Eli Planas
301359051

CMPT 225
Assignment 1: List Operations

Part 1

List.visitAll() Operation:

The visitAll function for lists works by using a temporary node pointer that first points at the same node the head is pointing to. In a while loop, the temporary pointer will traverse from one node to its next node until the temporary point is pointing to the same node the tail is pointing to. This has the time complexity of O(n) because the time to visit all elements of the list increase linearly as the amount of elements increases due to the use of a do-while loop.

Vector.visitAll() Operation:

The visitAll function for vectors works by iterating though each element using a for loop and objects[element]. To make sure the compiler will not ignore the function, I interact with each element using the swap function. This has the time complexity of O(n) because the time to visit all elements of the vector increases linearly as the amount of elements increases due to the use of a for loop.

List construction using push_back() operation:

The list to be tested is constructed by using a for loop that will push_back() as many times as there are elements for the list. The push_back() operation itself has the time complexity of O(1) due to not increasing no matter how large the amount of input data. Since this operation is used in a for loop, the construction of the whole list has the time complexity of O(n) because the time complexity increases linearly as the amount of elements to push_back() increases.

List construction using push_back() operation:

The vector is created in the same manner as the list is made, using a for loop and the push_back() function. The time complexity of the push_back() function is O(1) but the use of the loop causes the construction of the complete vector to have the time complexity of O(n).

Part 2

Vector.push_front() Operation:

The push_front() function made for vectors works by first checking of theSize (amount of elements) is equal to theCapacity(maximum amount of elements the vector can currently hold) and expand the vector by two times if the vector is full. This task takes O(1) because it is only an if statement. After that, the operation will shift every element to the right by one by using a for loop and then assign the first (or 0th) element with the new value to be added. This for loop causes the push_front() function to have the time complexity of O(n).

List construction using push_front() operation:

The second list to be tested is constructed by using a for loop that will push_front() as many times as there are elements for the list. The push_front() operation itself has the time complexity of O(1) due to not increasing no matter how large the amount of input data. Since

Eli Planas
301359051

this operation is used in a for loop, the construction of the whole list has the time complexity of O(n) because the time complexity increases linearly as the amount of elements to push_front() increases.

List construction using push_front() operation:

The vector is created using a for loop with the push_front() operation. The push_front() function has the time complexity of O(n), so with it nested in a for loop causes the whole construction of the vector to have the time complexity of O(n^2).

Time measured to construct the list data structures with various generic types and various amounts of elements:

Type	# of Elements	Push_back vector insertion	Push_back list insertion	Vector visitAll	List visitAll
Int	1000	0.031 ms	0.103 ms	0.016 ms	0.007 ms
Char	10	0.009 ms	0.008 ms	0.005 ms	0.010 ms
String	100	0.042 ms	0.034 ms	0.007 ms	0.006 ms

Type	# of Elements	Push_front vector insertion	Push_front list insertion	Vector visitAll	List visitAll
Int	1000	2.043 ms	0.108 ms	0.017 ms	0.008 ms
Char	10	0.01 ms	0.009 ms	0.005 ms	0.011 ms
String	100	0.114 ms	0.020 ms	0.004 ms	0.003 ms