Insertion Sort

1. Example:

```
# Python program for implementation of Insertion Sort

# Function to do insertion sort
def insertionSort(arr):

    # Traverse through 1 to len(arr)
    for i in range(1, len(arr)):

        key = arr[i]

        # Move elements of arr[0..i-1], that are
        # greater than key, to one position ahead
        # of their current position
        j = i-1
        while j >= 0 and key < arr[j] :
                arr[j + 1] = arr[j]
                j -= 1
        arr[j + 1] = key



# Driver code to test above
arr = [12, 11, 13, 5, 6, 1]
insertionSort(arr)
for i in range(len(arr)):
    print ("% d" % arr[i])
```

2. Time complexity For the Example

1. Assign the key
2. Compare 2nd element (key) to the element before it
3. Insert element in key's place
4. (Repeat until key is bigger than the element compared)
5. Insert key in the smaller element's place
6. Repeat n times iterating one more element each time

For a 6 element array [plus key assignment] [when key is smaller: + placing the key]:
[12, 11, 13, 5, 6, 1]: 1 step comparison + 1 step swapping
[11, 12, 13, 5, 6, 1]: 0 steps
[11, 12, 13, 5, 6, 1]: 3 steps comparison + 3 step swapping
[5, 11, 12, 13, 6, 1]: 4 steps comparison + 3 step swapping
[5, 6, 11, 12, 13, 1]: 5 steps comparison + 5 step swapping

25 steps (swapping/comparison)
5 key assignments
4 key placements

Total = 33 steps

3. Generalization:

Last iteration: n-1
Second to last: n-2
...
First iteration: 1

So:

$(1+2+3+\cdots+(n-1)) = (n-1+1)((n-1)/2) = (n^2/2)-n/2$

In asymptotic notation:

O(n^2)

4. Edge Cases

Sorted: it iterates through the array one time, so linear time complexity O(n). Very good.

Sorted backwards: Very bad, this is the worst case scenario as it will have to reorder every single item.

All items in the array are the same: time complexity O(n). Very good.

Almost sorted: Not bad. Number of steps would be:

n +(the slot number occupied by each unsorted element x 2)

(Times two because we need to compare and also to place element)