**CSE 212 – Programming with Data Structures**

**W02 Prove – Response Document**

|  |  |
| --- | --- |
| **Name:** | Devun Durst |
| **Date:** | 1/18/24 |
| **Teacher:** | Brother Alvey |

*It is a violation of BYU-Idaho Honor Code to post or share this document with others or to post it online. Storage into a personal and private repository (e.g. private GitHub repository, unshared Google Drive folder) is acceptable.*

**Question 1: From Part 1, what is the big O notation for the sort\_list function?**

**O(n^2)**

**Question 2: From Part 1, what is the big O notation for the standard\_deviation\_1 function?**

O(2n) >>>> O(n)

**Question 3: From Part 1, what is the big O notation for the standard\_deviation\_2 function?**

O(n^2)

**Question 4: From Part 1, what is the big O notation for the standard\_deviation\_3 function?**

O(n)

**Question 5: From Part 1, put the following big O notations in order from best performance to worst performance: O(n^2), O(1), O(2^n), O(n log n), O(log n), O(n).**

O(1) (best performance), O(log n), O(n), O(n log n), O(n^2), O(2^n) (worst performance)

**Question 6: From Part 2, what is the performance (using big O notation) for the search\_sorted\_1 function?**

**The performance seems to run as O(n) it will run n times until it hits the correct data**

**Question 7: From Part 2, what is the performance (using big O notation) for the search\_sorted\_2 function?**

O (Log N) n is the length of the data. This is true because it is a recursive function and it will continue to half and get closer to zero making it have a better performance.

**Question 8: From Part 2, which function (search\_sorted\_1 or search\_sorted\_2) has the better performance?**

**Search\_sort\_2 has the better performance.**

**Question 9: From Part 2, for both functions (search\_sorted\_1 and search\_sorted\_2), explain in detail how you determined the big O notation by just looking at the code without the benefit of observing actual execution results?**

I looked at each of the functions and kind of broke down what they were doing. The first I saw immediately was O(n) because it was just doing a line-by-line search in the data. The second was constantly halving itself making it not O(n) or O(1) but O(log n) getting closer and closer to the target over time.

**Question 10: From Part 2, it is possible in the best case for each of these functions (search\_sorted\_1 and search\_sorted\_2) to complete in O(1) time even if the size of the list was very large. What input scenarios would give this result for both functions?**

Yes, there is a best case for each function. The best case scenario for search\_sort\_1 is that the target element is the first element in the list and the Best case scenario for search\_sort\_2 is that the target element is the middle element of the list. The first check lead to the constant time complexity of O(1).