Project 1 Assignment - Sorting

1. **Modified bubble sort function**

```
def modified_bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        swapped = False
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
                swapped = True
        if not swapped:
            break
    return arr
# testing in random order
import random
test_list = random.sample(range(1, 21), 20)
print("Original list:", test_list)
sorted_list = modified_bubble_sort(test_list.copy())
print("Sorted list:", sorted_list)
```

2. In essence, the number of data exchanges in both the selection sort and bubble sort analysis reflects the number of times that elements are swapped to possess this final sorted order. The number of exchanges in the worst case is O(n^2), which is proportional to the quadratic of the number of elements, in both algorithms. Where the size of the data

objects does not pay any significant role in the analysis, rather the time complexity is dominated by the order of magnitude of the number of elements being sorted.

3. The optimized bubble sort still exhibits the O(n^2) behavior on average because while the best case performance is improved to linear when the list already lies sorted, the average case still passes through a quadratic number of comparisons and swaps. The optimization only eliminates needless passes when the list already lies sorted.
4. Insertion sort in partially sorted lists works well owing to being effective at handling easily the situation where only a few elements are out of place. It performs well for the input data that is already partially sorted with a few comparisons and swaps to insert each element in its correct position. This makes it more efficient than bubble sort or selection sort algorithms in situations, which data is almost sorted.

5. Modified of the selectionSort function

```
def selectionSort(arr, reverse=False):
    n = len(arr)
    for i in range(n-1):
        min_index = i
        for j in range(i+1, n):
            if reverse:
                if arr[j] > arr[min_index]:
                    min_index = j
            else:
```

```
            if arr[j] < arr[min_index]:
                min_index = j
        arr[i], arr[min_index] = arr[min_index], arr[i]
    return arr

# Example usage:
arr = [64, 25, 12, 22, 11]
sorted_arr = selectionSort(arr, reverse=True)
print("Sorted array in descending order:", sorted_arr)
```