CS 2413 Data Structures
Dhruv Maniar

Q1)

```
QUICKSORT (ARR, BEG, END)
        if BEG < END
                var = PARTITION(ARR, BEG , END)
                QUICKSORT (ARR, BEG, var-1)
                QUICKSORT (ARR, var+1, END)
PARTITION (ARR, BEG, END)
        varvalue = ARR [END]
        i = BEG - 1
        for j= BEG to END -1
                if ARR[j] >= varvalue
                        i=i+1
                        exchange ARR[i] with ARR[j]
        exchange ARR[i+ 1] with ARR[END]
        Return i+1
```

Q2)

Insertion Sort:

Insertion sort is a simple algorithm that sorts an array by iterating over the elements and inserting each element into its proper position in the sorted subarray to its left. The algorithm maintains a sorted subarray and repeatedly inserts the next unsorted element into the subarray, shifting the other elements to make room for the new element.

The time complexity of insertion sort is O(n^2) in the worst case, where n is the number of elements in the array. However, in practice, the algorithm is much faster than the worst-case bound suggests. The average-case time complexity of insertion sort is O(n^2), but the constant factor hidden in the O(n^2) term is much smaller than that of selection sort.

Selection Sort:

Selection sort is another simple algorithm that sorts an array by repeatedly finding the minimum element in the unsorted subarray and swapping it with the first element in the unsorted subarray. The algorithm maintains two subarrays, one sorted and one unsorted, and repeatedly selects the smallest element from the unsorted subarray and swaps it with the first element in the unsorted subarray.

The time complexity of selection sort is also O(n^2) in the worst case and the average case. The algorithm must compare every element in the unsorted subarray with every other element to find the minimum, leading to a large constant factor in the time complexity.

There are several reasons why insertion sort is faster than selection sort on average:

1. Fewer Comparisons: Insertion sort performs fewer comparisons than selection sort. In the best case, where the input array is already sorted, insertion sort only performs n - 1 comparisons, while selection sort still performs n^2/2 comparisons. In the worst case, both algorithms perform the same number of comparisons, but insertion sort is still faster because of the next reason.

2. Data Movement: Insertion sort performs less data movement than selection sort. Insertion sort moves elements only once, while selection sort moves elements multiple times. In insertion sort, an element is moved to its final position in the sorted subarray only once, while in selection sort, an element can be moved multiple times until it reaches its final position.

3. Cache Efficiency: Insertion sort has better cache efficiency than selection sort. Insertion sort accesses elements in a contiguous manner, which is cache-friendly, while selection sort accesses elements in a scattered manner, which is cache-unfriendly. This can lead to a significant performance improvement, especially for large arrays.

While the number of comparisons is a factor, it is not the only reason why insertion sort is faster than selection sort on average. In addition to making fewer comparisons when the input array is partially sorted, insertion sort also performs less data movement and has better cache efficiency than selection sort. This is because insertion sort moves each element only once to its final position in the sorted subarray, while selection sort may move an element multiple times before it reaches its final position. Insertion sort also accesses elements in a contiguous manner, making it more cache-friendly than selection sort, which accesses elements in a scattered manner. These factors contribute to a lower constant factor in the time complexity of insertion sort, resulting in faster average performance compared to selection sort.

Overall, insertion sort is faster than selection sort on average because it performs fewer comparisons, less data movement, and has better cache efficiency. The constant factor hidden in the O(n^2) term is much smaller in insertion sort than in selection sort, leading to faster average performance.

Q3)
3.1)
To sort n/k subarrays of length k using insertion sort, the time complexity (in Θ notation) is Θ(nk). This is because there are n/k subarrays of length k that need to be sorted, and the worst-case time complexity of insertion sort for k elements is Θ(k^2). Thus, the total number of elements to be sorted is nk, resulting in a time complexity of Θ(nk).

3.2)
The time complexity in Θ notation to merge all subarrays in the "divide-conquer-merge" framework of merge sort is Θ(nlogn). This is due to the fact that there are logn levels of recursion, and at each level, n elements are required to be merged. Since each merge operation takes Θ(n) time, the overall time complexity becomes Θ(nlogn).