Byron Laferriere

CS-260

28 FEB 2021 ( 30-JAN-2022)

Final Project

**I. Data Structures**

**A. Vectors**

The file that best exemplifies my understanding of the vector data structure type would be Lab2-1. In this assignment I was tasked with creating a vector data structure that would hold bids that were being placed or had already been placed. Much like an array, a vector can be used to store an ordered list of data items, with having direct access to each of the items. The advantage of using vectors in this scenario is that vectors are resizable whereas arrays are fixed (Rajput, 2018). This means that the vector storage could grow and shrink as the data being processed changed as well. This also allows for automatic deallocation of the storage if there are no variables being stored within the vector, whereas arrays must be explicitly deallocated (Rajput, 2018). Once this vector was created, I created a data structure within the vector to hold data from each row entry. Four variables were created within the vector to store the values entered by the program. When tested in Powershell, the program successfully stored and displayed the entered information. This assignment also required learning to loop through the vector to display all the bids stored. The program successfully stored the entered items and was able to search and locate them upon request. This exemplified my understanding of how to implement and initialize vectors successfully and why the vector is a more powerful choice of data structure in this scenario than using something like an array.

**B. Hash tables**

The document in the portfolio that best exemplifies my knowledge and understanding of hash tables and how they work would be the file named Lab5-1. In this assignment, we were tasked with creating a structure that could take unordered information and store each item by mapping it to a location. In this type of scenario, hashing the data into a hash table helps create much quicker search results (Borowski, 2018). Hash tables can process large amounts of data in O(1) or worst case O(n) time. Using Big O time to discern what this means, a data set of 1,000,000 files would run in seconds compared to days when using a loop (Borowski, 2018). I displayed my understanding of how hash tables work with hashing by creating structures to hold all bid information and then implementing code to calculate the hash value assigned for the bid information into a way that the table could be read quickly. The modulo operator was used to calculate a hash value determined by the size of the table. After this, I also implemented logic to insert new bids into the hash table, while the modulo operator was constantly updating the table as necessary. Lastly, in this assignment, I implemented the code to remove items from the hash table. Using the logic hash() and ‘atoi’ or askew to integer allowed for this to successfully happen. This lab clearly exemplifies that I have a keen understanding of how to operate a hash table and why the hash table can be a powerful data structure to incorporate into projects dealing with large amounts data processing, that need quick iterations.

**C. Tree structures**

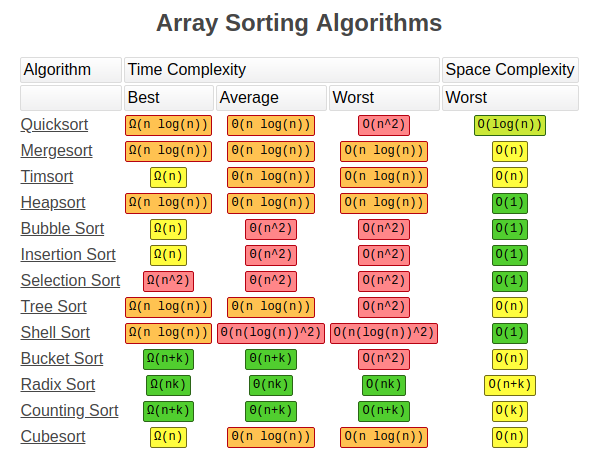
The importance of understanding when and where to use tree structures instead of arrays takes an experienced programmer. Although arrays hold a constant Big O notation time of O(1), this can also be considered its weakness with insertions on large projects (Ang, 2017). Because of a Binary Search Trees ability to maintain and manage a dynamic set of data, it makes this the better option by conforming to the O(log(n)) notation (Ang, 2017). This data structure needs to be chosen carefully according to the needs but can be the best choice in certain scenarios. The document that best exemplifies my mastery and knowledge of tree structures and how they operate with roots, parents, internals, leafs, and managing a changing set of data would be the assignment from week 6 titled Lab6-2. In this assignment I was tasked with creating a binary search tree (BST) and filling nodes with new and current bids from the sample bids file. The BST was chosen for this assignment because of the ever-changing value of best or worst bid coming in, there would be no set root node etc…. The code begins by implementing the root node and pointers going left and right. After this I created a left and right variable that was assigned to 0 to begin. The second task completed in this assignment worked with inserting new bids into the BST. Using an if loop, the program was easily able to work through filling in the tree by determining if the root was equal to zero. Otherwise, it would fill in the bid into the proper node accordingly by working through left and right comparisons in the tree. I also created a while loop to create a search method through the BST in this week’s assignment. Within the while loop I had an if loop and a nested if loop that controlled the search parameters through the tree. I believe that this verifies that this assignment best exemplifies my understanding and knowledge of tree structures and proper use cases that maximize their potential over other available data structures.

**II. Algorithms:**

1. **Search**

The document that best exemplifies my mastery of how to implement the search algorithm in C++ would be the assignment from Week 3 in the folder Lab3-2. Although there are much more sophisticated algorithms that can be used, this assignment tasked the student with using a linear search algorithm to work through the linear linked lists and return a match for the specified bid being searched for. Using Linear Search instead of a Binary Search was the best in this case because of the time complexities performing the search. Best case scenario using a linear search on a linear linked list is O(1) and worst case is O(2), while the average case is O(2) (Jeyaraman, 2020). These metrics beat out the other option, binary search, because a binary search requires sorted lists and can reach a worst case of O(log^2(n)) (Tech Differences, 2019). This assignment required us to complete many tasks within a linked list that was created. Of these tasks, the implemented search logic was created using a while loop that checked for the entered value against 0 using the compare() command. From there, it looped over each node looking for a match. When tested in Powershell this piece of the program ran very smoothly and quickly, proving that this assignment best displays my understanding of the search algorithm. Not only did it prove my understanding of how to use the search algorithm, but it showed my mastery of diagnosing the best use case for the linked list that was used and being able to discern between the linear search and binary search.

1. **Sort**



Picture from: [Which Sorting Algorithm Is Best?. There are tons of sorting algorithms… | by Joshycsm | Medium](https://joshycsm.medium.com/which-sorting-algorithm-is-best-ca83e3cc3ca0)

The assignment that best exemplifies my understanding and knowledge of sorting algorithms is the assignment from Week 4 found in the file Lab4-2 that implements two sorting algorithms over a vector. Although there are many other options to use when sorting, it is important to understand why quicksort was chosen for this assignment. There are both time and space complexity-wise faster algorithms that can be used but quicksorts average time (O(n log n)) when combined with a proper pivot method can make it an extremely effective option (Wang, 2016). Of course, this does make quicksort the “fastest” algorithm available to use, but this term cannot be universally applied because of the variety of use cases and specific need of projects (Wang, 2016). This assignment required us to implement and invoke both the quicksort algorithm and the selection sort algorithm to a vector of bids. The quicksort algorithm was implemented by using multiple while loops to check against the pivot to determine whether to decrement or increment. Selection sort was completed by using if and a nested if that used compare() to sort bids. This assignment showed the speed of the quicksort method over the selection sort and exemplifies my knowledge of algorithm choices and proper scenarios for usage.

1. **Hash/Chaining**

The assignment that best displays my understanding and knowledge of the hashing with chaining techniques to avoid collisions would be from Week 5 in the file Lab5-1 where we used chaining to handle collisions in the hash table by creating temporary buckets in a linked list format. A collision occurs when two keys are hashed to the same index in a hash table. Collisions are a problem because every slot in a hash table is supposed to store a single element (Dopico, 2021). In this assignment I created a linked list named ‘temp’ to store these values as collisions happened during inserts and reconfigurations. Hashing with separate chaining, like this, has an advantage that it is less sensitive to a hash function (Dopico, 2021). By creating the temp value to hold the information while it is being moved, I was able to avoid any difficulties that could be experienced when performing this type of sorting of hash tables. Then after the temp variable has completed its job, the memory is wiped to help save storage space. I believe this lab best exemplifies my mastery of the hashing with chaining concept and how I can use this algorithmic methodology to help protect against collision error.

**III. Student’s Choice:**

The document in the portfolio that was my favorite program was the assignment from Week 5 in the file Lab5-1 that required the creation of a data structure called a hash table and combined the chaining algorithm. This program created a hash value for each bid by using the modulo operator to continuously convert values according to the table size. The program was able to do so successfully upon testing PowerShell, when adding new bids into the table and seeing the chaining algorithm create a temporary bucket to store bids as they were passed through the hash table. There are well-written in-line comments that lay out the decisions that were made while writing the code for this program. This is most evident in the design of the structure Node() under FIXME1. The code is concise and easy to read, which makes it reusable and easy to edit as needed. The area of code which best highlights this fact is the implementation of the erase() and resize() logic. No matter how the table changes, the code is prepared to adapt. Modular composition is achieved throughout the code as it has been spaced out into easy-to-comprehend modules of code according to the needs of the program. Once this assignment was completed, I had a better understanding of the importance of data structures and algorithms in programming, which I why I decided to use this assignment as my proof of concept.

**IV. Conclusions:**

Data structures play a very important role in computer programming because of their foundational properties that they supply to a program. Programs need data structures to collect and organize data within the program, to run successfully. Depending on the needs of the program, different data structures can be more appropriate for each individual case needing to be satisfied. The assignments in this course required us to explore vectors, hash tables, and different types of trees. Throughout the semester, we learned how these different types of structures were both similar, yet different in many ways at the same time. Algorithms play an important role in interacting with the chosen data structures. Program effectiveness relies on this combination of algorithm and data structure to handle large amounts of data for processing purposes when necessary.

This semester, we covered different algorithms that do this like linear and binary search, many different sorting algorithms, and hash tables and hashing with chaining. The assignments had us combine data structures like vectors and sorting algorithms and showed the power of using quick sort over a vector, when compared to selection sort combined with a vector. There was a 14 second differential between the two when only dealing with 78k files. I can only imagine the impact this type of decision could have on a large program’s processing speeds. This assignment highlighted the importance of properly combining data structures and algorithms to avoid slowing down the program. This can be compared to the balance that is needed in our daily lives to ensure efficiency. Going out drinking every night, will never mix with professional needs and will only slow you down in the long run. All while blocking personal ability to perform, from choosing the wrong ‘algorithm’, as you should be able to and slowing down your own personal ‘program.’

**References:**

Ang, N. (2017, December 10). *Why use binary search tree?* Nick Ang. Retrieved January 29, 2022, from <https://www.nickang.com/2017-12-10-why-use-binary-search-tree/>

Borowski, D. (2018, September 3). *The importance of hash tables*. Medium. Retrieved January 29, 2022, from <https://medium.com/coderbyte/importance-of-hash-tables-c429a2b523b8>

*Difference between linear search and binary search (with comparison chart)*. Tech Differences. (2019, December 28). Retrieved January 29, 2022, from <https://techdifferences.com/difference-between-linear-search-and-binary-search.html>

Dopico, A. (2021, February 20). *Home*. – JanetPanic.com. Retrieved January 30, 2022, from <https://janetpanic.com/what-is-hash-table-chaining/>

Jeyaraman, H. (2020, May 23). *Linear Search in C programming language*. OpenGenus IQ: Computing Expertise & Legacy. Retrieved January 29, 2022, from <https://iq.opengenus.org/linear-search-in-c/>

Rajput, A. (2018, October 30). *Advantages of vector over array in C++*. GeeksforGeeks. Retrieved January 29, 2022, from <https://www.geeksforgeeks.org/advantages-of-vector-over-array-in-c/#:~:text=%20Advantages%20of%20Vector%20over%20arrays%20%3A%20,can%20grow%20and%20shrink%20as%20vectors...%20More%20>

Unknown. (n.d.). *CS: 260 Data Structures and Algorithms*. Zybooks.

Wang, E. (2016, April 17). *Quicksort - the best sorting algorithm?* Medium. Retrieved January 29, 2022, from <https://medium.com/human-in-a-machine-world/quicksort-the-best-sorting-algorithm-6ab461b5a9d0>