**Evaluation**

**Big – O Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| Reading File and Creating courses | Line Cost | # of times executed | Total Cost |
| Declare variable | 1 | 1 | 1 |
| Initialize fs fileStream to get contents of file | 1 | 1 | 1 |
| Initialize string line to hold a single line from file | 1 | 1 | 1 |
| Loop while not end of file | 1 | n | n |
| String line = getline(fs) | 1 | n | n |
| Check to make sure we have a line | 1 | n | n |
| Create new Course | 1 | n | n |
| Invoke Parse that is a Course object member previously defined. | n | n | n^2 |
| Push course on courseVector | 1 | 1 | 1 |
| Validate that the prerequisites are courses done with nested loops | n | n | n^2 |
| loop courses.sort() | n | Log n | n log n |
| currCourse.print() | 1 | n | n |
| Total cost | | | 3 + 5n + 2n^2 +  n log n |
| Runtime | | | O(n log n) |

**Vector**

|  |  |  |  |
| --- | --- | --- | --- |
| Creating course Objects | Line Cost | # of times executed | Total Cost |
| int courseNumber | 1 | 1 | 1 |
| string courseName | 1 | 1 | 1 |
| vector<string> prerequisite | 1 | 1 | 1 |
| Parse(string str, char del) | n | n | n^2 |
| vector tempVect = split(str, del) | 1 | n | n |
| for (int i=0 i<tempVect.size i++) | 1 | n | n |
| if (i == 0) | 1 | 1 | 1 |
| this.CourseNumber = atoi(tempVect[i]) | 1 | n | n |
| else if (i == 1) | 1 | 1 | 1 |
| this.CourseName = tempVect[i] | 1 | 1 | 1 |
| else  push on prequist vector | 1 | 1 | 1 |
| Print() | 1 | 1 | 1 |
| cout >> courseNumber >> " " >> courseName >> " " >> prequist.sort(asc) | 1 | 1 | 1 |
| Total Cost | | | 7 + 3n + n^2 |
| Runtime | | | O(n^2) |

**HashTable**

|  |  |  |  |
| --- | --- | --- | --- |
| Creating Course Objects | Line Cost | # of Times run | Total Cost |
| Create key for acourse by hashing acourse's courseNumber | 1 | 1 | 1 |
| Create Node\* node to retrieve node using key | 1 | 1 | 1 |
| If node is null pointer | 1 | 1 | 1 |
| ode\* next = new Node(acourse, key) | 1 | 1 | 1 |
| nodes.insert(nodes.begin() + key, (\*next)) | 1 | 1 | 1 |
| Else  if (current->key == UINT\_MAX) | 1 | 1 | 1 |
| current->key = key | 1 | 1 | 1 |
| current->course = acourse | 1 | 1 | 1 |
| current->next = nullptr | 1 | 1 | 1 |
| Else while (current->next != nullptr) | 1 | n | n |
| current = current->next | 1 | 1 | 1 |
| Total Cost | | | 10 + n |
| Runtime | | | O(n) |

**Binary Search Tree**

|  |  |  |  |
| --- | --- | --- | --- |
| Creating course Objects | Line Cost | # of times executed | Total Cost |
| If node is larger than course’s courseId | 1 | 1 | 1 |
| Add to left subtree | 1 | 1 | 1 |
| If left node is null | 1 | 1 | 1 |
| Create new node and make left pointer | 1 | 1 | 1 |
| Else left node is not null | 1 | 1 | 1 |
| Recursively traverse left sub-tree | 1 | n | n |
| Add node and pass left pointer and course | 1 | 1 | 1 |
| Else  If right node is null | 1 | 1 | 1 |
| Create new node and make right pointer | 1 | 1 | 1 |
| Else right node is not null | 1 | 1 | 1 |
| Recursively traverse left subtree | 1 | n | n |
| Add node and pass right pointer and course | 1 | 1 | 1 |
| Total cost | | | 10 + 2n |
| Runtime | | | O(n) |

**Advantages and disadvantages**

Vector

Advantages:

1. Implementation is fairly easy and straightforward
2. Can be searched in O(logn) time is sorted with binary search
3. Rear insertion can be done in constant time.

Disadvantages:

1. Must be sorted to take utilize to full potential
2. Removing from the front is done in linear time due to shifting of indexes
3. Vectors can cause a lot of overhead as they typically allocate more memory than is required.

HashTable

Advantages:

1. Direct access to items
2. Insert and delete are done in constant time no matter size of table
3. Access time of element is O(1)

Disadvantages:

1. A high number of collisions can cause poor performance
2. Can not avoid collisions
3. Do not allow null values as key.

Tree

Advantages:

1. When balanced they allow for fast insertion and deletion
2. Easy to implement when compared to some other data structures
3. BST have a worse case search of O(n)

Disadvantages

1. Must remain balanced and for optimal performance
2. Use of recursion can cause size and runtime issues