

This is the final project for the final project for data structures. There is a table of contents on how everything is broken down. This is for time complexity and online auction solution.

Final Project: Online Auction Solution

Data Structures

Peter Vaughan

Contents

Final Project: Online Auction Solution	2
Overall Grade	2
Part A Expectations: Complexity [5%]	3
Complexity	4
My choice	6
Part B Expectations: Data Structures [35%]	7
Three Parts	7
Expectations	8
Overview	8
Goals and non-goals	8
Milestones	8
Proposed Solution	9
Changing my solution in tie?	9
Existing Solution / Alternative Solutions	9
Testability, Monitoring and Alerting	10
Optional Code	11
Submission	12
My submission	12
Bibliography	12

Final Project: Online Auction Solution

Overall Grade

Grade: 40%

Two Parts of this project:

- Part A Expectations: Complexity [5%]
- Part B Expectations: Data Structures [35%]
- Optional Code

Part A Expectations: Complexity [5%]

Time complexities of the five algorithms that can be used to solve a computational problem is characterized as the following:

a) $O(n^3)$ b) $\Omega(n^3)$ c) $O(n^2 \lg n)$ d) $\Theta(n^2 \lg n)$ e) $O(2n)$

Which one would you choose as the best for relatively large n ? Explain.

Big O vs. Big Theta vs. Big Omega Explained (Austin, 2024)

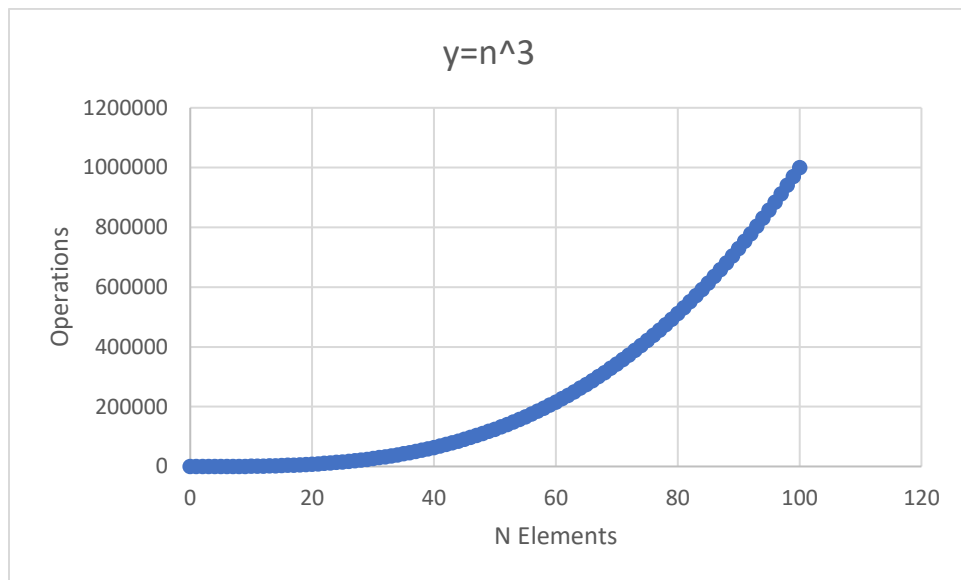
- **Big O:** This represents the worst-case performance for an algorithm, setting an upper bound on how slow your code can be. It's noted as $O(n^2)$.
- **Big Theta (Θ):** This represents the average, typical case performance for an algorithm. It's noted as $\Theta(n \times p)$.
- **Big Omega (Ω):** This represents the best-case performance for an algorithm, setting a lower bound on how fast the code can perform. It's noted as $\Omega(n)$.

Something info to consider for future products:

What are the bounds of the equations? Everything has a max capability even if it can deal with many numbers at any given moment. When do equations actually get cut off? I will not consider bounds of this application besides the computational problem of the large n .

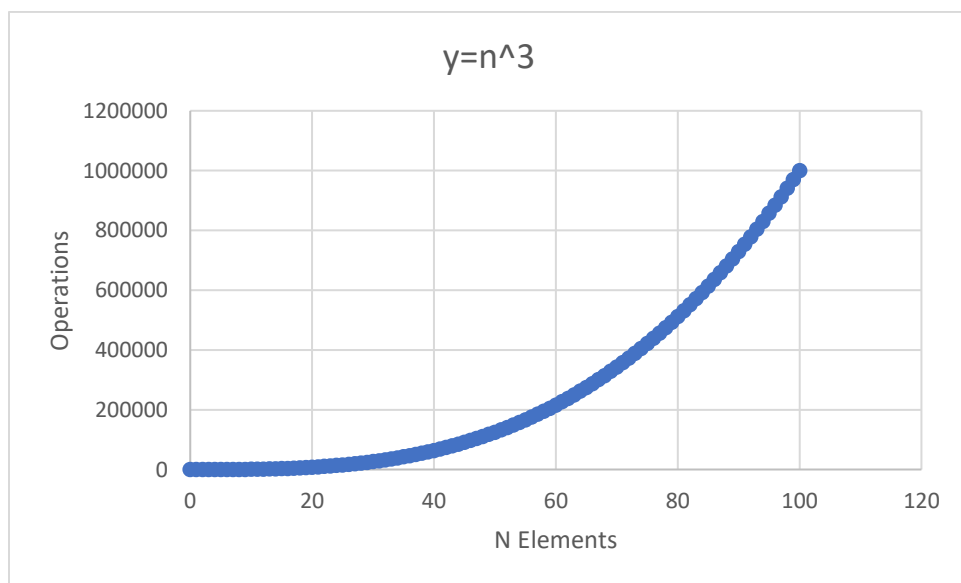
Complexity

a) $O(n^3)$



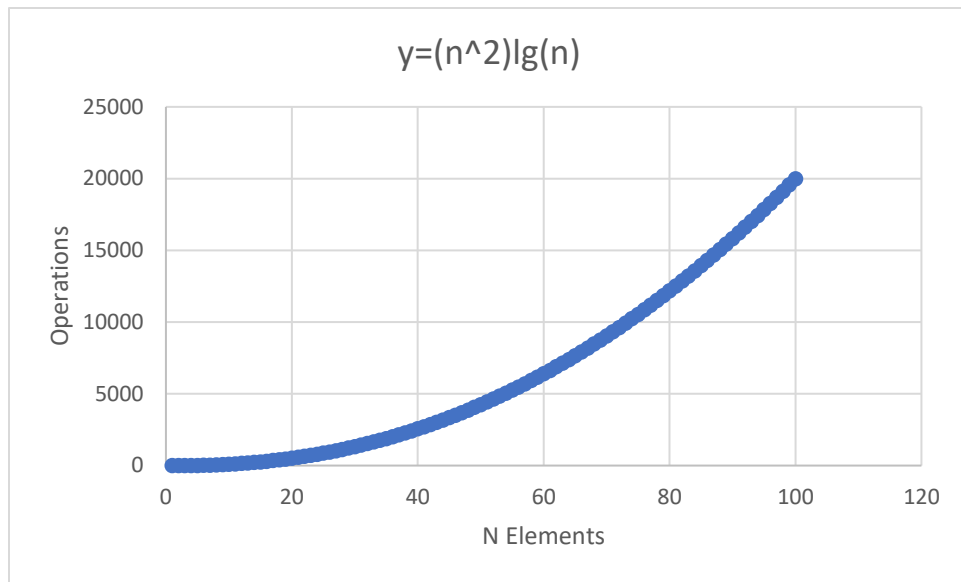
While it does do a lot of operations with larger numbers. However, it might be wasting time doing needless or inefficient operations. This would be my first option if it was doing a range of tasks. Also, this is worst-case scenario, so this would be better than option B because it is the best-case scenario. This would be third option. This does not take into account the CPU or computer.

b) $\Omega(n^3)$



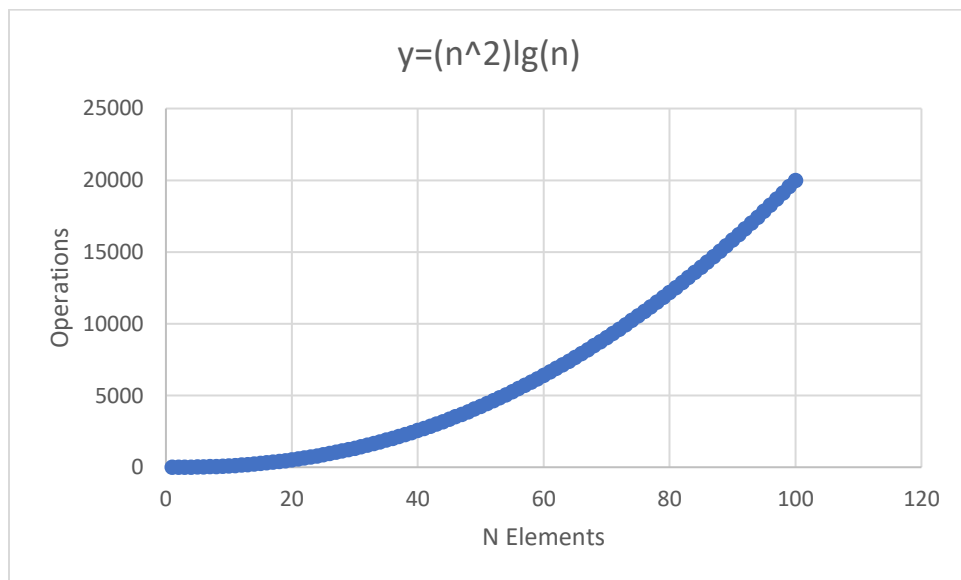
This would be my close forth option.

c) $O(n^2 \lg n)$



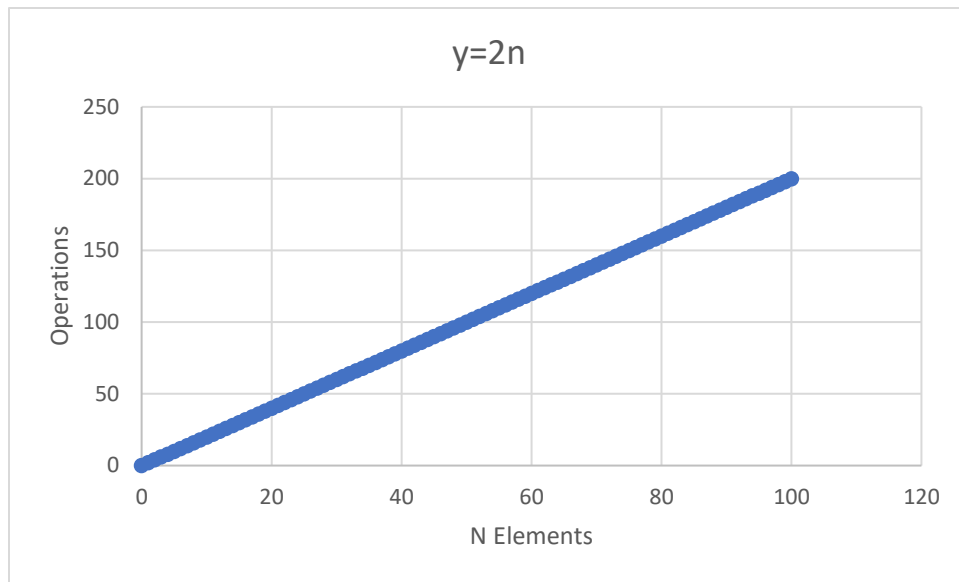
Option C is better than Option D because option D would be the best case scenario, despite being the same equation. Option C is the worst case while option D is the best-case. Therefore, option C would be better than option D. If it were both given the same numbers, option C would win every time. Option C is my top choice because it is not inefficient and deals with large numbers of numbers. This does not take into account the CPU or computer.

d) $\Theta(n^2 \lg n)$



This is my close second option.

e) $O(2n)$



This type of algorithm would be more ideal for smaller numbers. This would not be a good option for large quantities of numbers. This would be my fifth-place option. We were told $2n$, which in math terms is automatically $2*n$ and not 2^n .

My choice

My choice would be option C because there will be lots of operations to solve the problem with the information without error. Option C would be a close second. They would not take up too much of the computer resources while solving the problem, such as sorting.

Option A and Option B takes too much of the resources to do the same task of Option C and Option D. Option E might not verify all the information with the lack of operations.

Part B Expectations: Data Structures [35%]

Consider the following Unique Bid Blind On-line Auction scenario:

For every item auctioned, the procedure is the same:

- Bidders, identified by their IDs, submit their bids until the auction is closed.
- Everyone can submit a number of bids.
- Bids must be integer values, in Canadian dollars, and higher than the minimum bid set (bids of less than the required minimum is discarded).
- As soon as the auction is closed the highest bidders are found
 - if there is a tie, the auction is restarted with the minimum bid of the current highest bid + \$1
 - if the highest unique bid is found, and the bidder who placed it wins

Three Parts

1. **[20%]** Propose a solution for storing & processing bids efficiently.
Identify data structures and/or algorithms used, and explain how they will be used.
Make sure that you will be able to handle large numbers of bids, often exceeding thousands of bids.
Do not concern yourself with receiving data from network connections; assume that your `placeBid(Bid)` function (method) will be called when a bid request is received, and with its help, you will record bids, one at a time.
2. **[15%]** Would your solution need to change if, in the case of a tie, the bidder who placed the highest bid first were to win?
3. **[10%]** [Optional] Try to implement your solution in any programming language.

Expectations

Overview

We are an auction company, much like eBay. We want to start a new bidding system. For every item we auction online, the procedure is always the same:

- Bidders, identified by their IDs, submit their bids until the auction is closed.
- Everyone can submit any number of bids on the same item.
- Bids must be (integer) whole values, in Canadian dollars, and higher than the minimum.
- Bid set (bids of less than the required minimum is discarded).
- As soon as the auction is closed the highest bidders are found. If there is a tie, the auction is restarted with the minimum bid of the current highest bid + \$1. Otherwise, if the highest unique bid is found, and the bidder who placed it wins.

Goals and non-goals

Main goals:

- We need to follow the rules in the overview.
- We need to keep track on people's info including their IDs and item info with their IDs.
- We need some kind of security for privacy reasons and fraudsters.
- Verify winners before e-mailing them that they won.

Non-goals:

- Give people suggestions to what items to bid on much like Amazon.
- E-mail people if they need to potentially bid more.
- Fancy graphical interface to encourage people to bid more while still making it easy to bid.

Milestones

Milestone 1 would be two main core tasks. We need to get a program to follow the rules in the overview. Then we need to keep track on people's info including their IDs and item info with their IDs. This is the most important part of the part of bidding.

Milestone 2 is adding some kind of security for privacy reasons and fraudsters. Verify winners before they are e-mailed about winning. We cannot have a bidding system without customers to sell and buy items.

Milestone 3 is adding fancier features to attract people to come back. Also, when users create accounts, allow them to choose concepts like bid suggestions, or to get notifications to bid more. If you run out of time with this milestone, these concepts are not needed because they are add-on features.

Proposed Solution

Vector HashMapping sort would give a fast and reliable answer. We need a reliable way to find the highest bid along with an ID. With `HashMap<Integer, Integer>` or to a similar concept, there can be a customer ID and bid amount. Customers can add as many bid as they want and customer IDs can be letter and numbers. Before customer IDs and integers are saved in a vector `HashMap`, bids must be a verified integer values, in Canadian dollars, and higher than the minimum. The bids that are less than the required minimums are discarded and not saved. Otherwise, it is saved one at that time. There needs to be a timer to let the program know when the bidding time is over. People's personal bids can be seen on their accounts and stored in a database.

Based on my sample code, vector HashMapping can handle large numbers of bids, often exceeding thousands of bids. After getting the info into `HashMaps`, there is a potential option for highest bids and a potential case for a tie. I would do a comparative-style search algorithm for the highest number with a counter. Once it gets a new high number, the counter is set to 0. Once it reaches the same high number, increase counter. Until a new high number is reached and is now the new high number and the counter is set to 0 – 0 mean no bids. If the counter is bigger than 1, there is a tie. It is important to find the highest bid and to find if there is a potential tie.

Once the program searches the bid list, and if there is a tie, we would be placing the big at highest bid +1. Otherwise, it will be sold to the highest bidder, identified by ID. On my code, I did put a cap on bidding to showcase the program working.

At the end when there is a winner, we need to gather the info. We need the winner information, bid number and item info would be store in a database. Then the winner is verified and notified of their winnings and process the payment.

Changing my solution in tie?

My solution would not change if, in the case of a tie, the bidder who placed the highest bid first were to win. With the right verification process, my solution should be acceptable. It will sort the information correctly, so it is easy to find the highest number. I will consider changing my method if there is an issue connecting the `HashMap` to a database table.

Existing Solution / Alternative Solutions

While there are options out there, my consideration is for the program to be done partially in house. We can connect this program to other services, such as payment services or a third-party database. If a particular service goes out of business or gets too expensive, then we can change out program to fit the new system. There is the potential for the company to grow and we eventually do more in-house to save money.

We might consider initially to rely primarily on third party software but this would be expensive. Then consider my potential solution.

Testability, Monitoring and Alerting

Three main tasks we need to keep up-to-date is the testability, monitoring and alerting. This is very important to keep the business running smoothly and with high satisfaction.

With testability, we will do regular unit testing. There need to be testing on our milestone program building. Also, we need to continue to develop our program and to provide correct unit-testing for these segments of code.

With monitoring of information and keeping the website online, there should be a security to keep the website online and working safely. For someone like me, I would have to get trained more programming security. I am familiar with customer security due to working in sensitive areas. Also, we should get an alert if the web site is down. We should have back-up power to solve the potential power outages. But we need monitoring our security to prevent cyber-attacks.

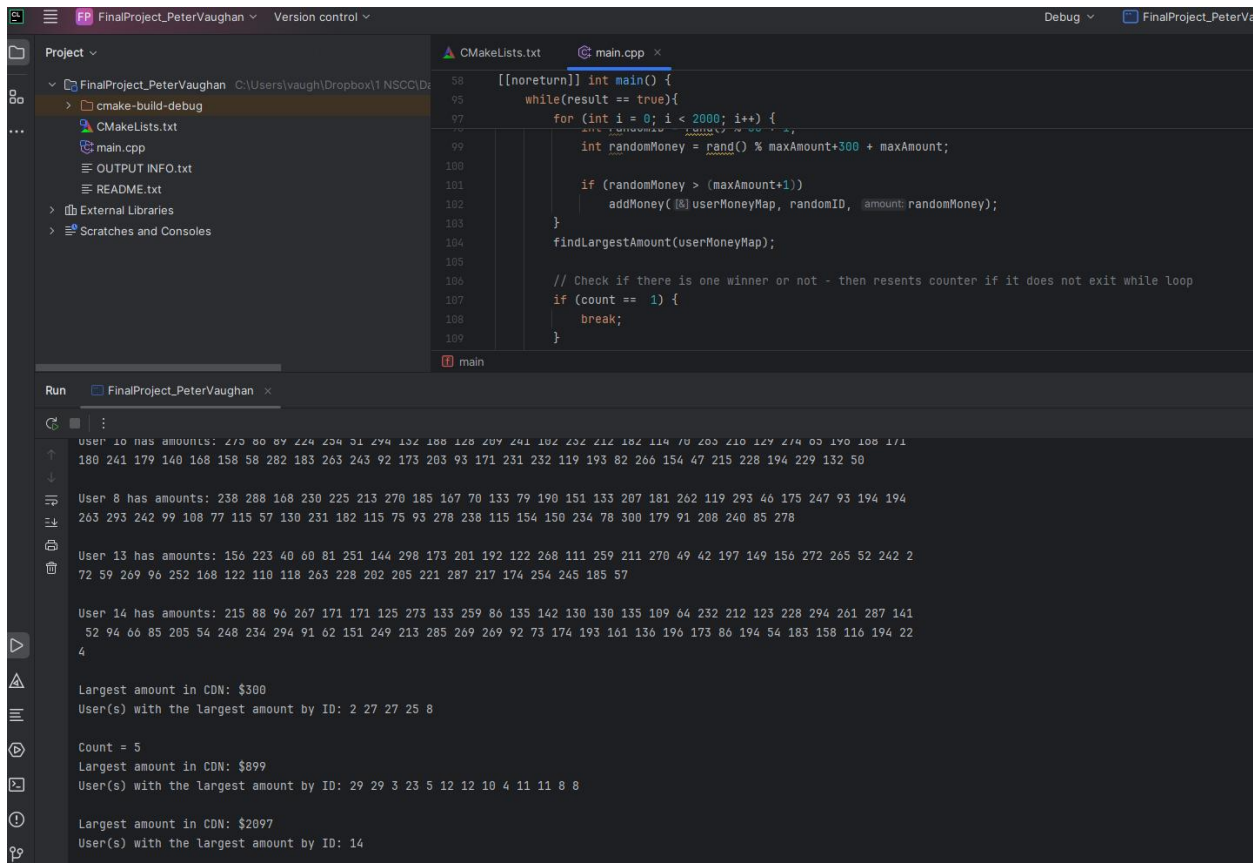
All the relevant people need to be alerted if there are any potential issues such as the web site being down or a cyber-attack. We should value our customers privacy and business bidding transaction. An alerting system should be flexible to add more alert parameters if needed to the correct people.

Optional Code

For the code, I will start by welcoming people to the auction. There will be instructions such as bidder's will be using their IDs, people submit their bids until the auction is closed, and everyone can submit a undetermined number of bids. These bids must be integer values, in Canadian dollars, and higher than the minimum.

With my code, it is to demonstrate the concept of a bid. There will be a random number generator to get an ID and bid number and if it is higher than the minimum, insert a number to be compared later. Repeat this action about 1,000+ times. In a properly working program, IDs should be verified first for person's identity and errors. But it would not be part of the bidding system. If there are two or more winners, this will restart the bid at highest bid +1. The same situation will happen from the start. Otherwise, display winner's ID. On my code, I did put a cap on bidding to showcase the program working within its capabilities.

This is sample code, assuming there is only one item. If I would to build this more, I need to connect this to a database and associate this with other info. We would ideally have to connect this winner to an item. The creator might want to put this into a database with an appropriate item ID. There are many options for this.



```

58 [[noreturn]] int main() {
59     while(result == true){
60         for (int i = 0; i < 2000; i++) {
61             // Random ID and bid number
62             int randomMoney = rand() % maxAmount+300 + maxAmount;
63
64             if (randomMoney > (maxAmount+1))
65                 addMoney(&userMoneyMap, randomID, amount: randomMoney);
66         }
67         findLargestAmount(userMoneyMap);
68
69         // Check if there is one winner or not - then resents counter if it does not exit while loop
70         if (count == 1) {
71             break;
72         }
73     }
74 }

```

Run FinalProject_PeterVaughan

```

User 10 has amounts: 270 80 89 224 234 51 294 132 188 126 209 241 182 252 212 182 114 70 203 210 129 274 65 190 108 171
180 241 179 140 168 158 58 282 183 263 243 92 173 203 93 171 231 232 119 193 82 266 154 47 215 228 194 229 132 50

User 8 has amounts: 238 288 168 230 225 213 270 185 167 70 133 79 190 151 133 207 181 262 119 293 46 175 247 93 194 194
263 293 242 99 108 77 115 57 130 231 182 115 75 93 278 238 115 154 150 234 78 300 179 91 208 248 85 278

User 13 has amounts: 156 223 40 60 81 251 144 298 173 201 192 122 268 111 259 211 270 49 42 197 149 150 272 265 52 242 2
72 59 269 96 252 168 122 110 118 263 228 282 205 221 287 217 174 254 245 185 57

User 14 has amounts: 215 88 96 267 171 171 125 273 133 259 86 135 142 130 130 135 109 64 232 212 123 228 294 261 287 141
52 94 66 85 205 54 248 234 294 91 62 151 249 213 285 269 269 92 73 174 193 161 136 196 173 86 194 54 183 158 116 194 22
4

Largest amount in CDN: $300
User(s) with the largest amount by ID: 2 27 27 25 8

Count = 5
Largest amount in CDN: $899
User(s) with the largest amount by ID: 29 29 3 23 5 12 12 10 4 11 11 8 8

Largest amount in CDN: $2097
User(s) with the largest amount by ID: 14

```

Submission

GitHub Repo: <https://classroom.github.com/a/SbMTIEZo>

Submission: Prepare a Technical Solution/Design document to support your viewpoint, and GitHub code repo for your solution. Also prepare your classroom presentation or uploading the recording.

Reference URLs:

- <https://www.nuclino.com/articles/software-design-document>
- <https://www.freecodecamp.org/news/how-to-write-a-good-software-design-document-66fcf019569c/>
- <https://softwaredominos.com/home/software-design-development-articles/write-solution-design-document/>

My submission

Video: Done!

PDF: Done along with an additional txt file!

Coding Knowledge: Shown!

Read Me File: Done!

Repo: <https://github.com/Winter2024-NSCC-ECampus/final-project-online-auction-solution-cadalac-don>

Problem Solving and Efficiency: Shown!

Program Structure: Shown!

Bibliography

Austin, D. D. (2024, December 09). *Big O vs. Big Theta vs. Big Omega Notation Differences Explained*. Retrieved from Built-in: <https://builtin.com/software-engineering-perspectives/big-o-vs-big-theta>