Prabhav Panicker
D-28/2016906 Tutorial - 3

Sol 1

```
int LinearSearch(int *arr, int n, int key)
    for i >= 0 to n-1 {
        if arr[i] = key
            return i
    } return -1.
```

Sol 2

Iterative Insertion Sort

```
void insertion-sort(int arr[], int n)
    int j, temp, g i
    for j < 1 to n.
        temp ← arr[i]
        j ← i-1.
    while(g >= 0 AND arr(j) > temp)
        arr(j+1] ← arr(j)
    arr(j+1] ← temp
```

Recursive Insertion Sort

```
void insertion-sort(int arr[] int n)
    if(n <= 1) return.
    Insertion sort(arr, n-1)
        last = arr[n-1]
                g=n-2
while(j >= 0 && arr[j] > last)
    arr[j+1] = arr[j].
        j--
    arr[j+1] = last.
```

Q 3 Selection Sort:

Time Complexity: Best Case: $O(n^2)$, worst case = $O(n^2)$

Space Complexity: $O(1)$

Insertion Sort

TC: best Case : $O(n)$, worst cast = $O(n^2)$

SC: $O(1)$

Merge Sort:

TC => Best => $O(n \log n)$

Worst => $O(n \log n)$

SC => $O(n)$

Quick Sort:

TC => Best $O(n \log n)$

Worst $O(n^2)$

SC => $O(n)$

Prakhar Pande
D-28, 2016906

Sol^n 4

Sorting	inplace	Stable
Selection	✓	
Insertion	✓	✓
Merge	✓	
Quick	✓	
Heap	✓	
Bubble	✓	✓

Sol^n 5 Quick Sort is the fastest general purpose sort, In most practical situation quick sort is the method of choice If stability is important and space is bubble sort, merge sort, might be best.

Sol 6 The worst TC of quicksort is $O(n^2)$. The worst case occur when the picked pivot is always an extreme (smallest or largest) element. This happens when input array is sorted or reverse sorted array and either 1st or last element is picked as pivot

Sol Recurrence relation of:

a) Merge Sort => $T(n) = 2T(n/2) + n$

b) Quick Sort => $T(n) = 2T(n/2) + n$

Merge Sort is more efficient & works faster than quick sort in case of large array size or data sets.