**CSE 212 – Programming with Data Structures**

**W02 Prove – Response Document**

|  |  |
| --- | --- |
| **Name:** | Jorge Alberto Chavez Ponce |
| **Date:** | 09/23/2023 |
| **Teacher:** | Ephraim Kunz |

*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?**

* The big O notation is O(n2)

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

* The big O notation is O(n)

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

* The big O notation is O(n2)

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

* The big O notation is 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) - Constant Time
* O(log n) - Logarithmic Time
* O(n) - Linear Time
* O(n log n) - Logarithmic Linear Time
* O(n^2) - Quadratic Time
* O(2^n) - Exponential Time

A graph of a function

Description automatically generated

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

* The big O notation is O(n)

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

* The big O notation is O(log(n))

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

The search\_sorted\_2 function with binary search has a better performance compared to the search\_sorted\_1 function with linear search, especially for larger lists.

**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?**

* search\_sorted\_1 (Linear Search):

The loop iterates once for each element in the data list. In the worst case, if the target is not present in the list, it will iterate through all the elements. The loop increments the count variable for each iteration, which represents the number of iterations or comparisons made.

If the target is found, the function returns early.

Since the loop's number of iterations directly depends on the length of the data list, the time is linear, denoted as O(n).

* search\_sorted\_2 (Binary Search):

This function employs a recursive binary search algorithm to find the target element in the sorted list. In each step, it reduces the size of the input list to half by choosing the middle element and comparing it with the target. The key insight here is that with each step, the algorithm eliminates one-half of the remaining elements from consideration.

In the worst-case scenario, the binary search continues until it either finds the target or determines that it's not in the list. This means that the number of steps required is proportional to the logarithm of the size of the input list (n). The time of binary search is O(log n) because it exhibits logarithmic growth, which is much more efficient than linear growth as the input size increases.

**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, it is possible for both search\_sorted\_1 and search\_sorted\_2 functions to complete in O(1) time in the best case. The best-case scenarios for each function:

* For search\_sorted\_1:

In the search\_sorted\_1 function, the best case for O(1) time occurs when the target element is found at the very beginning of the input list, the first element, In this case: The loop iterates just once.

The target element is found, and the function returns immediately. The number of iterations or comparisons in this best case is constant, regardless of the size of the input list. Therefore, the time in this scenario is O(1).

* For search\_sorted\_2:

In the search\_sorted\_2 function, the best case for O(1) time occurs when the target element is found at the middle of the input list. In this case: The function calculates the middle index and checks the middle element. The target element is found, and the function returns immediately.

Like search\_sorted\_1, the number of recursive calls and comparisons in this best case is constant, regardless of the size of the input list. Therefore, the time in this scenario is also O(1).