CSCE 221-200: Honors Data Structures and Algorithms
Assignment Cover Page
Spring 2021

Name:	Alexander Akomer
Email:	alexakomer@tamu.edu
Assignment:	PA 3
Grade (filled in by grader):	

Please list below all sources (people, books, webpages, etc) consulted regarding this assignment (use the back if necessary):

CSCE 221 Students	Other People	Printed Material	Web Material (give URL)	Other Sources
1.	1.	1.	1. https://www.geeksforgeeks.org/hashing-set-2-separate-chaining/	1.
2.	2.	2.	2.	2.
3.	3.	3.	3.	3.

Recall that TAMU Student Rules define academic misconduct to include acquiring answers from any unauthorized source, working with another person when not specifically permitted, observing the work of other students during any exam, providing answers when not specifically authorized to do so, informing any person of the contents of an exam prior to the exam, and failing to credit sources used. *Disciplinary actions range from grade penalty to expulsion.*

"On my honor, as an Aggie, I have neither given nor received unauthorized aid on this academic work. In particular, I certify that I have listed above all the sources that I consulted regarding this assignment, and that I have not received or given any assistance that is contrary to the letter or the spirit of the collaboration guidelines for this assignment."

Signature:	Alexander Akomer
Date:	04/02/2021

Alexander Akomer

PA 3 Report

1. *Introduction:*

This assignment is an early exercise in calculating efficiency of different data structures. We were tasked with creating three different hash tables implementations– one following the method of separate collisions, one following the method of open addressing and linear probing, and the last following the method of open addressing and double hashing. To compare the time efficiency of the three, a constant output text size of 20,000 characters was maintained while adjusting the window size and measuring the time required to do the combination of three tasks: inputting data from a file into a hash table implementation, searching for data within the structure, and randomly outputting that data into a file. This was done to measure the time complexity of each structure so that the advantages and disadvantages of each could be easily displayed. The results are as follows.

2. *Theoretical analysis:*

Two different functions were required to be implemented across each of the three hash table types. These two functions are the search and insert operations, which are each known to have O(1) or constant time across all three required hash table implementations (separate chaining, linear probing, and double hashing).

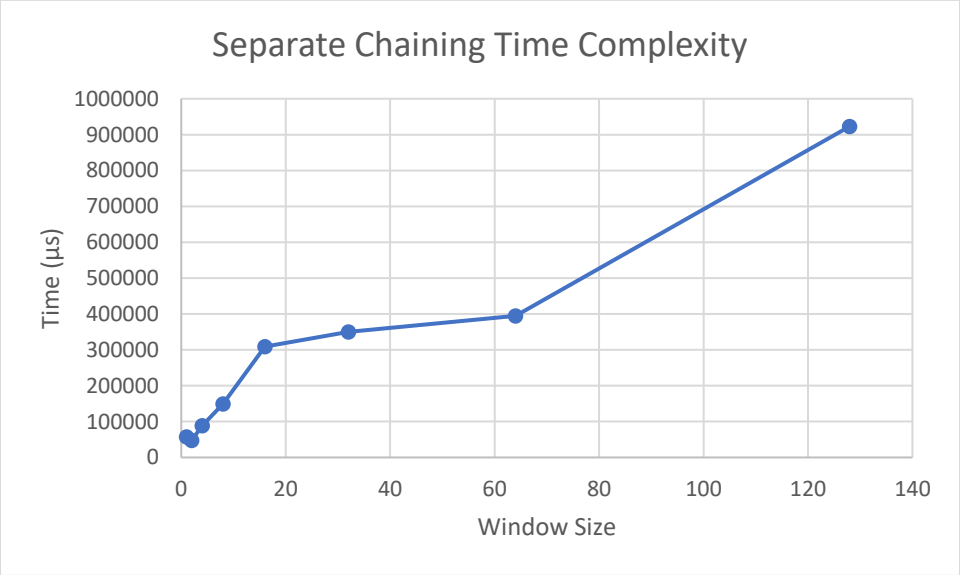
The three hash table implementations considered for this assignment each possess an average case time complexity of O(1) for searching and insertion as stated above; however, each suffer from an O(n) worst-case time complexity for these two operations when the table has achieved a state in which it has passed its load balance or when it has hashed too many values to the same key, which results in large clusters that function similarly to arrays. Amortized analysis of these functions will

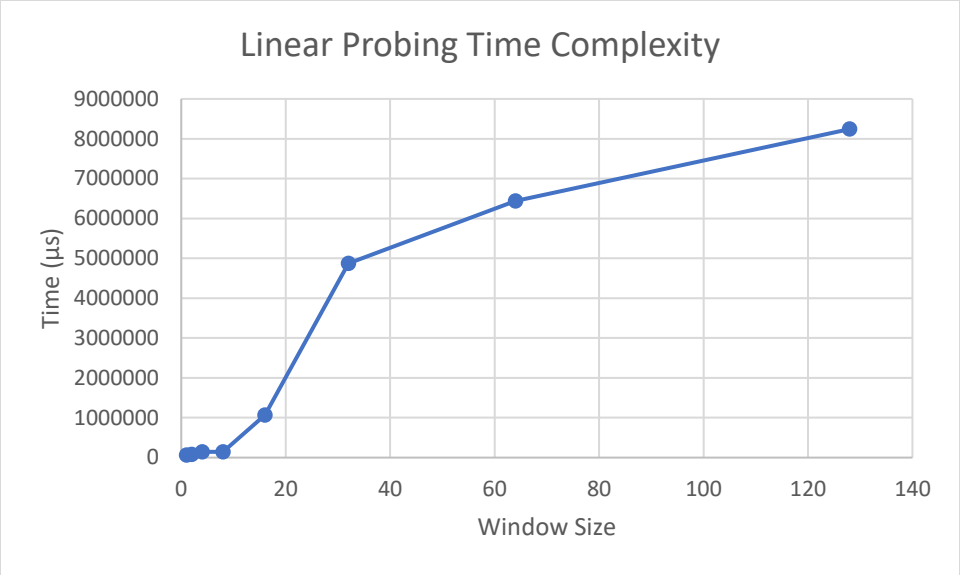
reveal whether the given hash function and load balance are well-suited to the task at hand based on observation of linear or constant time complexity.

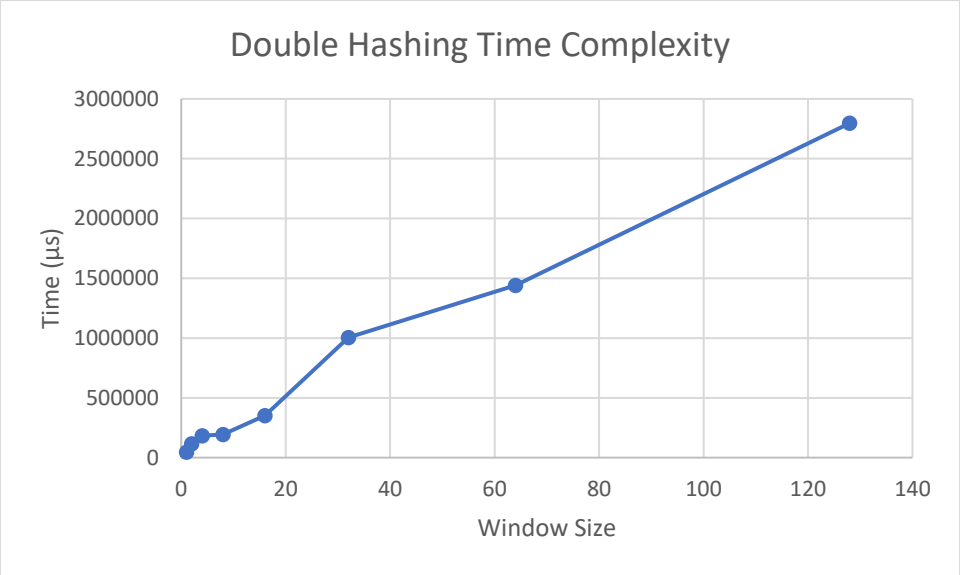
3. *Experimental Setup:*

The machine the program was run on for my trials was a Surface Book 2 with 16 GB of RAM, an i7-8650U @ 1.90 GHz, and 106 GB / 475 GB free on the C: drive. My timing mechanism was the code attached on Canvas, which included the chrono package. This package allowed me to run a high-resolution clock and output the runtime in microseconds for each trial. Because the instructions only called for one trial of each function call for each data structure, that is the all of the data that was recorded, which may lead to faults in accuracy.

4. *Experimental Results:*




Linear Probing Time Complexity
Time (µs)
9000000
8000000
7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
0
20
40
60
80
100
120
140
Window Size


Double Hashing Time Complexity
Time (µs)
3000000
2500000
2000000
1500000
1000000
500000
0
0
20
40
60
80
100
120
140
Window Size

After testing the search and insert functions (combined in a function named fillTable()) for each of the following tree implementations, the following was concluded:

Separate Chaining: Worst/Average Case O(n)

Linear Probing: Worst/Average Case O(n)

Double Hashing: Worst/Average Case O(n)

The runtime data somewhat supports the theoretical analysis of each function. Though the average case of each hash table implementation is supposed to be O(1), each of my implementations acted closer in accordance to O(n). This is proof that in many instances, the load balance was not properly maintained and that clusters formed where many values were mapped to the same key. This may be due to a poor implementation on my end, a discrepancy in background applications affecting CPU usage, or the fact that tests were computed based on my fillTable() function (which included two calls to the search() function, one call to the insert() function, and one call to the occurs() function). Constant time could very easily have been maintained somewhere in the function, but any linear time operations within fillTable() would blow the data out of proportion. Despite discrepancies in performance where I would have otherwise expected constant time, each function was bounded exactly as depicted in the worst-case theoretical analyses.

Based off of my implementations of the different types of hash tables, the separate chaining hash table was by far the fastest for this application, which was very insert-heavy. The reason this implementation was so much faster than the other two is because the other two relied heavily on open addressing. The sequential chaining method uses linked lists at each key to avoid ever resizing. This collision-handling technique proved to be faster in this application, because many

of the elements ended up hashing to the same key (due to the repetition of certain words and phrases). The open addressing methods fell behind in several areas – where linear probing has great cache performance, it is very prone to clustering (which can be seen at the lower window sizes, where time complexity grew rapidly as many values had to resort to linear probing in a clustered area). The other open addressing method, double hashing, did not suffer as greatly as linear probing because it has virtually no clustering, but struggled in performance due to much more extensive calculation periods. Asymptotically, this left double hashing very similar to separate chaining with a higher multiplicative constant. This method isn't necessarily faster in all insertion cases, but proved effective for Shakespeare. In conclusion, separate chaining is far superior for smaller window sizes, where clustering proves to be the largest issue for linear probing. Beyond that, linear probing and separate chaining appear to be very similar in terms of their growth as window sizes increase (and therefore collisions and clusters decrease).