CSC 225 Assignment 2

1. Insertion Sort code:

```
void sort(int arr[]) {
    int n = arr.length;
    for (int i=1; i<n; i++) {
        int key = arr[i];
        int j = i-1;
        while (j>=0 && arr[j]>key) {
            arr[j+1] = arr[j];
            j = j-1;
        }
        arr[j+1] = key;
    }
}
```

From the insertion sort code we can see that every time while loop runs, reduce one inversion. The running time of insertion sort is $O(n+f(n))$, n is outter loop running time, $f(n)$ is how many inversion need to reduce. There are k inversions, so the running time of insertion sort when it's used to sort A is $O(n+k)$.

2. ① mid = (0 + last index of array A)/2
   ② Divide the array A into array B and array C, the elements of array B is from the elements of index 0 to mid of array A, the array C is from the elements of index mid+1 to last index of Array A.
   ③ Use array B and array C to call mergeSort, and count the inversions of the array B and array C.
   ④ Merge the array B and array C to a new array, during the merge, let l be the index of array B, let r be the index of array C. if $b[l] > c[r]$, then there are mid - l inversions.
   ⑤ So the inversions is the sum of inversions of array B and array C and during the merge step.

3. To compute the Kendall tau distance we use question 2 to count the inversions of each array, and keep track of what pair of indexes that are not in order.

4. Using Radix Sort

Take every digit i from the least significant digit to the the most significant digit and do bubble sort until the ith digit.

Radix Sort takes $O(d(n+b))$ time, d is d digits in input integers. b is the base for representing numbers.

$\because$ there are n integers and n^2-1 is the maximum

$\therefore d = O(\log_b n)$

$\because b = n$

$\therefore d = O(1)$

$\therefore$ Overall the running time is $O(n)$

5. for an n-element heap has height $\lfloor \log n \rfloor$. for insert method, the worst case we have to do $\lfloor \log n \rfloor$ comparisons for the element at the bottom of the heap. So it's impossible for insert take $O(\log(\log n))$ time, since $\lfloor \log n \rfloor > \log \log n$

(a)

(b) For the removeMin method, it's similal with insert method. After we use the element at the end of heap to replace the root (the min. element). If the worst case we do $\lfloor \log n \rfloor$ comparisons for the element bubbledown, then it's impossible removeMin method take $O(\log(\log n))$ time, since $\lfloor \log n \rfloor > \log \log n$