function `find-minimum` that takes a `(Listof Integer)` and outputs the smallest number in the list.

Hint: you can do this in one line with `foldr`

(b) Make a helper function `remove-elem` that takes a `(Listof Integer)` and an `Integer` and returns the list with the given integer removed. Please don't use any built-in to do this exercise. Your algorithm should be recursive.

(c) Use your helper functions to write the `selection-sort` function.

(d) Write down a runtime recurrence for any of the above functions that are recursive, and explain why the different parts of the runtime recurrence are there. On a list of length n, what is the runtime of `find-minimum` and `remove-elem`? What is the runtime of `selection-sort`? Solve the recurrences using the tree method, or by citing a function with the same runtime recurrence from class. Explain your answers fully, but use big O notation to avoid being too tedious with the counting.

If you used `foldr` in part (a), you may use that `(foldr f base my-list)` has runtime $O(n \cdot (\text{runtime of } f))$ when `my-list` has length n.

(e) Do an informal speed test of your algorithm by putting the following lines of code at the end of your file. This code constructs a random list of length n and sorts it.

```
(: n : Integer)
(define n 100)

"Building list"

(: big-list : (Listof Integer))
(define big-list (build-list n (lambda ([i : Integer]) (random n))))

"Sorting list"

(empty? (selection-sort big-list))

"Finished sorting"
```

When you run your code, you should see "Building list", "Sorting list", "Finished sorting" appear in the interaction window. Increase the value of n. Approximately how big can you make n before it takes more than 10 seconds between "Sorting list" and "Finished sorting"?

(f) Now replace the call to `selection-sort` with a call to Racket's built-in `sort` function (and add the comparison function argument, because Racket's sort function is polymorphic),

```
(empty? (sort big-list <))
```

How large of a list can you sort now? Feel free to increase your memory limit – this may make your computer slower, but it's only temporary. After the code finishes running, your computer will start to reclaim memory from DrRacket, and it will get faster again.