Name: \_\_\_\_Tristan Wong\_\_\_\_\_\_

CSE 212 Data Structure “Cheat Sheet”

*Note: This document should contain your own work. It is a violation of the BYU-Idaho honor code to share this document or copy in whole or in part from another.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Structure | Graphic | Purpose and Example | Time Complexity of Common Operations | |
|  |  |  | Insert | Access |
| Dynamic Array | A screenshot of a tablet  Description automatically generated | Dynamic arrays can grow or shrink in size automatically, allowing adding and removing elements without worrying about the size. Moreover, the methods to operate on a dynamic array are simple. Any items in the dynamic array can be accessed with high efficiency by index.  Example: To-do list, each task is an individual item in the list. When the user completes a task, he can remove it. He can also add another task. As a dynamic array has no capacity restrictions, he can add as many tasks he wants. | At the beginning: O(n)  At the end: O(1) | Find by value: O(n)  Find by index: O(1) |
| Linked List (Doubly-linked) | A screenshot of a tablet  Description automatically generated | Nodes in a doubly-linked list are connected in both directions (to the previous and next node). Connecting in both directions, allows the code to traverse through the list in both directions, boosting efficiency. This provides flexibility as inserting at the beginning and at the end is efficient, allowing it to be operated on a larger scale, but preserving efficiency.  Example: Playlist management, each song contains a reference to the next song and the previous song, so that when the use presses “previous” or “next”, the code can immediately locate the songs around the current song. | At the beginning: O(1)  At the end: O(1) | Find by value: O(n)  Find by index: O(n) |
| Stack | A screenshot of a tablet  Description automatically generated | Last In, First Out. A stack can only insert items at the end and remove them at the end. This allows the data to be tracked in a reversed order, in which a newly added item will be removed with methods for a stack. There is also an “undo” option for a stack, according to the LIFO property, when the item is inserted, but maybe it is a mistake, the items at the end can be removed until the one with the error is removed.  Example: Web browser history, when browsing on the Internet, the stack will keep track of the latest history. If the user presses “back”, the stack can remove the latest one and keep track of the previous page. | O(1) | The top: O(1) |
| Queue | Screens screenshot of a tablet  Description automatically generated | First In, First Out. A queue can only insert items at the end and remove them at the top. This allows the code to keep track of the data in the same order they were added to the queue.  Example: A print job manager, the queue will process the first job entered into the system, while still able to take new jobs and adding to the back of the queue. | O(1) | The front: O(1) |
| Map | Screens screenshot of a tablet  Description automatically generated | A map is a dictionary. It has 2 components, the key and the value. The map allows the code to keep track of the data in pairs, or a value associated with a key. A key must be unique, no duplicates are allowed.  Example: An employee directory, the employee ID can be used as the key, and the employee name can be used as the value. Since a key in a dictionary must be unique, this ensures the ID is only for 1 employee. With a dictionary, the name and ID can be linked together, which will be more convenient when finding 1 of the 2 data. | O(1) | O(1) |
| Balanced BST | Screens screenshot of a tablet  Description automatically generated | A balanced BST allows efficient search due to its balanced height (how many levels of child nodes extended from the root node). As we know the left child node must be smaller than its parent node, and the right child node must be larger than its parent node. The data is structured and sorted.  Example: A library management system, the ISBN can be used as a node. When the librarian needs to search for a certain book by ISBN, since the tree is balanced, it is efficient. | O(log n) | Contains: O(log n) |