Problem Specification

Problem Specification

Read and parse an XML document which contains stock performance information. Using this stored information read in another document which analyses the previous stock information. This stock information once analyzed is output into an HTML file which utilized Google's Visualization line chart API. The graph shows two lines the first being the day to day closing price information while the second is the line of linear regression.

Example Stock Information from Hist.txt:

```
<sr>
      <tk>AA</tk>
      <td>20090821</td>
      <hp>12.73</hp>
      <lp>12.49</lp>
      <op>12.64</op>
      <cp>12.56</cp>
      <tr>338295</tr>
</sr>
<sr>
      <tk>AA</tk>
      <td>20090824</td>
      <hp>12.83</hp>
      <lp>12.36</lp>
      <op>12.76</op>
      <cp>12.42</cp>
      <tr>307627</tr>
</sr>
<sr>
      <tk>AA</tk>
      <td>20090825</td>
      <hp>12.66</hp>
      <lp>12.3</lp>
      <op>12.57</op>
      <cp>12.35</cp>
      <tr>246836</tr>
</sr>
```

Stock Record	<SR>	- record of stock prices and shares traded on a specified day for a specified stock	<SR>*stock record*</SR>
Ticker	<TK>	- unique code comprised of alphabetic letters, digits, and/or periods - designates a corporation whose stock is traded on the market	<OP>*decimal numera*l</OP>
Opening Price	<OP>	- price of a stock at the time the market opens on a trading day	<OP>*decimal numera*l</OP>
Highest Price	<HP>	- highest price for which a stock sold on a trading day	<HP>*decimal numera*l</HP>
Lowest Price	<LP>	- lowest price for which a stock sold on a trading day	<LP>*decimal numera*l</LP>
Closing Price	<CP>	- price of a stock at the time the market closes on a trading day	<CP>*decimal numera*l</CP>
Shares Traded	<TR>	- number of shares of a stock traded on a trading day	<TR>*decimal numera*l</TR>
Analysis Request	<AR>	- request for analysis chart	<AR>*analysis request*</AR>
Trading Date	<TD>	- date of stock prices	<TD>*YYYYMMDD*</TD>
Starting Date	<SD>	- first day of analysis period	<SD>*YYYYMMDD*</SD>
Ending Date	<ED>	- last day analysis period	<ED>*YYYYMMDD*</ED>

Analysis File Input Example

Ticker, Start-date,End-date

```
<AR>ADI,20090821,20100820</AR>
<AR>ADP,20090821,20100820</AR>
<AR>ADSK,20090821,20100820</AR>
<AR>AKAM,20090821,20100820</AR>
<AR>ALTR,20090821,20100820</AR>
<AR>AMD,20090821,20100820</AR>
<AR>AMZN,20090821,20100820</AR>
<AR>ANF,20090821,20100820</AR>
<AR>APA,20090821,20100820</AR>
<AR>AVP,20090821,20100820</AR>
<AR>AZO,20090821,20100820</AR>
<AR>BBBY,20090821,20100820</AR>
<AR>BBY,20090821,20100820</AR>
<AR>BDK,20090821,20100820</AR>
<AR>BMC,20090821,20100820</AR>
<AR>BSX,20090821,20100820</AR>
<AR>CBS,20090821,20100820</AR>
<AR>CCE,20090821,20100820</AR>
<AR>CL,20090821,20100820</AR>
<AR>CLX,20090821,20100820</AR>
<AR>CMCSA,20090821,20100820</AR>
<AR>COl,20090821,20100820</AR>
<AR>CPWR,20090821,20100820</AR>
<AR>CSC,20090821,20100820</AR>
<AR>CSCO,20090821,20100820</AR>
<AR>DELL,20090821,20100820</AR>
<AR>DIS,20090821,20100820</AR>
<AR>DPS,20090821,20100820</AR>
<AR>DTV,20090821,20100820</AR>
```

Team Design

Design Steps

1. **Read in file =>hist.txt**

   a. Typical function to read file as string

2. **Parsing "hist.txt"**

   a. Remove newlines/spaces

   b. Packets on <SR>
   This will deliver us a list of each all of the stock tickers in the file in list form

   c. Packets_set </TK>, </RT> ….
   This will provide each of the tags within the list generated by step b.

   d. Create tuples on Needed information
   Pull just the information within the tags <TK>, <CP>, <TD> such data structure will look as the following:

   ((TK,CP,TD),(TK,CP,TD),(TK,CP,TD))
   where each tag represents the actual data parsed from the file

   Example: **((GOOG,100,20121211),(AMEX,200,20121130)).......**

3. **Build Tree**

   a. Use all the data created in 2c to build our tree. This tree will input data keyed off of the name of the stock ticker which each node of the tree will have the following data structure

   (TK, (Subtree TD, CP))

   Example on AMEX: **("AMEX" ,((20121031,500),(20121101,501)))**

4. **Read in Request File =>request.txt**

   a. Parse this file Accordingly return list of lists
   ((ticker,start_date,end_date),(ticker,start_date,end_date))

5. **Prune Tree according to list given from request.txt**

   a. This will grab all of the dates between the start and end date.

   b. Increasing on the dates by one to check the tree for the correct information this is assuming that all days are 31 the function **check-date** checks whether the date is 31 and if so increase month or year appropriately.

   c. Important note the information for dates that are not read will be input when the tree is pruned.

6. **Calculate Graph Lines**

   a. *Graph line 1*: The first line shows the day-by-day total closing price of the stocks in the group. If some of the days in the period are missing for a particular stock, assume that its closing price on that day is the same as it was on the next previous or next following day for which a closing price is recorded.

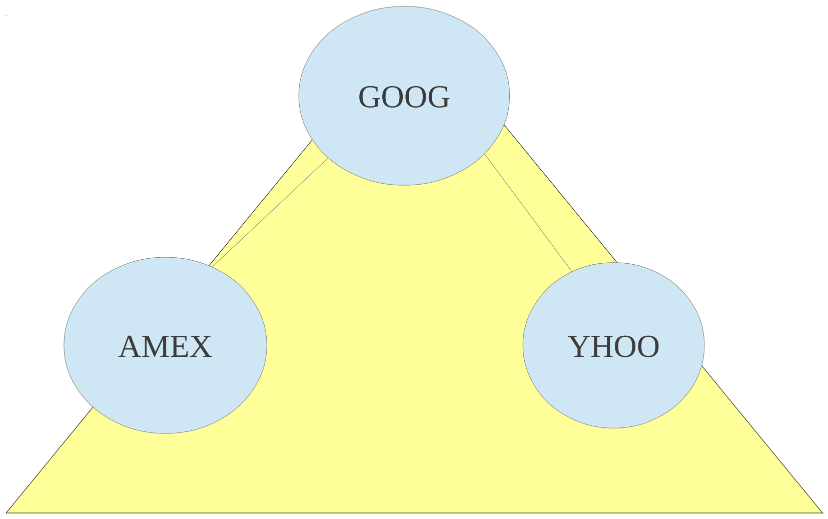
   b. *Graph line 2*: The second line is the linear least-squares regression line of the total closing-price figures in Line 1.

7. **Output to HTML File**

   a. Create our output string for the file

   b. Write the string out to file and view in browser

Original Tree Example



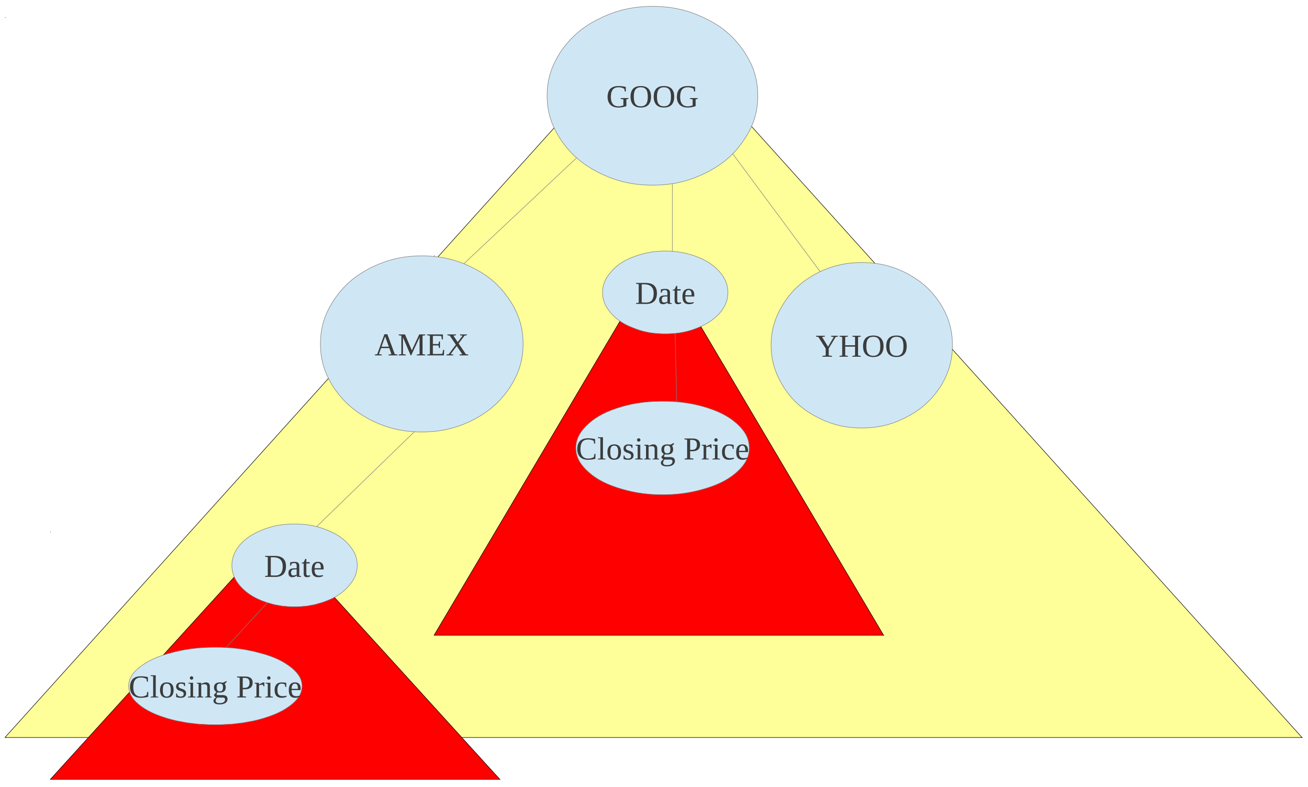
Analysis

- Where each Node is of the Following Construct

  (GOOG, ((20121031,500),(20121101,600)))......

- This original plan did not pan out as expected as this would have been a more difficult way of retrieving the dates we need so we redesigned how our data is stored

Refined Tree Example



Analysis

- We have our main tree where search keys on the tree are the names of the stock tickers.
- Once the ticker key is found in the tree the datum for the node is a subtree consisting of dates which are the key and the datum for the subtree is the Closing Price

-Example:

Key	Datum
(“GOOG”,	**(Sub tree of Dates))**
(Date,	**Closing-Price)**

Module Overview

1. **Main Module**
    - main-function: Specifies the main entry point for the program calls all the needed modules

2. **Read-and-Parse Module (Minput.txt)**
    - **read-in-file**
        - file ---> string
        - reads in file returns a string
        - private

    - **packets-list**
        - string --> list of strings
        - Strings delimited by <SR>
        - private

    - **packets-set**
        - (list of delimiters, list of strings) → list of tags needed for tree
        - Tags needed are (TK,TD, CP)
        - private

    - **build-tree (mbuild-tree.lisp)**
        - (list of needed tags)---> full tree
        - builds tree with the nodes from packets-set
        - private

3. **Read-and-Sort Module (Mread-and-sort.lisp)**
    - **read-in-request**
        - file → string
        - reads in the request file
        - private

    - **parse-requests**
        - string → list of companies and dates
        - turns string to list of companies and dates
        - private
    - **fix-dates**
        - since the search for a particular ticker leads to many dates increment by one assuming 31 days in each month

  - This function also handles month/year change

- **get-nearest-date**
  - Since not all information occurs on all days we have to assume the CP is the same from the day before.
  - This function handles that by searching for the previous date.

- **prune-tree**
  - (list of companies and dates, tree) → pruned-tree
  - takes in the requests and prunes tree accordingly
  - public

4. **Linear-Regression Module (Mlinear-regression-functions.lisp)**
   - **Calculates the line of linear-regression**
     - Uses statistical methods to implement.

5. **HTML-Graph Module (Mhtml.lisp)**
   - **Write-out-html**
     - string->file
     - writes out the string to a html file
     - private
   - **build-string-graph1**
     - tuple->string
     - the tuple contains the data needed
     - private
   - **build-string-linear-reg**
     - string->file
     - writes out the string to a html file
     - private
   - **Write-out-html**
     - string->file
     - writes out the string to a html file
     - private

6. **AVL-Tree (Mavl-string-keys.lisp)**
   - **AVL-tree**
     - Typical AVL tree using strings as keys
     - Insert, Delete, Rotation, Flatten operations
     - Uses Strings as keys

Work Delegation

Shane Moore:

- Responsible for Mread-sort.lisp

Jakob Griffith:

-Responsible for Mmain_module.lisp / Tied all parts together

Femi Fashanu:

- Responsible for Mbuild-tree.lisp

Cezar Delucca:

-Responsible for Mhtml.lisp

Clayton Miller:

-Responsible for Minput.lisp

ACL2 Modules

Main-module.lisp

```
;75 Chars
*****************************************************************

;Team Van Rossum
;Software Engineering 1
;defines main interface with user

;how to use this file
; execute the command (do) with three parameters
; out_html - the file name of the output html
; in_req   - the file name of the input requirements file
; in_hist  - the file name of the ticker statistics

(require "mavl-string-keys.lisp")
(require "mlinear_regression_functions.lisp")
(require "minput.lisp")
(require "mbuild-tree.lisp")
(require "mread-sort.lisp")
(require "mcalc_slope_intercept.lisp")
(require "mhtml.lisp")

; main interface
(interface Imain
  (sig do(out_html in_req in_hist)))

; main module
(module Mmain-private
  (import Iinput)
  (import Ibuild-tree)
  (import Iread-sort)
  (import Icalc_slope_intercept)
  (import I-HTMLGenerator)

  (set-state-ok t)

  ;simply stack everything as a single command
  (defun do (out_html in_req in_hist)
    (writeHTML out_html
               (get_list_and_slope_intercept
                (prune (read-req-file in_req)
                       (delegate-into-tree
                        (parse-input (file->tuples in_hist state))
   )))))

  (export Imain)
  )

; build all the correct links
(link Mmain
  (import Iavl-string-keys
          Ilinear_regression_functions
          Iinput
          Ibuild-tree
          Iread-sort
          Icalc_slope_intercept
          I-HTMLGenerator
          )
  (export Imain)
  (Mmain-private))

(link Rmain
      (import)
      (export Imain)
      (   Mavl-string-keys
          Mlinear_regression_functions
          Minput
          Mbuild-tree
          Mread-sort
          Mcalc_slope_intercept
          M-HTMLGenerator
          Mmain))
; run
(invoke Rmain)
```

Mavl-string-keys.lisp

```
;75 Chars
*****************************************************************

;Team Van Rossum
;Software Engineering 1
;defines interface for mavl-string-key module
;build off of pages model

(interface Iavl-string-keys
  ; function def's
    (sig avl-retrieve (tr k))
    (sig empty-tree ( ))
    (sig avl-insert (tr new-key new-datum))
    (sig avl-delete (tr key))
    (sig avl-flatten (tr))
    (sig avl-flatten-both (tr)))

#|=======  Module: AVL trees
=======================================
 Defining
   (avl-retrieve tr key)
   (empty-tree)
   (avl-insert tr key datum)
   (avl-delete tr key)
   (avl-flatten tr)

=========  Usage Notes
=============================================
 To make definitions in this module available in an importing module,
 put the following commands in that module:
   (include-book "avl-rational-keys" :dir :teachpacks)
=====================================================================
=
 Function usage notes

 1. (avl-retrieve tr key)
   assumes
     tr         has been constructed by one of the AVL-tree
                constructors (empty-tree, avl-insert, and avl-delete)
     new-key    is a rational number
   delivers
     either a two element list (k d)
       such that k equals key and
                 tr contains a subtree with k and d in its root
     or nil, in case key does not occur in tr

 2. (empty-tree)
   delivers an empty AVL-tree

 3. (avl-insert tr key datum)
   assumes
      tr       has been constructed by the AVL-tree constructors
               (empty-tree, avl-insert, or avl-delete)
      key      is a rational number
   delivers an AVL tree with the following property
     (and (equal (avl-retrieve (avl-insert tr key datum) key)
                 (list key datum))
          (iff (avl-retrieve (avl-insert tr key datum) k)
               (or (avl-retrieve tr k)
                   (key= k key))))

 4. (avl-delete tr key)
   assumes
      tr       has been constructed by the AVL-tree constructors
               (empty-tree, avl-insert, and avl-delete)
      key      is a rational number
   delivers an AVL tree with the following property
     (equal (avl-retrieve (avl-delete tr key) key)
            nil)

 5. (avl-flatten tr)
   assumes
      tr       has been constructed by the AVL-tree constructors
               (empty-tree, avl-insert, and avl-delete)
   delivers a list of cons-pairs with the following properties
      (and (implies (occurs-in-tree? k tr)
                    (and (occurs-in-pairs? k (avl-flatten tr))
                         (meta-property-DF tr k)))
           (implies (not (occurs-in-tree? k tr))
                    (not (occurs-in-pairs? k (avl-flatten tr)))))
           (increasing-pairs? (avl-flatten tr)))
     where (meta-property-DF tr k) means that one of the elements, e,
     in the list (avl-flatten tr) satisfies (equal (car e) k)) and
     (cadr e) is the datum at the root of the subtree of tr where k
```

```
     occurs
==========   Environment setup    ===================================|
#
;
=====================================================================


(module Mavl-string-keys-private

  ; Extractors (and empty-tree detector)
  (defun empty-tree? (tr) (not (consp tr)))
  (defun height (tr) (if (empty-tree? tr) 0 (car tr)))
  (defun key (tr) (cadr tr))
  (defun data (tr) (caddr tr))
  (defun left (tr) (cadddr tr))
  (defun right (tr) (car (cddddr tr)))
  (defun keys (tr)
    (if (empty-tree? tr)
        nil
        (append (keys (left tr)) (list (key tr)) (keys (right tr)))))

  ; Constructors
  (defun empty-tree ( ) nil)
  (defun tree (k d lf rt)
    (list (+ 1 (max (height lf) (height rt))) k d lf rt))

  ; Contstraint detectors and key comparators
  (defun key? (k) (stringp k))     ; to change representation of
keys
  (defun key< (j k) (string< j k))   ;    alter definitions of key?
and key<
  (defun key> (j k) (string< k j))
  (defun key= (j k)       ; note: definitions of
    (and (not (key< j k))           ;    key>, key=, and key-member
         (not (key> j k))))    ;        get in line automatically
  (defun key-member (k ks)
    (and (consp ks)
         (or (key= k (car ks))
             (key-member k (cdr ks)))))
  (defun data? (d)
    (if d t t))
  (defun tree? (tr)
    (or (empty-tree? tr)
        (and (natp (height tr))               ; height
             (= (height tr)                      ;   constraints
                (+ 1 (max (height (left tr))
                          (height (right tr)))))
             (key? (key tr))                     ; key constraint
             (data? (data tr))                   ; data constraint
             (tree? (left tr))                   ; subtree
             (tree? (right tr)))))               ;   constraints

    ; Key occurs in tree detector
    (defun occurs-in-tree? (k tr)
      (and (key? k)
           (tree? tr)
           (key-member k (keys tr))))
    (defun alternate-occurs-in-tree? (k tr)
      (and (key? k)
           (tree? tr)
           (not (empty-tree? tr))
           (or (key= k (key tr))
               (alternate-occurs-in-tree? k (left tr))
               (alternate-occurs-in-tree? k (right tr)))))

    ; all-key comparators
    (defun all-keys< (k ks)
      (or (not (consp ks))
          (and (key< (car ks) k) (all-keys< k (cdr ks)))))

    (defun all-keys> (k ks)
      (or (not (consp ks))
          (and (key> (car ks) k) (all-keys> k (cdr ks)))))

    ; definitions of ordered and balanced, and avl-tree detector
    (defun ordered? (tr)
      (or (empty-tree? tr)
          (and (tree? tr)
               (all-keys< (key tr) (keys (left tr)))
               (all-keys> (key tr) (keys (right tr)))
               (ordered? (left tr))
               (ordered? (right tr)))))

    (defun balanced? (tr)
      (and (tree? tr)
           (or (empty-tree? tr)
               (and (<= (abs (- (height (left tr))
                                (height (right tr)))) 1)
                    (balanced? (left tr))
                    (balanced? (right tr))))))

    (defun avl-tree? (tr)
      (and (ordered? tr)
           (balanced? tr)))

    ; rotations
    (defun easy-R (tr)
      (let* ((z (key tr)) (dz (data tr))
                          (zL (left tr)) (zR (right tr))
                          (x (key zL)) (dx (data zL))
                          (xL (left zL)) (xR (right zL)))
        (tree x dx xL (tree z dz xR zR))))

    (defun easy-L (tr)
      (let* ((z (key tr)) (dz (data tr))
                          (zL (left tr)) (zR (right tr))
                          (x (key zR)) (dx (data zR))
                          (xL (left zR)) (xR (right zR)))
        (tree x dx (tree z dz zL xL) xR)))

    (defun left-heavy? (tr)
      (and (tree? tr)
           (not (empty-tree? tr))
           (= (height (left tr)) (+ 2 (height (right tr))))))

    (defun outside-left-heavy? (tr)
      (and (left-heavy? tr)
           (or (= (height (left (left tr)))
                  (height (right (left tr))))
               (= (height (left (left tr)))
                  (+ 1 (height (right (left tr))))))))

    (defun right-rotatable? (tr)
      (and (ordered? tr)
           (not (empty-tree? tr))
           (balanced? (left tr))
           (balanced? (right tr))
           (not (empty-tree? (left tr)))))

    (defun right-heavy? (tr)
      (and (tree? tr)
           (not (empty-tree? tr))
           (= (height (right tr)) (+ 2 (height (left tr))))))

    (defun outside-right-heavy? (tr)
      (and (right-heavy? tr)
           (or (= (height (right (right tr)))
                  (height (left (right tr))))
               (= (height (right (right tr)))
                  (+ 1 (height (left (right tr))))))))

    (defun left-rotatable? (tr)
      (and (tree? tr)
           (not (empty-tree? tr))
           (balanced? (left tr))
           (balanced? (right tr))
           (not (empty-tree? (right tr)))))

    (defun hard-R (tr)
      (let* ((z (key tr))
             (dz (data tr))
             (zL (left tr))
             (zR (right tr)))
        (easy-R (tree z dz (easy-L zL) zR))))

    (defun hard-L (tr)
      (let* ((z (key tr))
             (dz (data tr))
             (zL (left tr))
             (zR (right tr)))
        (easy-L (tree z dz zL (easy-R zR)))))

    (defun inside-left-heavy? (tr)
      (and (left-heavy? tr)
           (= (height (right (left tr)))
              (+ 1 (height (left (left tr)))))))

    (defun hard-R-rotatable? (tr)
      (and (right-rotatable? tr)
           (left-rotatable? (left tr))))

    (defun inside-right-heavy? (tr)
      (and (right-heavy? tr)
           (= (height (left (right tr)))
              (+ 1 (height (right (right tr)))))))
    (defun hard-L-rotatable? (tr)
      (and (left-rotatable? tr)
           (right-rotatable? (right tr))))

    (defun rot-R (tr)
      (let ((zL (left tr)))
        (if (< (height (left zL)) (height (right zL)))
            (hard-R tr)
            (easy-R tr))))

    (defun rot-L (tr)
      (let ((zR (right tr)))
        (if (< (height (right zR)) (height (left zR)))
            (hard-L tr)
            (easy-L tr))))

    ; insertion
    (defun avl-insert (tr new-key new-datum)
      (if (empty-tree? tr)
          (tree new-key new-datum (empty-tree) (empty-tree))
          (if (key< new-key (key tr))
              (let* ((subL (avl-insert (left tr) new-key new-datum))
                     (subR (right tr))
                     (new-tr (tree (key tr) (data tr) subL subR)))
                (if (= (height subL) (+ (height subR) 2))
                    (rot-R new-tr)
                    new-tr))
              (if (key> new-key (key tr))
                  (let* ((subL (left tr))
                         (subR (avl-insert (right tr) new-key new-
  datum))
                         (new-tr (tree (key tr) (data tr) subL subR)))
                    (if (= (height subR) (+ (height subL) 2))
                        (rot-L new-tr)
                        new-tr))
                  (tree new-key new-datum (left tr) (right tr))))))

    ; delete root - easy case
    (defun easy-delete (tr)
      (right tr))

    ; tree shrinking
    (defun shrink (tr)
      (if (empty-tree? (right tr))
          (list (key tr) (data tr) (left tr))
          (let* ((key-data-tree (shrink (right tr)))
                 (k (car key-data-tree))
                 (d (cadr key-data-tree))
                 (subL (left tr))
                 (subR (caddr key-data-tree))
                 (shrunken-tr (tree (key tr) (data tr) subL subR)))
            (if (= (height subL) (+ 2 (height subR)))
                (list k d (rot-R shrunken-tr))
                (list k d shrunken-tr)))))

    (defun raise-sacrum (tr)
      (let* ((key-data-tree (shrink (left tr)))
             (k (car key-data-tree))
             (d (cadr key-data-tree))
             (subL (caddr key-data-tree))
             (subR (right tr))
             (new-tr (tree k d subL subR)))
        (if (= (height subR) (+ 2 (height subL)))
            (rot-L new-tr)
            new-tr)))

    ; delete root - hard case
    (defun delete-root (tr)
      (if (empty-tree? (left tr))
          (easy-delete tr)
          (raise-sacrum tr)))

    ; deletion
    (defun avl-delete (tr k)
      (if (empty-tree? tr)
          tr
          (if (key< k (key tr))            ; key occurs in left subtree
              (let* ((new-left (avl-delete (left tr) k))
                     (new-tr (tree (key tr) (data tr)
                                   new-left (right tr))))
                (if (= (height (right new-tr))
                       (+ 2 (height (left new-tr))))
                    (rot-L new-tr)
                    new-tr))
              (if (key> k (key tr))        ; key occurs in right subtree
                  (let* ((new-right (avl-delete (right tr) k))
                         (new-tr (tree (key tr) (data tr)
                                       (left tr) new-right)))
                    (if (= (height (left new-tr))
                           (+ 2 (height (right new-tr))))
                        (rot-R new-tr)
                        new-tr))
                  (delete-root tr)))))  ; key occurs at root

    ; retrieval
    (defun avl-retrieve (tr k)  ; delivers key/data pair with key = k
      (if (empty-tree? tr)      ; or nil if k does not occur in tr
          nil               ; signal k not present in tree
          (if (key< k (key tr))
              (avl-retrieve (left tr) k)   ; search left subtree
              (if (key> k (key tr))
                  (avl-retrieve (right tr) k) ; search right subtree
                  (cons k (data tr))))))      ; k is at root,
    ; deliver key/data pair

    (defun avl-flatten (tr)  ; delivers all key/data cons-pairs
      (if (empty-tree? tr)   ; with keys in increasing order
          nil
          (append (avl-flatten (left tr))
                  (list (cons (key tr) (data tr)))
                  (avl-flatten (right tr)))))

    (defun occurs-in-pairs? (k pairs)
      (and (consp pairs)
           (or (key= k (caar pairs))
               (occurs-in-pairs? k (cdr pairs)))))

    (defun increasing-pairs? (pairs)
      (or (not (consp (cdr pairs)))
          (and (key< (caar pairs) (caadr pairs))
               (increasing-pairs? (cdr pairs)))))

      (defun avl-flatten-both (tr)  ; delivers all key/data cons-pairs
      (if (empty-tree? tr)        ; with keys in increasing order
          nil                     ; and datum is also a flattened tree
          (append (avl-flatten-both (left tr))
                  (list (cons (key tr) (list (avl-flatten (data tr)))))
                  (avl-flatten-both (right tr)))))

    (export Iavl-string-keys)
    )
```

```
(link Mavl-string-keys
      (import)
      (export Iavl-string-keys)
      (Mavl-string-keys-private))
```

Mbuild-tree.lisp

```
;75 Chars
*****************************************************************

;Team Van Rossum
;Software Engineering 1
;defines how to build a tree after reading in the file

; Defines the Module build tree

(require "mavl-string-keys.lisp")

(interface Ibuild-tree

  (sig insert-into-tree (xs tree))
  (sig delegate-into-tree (xs))
  (sig parse-input (xs))
  )

(module Mbuild-tree-private
  (import Iavl-string-keys)
  (include-book "list-utilities" :dir :teachpacks)
  (include-book "io-utilities" :dir :teachpacks)

  (set-state-ok t)

  (defun insert-into-tree (xs tree)
    (let* ((tk (car xs))
           (td (cadr xs))
           (cp  (caddr xs))
           (old-tree (avl-retrieve tree tk))
           (new-tree
            (if (equal old-tree nil)
                (avl-insert tree tk (avl-insert (empty-tree) td cp))
                (avl-insert tree tk (avl-insert (cdr old-tree) td
cp)))))
      new-tree))

  ; this fuction takes a list of lists where each list of strings
  ; contains the ticker, the closing price and the trade date.
  ;Then it builds an AVL
  ;tree using the tk as the key, and the closing price
  ; and trade date as the data
  (defun delegate-into-tree-helper (xs tree)
  (if (consp (cdr xs))
        (delegate-into-tree-helper (cdr xs) (insert-into-tree (car 
xs) tree))
        (insert-into-tree (car xs) tree)))

  (defun delegate-into-tree (xs)
    (delegate-into-tree-helper xs (empty-tree)))

  (defun parse-input (xs)
    (if (consp xs)
        (let* ((cp (chrs->str (car(tokens '(#\< #\c #\p #\>)
                                       (str->chrs (cadr(car xs)))))))
               (td (chrs->str (car(tokens '(#\< #\t #\d #\>)
                                      (str->chrs (caddr(car xs)))))))
               (key (chrs->str (car(tokens '(#\< #\t #\k #\>)
                                      (str->chrs (caar xs)))))))
          (cons (list key td cp) (parse-input (cdr xs))))
        nil))

  (export Ibuild-tree)
  )

(link Mbuild-tree
      (import Iavl-string-keys)
      (export Ibuild-tree)
      (Mbuild-tree-private))
```

Mcalc-slope-intercept.lisp

```
;75 Chars
*****************************************************************

;Team Van Rossum
;Software Engineering 1
;defines how to take a pruned tree and calculate slope and intercept

(require "mavl-string-keys.lisp")
(require "mlinear_regression_functions.lisp")

(interface Icalc_slope_intercept

  (sig build-total-child-helper (flat-sub-tree return-tree))
  (sig build-total-helper (flat-tree return-tree))
  (sig build-total-date-tree (flat-tree))
  (sig get_xs (flat_dates_values number))
  (sig get_ys (flat_dates_values))
  (sig get_list_and_slope_intercept (pruned_tree)))

(module Mcalc_slope_intercept-private
  (import Iavl-string-keys)
  (import Ilinear_regression_functions)
  (include-book "io-utilities" :dir :teachpacks)
  (include-book "list-utilities" :dir :teachpacks)
  (set-state-ok t)

  (defun build-total-child-helper (flat-sub-tree return-tree)
    (if (consp flat-sub-tree)
        (let* ((key_date  (caar flat-sub-tree))
               (key_value (cdar flat-sub-tree))
               (old_tree  (avl-retrieve return-tree key_date))
               (old_value (cdr old_tree))
               (new-return-tree
                (if (equal old_tree nil)
                    (avl-insert return-tree key_date (str->rat
key_value))
                    (avl-insert (avl-delete return-tree key_date)
key_date
                                (+ old_value (str->rat key_value)))
                    )))
          (build-total-child-helper (cdr flat-sub-tree) new-return-
tree))
        return-tree))

  (defun build-total-helper (flat-tree return-tree)
    (if (consp flat-tree)
        (let* ((key_ticker (caar flat-tree))
               (date_list  (cadar flat-tree))
               (new-return
                (build-total-child-helper date_list return-tree)))
          (build-total-helper (cdr flat-tree) new-return)
          )
        return-tree))

  ;spits out a tree where the datum are total ints of all keys
  ;and the keys are string dates
  (defun build-total-date-tree (flat-tree)
    (build-total-helper flat-tree (empty-tree)))

  (defun get_xs (flat_dates_values number)
    (if (consp flat_dates_values)
        (cons number (get_xs (cdr flat_dates_values) (+ 1 number)))
        nil))

  (defun get_ys (flat_dates_values)
    (if (consp flat_dates_values)
        (cons (cdar flat_dates_values) (get_ys (cdr
flat_dates_values)))
        nil))

  ;this is the method to call
  (defun get_list_and_slope_intercept (pruned_tree)
    (let* ((flat_dates_values
            (avl-flatten (build-total-date-tree
                          (avl-flatten-both pruned_tree))))
           (xs (get_xs flat_dates_values 1))
           (ys (get_ys flat_dates_values))
           (b_a (compute_slope_intercept xs ys)))
      (list flat_dates_values b_a)))

  (export Icalc_slope_intercept)
  )

(link Mcalc_slope_intercept
      (import Iavl-string-keys Ilinear_regression_functions)
      (export Icalc_slope_intercept)
      (Mcalc_slope_intercept-private))
```

Mhtml.lisp

```
;75 Chars
*****************************************************************

;Team Van Rossum
;Software Engineering 1
;defines how to writeout the data to an HTML file

;Interface signatures for the HTMLGenerator module.
(interface I-HTMLGenerator
  (sig regressionList-Helper(number slope intercept))
  (sig regressionList (xs slope intercept))
  (sig data->str (datesPrices regressionVals))
  (sig stringBuilder (valuesStr))
  (sig writeHTML(fileName datesPricesReg))

  ;regressionList contract
  (con regressionList-properties
       (if (not (consp xs))
           (equal (regressionList xs slope intercept) nil)
           ;empty list returns nil
           (= (len xs) (len (regressionList xs slope intercept)))))
  ;otherwise length should be the same as original
  )

(module M-HTMLGenerator-private

  (include-book "arithmetic-3/top" :dir :system)
  (include-book "io-utilities" :dir :teachpacks)
  (include-book "list-utilities" :dir :teachpacks)
  (include-book "doublecheck" :dir :teachpacks)
  (include-book "testing" :dir :teachpacks)
  (set-state-ok t)

  ;regressionList (xs)
  ;This function takes in a list of numbers and
  ;generates a list of regression values based on
  ;a slope and intercept (mx+b)
  ;xs = the list of values
  ;slope = the calculated slope (m) from the regression module (s)
  ;intercept = the calculated intercept (b) from the regression
module (s)
```

```
  (defun regressionList-helper (number slope intercept)
    (if (<= number 0)
        nil
        (cons  (+ (* slope number) intercept) (regressionList-helper
                                 (- number 1) slope intercept))))

  (defun regressionList (xs slope intercept)
    (reverse (regressionList-helper (len xs) slope intercept)))

  ;data->str (dates totalPrices reg)
  ;This function takes in the data and converts it to
  ;a string representation of a matrix to be used for
  ;the Google visualization API.
  ;dates = the analysis request dates (x-axis)
  ;totalPrices = the total closing prices of the stocks
  ;within the analysis file for a specific closing date (y-axis 1)
  ;reg = the linear regression data set (y -axis 2)

  ; new format gives a date, value, null null, linear regression,
null, null
  ;[new Date(2008, 1 ,1), 30000, null, null, 40645, null, null],

  ;Note: both datesPrices and regressionVals will be indexed the same
,
  ; so checking if we haved reachedthe end of regressionVals
  ; should be sufficient.
  ;rat->str requires a number of sig figs, for now im using 4...
  (defun data->str (datesPrices regressionVals)
    (if (endp regressionVals)
        nil
        (let* ((date-dgts (caar datesPrices))
               (mnth   (subseq date-dgts 4 6))
               (day    (subseq date-dgts 6 8))
               (year   (subseq date-dgts 0 4))
               (full_s (concatenate 'string "[new Date(" year ","
mnth ","
               day ")," (rat->str (car regressionVals) 4) ", null,
null, "
                (rat->str (cdar datesPrices) 4)  ", null, null]")))
          (if (equal (cdr regressionVals) nil)
              ;if this is the last element,
              ;format without a comma.
              full_s
              (concatenate 'string full_s ", \n"
                  (data->str (cdr datesPrices)(cdr regressionVals)))
              ))))


  ;stringBuilder (valuesStr)
  ;This function builds an HTML string from a data string
  ;that will generate a Google visualization line graph.
  ;valuesStr = input string with values to graph
  (defun stringBuilder (valuesStr)
    (list

     "<html>
  <head>
    <script type='text/javascript' src='http://www.google.com/jsapi'>
</script>
    <script type='text/javascript'>
      google.load('visualization', '1',
 {'packages':['annotatedtimeline']});
      google.setOnLoadCallback(drawChart);
      function drawChart() {
        var data = new google.visualization.DataTable();
        data.addColumn('date', 'Date');
        data.addColumn('number', 'Linear Regression');
        data.addColumn('string', 'title1');
        data.addColumn('string', 'text1');
        data.addColumn('number', 'Total Values');
        data.addColumn('string', 'title2');
        data.addColumn('string', 'text2');
        data.addRows(["

     (concatenate 'string valuesStr)

     "]);

        var chart = new google.visualization.AnnotatedTimeLine(
document.getElementById('chart_div'));
        chart.draw(data, {displayAnnotations: true});
      }
    </script>
  </head>

  <body>
    <div id='chart_div' style='width: 1200px; height: 800px;'></div>
  </body>
</html>"))

  ;writeFile (fileName dataStrList)
  ;This function writes day-by-day total closing price of the stocks
  ;within an analysis request along with a linear least-squares
  ;regression line
  ;to an HTML file. This content is then transformed
  ;by the Google visualization
  ;API into a line graph representation.
  ;fileName = the output file to write to.
  ;datesPricesReg = (((td, sumcp)(td2, sumcp2) ...) slope intercept)
  ;
  ;Note: This is the entry point for this module.
  ;This function should be the only one called externally.
  (defun writeHTML (fileName datesPricesReg)
    (let* ((intercept (cadadr datesPricesReg))
           (slope (caadr datesPricesReg))
           (xs (car datesPricesReg))
           (regressionVals (regressionList xs slope intercept))
           (dataStrConversion (data->str (car datesPricesReg)
                                         regressionVals))
           (htmlResult (stringBuilder dataStrConversion)))
      (string-list->file fileName htmlResult state)
      ))

  (export I-HTMLGenerator)
  )

(link M-HTMLGenerator
      (import)
      (export I-HTMLGenerator)
      (M-HTMLGenerator-private))
```

Minput.lisp

```
;75 Chars
*****************************************************************

;Team Van Rossum
;Software Engineering 1
;defines how a file is imported for tree creation

(interface Iinput
  (sig break-on-<sr> (filename state))
  (sig extract-fields (xs))
  (sig generate-tuples (xs))
  (sig file->tuples (filename state)))

(module Minput-private
  (include-book "list-utilities" :dir :teachpacks)
  (include-book "io-utilities" :dir :teachpacks)
  (set-state-ok t)

  ; (break-on-<sr> filename state)
  ; This function takes in a file and a state and delivers a list of
  ; lists of strings
  ; filename = string representation of the file to read
  ; state = file->string state
  (defun break-on-<sr> (filename state)
    (let* ((rawinput (car (file->string filename state)))
           (srsplit (packets "<sr>" (words rawinput))))
      (cdr srsplit)))

  ; (extract-fields xs)
  ; This function takes a list of strings and returns a list of three
  ; strings representing the ticker, closing price, and trading date
  ; of a particular stock record, respectively
  ; xs = list of strings representing fields in a stock record
  (defun extract-fields (xs)
    (let* ((tk (nth 0 xs))
           (cp (nth 5 xs))
           (td (nth 1 xs)))
      (cons tk (list cp td))))

  ; (generate-tuples xs)
  ; This function takes a list of a list of strings, where each list
  ; of strings represents a stock record, and each string is a field.
  ; It returns a list of lists of strings, in which each list of
  ; strings only contains the ticker, closing price, and trading
  ; date of the corresponding stock record, respectively
  ; xs = list of lists of strings representing stock records
  (defun generate-tuples (xs)
    (if (consp xs)
        (let* ((first (extract-fields (car xs)))
               (rest (generate-tuples (cdr xs))))
          (cons first rest))
        nil))

  ; (file->tuples filename state)
  ; This function takes a file and a state and returns a list of
lists
  ; of strings, in which each list of strings contains the ticker,
  ; closing price, and trade date of a given stock record,
respectively
  ; filename = string representation of the file to read
  ; state = file->string state
  (defun file->tuples (filename state)
    (generate-tuples (break-on-<sr> filename state)))

  (export Iinput)
  )

(link Minput
      (import)
      (export Iinput)
      (Minput-private))
```

Mlinear-regression-functions.lisp

```
;Jakob Griffith
;functions
(interface Ilinear_regression_functions
 (sig compute_slope_intercept (xs ys)))

(module Mlinear_regression_functions-private

(include-book "io-utilities" :dir :teachpacks)
(include-book "list-utilities" :dir :teachpacks)
(set-state-ok t)

;helper function for different avg
(defun avg-helper (xs length-xs)
  (if (consp xs)
      (+ (/ (car xs) length-xs) (avg-helper (cdr xs) length-xs))
      0))

;function calculates avg in a different way
(defun avg(xs)
  (if (consp xs)
      (avg-helper xs (length xs))
  nil))

;helper function for adding scalar to a list
(defun vector_plus_scalar-helper (xs x)
  (if (consp xs)
      (cons (+ (car xs) x) (vector_plus_scalar-helper (cdr xs) x))
      nil))

;function adds a scalar to a list.
(defun vector_plus_scalar (xs x)
  (if (and (rationalp x) (consp xs))
      (vector_plus_scalar-helper xs x)
      nil))

;helper function for adding scalar to a list
(defun vector_mul_vector-helper (xs ys)
  (if (and (consp ys) (consp xs))
      (cons (* (car xs) (car ys))
            (vector_mul_vector-helper (cdr xs) (cdr ys)))
      nil))
```

```
;function adds a scalar to a list.
(defun vector_mul_vector (xs ys)
  (if (and (consp ys) (consp xs) (equal (length ys) (length xs)))
      (vector_mul_vector-helper xs ys)
      nil))

;function computers the r value
(defun compute_r_xvar (xs ys xavg yavg)
  (if (and (rationalp xavg) (rationalp yavg) (consp ys)
           (consp xs) (equal (length ys) (length xs)))
      (let* ((x_m_a   (vector_plus_scalar xs (- 0 xavg)))
             (y_m_a   (vector_plus_scalar ys (- 0 yavg)))
             (x_mul_y (vector_mul_vector x_m_a y_m_a)))
      (avg x_mul_y))
      nil))

;quick wrapper for xvar
(defun compute_xvar (xs xavg)
  (if (and (rationalp xavg) (consp xs))
      (compute_r_xvar xs xs xavg xavg)
      nil))

;function computers slope and interectp and puts them into a list r.
;b is slope
;a is intercept
(defun compute_slope_intercept (xs ys)
  (if (and (consp xs) (consp ys))
      (let* ((yavg (avg ys))
             (xavg (avg xs))
             (r    (compute_r_xvar xs ys xavg yavg))
             (xvar (compute_xvar xs xavg))
             (b (/ r xvar))
             (a (- yavg (* b xavg))))
        (list b a))
      nil))


;************************************
;everything below this line is IO
;************************************


;convert a list of chars into a list of rational numbers
```

```
(defun chrs_list->ration_list (xs)
  (if (consp xs)
      (cons (str->rat (chrs->str (car xs))) (chrs_list->ration_list
                                             (cdr xs)))
       nil))

;function splits a list of numbers into a list of xs and ys
(defun split_xs_ys (xs)
  (if (consp (cdddr xs))
      (let* ((tfirst  (car xs))
             (tsecon  (cadr xs))
             (remain  (cddr xs))
             (calced  (split_xs_ys remain))
             (calfir  (car calced))
             (calsec  (cadr calced)))
        (list (cons tfirst calfir) (cons tsecon calsec)))
      (list (list (car xs)) (list (cadr xs)))))

;function takes a list of chars and counts how many are after decimal
(defun count_after_decimal (xs)
  (if (consp xs)
      (length (drop-past #\. xs))
      0))

;function finds the maximum sig figs from a list of chars
(defun max_decimals (xs)
  (if (consp xs)
      (max (count_after_decimal (car xs)) (max_decimals (cdr xs)))
      0))

;helper function for read-transform-write
(defun your-transform (input-string)
  (let* ((LF             (code-char 10))
         (CR             (code-char 13))
         (whitespace     (list #\Space #\Newline #\Tab LF CR))
         (string_numbers (tokens whitespace (str->chrs input-
string)))
         (ration_numbers (chrs_list->ration_list string_numbers))
         (max_decimals_c (max_decimals string_numbers))
         (split_xsys     (split_xs_ys ration_numbers))
         (xs             (car split_xsys))
         (ys             (cadr split_xsys))
         (ab             (compute_slope_intercept xs ys))
         (slope          (car ab))
         (intercept      (cadr ab)))
    (list "Linear Regression Coefficients (slope, intercept)"
          (concatenate 'string
           (rat->str slope max_decimals_c)
           " "
           (rat->str intercept max_decimals_c))
          "x-y Data"
          input-string)))

;predefined function from the slides
(defun read-transform-write (f-in f-out state)
  (mv-let (input-as-string error-open state)
          (file->string f-in state)
     (if error-open
         (mv error-open state)
         (mv-let (error-close state)
                 (string-list->file f-out
                                    (your-transform input-as-string)
                                    state)
            (if error-close
                (mv error-close state)
                (mv (string-append "input file: "
                     (string-append f-in
                      (string-append ", output file: " f-out)))
                    state))))))

;our wrapper for easily making input file and output file name match
(defun entry_point (f-in state)
  (read-transform-write f-in (concatenate 'string
      (chrs->str (car (packets #\. (str->chrs f-in))))
      "withLRcoeffs.txt") state))

  (export Ilinear_regression_functions)
  )

(link Mlinear_regression_functions
      (import)
      (export Ilinear_regression_functions)
      (Mlinear_regression_functions-private))
```

Mread-sort.lisp

```
;75 Chars
*****************************************************************

;Team Van Rossum
;Software Engineering 1
;defines interface for mread-sort module

(require "mavl-string-keys.lisp")

(interface Iread-sort

  (sig check-day (date))
  (sig fmt-date-helper (year mnth day))
  (sig fmt-date (date))
  (sig plus_one_date (date))
  (sig get_nearest_date_< (date rev_flat_tree))
  (sig get_nearest_date_> (date flat_tree))
  (sig date_difference_helper (date_1 date_2 count))
  (sig date_difference (date_1 date_2))
  (sig get_nearest_date (date sub_tree))
  (sig  get-by-dates (start end tree ret-tree))
  (sig prune-clean (reqs tree ret-tree))
  (sig  prune (reqs tree))
  (sig parse-analysis-req (reqs))
  (sig to-search-structure (xs))
  (sig  split-csv-style (data))
  (sig read-req-file (filename))

  )

(module Mread-sort-private
  (import Iavl-string-keys)
  (include-book "io-utilities" :dir :teachpacks)
  (include-book "list-utilities" :dir :teachpacks)
  (set-state-ok t)

  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;Date Formatting;;;;;;;;
  ;Checks the day on whether
  ; it is a 31 will be used to see if needed next month
  (defun check-day (date)
    (if (equal 31 (str->int (subseq date 6 8)))
        t
        nil))
  ;Formats the date appropriately
  ;called from fmt-date ahead a month or year if needed
  (defun fmt-date-helper (year mnth day)
    (if (equal 12 mnth)
        (append (int->dgts (1+ year)) '(0 1 0 1))
        (if (< mnth 9)
            (append (int->dgts year) (cons 0 (int->dgts (1+ mnth)))
'(0 1))
            (append (int->dgts year) (int->dgts (1+ mnth))  '(0
1)))))

  ;Formats the date to a format that we can work with
  (defun fmt-date (date)
    (let* ((date-dgts  (int->dgts (str->int date)))
           (mnth (dgts->int (subseq date-dgts 4 6)))
           (day (dgts->int (subseq date-dgts 6 8)))
           (year (dgts->int (subseq date-dgts 0 4))))
      (int->str (dgts->int (fmt-date-helper year mnth day)))))



  ;;;;;;;;;;;;; Add these nodes to the tree if not in;;;;;;;;;;
  ;Increase the date by one
  (defun plus_one_date (date)
    (int->str (1+ (str->int date))))

  (defun get_nearest_date_< (date rev_flat_tree)
    (let* ((top_element (car rev_flat_tree))
           (top_date    (car top_element)))
      (if (consp rev_flat_tree)
          (if (string< top_date date)
              top_element
              (get_nearest_date_< date (cdr rev_flat_tree)))
          nil)))

  (defun get_nearest_date_> (date flat_tree)
    (let* ((top_element (car flat_tree))
           (top_date    (car top_element)))
      (if (consp flat_tree)
          (if (string> top_date date)
              top_element
              (get_nearest_date_> date (cdr flat_tree)))
          nil)))

  (defun date_difference_helper (date_1 date_2 count)
    (if (>= (str->int date_1) (str->int date_2))
        count
        (let* ((start-date (if (check-day date_1)
                               (fmt-date date_1)
                               date_1)))
          (date_difference_helper (plus_one_date start-date) date_2
(1+ count)))))

  ; Difference between dates
  (defun date_difference (date_1 date_2)
    (if (> (str->int date_1) (str->int date_2))
        nil
        (date_difference_helper date_1 date_2 0)))

  (defun get_nearest_date (date sub_tree)
    (let* ((flat_tree (avl-flatten sub_tree))
           (nearest<  (get_nearest_date_< date (reverse flat_tree)))
           (nearest>  (get_nearest_date_> date flat_tree))
           (date<     (car nearest<))
           (date>     (car nearest>))
           (amount<
            (if (equal date< nil)
                9999999999999999
                (date_difference date< date)))
           (amount>
            (if (equal date> nil)
                9999999999999999
                (date_difference date date>))))
      (if (< amount< amount>) ;because fuck you
          nearest<
          nearest>)))


  ;;;;;;;;;;;;;;Now to get the stuff outta tree

  ; Searching subtree for the corret dates
  ; First check if we have matched the start
  ; make sure date format is correct
  (defun get-by-dates (start end tree ret-tree)
    (if (equal start end)
        ret-tree
    (let* ((start-date (if (check-day start)
                               (fmt-date start)
                               start))
               (old-ret-tree (avl-retrieve tree start-date))
               (start_value (cdr old-ret-tree))
               (new-ret-tree (if (equal old-ret-tree nil)
                                 (let* ((nearest_element
(get_nearest_date start-date tree))
                                        (nearest_value   (cdr
nearest_element)))
                                   (avl-insert ret-tree start-date
nearest_value))
                                 (avl-insert ret-tree start-date
start_value))))
          (get-by-dates (plus_one_date start-date) end tree
                        new-ret-tree))))

  ; After the retrieval file is parsed
  ; Prepare the data to search the tree
  ; Once the correct ticker name is found
  (defun prune-clean (reqs tree ret-tree)
    (if (consp reqs)
        (let* ((req (car reqs))
               (ticker (car req))
               (start (cadr req))
               (end (caddr req))
               (start-tree (avl-retrieve tree ticker))
               (sub-tree (cdr start-tree))
               (new-ret-tree
                (if (equal start-tree nil)
                    ret-tree
                    (avl-insert ret-tree ticker (get-by-dates start
end sub-tree (empty-tree))))))
          (prune-clean (cdr reqs) tree new-ret-tree))
        ret-tree))

  ;Call this function to start the whole prune process
  (defun prune (reqs tree)
    (prune-clean reqs tree (empty-tree)))

  ;So here we will pretty much have a list
  ;where in fact every even numbered list part
  ; will be the stuff we need
  (defun parse-analysis-req (reqs)
  (if (consp reqs)
        (cons (cdddr (car reqs)) (parse-analysis-req (cddr reqs)))
        nil))

  ; Get the data into the proper structure for
  ; Traversing tree
  (defun to-search-structure (xs)
    (if (consp xs)
        (cons (chrs->str (car xs)) (to-search-structure(cdr xs)))
        nil))

  ; Just plain csv splitter to you know
  ; split comma delited stuff
  (defun split-csv-style (data)
    (if (consp data)
        (cons (to-search-structure (packets #\, (car data)))
              (split-csv-style (cdr data)))
        nil))

  ; Reads in file and sends to pruner
  (defun read-req-file (filename)
    (if (stringp filename)
        (split-csv-style
         (parse-analysis-req
          (cdr(packets-set '(#\<,#\A,#\R,#\>)
                             (str->chrs (car (file->string filename
state)))
                             ))))
        nil))

  (export Iread-sort)
  )

(link Mread-sort
      (import Iavl-string-keys)
      (export Iread-sort)
      (Mread-sort-private))
```

Module Tests

Thelper.lisp

```
;75 Chars
*****************************************************************

;Team Van Rossum
;Software Engineering 1
;helper function that builds some trees for testing

(require "mavl-string-keys.lisp")

(interface Ihelper
  (sig make-subtree())
  (sig make-different-subtree())
  (sig make-tree())
  (sig make-complex-tree()))

(module Thelper-private
  (import Iavl-string-keys)

  (defun make-subtree()
    (let* ((start-tree (empty-tree))
           (start-tree2 (avl-insert start-tree "20120101" "2.0"))
           (start-tree3 (avl-insert start-tree2 "20120202" "3.0"))
           (start-tree4 (avl-insert start-tree3 "20130101" "1.0")))
      start-tree4))

  (defun make-different-subtree()
    (let* ((start-tree (empty-tree))
           (start-tree2 (avl-insert start-tree "20090101" "2.0"))
           (start-tree3 (avl-insert start-tree2 "20090202" "3.0"))
           (start-tree4 (avl-insert start-tree3 "20080101" "1.0")))
      start-tree4))

  (defun make-tree ()
    (avl-insert (empty-tree) "GOOG"
                (make-subtree)))

  (defun make-complex-tree ()
    (let* ((first
            (avl-insert (make-tree) "AA" (make-subtree)))
           (second
            (avl-insert first "AAPL" (make-subtree))))
      second))

  (export Ihelper)
  )

(link Thelper
      (import Iavl-string-keys)
      (export Ihelper)
      (Thelper-private))
```

Thtml.lisp

```
;75 Chars
*****************************************************************

;Team Van Rossum
;Software Engineering 1
;Testing for the HTML generator

(require "mhtml.lisp")

(module T-HTMLGenerator-private
  (import I-HTMLGenerator)

  (include-book "testing" :dir :teachpacks)
  (include-book "doublecheck" :dir :teachpacks)

  ;check expects
  (check-expect (regressionList '(1 2 3 4)  2 3) '(5 7 9 11))
  (check-expect (regressionList '()  2 3) nil)
  (check-expect (regressionList '(1 2 3 4 5)  0 0) '(0 0 0 0 0))
  )

(link T-HTMLGenerator
      (M-HTMLGenerator T-HTMLGenerator-private))

(invoke T-HTMLGenerator)
```

Tread-sort.lisp

```
;75 Chars
*****************************************************************

;Team Van Rossum
;Software Engineering 1
;Testing for the read and sort module

(require "Mread-sort.lisp")

(module Tread-sort-private
  (import Iavl-string-keys)
  (import Iread-sort)
  (include-book "testing" :dir :teachpacks)
  (include-book "doublecheck" :dir :teachpacks)



  (check-expect (check-day "20121131") t)
  (check-expect (check-day "20120101") nil)
  (check-expect (check-day "20120214") nil)


  (check-expect (fmt-date "20121131") "20121201")
  (check-expect (fmt-date "20121231") "20130101")
  (check-expect (fmt-date "20120931") "20121001")
  (check-expect (fmt-date "20120831") "20120901")
  (check-expect (fmt-date "19991231") "20000101")


  (check-expect (split-csv-style nil) nil)
  (check-expect (split-csv-style '((#\G #\O #\O #\G #\,
             #\2 #\0 #\1 #\2 #\1 #\1 #\1 #\3 #\,
                            #\2 #\0 #\1 #\2 #\0 #\3 #\0 #\5)))
                '(("GOOG" "20121113" "20120305")))

  (check-expect (split-csv-style '(
                                  (#\G  #\O  #\O  #\G  #\,
                         #\2  #\0  #\1  #\2  #\0  #\1  #\0  #\5  #\,
                              #\2  #\0  #\1  #\2  #\0  #\3  #\0  #\5)
                            (#\A  #\M  #\Z  #\N  #\,
                          #\2  #\0  #\1  #\2  #\0  #\1  #\0  #\5  #\,
                              #\2  #\0  #\1  #\2  #\0  #\3  #\0  #\5)
                             (#\Y  #\H  #\O  #\O  #\,
                          #\2  #\0  #\1  #\2  #\0  #\1  #\0  #\5  #\,
                              #\2  #\0  #\1  #\2  #\0  #\3  #\0  #\5)
                             (#\A  #\P  #\P  #\L  #\,
                         #\2  #\0  #\1  #\2  #\0  #\1  #\0  #\5  #\,
                         #\2  #\0  #\1  #\2  #\0  #\3  #\0  #\5)
                            (#\V  #\Z  #\W  #\,
                         #\2  #\0  #\1  #\2  #\0  #\1  #\0  #\5  #\,
                              #\2  #\0  #\1  #\2  #\0  #\3  #\0  #\5)
                               (#\G  #\O  #\O  #\G  #\,
                         #\2  #\0  #\1  #\2  #\0  #\1  #\0  #\5  #\,
                              #\2  #\0  #\1  #\2  #\0  #\3 #\0 #\5)))
                '(("GOOG" "20120105" "20120305")
                  ("AMZN" "20120105" "20120305")
                  ("YHOO" "20120105" "20120305")
                  ("APPL" "20120105" "20120305")
                  ("VZW" "20120105" "20120305")
                  ("GOOG" "20120105" "20120305"))


(check-expect (to-search-structure nil) nil)

  (check-expect (read-req-file nil) nil

  (defrandom random-date ()
    (random-between 19000101 20130101))

  (defrandom random-cp ()
    (random-between 50 700))
  (defrandom random-tick ()
    (random-string))

  (defrandom random-tree-input ()
    (list (random-tick) (random-date) (random-tick)))

  (defrandom random-tree-input-list ()
    (random-list-of (random-tree-input)))
  )

(link Tread-sort
      (Mavl-string-keys Mread-sort Tread-sort-private))

(invoke Tread-sort)
```

Tcalc-slope-intercept.lisp

```
;75 Chars
*****************************************************************

;Team Van Rossum
;Software Engineering 1
;Testing for the slope and intercept module

(require "mcalc_slope_intercept.lisp")
(require "thelper.lisp")

(module Tstcalc_slope_intercept
  (import Iavl-string-keys)
  (import Ilinear_regression_functions)
  (import Ihelper)
  (import Icalc_slope_intercept)
  (include-book "testing" :dir :teachpacks)
  (include-book "doublecheck" :dir :teachpacks)

  ; Test suite for avl-flatten-both
  (check-expect (avl-flatten-both (make-subtree))
                '(("20120101" nil)
                  ("20120202" nil)
                  ("20130101" nil)))
  (check-expect (avl-flatten-both (make-different-subtree))
                '(("20080101" nil)
                  ("20090101" nil)
                  ("20090202" nil)))
  (check-expect (avl-flatten-both (make-tree)) '(("GOOG"
                               (("20120101" . "2.0")
                               ("20120202" . "3.0")
                               ("20130101" . "1.0")))))
  (check-expect (avl-flatten-both (make-complex-tree))
                '(("AA"(("20120101" . "2.0") ("20120202" . "3.0")
                                             ("20130101" . "1.0")))
                  ("AAPL" (("20120101" . "2.0") ("20120202" . "3.0")
                                                ("20130101" .
"1.0")))
                  ("GOOG" (("20120101" . "2.0") ("20120202" . "3.0")
                                                ("20130101" .
"1.0")))))
```

```
  ; Test suite for build-total-date-tree
  (check-expect (build-total-date-tree (avl-flatten-both (make-
tree)))
                '(2 "20120202" 3 (1 "20120101" 2 nil nil)
                    (1 "20130101" 1 nil nil)))
 (check-expect (build-total-date-tree (avl-flatten-both
                                        (make-complex-tree)))
                '(2 "20120202" 9 (1 "20120101" 6 nil nil)
                    (1 "20130101" 3 nil nil)))

  ; Test suite for get_xs
  (check-expect (get_xs (build-total-date-tree
                (avl-flatten-both (make-tree))) 1)
                '(1 2 3 4 5))
  (check-expect (get_xs (build-total-date-tree
             (avl-flatten-both (make-complex-tree))) 1)
                '(1 2 3 4 5))
  (check-expect (get_xs (avl-flatten (build-total-date-tree
                      (avl-flatten-both (make-tree)))) 1)
                '(1 2 3))
  (check-expect (get_xs (avl-flatten (build-total-date-tree
                 (avl-flatten-both (make-complex-tree)))) 1)
                '(1 2 3))

  ; Test suite for get_ys
  (check-expect (get_ys (avl-flatten
        (build-total-date-tree (avl-flatten-both (make-tree)))))
                '(2 3 1))
  (check-expect (get_ys (avl-flatten
        (build-total-date-tree (avl-flatten-both (make-complex-
tree)))))
                '(6 9 3))
  )

(link Tcalc_slope_intercept
      (Mavl-string-keys Mlinear_regression_functions
                        Thelper Mcalc_slope_intercept
                        Tstcalc_slope_intercept))

(invoke Tcalc_slope_intercept)
```

Example Input and Output

Sample Input

Example Hist.txt:

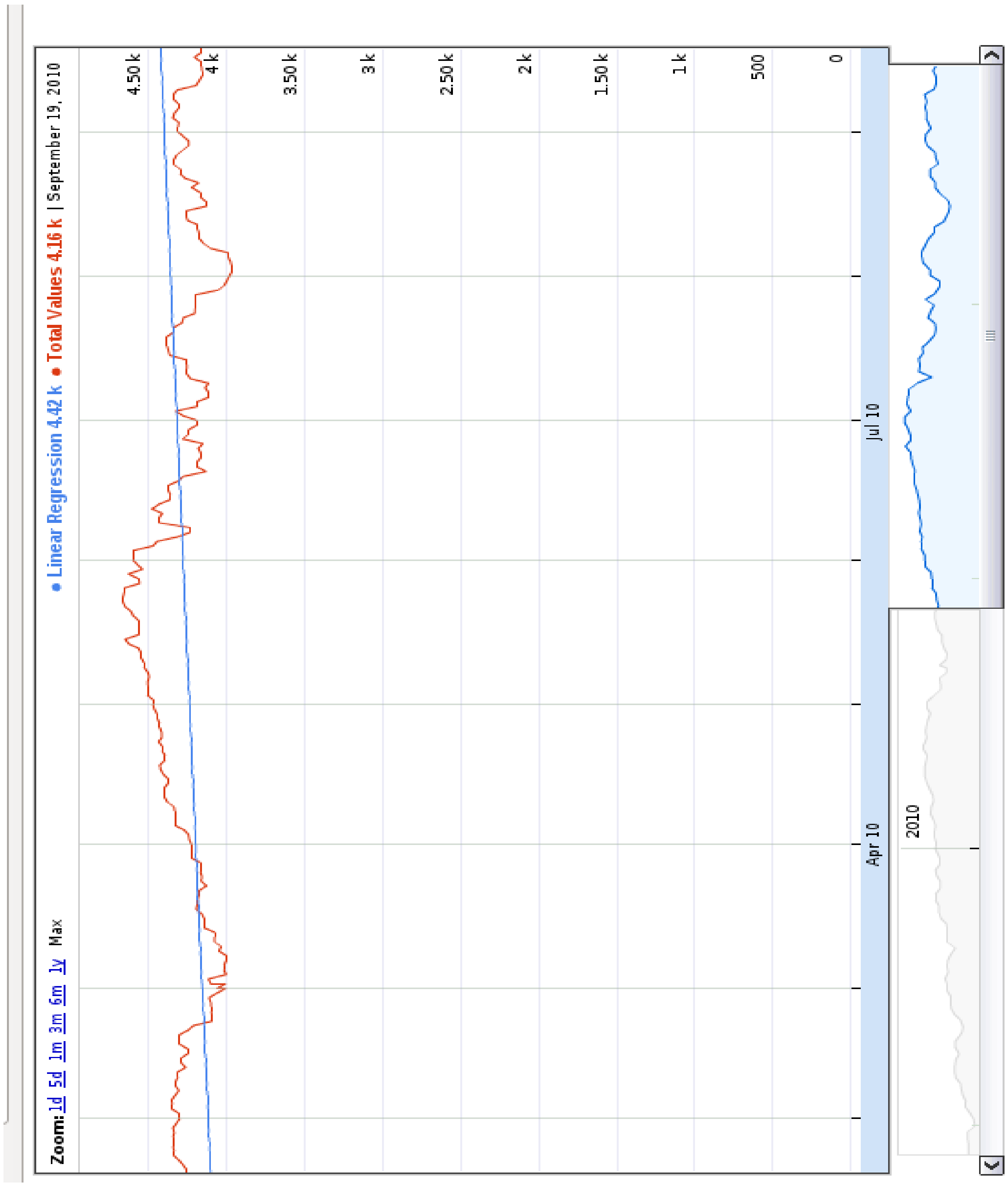
```
<sr>
      <tk>AA</tk>
      <td>20090821</td>
      <hp>12.73</hp>
      <lp>12.49</lp>
      <op>12.64</op>
      <cp>12.56</cp>
      <tr>338295</tr>
</sr>
<sr>
      <tk>AA</tk>
      <td>20090824</td>
      <hp>12.83</hp>
      <lp>12.36</lp>
      <op>12.76</op>
      <cp>12.42</cp>
      <tr>307627</tr>
</sr>
<sr>
      <tk>AA</tk>
      <td>20090825</td>
      <hp>12.66</hp>
      <lp>12.3</lp>
      <op>12.57</op>
      <cp>12.35</cp>
      <tr>246836</tr>
</sr>
```

Sample Analysis.txt:

```
<AR>ADI,20090821,20100820</AR>
<AR>ADP,20090821,20100820</AR>
<AR>ADSK,20090821,20100820</AR>
<AR>AKAM,20090821,20100820</AR>
<AR>ALTR,20090821,20100820</AR>
<AR>AMD,20090821,20100820</AR>
<AR>AMZN,20090821,20100820</AR>
<AR>ANF,20090821,20100820</AR>
<AR>APA,20090821,20100820</AR>
<AR>AVP,20090821,20100820</AR>
<AR>AZO,20090821,20100820</AR>
<AR>BBBY,20090821,20100820</AR>
<AR>BBY,20090821,20100820</AR>
<AR>BDK,20090821,20100820</AR>
```

Sample Output

(Note input does not match output they are used for example)



PSP Logs

Team Log

name: Van Rossum
date: December 2, 2012
program: tProject
instructor: Dr. Page

actual base lines: 755
actual added lines: 1033
actual modified lines: 0
actual removed lines: 0

time log:

```
    - date: November 1, 2012
      start time: 3:02PM
      end time: 4:16PM
      phase: Group
      comment: Working on initial design of project.

    - date: November 6, 2012
      start time: 3:10PM
      end time: 4:01PM
      phase: Group
      comment: Revising the design of the project.

    - date: November 20, 2012
      start time: 3:07PM
      end time: 3:46PM
      phase: Group
      comment: Delegating work and final design decisions.

    - date: November 27, 2012
      start time: 3:01PM
      end time: 4:15PM
      phase: Group
      comment: Group work day.

    - date: November 29, 2012
      start time: 3:11PM
      end time: 4:18PM
      phase: Group
      comment: Group work day.
```

new objects:

```
- name: Wrapper_main_function and Module by Jakob
  estimated lines: 175
  type: Module

- name: Read_and_parse and Module by Femi
  estimated lines: 175
  type: Module

- name: Read_and_proune and Module by Shane
  estimated lines: 175
  type: Module

- name: Linear_regression and Module by Clay
  estimated lines: 175
  type: Module

- name: HTML_output by and Module Cezar
  estimated lines: 175
  type: Module
```

reused objects:

```
- name: Wrapper_main_function by Jakob
  estimated base: 151
  type: Function_Set

- name: Read_and_parse by Femi
  estimated base: 151
  type: Function_Set

- name: Read_and_proune by Shane
  estimated base: 151
  type: Function_Set

- name: Linear_regression by Clay
  estimated base: 151
  type: Function_Set

- name: HTML_output by Cezar
  estimated base: 151
  type: Function_Set
```

defect log:

- date: Nov 23, 2012
  type: Conceptual
  fix time: 26
  comment: Prune is not working with the test data I supply it my key is not working properly fixed appropriately

- date: Nov 23, 2012
  type: Conceptual/Technical
  fix time: 22
  comment: PAckets is not working the way I expected it to so I fixed it to work with characters rather than string

- date: Nov 24, 2012
  type: Conceptual/Technical
  fix time: 26
  comment: Parsing of the <AR> is not working the way I expected it to still have a AR>DATA..... in there my fix was to just trim off the front part

- date: Nov 24, 2012
  type: Conceptual/Technical
  fix time: 103
  comment: My data structure is not being put together correctly in the parse function so when it is pased to prune it has (GOOG DATE1 DATE2) rather than (GOOG (DATE1) (DATE2))

- date: Nov 27, 2012
  type:Technical
  fix time: 32
  comment: Cannot test correctly if my pruning function is working made some sample input and tested worked correctly after minor fix i car'd when u shoulda caar'd

- date: Nov 27, 2012
  type:Technical
  fix time: 103
  comment: Date is not going to next year problem was that it did not account for the 0101 for january 1st fixed

- date: Nov 27, 2012
  type: Technical
  fix time: 15
  comment: Date formatting with years and months not working properly because I was expecting an integer to be passed changed to string

- date: Nov 28, 2012

type: Conceptual/Technical
fix time: 21
comment: Problem occuring with interface not meshing with my module reason was because I did not do the require at the top

- date: Nov 27, 2012
  type: bug
  fix time: 15
  comment: String injection was not injecting anything. Was off by one.

- date: Nov 27, 2012
  type: bug
  fix time: 5
  comment: RegressionList was returning incorrect values.

- date: Dec 2, 2012
  type: bug
  fix time: 10
  comment: contract had multiple syntax errors that were not clearly visible.

- date: Dec 2, 2012
  type: bug
  fix time: 13
  comment: After converting to modular form, there were some issues with linking the software properly.

- date: Nov 26, 2012
  type: logical
  fix time: 1
  comment: Accidentally fetched shares traded instead of closing price for the second field in extract-fields. Simply changed a 6 to a 5 to correct it.

- date: Nov 25, 2012
  type: coding
  fix time: 5
  comment: got errors using the chrs->str function because I attempted to pass in a list of characters instead of a list of lists of charachters

- date: Nov 26, 2012
  type: Build Tree
  fix time: 15
  comment: program crashes on attempt to build the AVL tree. Need to reformat the features of the tree

- date: Nov 28, 2012

type: AVL Tree
fix time: 20
comment: when building the AVL tree, stock records with the same name are added only once to the tree, and everything else ignored. Fixed by putting stock records with the same names as subtrees in one stock record of the same name

- date: November 1, 2012
  type: People
  fix time: 1
  comment: Unable to write wrapper without stubs.

- date: November 30, 2012
  type: Code
  fix time: 2
  comment: Delegation prune should be recursive.

- date: November 30, 2012
  type: Code
  fix time: 1
  comment: Use the CDR of avl-retrieve to get the datum. not the cadr.

- date: November 30, 2012
  type: Code
  fix time: 7
  comment: Use str->int not dgts->int.

- date: November 30, 2012
  type: Code
  fix time: 2
  comment: Cannot add 1 to a string without str->int first.

- date: November 30, 2012
  type: Code
  fix time: 8
  comment: When making a tree, use strings or ints always as key, don't mix the two.

- date: November 30, 2012
  type: Code
  fix time: 1
  comment: Double check equal comparisons with trees.

- date: November 30, 2012
  type: Code
  fix time: 4
  comment: Cannot make straight int comparisons with dates. they must be compared

with a date comparitor.

```
- date: November 30, 2012
  type: Code
  fix time: 25
  comment: Be sure to flatten datum at the same time as the tree when avl-flattening.

- date: November 30, 2012
  type: Code
  fix time: 2
  comment: The difference between empty-tree and return-tree is one is empty (duh).

- date: November 30, 2012
  type: Code
  fix time: 6
  comment: Old values are not to be trusted. Ever.

- date: November 30, 2012
  type: Code
  fix time: 1
  comment: Dont muddle HTML output with data. Make it clean with strings.

- date: November 30, 2012
  type: Code
  fix time: 10
  comment: Linear regression should be based on the count, not actual values of xs.

- date: December 2, 2012
  type: Code
  fix time: 13
  comment: Be sure to give everything a link.

- date: December 2, 2012
  type: Code
  fix time: 62
  comment: Don't double link anything except for the final runnable module.

- date: December 2, 2012
  type: Code
  fix time: 2
  comment: Also dont use in-acl2 with modular c
```

Shane's Log

name: Shane Moore
date: Nov 11,2012
program: tProject Stocks
instructor: Dr. Page
actual added lines: 294

time log:

- date: Oct 30, 2012
  start time: 3:00PM
  end time: 4:13PM
  phase: design
  comment: Worked with team to come up with initial design for project. Discussed data structures, I/O, and parsing methods.

- date: Nov 1, 2012
  start time: 3:09PM
  end time: 4:14PM
  phase: design
  comment: Worked with the team to design the flow and layout of the paper and project

- date: Nov, 5 2012
  start time: 4:30PM
  end time: 7:30PM
  phase: documentation
  comment: Worked on the format for the design document waiting to discuss with team what else needs to be put in before we hand in

- date: Nov 6, 2012
  start time: 3:08PM
  end time: 4:15PM
  phase: design
  comment: Team design documentation, and decided who would be working on which modules.

- date: Nov 20, 2012
  start time: 3:00PM
  end time: 4:15PM
  phase: planning
  comment: Discussed who would work on which parts of the project, and what code each person would need to write.

- date: Nov 22,2012
  start time: 5:26PM
  end time: 7:13PM
  phase: Code/Design
  comment: Spending time on my flight to Phoenix writing the initial stubs for code wrote prune, parse and read in parts and writing out design

- date: Nov 23,2012
  start time: 8:41PM
  end time: 10:46PM
  phase: Code
  comment: Wrote the part for the prune works good for input I am giving it need to put to gether with rest of projects

- date: Nov 24,2012
  start time: 8:54AM
  end time: 9:32AM
  phase: Code
  comment: Wrote the read in and set it up to be passed to the parsing function then to prune

- date: Nov 24,2012
  start time: 10:36AM
  end time: 11:46AM
  phase: Code
  comment: Stuck on the parsing it is not working the way I am expecting so I am going to take a break eat a sandwhich and come back to it

- date: Nov 24,2012
  start time: 1:14PM
  end time: 3:06PM
  phase: Code
  comment: Got all the parsing done and it is working being passed to the prune function committing

- date: Nov 27,2012
  start time: 3:00PM
  end time: 4:15PM
  phase: Code
  comment: Team meeting hashing out the minor details of who does what and did some coding as well

- date: Nov 27,2012
  start time: 4:53PM
  end time: 6:02PM

phase: Code
comment: Working on the pruning of the tree went well got the functions needed to work with parsing module

- date: Nov 27,2012
  start time: 08:33PM
  end time: 11:32PM
  phase: Code
  comment: Did all of the date formatting functions needed for the project so we can traverse the tree right

- date: Nov 28,2012
  start time: 2:24PM
  end time: 5:23PM
  phase: Code
  comment: Added check-expects to my part of the project all tests went through correctly.

- date: Nov 28,2012
  start time: 11:15PM
  end time: 2:45PM
  phase: Code
  comment: Modularized all of my functions and tests so the project will mesh together with the rest of the modules

- date: Nov 29,2012
  start time: 3:00PM
  end time: 4:15PM
  phase: Meeting
  comment: Team meeting today worked out minor bugs everyone was having so we can put all the modules together

- date: Dec 2,2012
  start time: 1:30PM
  end time: 2:45PM
  phase: Documentation
  comment: Wrote the majority of the documentation for Timpl just need to add the psp out-put and can print

defect log:
- date: Nov 23, 2012
  type: Conceptual
  fix time: 26

comment: Prune is not working with the test data I supply it my key is not working properly fixed appropriately

- date: Nov 23, 2012
  type: Conceptual/Technical
  fix time: 22
  comment: PAckets is not working the way I expected it to so I fixed it to work with characters rather than string

- date: Nov 24, 2012
  type: Conceptual/Technical
  fix time: 26
  comment: Parsing of the <AR> is not working the way I expected it to still have a AR>DATA..... in there my fix was to just trim off the front part

- date: Nov 24, 2012
  type: Conceptual/Technical
  fix time: 103
  comment: My data structure is not being put together correctly in the parse function so when it is pased to prune it has (GOOG DATE1 DATE2) rather than (GOOG (DATE1) (DATE2))

- date: Nov 24, 2012
  type: Conceptual/Technical
  fix time: 103
  comment: My data structure is not being put together correctly in the parse function so when it is pased to prune it has (GOOG DATE1 DATE2) rather than (GOOG (DATE1) (DATE2))

- date: Nov 27, 2012
  type:Technical
  fix time: 32
  comment: Cannot test correctly if my pruning function is working made some sample input and tested worked correctly after minor fix i car'd when u shoulda caar'd

- date: Nov 27, 2012
  type:Technical
  fix time: 103
  comment: Date is not going to next year problem was that it did not account for the 0101 for january 1st fixed

- date: Nov 27, 2012
  type: Technical
  fix time: 15

comment: Date formatting with years and months not working properly because I was expecting an integer to be passed changed to string

- date: Nov 28, 2012
  type: Conceptual/Technical
  fix time: 21
  comment: Problem occuring with interface not meshing with my module reason was because I did not do the require at the top

new objects:

- name: read-req-file
  estimated lines: 6
  type: io

- name: parse-analysis-req
  estimated lines: 10
  type: non-io

- name: to-search-structure
  estimated lines: 7
  type: non-io

- name: split-csv-style
  estimated lines: 5
  type: non-io

- name: fmt-date-helper
  estimated lines: 4
  type: non-io

- name: check-day
  estimated lines: 3
  type: non-io

- name: fmt-date
  estimated lines: 5
  type: non-io

- name: prune
  estimated lines: 10
  type: non-io

```
- name: get-by-dates
  estimated lines: 15
  type: non-io
```

Jakob's Log

name: Jakob Griffith
date: November 1, 2012
program: tProject
instructor: Dr. Page

actual base lines: 513
actual added lines: 204
actual modified lines: 26
actual removed lines: 0

time log:

- date: November 1, 2012
  start time: 3:02PM
  end time: 4:16PM
  phase: Group
  comment: Working on initial design of project.

- date: November 6, 2012
  start time: 3:10PM
  end time: 4:01PM
  phase: Group
  comment: Revising the design of the project.

- date: November 20, 2012
  start time: 3:07PM
  end time: 3:46PM
  phase: Group
  comment: Delaging work and final design decisions.

- date: November 27, 2012
  start time: 3:01PM
  end time: 4:15PM
  phase: Group
  comment: Group work day, I layed out my main functions.

- date: November 29, 2012
  start time: 3:11PM
  end time: 4:18PM
  phase: Group
  comment: Group work day, I worked on changing avl rational to avl string keys.

- date: November 30, 2012

start time: 11:01AM
end time: 12:20PM
phase: Individual
comment: Working on combining everyones work together. Specifically Clay and Femi's code.

- date: November 30, 2012
  start time: 3:56PM
  end time: 10:42PM
  phase: Individual
  comment: After my break I resumed meshing work together. Specifically the above with Shane, my, and Cezar's code.

- date: December 2, 2012
  start time: 4:17PM
  end time: 7:30PM
  phase: Individual
  comment: Once everyones work was together, I modularized it.

new objects:

- name: Wrapper_main_function
  estimated lines: 10
  type: Function

- name: Slope intercept calculator function
  estimated lines: 20
  type: Function

- name: Sum of ticker cp function
  estimated lines: 20
  type: Function

- name: Nearest date calculator function
  estimated lines: 25
  type: Function

reused objects:

- name: Linear_regression_functions
  estimated base: 157
  type: Function

- name: AVL-rational-keys

estimated base: 356
type: Function

defect log:

- date: November 1, 2012
  type: People
  fix time: 1
  comment: Unable to write wrapper without stubs.

- date: November 30, 2012
  type: Code
  fix time: 2
  comment: Delegation prune should be recursive.

- date: November 30, 2012
  type: Code
  fix time: 1
  comment: Use the CDR of avl-retrieve to get the datum. not the cadr.

- date: November 30, 2012
  type: Code
  fix time: 7
  comment: Use str->int not dgts->int.

- date: November 30, 2012
  type: Code
  fix time: 2
  comment: Cannot add 1 to a string without str->int first.

- date: November 30, 2012
  type: Code
  fix time: 8
  comment: When making a tree, use strings or ints always as key, don't mix the two.

- date: November 30, 2012
  type: Code
  fix time: 1
  comment: Double check equal comparisons with trees.

- date: November 30, 2012
  type: Code
  fix time: 4
  comment: Cannot make straight int comparisons with dates. they must be compared with a date comparitor.

```
- date: November 30, 2012
  type: Code
  fix time: 25
  comment: Be sure to flatten datum at the same time as the tree when avl-flattening.

- date: November 30, 2012
  type: Code
  fix time: 2
  comment: The difference between empty-tree and return-tree is one is empty (duh).

- date: November 30, 2012
  type: Code
  fix time: 6
  comment: Old values are not to be trusted. Ever.

- date: November 30, 2012
  type: Code
  fix time: 1
  comment: Dont muddle HTML output with data. Make it clean with strings.

- date: November 30, 2012
  type: Code
  fix time: 10
  comment: Linear regression should be based on the count, not actual values of xs.

- date: December 2, 2012
  type: Code
  fix time: 13
  comment: Be sure to give everything a link.

- date: December 2, 2012
  type: Code
  fix time: 62
  comment: Don't double link anything except for the final runnable module.

- date: December 2, 2012
  type: Code
  fix time: 2
  comment: Also dont use in-acl2 with modular code.
```

Femi's Log

name: Olufemi Fashanu
date: December 3, 2012
program: groupProject
instructor: Dr. Page
language: Modular ACL2

actual added lines: 23
actual base lines: 29

time log:

- date: Oct 30, 2012
  start time: 3:05PM
  end time: 4:11PM
  phase: coding
  comment: made initial design plans for the team design review. Layed out plans for each part of the project.

- date: Nov 1, 2012
  start time: 3:07PM
  end time: 4:04PM
  phase: coding
  comment: expanded on each part of the design and detailed the plans for each part.

- date: Nov 6, 2012
  start time: 3:03PM
  end time: 4:12PM
  phase: coding
  comment: Finalized the final team design document.

- date: Nov 20, 2012
  start time: 3:04PM
  end time: 4:10PM
  phase: coding
  comment: Finalized plans to complete the project, and deligated taks to each member based on the team design.

- date: Nov 25, 2012
  start time: 10:22PM
  end time: 11:21PM
  phase: coding
  comment: Worked on splitting the information from the stock record into three tags (tickers, closing price and trade date)

- date: Nov 27, 2012
  start time: 3:06PM
  end time: 4:12PM
  phase: coding
  comment: worked on code to build the AVL tree.

- date: Nov 29, 2012
  start time: 3:12PM
  end time: 4:17PM
  phase: coding
  comment: defined a couple of helper functions to build the tree accoding to the document specifications; i.e creating sub-trees to as the data for nodes in the trees keyed off the same key.

new objects:

- name: insert-into-tree
  estimated lines: 7
  type: AVL tree helper

- name: delegate-into-tree
  estimated lines: 7
  type: AVL tree helper

- name: parse-input
  estimated lines: 15
  type: AVL tree

defect log:

- date: Nov 25, 2012
  type: coding
  fix time: 5
  comment: got errors using the chrs->str function because I attempted to pass in a list of characters instead of a list of lists of charachters

- date: Nov 26, 2012
  type: Build Tree
  fix time: 15
  comment: program crashes on attempt to build the AVL tree. Need to reformat the features of the tree

- date: Nov 28, 2012
  type: AVL Tree
  fix time: 20
  comment: when building the AVL tree, stock records with the same name are added only once to the tree, and everything else ignored. Fixed by putting stock records with the same names as subtrees in one stock record of the same name

Cezar's Log

name: Cezar Delucca
date: Oct 30, 2012
program: ACL2 Stock Analysis
instructor: Dr. Page
language: ACL2

actual added lines: 64
actual base lines: 55
actual modified lines: 24
actual removed lines: 13

time log:

```
 - date: Oct 30, 2012
   start time: 3:00PM
   end time: 4:13PM
phase: plan
   comment: Began brainstorming and learning project requirements.

 - date: Nov 1, 2012
   start time: 3:00PM
   end time: 4:18PM
phase: plan
   comment: Continued with initial design and completed overview of requirements.

 - date: Nov 6, 2012
   start time: 3:01PM
   end time: 4:12PM
phase: plan
   comment: Continued to plan out all of the modules and review design.

 - date: Nov 20, 2012
   start time: 3:04PM
   end time: 4:20PM
phase: plan
   comment: Finished reviewing design and began to delegate modules for each
member of the group.

 - date: Nov 26, 2012
   start time: 4:33PM
   end time: 9:43PM
```

```
          phase: individual portion
            comment: Worked on planning and implementing my portion of the project, the html
generation.

           - date: Nov 27, 2012
            start time: 1:12PM
            end time: 4:16PM
          phase: PSP
            comment: Merged PSP documents to satisfy project requirements and continued
individual implementations.

           - date: Nov 29, 2012
            start time: 9:44AM
            end time: 11:21PM
          phase: Individual work
            comment: Completed the HTML generator and documented the source.

           - date: Nov 29, 2012
            start time: 3:05PM
            end time: 4:18PM
          phase: PSP
            comment: Group meeting to work on our individual portions and provided aid to those
who needed it.

           - date: Nov 30, 2012
            start time: 3:35PM
            end time: 5:22PM
          phase: modularized
            comment: Converted the HTML generator into modular ACL2 and tested its
functionality.

           - date: Dec 2, 2012
            start time: 4:15PM
            end time: 6:18PM
          phase: testing
            comment: Wrote check-expects and a contract to verify regressionList's correctness.

defect log:

           - date: Nov 27, 2012
            type: bug
            fix time: 15
            comment: String injection was not injecting anything. Was off by one.

           - date: Nov 27, 2012
```

```
    type: bug
    fix time: 5
    comment: RegressionList was returning incorrect values.

  - date: Dec 2, 2012
    type: bug
    fix time: 10
    comment: contract had multiple syntax errors that were not clearly visible.

  - date: Dec 2, 2012
    type: bug
    fix time: 13
    comment: After converting to modular form, there were some issues with linking the
software properly.


new objects:

  - name: data->str
    estimated lines: 12
    type: non-io

  - name: stringBuilder
    estimated lines: 20
    type: non-io

  - name: writeHTML
    estimated lines: 5
    type: IO

  - name: regressionList
    estimated lines: 5
    type: non-io

  - name: check-expect1
    estimated lines: 3
    type: testing

  - name: check-expect2
    estimated lines: 3
    type: testing

  - name: check-expect3
    estimated lines: 3
    type: testing
```

```
- name: regressionList-properties
  estimated lines: 5
  type: contract
```

Clay's Log

name: Cezar Delucca
date: Oct 30, 2012
program: ACL2 Stock Analysis
instructor: Dr. Page
language: ACL2

actual added lines: 64
actual base lines: 55
actual modified lines: 24
actual removed lines: 13

time log:

```
 - date: Oct 30, 2012
   start time: 3:00PM
   end time: 4:13PM
phase: plan
   comment: Began brainstorming and learning project requirements.

 - date: Nov 1, 2012
   start time: 3:00PM
   end time: 4:18PM
phase: plan
   comment: Continued with initial design and completed overview of requirements.

 - date: Nov 6, 2012
   start time: 3:01PM
   end time: 4:12PM
phase: plan
   comment: Continued to plan out all of the modules and review design.

 - date: Nov 20, 2012
   start time: 3:04PM
   end time: 4:20PM
phase: plan
   comment: Finished reviewing design and began to delegate modules for each
member of the group.

 - date: Nov 26, 2012
   start time: 4:33PM
   end time: 9:43PM
phase: individual portion
   comment: Worked on planning and implementing my portion of the project, the html
```

generation.

```
        - date: Nov 27, 2012
          start time: 1:12PM
          end time: 4:16PM
        phase: PSP
          comment: Merged PSP documents to satisfy project requirements and continued
individual implementations.

        - date: Nov 29, 2012
          start time: 9:44AM
          end time: 11:21PM
        phase: Individual work
          comment: Completed the HTML generator and documented the source.

        - date: Nov 29, 2012
          start time: 3:05PM
          end time: 4:18PM
        phase: PSP
          comment: Group meeting to work on our individual portions and provided aid to those
who needed it.

        - date: Nov 30, 2012
          start time: 3:35PM
          end time: 5:22PM
        phase: modularized
          comment: Converted the HTML generator into modular ACL2 and tested its
functionality.

        - date: Dec 2, 2012
          start time: 4:15PM
          end time: 6:18PM
        phase: testing
          comment: Wrote check-expects and a contract to verify regressionList's correctness.

defect log:

        - date: Nov 27, 2012
          type: bug
          fix time: 15
          comment: String injection was not injecting anything. Was off by one.

        - date: Nov 27, 2012
          type: bug
          fix time: 5
          comment: RegressionList was returning incorrect values.

        - date: Dec 2, 2012
          type: bug
          fix time: 10
          comment: contract had multiple syntax errors that were not clearly visible.

        - date: Dec 2, 2012
          type: bug
          fix time: 13
          comment: After converting to modular form, there were some issues with linking the
software properly.


new objects:

        - name: data->str
          estimated lines: 12
          type: non-io

        - name: stringBuilder
          estimated lines: 20
          type: non-io

        - name: writeHTML
          estimated lines: 5
          type: IO

        - name: regressionList
          estimated lines: 5
          type: non-io

        - name: check-expect1
          estimated lines: 3
          type: testing

        - name: check-expect2
          estimated lines: 3
          type: testing

        - name: check-expect3
          estimated lines: 3
          type: testing

        - name: regressionList-properties
```

estimated lines: 5
type: contract

PSP Output

tProject

Personal Software Process Summary

Project Essentials

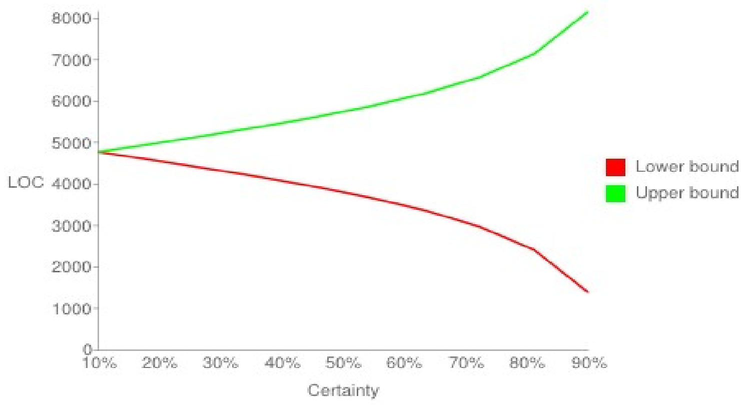
Name:	Van Rossum
Instructor:	Dr. Page
Date:	Dec 2, 2012
Language:	---

Lines of Code

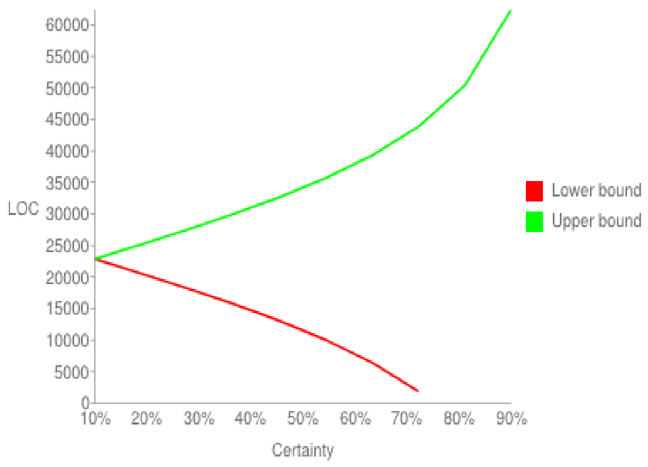
Type	Prediction by user	Actual
Added	875	1033
Base	755	755
Modified	0	0
Removed	0	0

PSP Projection

LoC Certainty

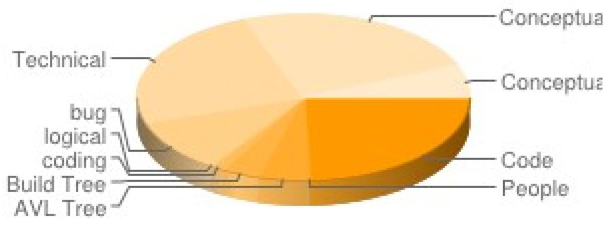


Time Certainty

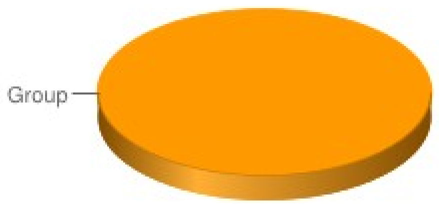


Project Data

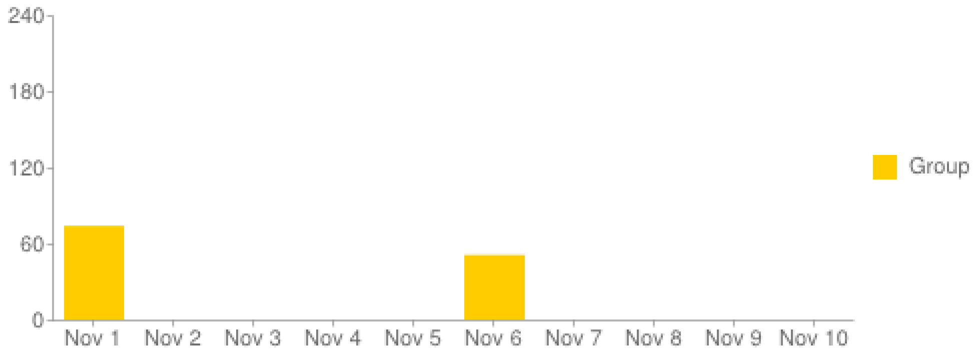
Time Per Defect Type

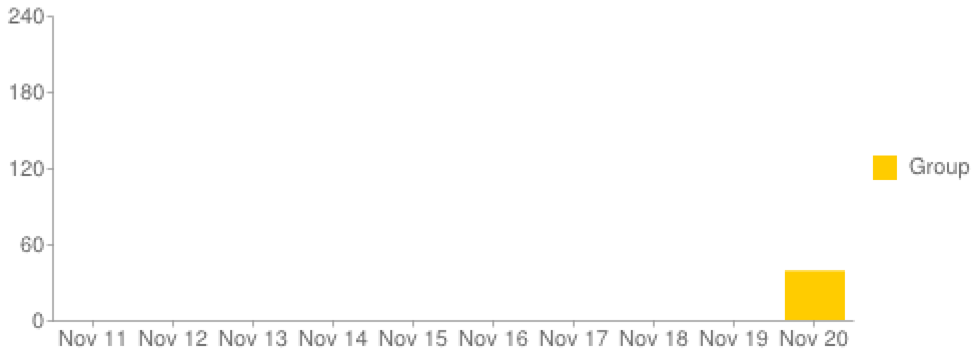


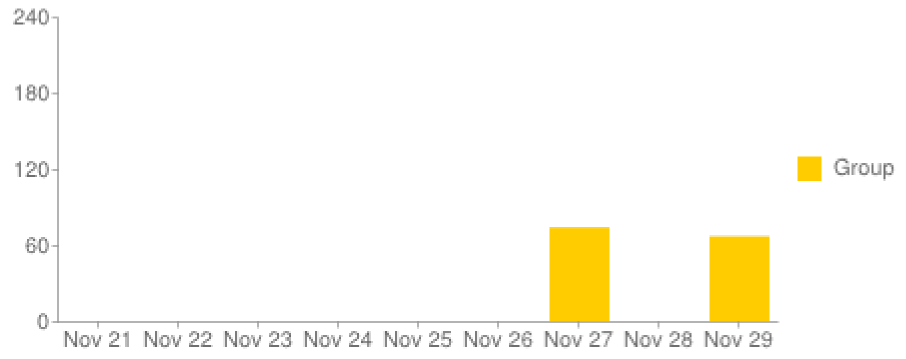
Time Per Phase



Time by Day

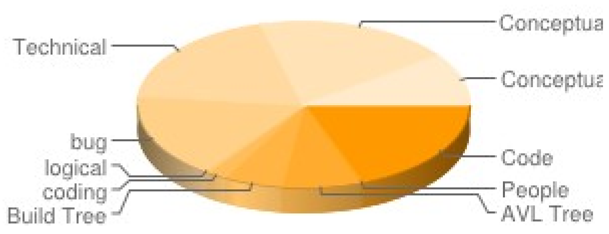




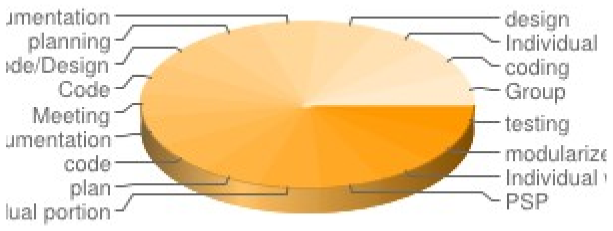


Cumulative Data

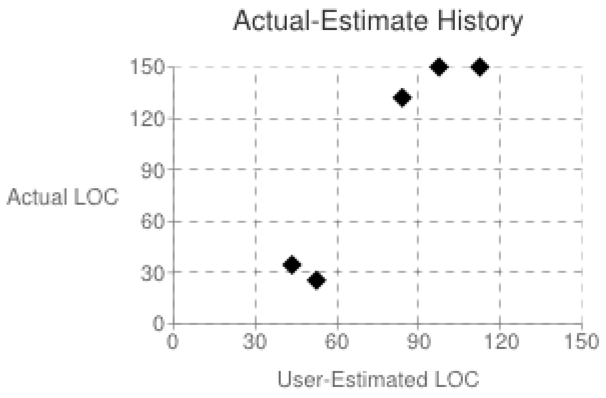
Time Per Defect Type



Time Per Phase



Actual vs Estimated LoC



Actual vs Estimated LoC

Project	Estimate	Actual
groupProject	29	23
tProject	75	230
tProject Stocks	65	294
tProject	35	17
ACL2 Stock Analysis	56	88

Time Log

Date	Type	Int. Time	Description
Nov 1, 2012, 3:02 PM - 4:16 PM	Group	0	Working on initial design of project.
Nov 6, 2012, 3:10 PM - 4:01 PM	Group	0	Revising the design of the project.
Nov 20, 2012, 3:07 PM - 3:46 PM	Group	0	Delegating work and final design decisions.
Nov 27, 2012, 3:01 PM - 4:15 PM	Group	0	Group work day.
Nov 29, 2012, 3:11 PM - 4:18 PM	Group	0	Group work day.

Defect Log

Date	Phase	Fix Time	Description
Nov 23, 2012	Conceptual	26	Prune is not working with the test data I supply it my key is not working properly fixed appropriately
Nov 23, 2012	Conceptual/Technical	22	PAckets is not working the way I expected it to so I fixed it to work with characters rather than string
Nov 24, 2012	Conceptual/Technical	26	Parsing of the
Nov 24, 2012	Conceptual/Technical	103	My data structure is not being put together correctly in the parse function so when it is pased to prune it has (GOOG DATE1 DATE2) rather than (GOOG (DATE1) (DATE2))
Nov 27, 2012	Technical	32	Cannot test correctly if my pruning function is working made some sample input and tested worked correctly after minor fix i car'd when u shoulda caar'd
Nov 27, 2012	Technical	103	Date is not going to next year problem was

			that it did not account for the 0101 for january 1st fixed
Nov 27, 2012	Technical	15	Date formatting with years and months not working properly because I was expecting an integer to be passed changed to string
Nov 28, 2012	Conceptual/Technical	21	Problem occuring with interface not meshing with my module reason was because I did not do the require at the top
Nov 27, 2012	bug	15	String injection was not injecting anything. Was off by one.
Nov 27, 2012	bug	5	RegressionList was returning incorrect values.
Dec 2, 2012	bug	10	contract had multiple syntax errors that were not clearly visible.
Dec 2, 2012	bug	13	After converting to modular form, there were some issues with linking the software properly.
Nov 26, 2012	logical	1	Accidentally fetched shares traded instead of closing price for the second field in extract-fields. Simply changed a 6 to a 5 to correct it.
Nov 25, 2012	coding	5	got errors using the chrs->str function because I attempted to pass in a list of characters instead of a list of lists of charachters
Nov 26, 2012	Build Tree	15	program crashes on attempt to build the AVL tree. Need to reformat the features of the tree
Nov 28, 2012	AVL Tree	20	when building the AVL tree, stock records with the same name are added only once to the tree, and everything else ignored. Fixed by putting stock records with the same names as subtrees in one stock record of the same name
Nov 1, 2012	People	1	Unable to write wrapper without stubs.
Nov 30, 2012	Code	2	Delegation prune should be recursive.
Nov 30, 2012	Code	1	Use the CDR of avl-retrieve to get the datum. not the cadr.
Nov 30, 2012	Code	7	Use str->int not dgts->int.
Nov 30, 2012	Code	2	Cannot add 1 to a string without str->int first.

Nov 30, 2012	Code	8	When making a tree, use strings or ints always as key, don't mix the two.
Nov 30, 2012	Code	1	Double check equal comparisons with trees.
Nov 30, 2012	Code	4	Cannot make straight int comparisons with dates. they must be compared with a date comparitor.
Nov 30, 2012	Code	25	Be sure to flatten datum at the same time as the tree when avl-flattening.
Nov 30, 2012	Code	2	The difference between empty-tree and return-tree is one is empty (duh).
Nov 30, 2012	Code	6	Old values are not to be trusted. Ever.
Nov 30, 2012	Code	1	Dont muddle HTML output with data. Make it clean with strings.
Nov 30, 2012	Code	10	Linear regression should be based on the count, not actual values of xs.
Dec 2, 2012	Code	13	Be sure to give everything a link.
Dec 2, 2012	Code	62	Don't double link anything except for the final runnable module.
Dec 2, 2012	Code	2	Also dont use in-acl2 with modular code.