Sorting

CS 151: Introduction to Computer Science I

Precise counting of operations

The *runtime* of a program is the number of operations it does.

Operations include: +, -, *, /, if, checking a `cond` case, performing a pattern match, substituting a value for a variable, etc

This definition is purposefully a bit unclear.
5 vs 6 not as important as 5,000 vs 500,000

Precise counting of operations

```
(* 2 (if (> x 5) (- x 1) (+ x 1)))
```

```
(match country
  ["United States" 325000000]
  ["Mexico" 127500000]
  ["Canada" 36000000]
  ...
```

```
(define (fact n)
  (cond
    [(= n 1) 1]
    [else (* n (fact (- n 1)))]))

(fact 5)
```

last

`first` gets the first element of a list. How about the last element?

```
(:  last :  (Listof Number) -> Number)
(define (last my-list)
  (cond
    [(empty?  my-list) (error "last:  empty list")]
    [(empty?  (rest my-list)) (first my-list)]
    [else (last (rest my-list))]))
```

Runtime: at most 10 operations per iteration, so at most $10n$ total on a list of length n

Big O notation

Formal definition: for $T(n)$ and $f(n)$ mathematical functions, T is $O(f)$ if there are constants C and N such

$$T(n) \leq C \cdot f(n)$$

for every $n \geq N$.

The following are $O(n)$:

$$n, 2n + 10, n - \sqrt{n} + \sqrt{\log_2 n}, 1000n$$

The following are not $O(n)$:

$$5n^2 - 10n + 15, 2n \log_2 n$$

Big O notation: length

Runtime of list length function

```
(:  length :  (Listof Number) -> Integer)
(define (length my-list)
  (cond
    [(empty?  my-list) 0]
    [else (+ 1 (length (rest my-list)))]))
```

$O(n)$ runtime on input of length n

bad-last

`first` gets the first element of a list. How about the last element?

```
(:  bad-last :  (Listof Number) -> Number)
(define (bad-last my-list)
  (cond
    [(empty?  my-list) (error "last:  empty list")]
    [(= (length my-list) 1) (first my-list)]
    [else (bad-last (rest my-list))]))
```

Runtime is $O(n^2)$

map and filter

```
(map add-one my-list)
```

```
(filter odd?  my-list)
```

Both have runtime $O(n)$

Find if all elements in a list are unique:

```
(andmap (occurs-once?  my-list) my-list)
```

Runtime is $O(n^2)$

Lab Attendance

When writing down students who attended lab, I want to know, how many students came to lab? However, sometimes I make a mistake and write someone twice.

```
(: lab : (Listof String) -> Integer)
(define (lab students)
  (if (not (unique? students))
      (error "lab: wrote down someone twice")
      (length students)))
```

Runtime is $O(n^2)$

Runtime Recurrences

```
(:  last :  (Listof Number) -> Number)
(define (last my-list)
  (cond
    [(empty?  my-list) (error "last:  empty list")]
    [(empty?  (rest my-list)) (first my-list)]
    [else (last (rest my-list))]))
```

Rutime recurrence:

$$T(n) = T(n-1) + 10$$

$$T(n) = T(n-1) + O(1)$$

Runtime Recurrences

Recursive functions give rise to recursive runtimes:

$$T(n) = T(n-1) + O(n)$$

$$T(n) = 2T(n-1) + O(1)$$

$$T(n) = T(n/2) + O(1)$$

Sorting Numbers

Common task: given a list of numbers, sort them

Sort a list of prices, sort a list of heights, sort `Activitys` in a `Calendar` by time of occurrence, ...

```
(:  sort :  (Listof Number) -> (Listof Number))
```

Insertion Sort

```
(define (insertion-sort my-list)
  (cond
    [(empty?  my-list) empty]
    [else (insert (first my-list)
                  (insertion-sort (rest my-list)))]))
(:  insert :  Number (Listof Number) -> (Listof
Number))
;; assumes my-list is sorted
(define (insert x my-list)
  (cond
    [(empty?  my-list) (list x)]
    [else (if (< x (first my-list))
              (cons x my-list)
              (cons (first-my-list) (insert x (rest
my-list))))]))
```

Merge Sort

A *divide-and-conquer* algorithm for sorting:

```
(:  merge-sort :  (Listof Number) -> (Listof
Number))
(define (merge-sort my-list)
  (cond
    [(empty?  my-list) empty]
    [else (merge (merge-sort (first-half my-list))
(merge-sort (second-half my-list)))]))
```

$$T(n) = 2T(n/2) + O(n)$$

Solution to this recurrence: $T(n) = O(n \log_2 n)$

What to know

- Big O notation, formal definition
- How to analyze the runtime of a piece of code
- Sorting: insertion, merge sort