# Project One

# Alexis Alexander

# Southern New Hampshire University

**Vector Pseudocode**

**Design pseudocode to define how the program opens the file, reads the** **data from the file, parses each line, and checks for file format errors.**

START

OPEN file (Using open function)

READ file (Using getLine)

IF one parameter

THEN alter file line to hold two parameters

ELSE IF prerequisite is not equal to courseNum

THEN print “ERROR with file (prerequisite issue)”

END

| Code | Line Cost | # Times Executes | Total Cost |
| --- | --- | --- | --- |
| Open file | 1 | 1 | 1 |
| Read each line | 1 | n | n |
| If only one parameter read | 1 | n | n |
| Create second parameter | 1 | 1 | 1 |
| Else if course != prerequisite | 1 | n | n |
| Print error message | 1 | 1 | 1 |
| Total Cost | | | 3n + 3 |
| Runtime | | | O(N) |

**Design pseudocode to show how to create course objects and store them** **in the appropriate data structure.**

START

CREATE Vector

WHILE file lines still available

IF file line is in correct format

ADD and STORE File line in Vector

END

| Code | Line Cost | # Times Executes | Total Cost |
| --- | --- | --- | --- |
| Create vector | 1 | 1 | 1 |
| Initialize vector | 1 | 1 | 1 |
| While file lines available | 1 | n+1 | n+2 |
| If file line is in correct format | 1 | n | n |
| Add line in vector | 1 | n | n |
| Total Cost | | | 3n + 4 |
| Runtime | | | O(N) |

**Design pseudocode that will search the data structure for a specific course and print out course information and prerequisites.**

START

IF data in Vector (Linear or Binary search)

Print Course Information

ELSE

Print “Course Data not available”

END

**Design pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order.**

START

LOAD data in Vector

SORT Vector using sort or mixedSort function

PRINT Vector data

**Hash Table Pseudocode**

**Design pseudocode to define how the program opens the file, reads the** **data from the file, parses each line, and checks for file format errors.**

START

OPEN file (Using open function)

READ file (Using getLine)

IF one parameter,

THEN alter file line to hold two parameters

ELSE IF prerequisite is not equal to courseNumber,

THEN print “ERROR with file (prerequisite issue)”

END

| Code | Line Cost | # Times Executes | Total Cost |
| --- | --- | --- | --- |
| Open file | 1 | 1 | 1 |
| Read each line | 1 | n | n |
| If only one parameter read | 1 | n | n |
| Create second parameter | 1 | 1 | 1 |
| Else if course != prerequisite | 1 | n | n |
| Print error message | 1 | 1 | 1 |
| Total Cost | | | 3n + 3 |
| Runtime | | | O(N) |

**Design pseudocode to show how to create course objects and store them in the appropriate data structure.**

START

CREATE and INITIAIZE Vector to store Bids

CALCULATE Hash Value

WHILE file lines still available

IF file line is in correct format

ADD and STORE Bid in Vector

END

| Code | Line Cost | # Times Executes | Total Cost |
| --- | --- | --- | --- |
| Create vector | 1 | 1 | 1 |
| Initialize vector | 1 | 1 | 1 |
| Calculate hash value | 1 | 1 | 1 |
| While file lines available | 1 | n+1 | n+2 |
| If file line is in correct format | 1 | n | n |
| Add line in vector | 1 | n | n |
| Total Cost | | | 3n + 5 |
| Runtime | | | O(N) |

**Design pseudocode that will search the data structure for a specific course and print out course information and prerequisites.**

START

IF Entry Found in Key

RETURN the course information

IF No Entry Found in Key

RETURN searched for course number

Print “Course Data not available”

WHILE the Entry is not assigned to a null pointer and you are currently on that node

IF the Entry is a match

RETURN the course information

END

**Design pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order.**

START

LOAD data in Hash Table

SORT Hash Table using sort naturally or sort algorithm

PRINT sorted Hash Table

**Binary Tree Pseudocode**

**Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors.**

START

OPEN file (Using open function)

READ file (Using getLine)

IF one parameter, then alter file line to hold two parameters

ELSE IF prerequisite is not equal to courseNumber, then print “ERROR with file (prerequisite issue)”

| Code | Line Cost | # Times Executes | Total Cost |
| --- | --- | --- | --- |
| Open file | 1 | 1 | 1 |
| Read each line | 1 | n | n |
| If only one parameter read | 1 | n | n |
| Create second parameter | 1 | 1 | 1 |
| Else if course != prerequisite | 1 | n | n |
| Print error message | 1 | 1 | 1 |
| Total Cost | | | 3n + 3 |
| Runtime | | | O(N) |

**Design pseudocode to show how to create course objects and store them in the appropriate data structure**.

START

CREATE and INITIAIZE Binary Search Tree

CREATE root

INSERT Bids into Binary Search Tree

WHILE file lines still available

IF course is greater than root

INSERT to the left

IF course is less than root

INSERT to the right

END

| Code | Line Cost | # Times Executes | Total Cost |
| --- | --- | --- | --- |
| Create Binary Search Tree | 1 | 1 | 1 |
| Initialize Binary Search Tree | 1 | 1 | 1 |
| Create Root | 1 | 1 | 1 |
| While file lines available | 1 | n+1 | n+2 |
| If course is greater the root | 1 | n | n |
| Insert course to left of root | 1 | n | n |
| If course is less than the root | 1 | n | n |
| Insert course to the right of root | 1 | n | n |
| Total Cost | | | 5n + 5 |
| Runtime | | | O(N) |

**Design pseudocode that will search the data structure for a specific course and print out course information and prerequisites.**

START

WHILE current Node is not NULL

IF Entry Found in Key

RETURN the course information

IF No Entry Found in Key

TRAVERSE down both sides of tree

IF Key found

RETURN the course information

ELSE

Print “Course Data not available”

END

**Design pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order.**

START

LOAD data in Binary Search Tree

CALL inOrder to root

PRINT Binary Search Tree

**Create pseudocode for a menu.**

**START**

**IF user Chooses 1**

**LOAD data structure**

**IF user Chooses 2**

**PRINT course list using print function that sorts courses**

**IF user Chooses 3**

**PRINT specific course using print function that searches for a course**

**IF user Chooses 9**

**EXIT**

**Evaluation**

**Based upon the feature of each data structure and the runtime analysis of each function, I would say that the data structures studied are in order of specificity. We have a vector, which is just a list of items of a specific data type. An advantage of a vector is that its simplicity allows us to do just about anything with the data. It’s like an array or dictionary in that its basic duty is to store elements. The main adisadvantage of the vector is that it is an object, so we’d have to worry about memory consumption. During the runtime analysis of the vector functions, it was shown to have the least worst-case running time.**

**We then have the hash table that has a greater worst-case running time than the vector data structure. An advantage f hash tables is that they map values to keys. Hash tables are better for searching for specific data more than other data structures because of this key to value feature. Disadvantages of hash tables include collisions within the table. When there are too many collisions, and this changes the complexity analysis of the hash table. However, this issue can be solved by chaining or probing within the table.**

**The last data structure is the binary search tree (BST). This data structure has the greatest worst-case running time of the three data structures. The greatest advantage of a BST in this case is that they are great at sorting and storing hierarchical data and don’t really have a max for how many nodes that can be stored. Disadvantages of BST’s involve the time that it takes to search through a tree and the runtime for some functions of binary search tree’s can become complex.**

**With this information in mind, my recommendation for the data structure used for this code would be a binary search tree. Although its runtime complexity has the possibility of getting higher than expected, I think it is the best structure for what the client would like to do with the program. We need to load data and ensure that it is formatted correctly which can be done with any data structure. But then we ultimately need to sort and print, search and print, and add data into our structure. I think this would be less complex with a BST as the data being added to the structure would have to be added alphanumerically or in some type of order anyway within the tree. This would make searching less difficult.Unlike a hash table, you don’t have to worry about collisions or too many nodes as the tree are ordered endlessly. Printing will not be difficult as the nodes will already be ordered and we can print the tree in any order that we want with our order functions.**