**CSE 212 – Programming with Data Structures**

**W02 Prove – Response Document**

|  |  |
| --- | --- |
| **Name:** | Jakob Hanson |
| **Date:** | 9/24/25 |
| **Teacher:** | Brother deBry |

*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 SortArray function?**

* This function has a nested for loop, thus the efficiency is **O(n^2)**

**Question 2: From Part 1, what is the big O notation for the StandardDeviation1 function?**

* This function contains 2 foreach loops, so the efficiency is O(2n) which simplifies to **O(n)**

**Question 3: From Part 1, what is the big O notation for the StandardDeviation2 function?**

* This function contains a nested foreach loop, so the efficiency is **O(n^2)**

**Question 4: From Part 1, what is the big O notation for the StandardDeviation3 function?**

* This function only makes use of a single loop of length n, thus the efficiency is **O(n)**

**Question 5: 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).**

We’re looking at these as n goes toward infinity (worst case scenario), so the order from best to worst performance is:

* O(1) - Runtime is independent of input size
* O(log n) - The runtime decreases logarithmically in relation to the input n
* O(n) - The runtime is linearly connected to the input
* O(n log n) - Runtime is a product of the previous two and grows quickly toward infinity in relation to input
* O(n^2) - Runtime squares proportionally to input
* O(2^n) - Runtime grows exponentially in relation to the function’s input

**Question 6: From Part 2, what is the performance (using big O notation) for the SearchSorted1 function?**

* This function uses a single foreach loop, and is thus directly dependent on the input, so the efficiency is **O(n)**

**Question 7: From Part 2, what is the performance (using big O notation) for the SearchSorted2 function?**

* This function uses recursion to go through each element, using the middle as the new ceiling if the middle is not the target, essentially halving the remaining values to check, meaning that it’s grows logarithmically with the input, so the efficiency is **O(log n)**

**Question 8: From Part 2, which function (SearchSorted1 or SearchSorted2) has the better performance?**

* As established in problem 5, **O(log n)** is more efficient than **O(n)** so SearchSorted2 is more efficient

**Question 9: From Part 2, for both functions (SearchSorted1 and SearchSorted2), explain in detail how you determined the big O notation by just looking at the code without the benefit of observing actual execution results?**

* The first function uses a foreach loop that iterates through each element, only exiting the loop if the target value is found. This means that in a worst case scenario, the function would iterate through every value, meaning the runtime is determined linearly by the length of the input, or in other words **O(n)**
* The second function uses a series of if statements to handle recursion, the first handles the end case (the starting index is greater than the end index), the second handles the found case, and the last handles the recursion by calling the function with an offset start. This cuts the number of possible indexes in half with each run, which means that the runtime grows logarithmically with respect to the length of the input.

**Question 10: From Part 2, it is possible in the best case for each of these functions (SearchSorted1 and SearchSorted2) 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?**

* SearchSorted1: [1,2,3,4,5,…], 1, 0, and the length of the array. This input would lead to the function finding the value on the first iteration of the loop.
* SearchSorted2: [1,2,3,4,5,…] (), value in the “middle” (exact middle, or either of the two middle values if an even length array), 0, and the length of the array. This input would lead to the function finding the value on the first call of the function without any need for recursion.