Bubble Sort:

General description: Each ‘pass’ through an array, we swap adjacent elements if they are in the wrong order. One pointer is initialized to the first index, while the next pointer is kept one index ahead. As we iterate through the array, both pointers are incremented until we reach the end of the array. This process is repeated a number of times equal to the number of elements in the array. The algorithm is called ‘bubble sort’ because, during each pass, the largest unprocessed element ‘bubbles’ rightwards towards the end of the array. Because there is both an outer loop and an inner loop whose number of iterations linearly with respect to the number of elements in the array, the time complexity of bubble sort is O(n^2). This is the average and worst case time complexity. Best case time complexity is O(n), when the array is already sorted and only one pass is made.

Solution Code:

```
def bubbleSort(arr):
    n = len(arr)
    # Traverse through all array elements
    for i in range(n):
        # use a flag variable to break out of code
        # if a swap doesn’t occur during a pass
        swapped = False
        for j in range(0, n-i-1):
            # traverse the array from 0 to
            # n-i-1 and swap if an element is larger than
            # it’s neighbor
            if arr[j] > arr[j+1] :
                arr[j], arr[j+1] = arr[j+1], arr[j]
                swapped = True

        # if no elements swapped
        # by inner loop, break out of outer loop
        if swapped == False:
            break
```