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

|  |  |
| --- | --- |
| **Name:** | Priyanshu Barnwal |
| **Email:** | [priyanshu@tamu.edu](mailto:priyanshu@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.Textbook | 1. | 1. |
| 2. | 2. | 2. | 2. | 2. |
| 3. | 3. | 3. | 3. | 3. |
| 4. | 4. | 4. | 4. | 4. |
| 5. | 5. | 5. | 5. | 5. |

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:** | Priyanshu Barnwal |
| **Date:** | 3/18/2021 |

**Introduction:**

The objective of this assignment is to learn and build hash tables using different methods to handle collisions. I will be using these hash tables to generate random characters using text from a play called “Merchant of Venice”.

**Theoretical Analysis:**

Search times for Hash Tables regarding separate chaining have a worst-case runtime of O(n), but this is very rare. O(lambda) or O(lambda + 1) are far more common, as most hash tables are somewhat evenly distributed. For Double Hashing, however, runtimes are a bit more complicated. For an unsuccessful search in the table, the runtime is O(1/(1 - lambda)) and for a successful search, it’s O((1 / lambda)\*lg (1 / (1 - lambda))). This is often slower than separate chaining. Linear probing is somewhere in the middle with a runtime of O(1+(1/(1-lambda))) for successful searches and O(1+(1/(1-lambda))^2) for unsuccessful ones.

**Experimental Setup:**

I am currently running this on a computer with 32GB RAM, however, Visual Studio only allocates 2 GB of processing memory for any given project. I am using Visual Studio 2019, which runs C++17 with experimental features of C++20. For my timing mechanism I used the high-resolution clock inbuilt function to accurately time how fast my program ran. I only ran each tree for one trial, but ran them 3 times by inserting increasing, decreasing, and random values.

**Experimental Results:**

As can be seen by the results in the graph, all three increase linearly with runtime. This is because of the way window sizes interact with the search functions. The differences in these cause each runtime to be almost identical, because of the way the window sizes work, but we can see that separate chaining is generally the fastest with the time complexity. Double Hashing and Separate chaining are near identical at first, however, a huge discrepancy can be noticed when larger input sizes are inserted. All three table increase with a similar time constant, and the numbers are almost interchangeable until the window size is set to 4 and the distinction between the three tables is clear. Separate chaining gives numbers far below the other two in terms of runtime and linear probing is second fastest, falling almost directly in between the other two.