Pseudocode

{MENU}()
{INPUT_PATH}()
Store input as [PATH]
If no input, load default [PATH]
Loop while [SELECTION] != '9'
{DISPLAY_MENU}()
Store User inpyt [MENU_OPTION]
Store user input [DATA_OPTION]
If [SELECTION] is invalid, throw error
If [SELECTION] == '1'
{LOAD_DATA}() based on [DATA_OPTION]
{DISPLAY_NUM_RECORDS}()
If [SELECTION] == '2'
{VALIDATE_DATA}() based on [DATA_OPTION]
If [SELECTION] == '3'
{USER_SEARCH}()
{PRINT_DATA}() based on [DATA_OPTION] and [USER_SEARCH]
If [SELECTION] == '4'
{PRINT_DATA}() based on [DATA_OPTION]
If [DATA_OPTION] == [VECTOR] or [HASH_TABLE]
{SORT_DATA}() based on [DATA_OPTION]
{PRINT_DATA}() based on [DATA_OPTION]
If [SELECTION] == '9'
Exit the program
{DISPLAY_GOODBYE}()
End

{COURSE}
[COURSE_CODE]
[COURSE_TITLE]
[P_COUNT]
[P_LIST]
{COURSE}()
[COURSE_CODE] = [COURSE_TITLE] = ""
[P_COUNT] = 0
[P_LIST] = ""

{B_TREE}
{NODE}
[COURSE]
[RIGHT]
[LEFT]

[ROOT]
{P_TREE}()
{B_TREE}()

{HASH_TABLE}
{BUCKET}
[COURSE]
[KEY]
[NEXT]
{HASH}()
{PRINT_TABLE}()
Public [List] [HASH_TABLE]

{SORT_DATA}()
Get [DATA], [LOW], [HIGH]
If [LOW] >= [HIGH], return
{PART}()
[LOW_END] == [PART]
{QUICKSORT}() [DATA], [LOW], [LOW_END]
{QUICKSORT}() [DATA], [LOW_END] + 1, [HIGH]
End

{PART}()
Get [DATA], [LOW], [HIGH]
Loop until [LOW] >= [HIGH]
Loop [DATA] [LOW] until element > pivot is found
Overwrite [LOW] with element position
Loop through [DATA] from [LOW]
[LOW] += 1
[HIGH] -= 1
Return [HIGH]
End

{PRINT_DATA}()
If [DATA_OPTION] == [VECTOR]
Loop through [DATA]
Output [COURSE_CODE] and [COURSE_TITLE]
Loop through [P_COUNT]
Output [COURSE_CODE]
Else If [DATA_OPTION] == [BINARY_SEARCH_TREE]
 [NODE] pointer == [ROOT_NODE]
Set == NULL
If [ROOT_NODE] == null, return
Print [COURSE_CODE] and [COURSE_TITLE]

Loop [P_COUNT]
Output [COURSE_CODE]
Else If [DATA_OPTION] == [HASH_TABLE]
[NODE] pointer == first node
Loop
Print [COURSE_CODE] from [COURSE]
Print [COURSE_TITLE] from [COURSE]
Loop through [P_COUNT]
Call {PRINT_COURSE}() passing [P_LIST]
End

Evaluation

Data Output

Data Structure	**Runtime Efficiency**	**Memory Efficiency**
Vector	O(1)	O(N)
Hash Table	O(1) - O(N)	O(N)
Binary Tree	O(log N)	O(N)

Searching

Data Structure	Runtime Efficiency	Memory Efficiency
Vector	O(N)	O(1)
Hash Table	O(1) - O(N)	O(N)
Binary Tree	O(log N) - O(N)	O(N)

Sorting

Data Structure	Runtime Efficiency	Memory Efficiency
Vector	O(N log N)	O(1)
Hash Table	O(N)	O(N)
Binary Tree	O(N)	O(N)

After analyzing the three provided data structures, I would recommend utilizing the binary search tree for this particular project. I say this because the size of the item list is not notably large, meaning any disparity in performance between the two data structures is expected to be minimal and unlikely to significantly affect the overall efficiency of the finalized code.