Module 3: Critical Thinking

Minimum Number

Nolan Byrnes

CSC506 – Design and Analysis of Algorithms

Colorado State University – Global Campus

Professor Isaac Gang

October 1st, 2022

Module 3: Critical Thinking

Algorithms can be written in different ways and solve the same problem, however, some algorithms run more efficiently than others. “Big O notation is a mathematical way of describing how a function (running time of an algorithm) generally behaves in relation to the input size” (Lysecky, 2019, para. 1). O(n) is classified as Linear, which means that the amount of time it takes to complete the algorithm increases linearly when using greater inputs. O(n²) is quadradic, which means that as the inputs increase, the amount of computations that the computer must make is squared. Using a nested loop can make an algorithm O(n²). I developed two algorithms to find the minimum number in a list of numbers, one where it the algorithm has an O(n²) and one that has an O(n). This paper discusses the two algorithms I developed to find the minimum number of a list, problems that I encountered, and if the algorithms I developed could have been implemented in a different way.

The logic in my first algorithm was to look at the first number of the list, then compare it to all the other numbers of the list to determine if the number I was currently looking at had a number that was smaller. After looking at each number, if there was not a smaller number than the current index I was on, then I declared that number as the smallest number. In the implementation of this algorithm, it had a O(n²) time complexity due to having a nested loop, since for each value of the list, I had to compare it to every other number in the list. You can view the resulting algorithm in figure 1. below.

Figure 1.

Python function to finding the Minimum Number with O(n²)

Text

Description automatically generated

Note. This function returns the minimum number from a given list with an O(n²) time complexity.

After implementing the first algorithm, I realized that the algorithm was inefficient because the algorithm was comparing each number to every other number, which was not necessary. With this implementation, it would compare the 0th index to the 1st index in the first iteration, and then in the second iteration, it would be comparing the 1st index to the 0th index. With the algorithm having a time complexity of O(n²), it is having to make redundant checks throughout its execution, which can be removed by refactoring the algorithm to have a time complexity of O(n).

For my second implementation of finding the smallest number in a list of numbers, I only included one loop, which gave the algorithm a O(n) time complexity. I was able to achieve this by assigning the first index of the list as the minimum\_value, and then iterating through the rest of the list. For each iteration of the loop, I had the algorithm check to see if the current value is smaller than the assigned minimum\_value. If true, then I would assign the minimum\_value to the value of the number of the iteration. You can see the resulting algorithm in figure 2 below.

Figure 2.

Python function to finding the Minimum Number with O(n)

Text

Description automatically generated

Note. This function returns the minimum number from a given list with an O(n) time complexity.

This algorithm could have been implemented in a different way by sorting the list in ascending order, and then returning the 0th index of the list. If using the Timsort, then grabbing the 0th index of the list for the minimum value, it would have a time complexity of O(N\*log N) (GeeksforGeeks, 2021).This solution could be the best solution to use if we also needed to find the maximum value or wanted to perform a binary search on the list as well. If the list is sorted, you could also find the maximum value by returning the -1 index of the sorted list.

Binary searches require the list to be in a sorted order, and work faster than a sequential search, so if there were plans to perform searches on the data as well, then it might be worth considering going the route of sorting the data, and then just using the 0th index to obtain the minimum number. (GeeksforGeeks, 2022).

The first implementation of the algorithm had a complexity of O(n^2) and was not the most efficient solution because to find the minimum value, the algorithm was comparing each number to every other number in a way that the same numbers were unnecessarily being compared against each other multiple times. By refactoring the algorithm, I was able to cut the complexity down to O(n) by comparing the value of each index of the list to the current value of the minimum\_value variable and assigning the new minimum value if a number was smaller than the current minimum value. We could also find the minimum value of a list by first sorting the list into ascending order, then returning the 0th index of the list, but with this implementation, the algorithm would have a complexity of O(log n), due to having to sort the list first. If there were other operations that we would want to perform on the list, such as returning the maximum value or performing a binary search, then there could be justification to use this third implementation of the algorithm. Otherwise, the best option would be to use the second implementation which had the O(n) complexity.

**REFERENCES**

GeeksforGeeks. (2022, July 11). Linear search vs binary search. <https://www.geeksforgeeks.org/linear-search-vs-binary-search/>

GeeksforGeeks. (2021, June 26). TimSort. GeeksforGeeks. <https://www.geeksforgeeks.org/timsort/>

Lysecky, R., & Vahid, F. (2019). 2.5 O notation. In Design and Analysis of Algorithms. zyBook.