**qwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnm**



|  |
| --- |
| Algorithm Profiler Program Documentation  Assignment work for Data Structures and Algorithm Analysis  8/14/2023  Submitted By:  1. Finhas Yohannes, ZD 0728  2. Yanet Abrham , PM9785  3. Zelalem Amare, NV33692  Submitted to: Tariku Worku |

Contents

[1. Introduction 1](#_Toc49147896)

[2. Bubble Sort Algorithm 1](#_Toc49147897)

[2.1. Best Case 1](#_Toc49147898)

[2.2. Worst Case 1](#_Toc49147899)

[2.3. Average Case 1](#_Toc49147900)

[3. Selection Sort Algorithm 1](#_Toc49147901)

[3.1. Best Case 1](#_Toc49147902)

[3.2. Worst Case 1](#_Toc49147903)

[3.3. Average Case 1](#_Toc49147904)

[4. Insertion Sort Algorithm 1](#_Toc49147905)

[4.1. Best Case 2](#_Toc49147906)

[4.2. Worst Case 2](#_Toc49147907)

[4.3. Average Case 2](#_Toc49147908)

[5. Conclusion 2](#_Toc49147909)

# Introduction

<In this section you describe what the assignment is for or about. What sorting is, what algorithm analysis is, what complexity is, what growth rate is and etc. this section must have at least 3 pages >

# Bubble Sort Algorithm

<Bubble sort description goes here with algorithm at most two pages.>

## Best Case

<Beast case analysis goes here describing what data set makes this algorithm to perform in its best. Table for data vs. time you used to plot the graph to be shown and described. At most two pages>

## Worst Case

<Worst case analysis goes here describing what data set makes this algorithm to perform in its worst. Table for data vs. time you used to plot the graph to be shown and described. At most two pages >

## Average Case

<Average case analysis goes here describing what data set makes this algorithm to perform in average. Table for data vs. time you used to plot the graph to be shown and described. At most two pages >

# Selection Sort Algorithm

<Selection sort description goes here with algorithm at most two pages.>

## Best Case

<Beast case analysis goes here describing what data set makes this algorithm to perform in its best. Table for data vs. time you used to plot the graph to be shown and described. At most two pages>

## Worst Case

<Worst case analysis goes here describing what data set makes this algorithm to perform in its worst. Table for data vs. time you used to plot the graph to be shown and described. At most two pages >

## Average Case

<Average case analysis goes here describing what data set makes this algorithm to perform in average. Table for data vs. time you used to plot the graph to be shown and described. At most two pages >

# Insertion Sort Algorithm

Insertion sort is a straightforward sorting algorithm that operates in the same manner that you would sort playing cards in your hands. The array is effectively divided into two parts: sorted and unsorted. Values from the unsorted section are selected and inserted in the right location in the sorted component.

It is important to note that the insertion sort does not perform any type of swapping but rather it does shifting operations. The way this shifting actually works is we first check if the element that we currently wanting to place let us call the “key” for convenience is smaller than the element right behind it. If that is the case then we do the following things:

Copy the element behind it to the position of the key, so if you were to take a snapshot of the array at that specific time then you would have the same element in the array twice. We then move on to the element right before the element we compared the key to. If that element is also bigger copy that element into the next slot. We do this until we find an element smaller than the key or we reach the first index of the array; index 0, then we place the key in that position.

Let's consider an illustration with 10 unsorted figures (7, 2, 9, 1, 5, 4, 8, 3, 6, and 10). We'll use the insertion sort algorithm to sort this array. We don’t start with the first element, 7. Since it's the only element, it's formerly considered sorted.

Move to the next element, 2. Compare it with the sorted portion (7). Since 2 is smaller in size, we need to shift 7 one position to the right to make space for 2. The array becomes (2, 7, 9, 1, 5, 4, 8, 3, 6, and 10).

Move to the third element, 9. Compare it with the sorted portion (2, 7). Since 9 is larger, it remains in its position.

Move to the fourth element, 1. Compare it with the sorted portion (2, 7, and 9). Since 1 is lower, we need to shift 2, 7, and 9 one position to the right. The array becomes (1, 2, 7, 9, 5, 4, 8, 3, 6, and 10). Now we can fit 1 at the beginning of the sorted portion.

Move to the fifth element, 5. Compare it with the sorted portion (1, 2, 7, and 9). Since 5 is larger than 2 and smaller than 7, we shift 7 and 9 one position to the right. The way this shifting actually works is we first check if the element that we are currently wanting to place is smaller than the element right behind it. If that is the case then we do the following things

Move to the sixth element, 4. Compare it with the sorted portion (1, 2, 5, 7, and 9). Since 4 is lower, we need to shift 5, 7, and 9 one position to the right. The array becomes (1, 2, 4, 5, 7, 9, 8, 3, 6, and 10). Now we can fit 4 in its correct position.

Move to the seventh element, 8. Compare it with the sorted portion (1, 2, 4, 5, 7, and 9). Since 8 is lesser than 9, we shift 9 one position to the right and place 8 in its position.

Move to the eighth element, 3. Compare it with the sorted portion (1, 2, 4, 5, 7, 8, and 9). Since 3 is lower, we need to shift 4, 5, 7, 8, and 9 one position to the right. The array becomes (1, 2, 3, 4, 5, 7, 8, 9, 6, and 10). Now we can fit 3 in its correct position.

Move to the ninth element, 6. Compare it with the sorted portion (1, 2, 3, 4, 5, 7, 8, and 9). Since 6 is lesser than 7,8 and 9 we shift those numbers one position to the right and place 6 in 7s original position.

Move to the tenth element, 10. Compare it with the sorted portion (1, 2, 3, 4, 5, 6, 7, 8, and 9). Since 10 is larger than 9, it remains in its position.

**After repeating through all the elements, the array is now sorted (1, 2, 3, 4, 5, 6, 7, 8, 9, and 10).**

The shifting operation in insertion sort involves moving elements one position to the right to make space for the current element being fitted. This shifting ensures that the current element finds its correct sorted position. By constantly performing this shifting and insertion process for each element, the entire array becomes sorted

## Best Case

<Beast case analysis goes here describing what data set makes this algorithm to perform in its best. Table for data vs. time you used to plot the graph to be shown and described. At most two pages>

## Worst Case

<Worst case analysis goes here describing what data set makes this algorithm to perform in its worst. Table for data vs. time you used to plot the graph to be shown and described. At most two pages >

## Average Case

<Average case analysis goes here describing what data set makes this algorithm to perform in average. Table for data vs. time you used to plot the graph to be shown and described. At most two pages >

# Conclusion

<Conclusion goes here. You describe your findings, observations and understandings here. At least 1 page>