Portfolio

=========

Document completion of the course's learning outcomes.

Instructions

====

The goal of this is to make it very easy for me (or an employer, or a teaching assistant) to see whether or not you have mastered the material of the class. The hope is that I would be able to grade your work without downloading and compiling your code. If you have recorded proper video demonstrations of your programs, that should be sufficient. You will also want to write a paragraph or so making your case for why you deserve full credit for particular learning outcome (or if you don't, then you should say so).

Important

=========

If you prefer, you may turn this in to me via email, instead of hosting it on GitHub. Please talk to me about it during office hours or before or after class.

Body of portfolio

====

7 - Create an implementation of a Queue

----

Possible sources of evidence (do any one of these):

I've implemented a queue and uploaded it to GitHub which can be found here:

https://github.com/Vutisat/03\_Queue\_Lab

7 - Create an implementation of a List

----

Possible sources of evidence (do any one of these):

I've implemented a queue and uploaded it to GitHub which can be found here:

https://github.com/Vutisat/04\_Linked\_List\_Lab

7 - Create an implementation of a Binary Search Tree

----

Possible sources of evidence (do any one of these):

I've implemented a BST and uploaded it to GitHub which can be found here:

<https://github.com/MiamiOH-CSE274/06_BST_Lab/blob/af4f0f307d4ae3b34e8bfd5ce9a265ebfb2f66bf/BST.ipp>

7 - Create an implementaiton of a Hash Table

----

I’ve worked on the lab to implement a hashing table, however not the whole lab is fully completed.

This is the link to the lab: <https://github.com/MiamiOH-CSE274/05_Hashing_Lab/tree/vutisat>

7 - Create an implementation of a Heap

----

I've implemented a Heap and uploaded it to GitHub which can be found here:

\*\*not pulled yet, fix link

<https://github.com/Vutisat/07_Heap_Lab/blob/96f496554d5a1eec0a861fa552dfadf186193fa5/Heap.ipp>

7 - Create an implementation of either Adjanency Lists or Adjacency Matrices

----

Possible sources of evidence (do any one of these):

\* Graph lab

\* Graph Algorithms lab

\* Graph project

\* Consult with Dr. Brinkman on an alternative project

7 - Implement graph algorithms

----

Possible sources of evidence (do any one of these):

\* Graph lab

\* Graph Algorithms lab

\* Graph project

\* Consult with Dr. Brinkman on an alternative project

21 - Determine space and time requirements of common data structure methods

-----

1. In the Linked List lab (link above) the add, remove, get, find, methods running time are O(1) if `n == 0` or if are O(n) `n >= 1`. This is because it has to go through each item in the list to the position, in order to add, remove, or get. Let say you wanted to add a new item into a list of 4 items, it will have to start running from the dummy node, use the next and go through each individual link to the right spot and then finally add your item.

The size, or get size function is O(1) because of the way it is implemented, I kept track of how many items there are in the list using a variable. However if there is not tracking of the size, the running time will be O(n) in order to go through all the nodes and count the numbers of items.

1. In the Binary Search Tree the functions find, add, and remove are Ɵ(lg n) if the tree is balanced. This is because when the tree is balanced the left and right branch would be very close to equal and as you move down one branch, you eliminate having to look at half of the tree.   
   Let’s say you want to find an item that is in the far right bottom branch of the tree, when you first start at the top of the tree, you realize that it will be in the right branch… you don’t have to do anything with the left side of the branch and you move down the tree and repeat the process.   
   However in the worst case scenario where the tree is a straight down from the root with no splitting branches the running time would be Ɵ(n) because you will have to go through and look at every items for these functions.
2. In the Queue lab the add() function is constant O(1) time because you can simply assign items into the index right away. However if the queue is full, grow() must be call and that will take O(n) time. It is linear because as the array gets larger and larger the les often the grow() method will have to be called upon. Because the number of items is being tracked with add() and remove(), the method to getNums is also constant O(1) because that variable can be pulled right out. Remove function is also constant like add.

5 - Describe memory management in C++, and correctly use dynamic variables, including destructors

----

In my Linked list lab : <https://github.com/Vutisat/04_Linked_List_Lab>

Unlike Java that has garbage collectors, in C++ programmers must deal with memory dynamically because they’re not automatically handled and taken care of. Unlike an array, the linked list can multiple nodes and the memory allocated does not to be one connected big chunk. Because is node node is separate each node can take up space at different parts of the memory. One node can get to another node using addresses so their place in the computer does not have to be side by side. The dummyNode is used when the list is empty and the beginning and the end to the list of nodes. Memory is first allocated in the constructor when the dummy node is created. More memory will be allocated when the add() function is called and new nodes are being created. Remove deletes the nodes individually when they are called and the memory that is being used for those nodes will be emptied. There are no dangling pointers because every node is taken care of properly. The deconstructor also makes sure that there is no leakage when destroyed. It does this by calling remove() of each node in the list which deletes them. In the end the dummyNode gets deleted in the deconstructor making the list completed emptied.

5 - Create collection classes using templates in C++

----

Templates being used in unique ways can be observed in many labs. The point of using templates in C++ is that it is a placeholder so that one can use other classes easily by replacing them ino the templates. Templates allow you to use the same code for implementation of things (in this case data structures) to deal with different data types. One could use my queue lab and implement data type of strings, or a class of persons, or dogs… whatever type of queue that is needed for the data being input.

Here is my queue lab : https://github.com/Vutisat/03\_Queue\_Lab

30 - Using time and space analysis, justify the selection of a data structure for a given application

----

\* Select a project for which there are multiple reasonable data structure designs. Describe two reasonable options, and explain the trade-offs between them. For each, describe an application where the data structure would be better. For example, if comparing KD-Trees to a Grid in the Starbucks problem, which one is better really depends on the input data set. Explain what the data would have to look like for the Grid to be a clear winner, and also what type of data would lead you to use a KD-Tree instead.

In the shuffle project there are two data structures that would fit to do the job. However there are no advantage in terms of running time between those two; linked list and an array. The add, remove, get, find, methods running time are O(1) if `n == 0` or if are O(n) `n >= 1`. So I chose to go with linked list. I first added the deck into a data structure of linked list then I used splice to split the linked list into two lists. TEST

\*\*Note: I attended the undergraduate research fair for extra credit\*\*