Adebayo Ogunmuyiwa U00837486 CSC225.

(1) Insertion sort takes time of $O(n^2)$ because it works by iterating over the array and growing the sorted array behind it.

So if we are inserting 1 into [2,3,9,5], the whole array would have to move one position to the right to accomodate this change.

Suppose an array contains elements $\{1, 2 \dots n\}$ the most inversions possible would be $\{n, n-1, n-2, n \dots 2, 1\}$, for all $1 \le i < j \le n$ then there is an inversion (i, j). The running time of such insertion sort is $n(n-1)/2$.

(2) We can use a modified Merge sort algorithm, where A is the array, and $p \le q < r$ such that p, q, and r are indices into the array.

```
Algorithm Merge-Inv(A, p, q, r)
n1 = q - p + 1
n2 = r - q
new L[n1+1], R[n2+1]
for i = 1 to n1
    L[i] = A[p + i - 1]
End for
for j = 1 to n2
    R[j] = A[q + j]
End for
i = 1
j = 1
Inversion = 0
count = false
```

2)

```
for k=p to r
  IF count==false && R[j] < L[i]
     inversion = inversion + n1 - i + 1
     count = True
  End IF
  IF L[i] ≤ R[j]
     A[k] = L[i]
     i = i + 1
  else A[k], R[j]
     j = j + 1
     counted = false
  end else
or                                  return inversion
```

```
Count_inversion (A, P, r)
  Inversion = 0
  IF p < r
     q = [(p+r)/2]
     inversion = inversion + Count_Inversion (A, p, q)
     Inversion = Inversion + count-inversion (A, q+1, r)
     Inversion = Inversion + Merge_Inversion (A, p, q, r)
```

~~first~~ we used ~~Merg~~ Modified Mergesort ~~to~~

(4) we ~~first~~ perform a radix sort on each of the n elements, Viewing ~~them~~ as pairs (i, j) ~~such that~~ i and j are integers in the range $[0, n-1]$.

Each number from $[0, n^2-1]$ can be represented by a two digit number in the number system with base n.

$(n-1) \cdot n + (n-1) = n^2 - 1$. then use radix sort in $O(2 \cdot n)$.

(5) (3) For insert and delete minimum to take only $O(\log(\log n))$ time it would require sorting each element of an array A into a minimum priority queue and then repeatedly remove the minimum the minimum element until all the element of the array A are placed in order. It would be impossible to do so because the quickest sorting known runs in $O(n \log n)$ time.