Aarav Dev
axd220001
Section 503

Assignment 1 README

ArrayStack Implementation:

For the implementation of the stack as an array, I approached the solution by making the class have a private array variable which would hold doubles and act as the main stack, a private top variable which would hold the index of the top of the stack, and a private size variable. Also, as instructed, I included a private final int INITIAL_CAPACITY variable which holds the starting amount of space in the stack, which I set as 10. In the default constructor, I set the array as a new double array with size INITIAL_CAPACITY, the top variable to -1, and the size variable to zero. The **resize** method is used when pushing to the stack when the stack is full. This is done by creating a new array which has double the capacity of the old stack and copying over the existing values into the new array, and then setting the created array as the new stack. The upper bound complexity of the resize method is O(N) because a loop through the stack is required when copying over the values, iterating over each of the elements in the stack. The **push** method adds a value to the top of the stack. In my implementation, this is done by first resizing the array if the stack is full, then increasing the top variable by 1, and then setting the the value of the array at index top to the value specified in the parameter of the method and increasing the size variable by 1. The upper bound complexity of the push method is O(N) because in the worst case, the array would need to be resized and the upper bound complexity of the resize method is O(N). The **pop** method returns and removes the value at the top of the stack. This is implemented by first checking if the stack is empty, in which case an EmptyStackException is thrown. If the stack is not empty, the size variable is decreased by 1, the value of top is decreased by 1, and the value at the index of top is returned. This operation takes constant time, so the upper bound complexity of the pop method is O(1). The **peek** method operates similarly to the pop method, except that the value at the top of the stack is not removed. This is done by throwing an EmptyStackException if the stack is empty, and if it is not empty, the value at the index of top is returned. The upper bound time complexity of the peek method is O(1), as it simply returns a known value. The **isEmpty** method returns a boolean representing if the stack is empty or not. This is implemented by checking if the top variable is -1, as that value represents an empty stack. The upper bound time complexity of the isEmpty method is O(1) because it involves only a single comparison. The **count** method simply returns the number of elements in the stack, which

is done by just returning the size variable which is updated with each push and pop call. The upper bound complexity of the count method is O(1) because all that is done is the returning of a value.

ListStack Implementation:

For the implementation of the stack as a linked list, I approached the solution by first creating a private node class within the ListStack class. The node class contained a double variable to hold the value, a pointer to the next node, and a constructor which set the value of the node. The ListStack class has a private node variable which acts as the head of the linked list, and a private int variable to keep track of the size. In the default constructor, the head is set to null and the size to zero. In the **push** method, a new node is created with its value being specified in the parameter. The new node's next pointer is set to the head, and the head is set to the new node, which in essence pushes the value to the top of the linked list. Finally, the size is increased by 1. The upper bound time complexity of the push method is O(1) because it involves updating a constant number of pointers, which takes constant time regardless of the size of the stack. The **pop** method first checks if the list is empty, in which case it throws an EmptyStackException. Otherwise, the value of the head is stored in a temporary variable, the head is set to its next pointer (removing the top value), and the size is decreased by 1. The value stored in the temporary variable is then returned. The upper bound time complexity for the pop method is O(1) because it involves updating a constant number of pointers (except when empty), which takes constant time regardless of the size of the stack. The **peek** method first checks if the list is empty, in which case it throws an EmptyStackException. Otherwise the value of the head is returned. The upper bound time complexity for the peek method is O(1) because it just simply returns a known value. The **isEmpty** method returns the comparison of head to null because an empty stack will have a head value of null as seen in the default constructor. The upper bound time complexity for the isEmpty method is O(1) because it simply returns a comparison of two values. Finally, the **count** method returns the size variable, as it represents the number of items in the linked list and is updated with each push and pop call. The upper bound time complexity for the count method is O(1) because it simply returns a known value.