**TIME AND SPACE COMPLEXITY**

**‘populate\_array’** FUNCTION

**TIME COMPLEXITY**

1. The function takes input from the user in a loop until the user inputs “done”. Each input is **O(1).**
2. Lets say the user enters **n** number of elements, the loop will run  **n** times. Therefore the total time complexity will be **O(1)**.

**SPACE COMPLEXITY**

1. The space complexity is determined by the storage required for the list **‘arr’.**
2. If the user enters **n**  elements, the list  **‘arr’** will have n elements, hence the space complexity is also **O(n).**

**‘bubble\_sort’** FUNCTION

**TIME COMPLEXITY**

1. Bubble sort involves nested loops. The outer loop runs  **n** times, and the inner loop runs **n - i – 1** times, where I ranges from **0** to **i - 1.**
2. Therefore, the time complexity of bubble sort is  **O(n^2).**

**SPACE COMPLEXITY**

1. Bubble sort is an in-place sorting algorithm, which means it does not require additional space proportional to the input size.
2. The only extra space required is for a few temporary variables used in swapping elements.
3. Hence, the space complexity is **O(1).**

**COMBINED COMPLEXITY**

Combining the complexities of both functions

**TIME COMPLEXIITY**

1. The total time complexity is the sum of the time complexities of both functions
2. i.e  **populate\_array** is  **O(n)**  and **bubble\_sort** is **O(n^2).**
3. Therefore, the total time complexity is **O(n) + O(n^2) = O(n^2).**

**SPACE COMPLEXITY**

* 1. The total space complexity is the sum of the space complexities of both functions.
  2. populate\_array is **O(n)** and bubble\_sort is **O(1).**
  3. Therefore, the combined space complexity is **O(n)+O(1)=O(n).**

**SUMMARY**

* **Time Complexity:** **O(n^2)**
* **Space Complexity:** **O(n)**