EduardoI Rodrigues [CS-300-R3289 DSA: Analysis and Design 24EW3](https://learn.snhu.edu/d2l/home/1460831)

# CS 300 Pseudocode Document

## Binary Search Tree Function Signatures

**Load Courses** of a File, F, and the Binary Search Tree, T

for each Course, C, in F

String array, S, is an array of all substrings of C

If the size of S is greater than 2

I is a string of S containing Course ID

I is stored in a linked list IL

P is a String array of S containing Course ID Prerequisites

P is stored in a Linked List PL

Insert C to T

**Inline Display Courses in Order** of C

M is equal to zero

For each P in PL

PC is the size of P

For each prerequisite, p, in P

For each I in IL

if I equal p

M increments by one

if M does not equal to PC

jump to EOF

else M is equal to zero

EOF

**Insert** Course, C, and Node, N, of Tree, T

if N is NULL

N is a new N of T

else

if N is not NULL and C of N is greater than C

if Left of N is NULL

Left of N is a new N of T

else

Insert C to Left of N

else if N is not NULL and C of N is less than C

if Right of N is NULL

Right of N is a new N of T

else

Insert C to Right of N

EOF

**Display Courses in Order** of Node, N, of Tree, T

if N is not NULL

Display Courses in Order of Left of N

print Course, C, of N

Display Courses in Order of Right of N

EOF

**Inline\_Display Courses in Order** of Node, N, of Tree, T

if N is not NULL

Display Courses in Order of Left of N

print Course, C, of N

Display Courses in Order of Right of N

EOF

## **Linked List Function Signatures**

**Link Courses** of a File, F, and the Linked List, L

for each Course, C, in F

String array, S, is an array of all substrings of C

If the size of S is greater than 2

I is a string of S containing Course ID

I is stored in a linked list IL

P is a String array of S containing Course ID Prerequisites

P is stored in a Linked List PL

Append C to L

M is equal to zero

For each P in PL

PC is the size of P

For each prerequisite, p, in P

For each I in IL

if I equal p

M increments by one

if M does not equal to PC

jump to EOF

else M is equal to zero

EOF

**Append** Course, C, and Node, N, of Linked List, L

If Head of L is NULL

Head of L is equal to N containing C

Tail of L is equal to N containing C

Or Else, The next node after Tail of L is N containing C

And Tail of L is N containing C

EOF

**Print List of Courses** of Node, N, of List, L

For each N in L

Print C in N

EOF

**sortList** of IL

For each position I in IL

For each position J in IL

If J+1 is larger

Switch J for J + 1

## **Hash Table Function Signatures**

**Load Courses** of a File, F, and the Hash Table, L

for each Course, C, in F

String array, S, is an array of all substrings of C

If the size of S is greater than 2

I is a string of S containing Course ID

I is stored in a linked list IL

P is a String array of S containing Course ID Prerequisites

P is stored in a Linked List PL

Make Hash Table of C

M is equal to zero

For each P in PL

PC is the size of P

For each prerequisite, p, in P

For each I in IL

if I equal p

M increments by one

if M does not equal to PC

jump to EOF

else M is equal to zero

EOF

**Make Hash Table of** C

Key, K of the Hash Table Vector, V, equals the Hash Function of C

If V at K is empty

Add C to Head of Linked List, L of V at K

Else If V at K is a Collision

While L has nodes, N to point to

Walk through each node

Add C to end of L

EOF

**Print Hash Table**

For each L in V

If L has a single node

Print S substrings in L at head

Else

While L has nodes, N to point to

Walk through each node

Print their respective S substrings

EOF

## Tree Runtime Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Load Courses** | 1 | P\*p\*C^2 | P\*p\*C^2 |
| **Insert** | 1 | C | C |
| **Display Courses in Order** | 1 | C | C |
|  |  |  |  |
|  |  |  |  |
| **Total Cost** | | | C^3\*P\*p |
| **Runtime** | | | O(C^2\*P\*p)  Ω(C\*P\*p) |

## Hash Table Runtime Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Load Courses** | 1 | P\*L^2 | C\*N\*P\*p |
| **Make Hash Table** | 1 | L\*N | C\*N |
| **Print Hash Table** | 1 | L\*N | C\*N |
|  |  |  |  |
|  |  |  |  |
| **Total Cost** | | | C^2\*N^2\*P\*p |
| **Runtime** | | | O(C\*N\*P\*p)  Ω(C\*P\*p) |

## Linked List Runtime Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Link Courses** | 1 | CPp | CPp |
| **Append** | 1 | 1 | 1 |
| **Print List of Courses** | 1 | N | C |
| **SortList** |  |  | I\*J |
|  |  |  |  |
| **Total Cost** | | | C\*N\*P\*p |
| **Runtime** | | | O(C^2\*P\*p)  Ω(C\*P\*p) |

## **Menu Function Signatures**

**Menu of null**

**Display options of user input**

**if load data structure**

Link Courses **of user input**

**if print course list alphabetically**

SortList of

Print List of Courses of IL

**if load data structure**

Print List of Courses of L

**of user input**

**Conculusion**

The best-case scenarios for each data structure for our use case are equivocal. Hash Tables lending themselves to being good for dictionary style searching within a discrete domain. Linked lists lend themselves to being convenient for various sorting strategies. Trees lend themselves to be convenient for gathering information about the list itself. In our implementation we can simply

For this operation, all three operations can be consolidated to the same best-case order, if we print as we add elements to the given data structure. This program structure limits the number operations manipulating the data into a linear order of time complexity. The order in which the comparison of a list of lists of prerequisites to each course in the course list is a matrix order of time complexity. Adding a bubble sorting mechanism will also add an additional matrix order of time complexity.