Gauss Trick - How can you compute the sum of all numbers from 1 to 100?

Write down the numbers first in ascending order and then in descending order as follows:

Ascending :	1	2	3	4	.....	99	100
Descending:	100	99	98	97		2	1
Sum:	101	101	101	101		101	101

The sum of all numbers in a given column will be equal to 101 (the sum of the first and the last number in our sequence). We have 100 sums (as many as the numbers in our sequence) of 101 for twice the sequence. Therefore, the sum of the numbers in [1, 100] is: 100 * 101 / 2.

In general, we can always compute the sum of all **n** elements in a given arithmetic progression as **n times the arithmetic mean of the first and last item in that progression.**

Insertion Sort – Worst Case Analysis

1. What is the worst case?

We've established in class that the worst case for insertion sort, the one which will cause insert(L, i) to perform the most number of steps is when L[i] < min(L[:i]). In other words, when the element we are trying to insert in the sorted part of our list L[:i], is smaller than all the elements in L[:i].

2. In the worst case on pass i of insertion sort, how many assignment statements are executed?

A single pass i of insertion sort corresponds to a single call to insert(L, i). Please note that a single pass of insertion sort can involve multiple iterations of a while-loop. The term “pass” is not equivalent to the term “iteration” here.

There are two statements within the while loop in the body of insert(L, i) function:

- L[i] = L[i – 1]
- i = i – 1

The number of times these two statements execute depends on the number of iterations of this while-loop. In the worst case scenario, the item to insert belongs at the front of the list. We can express this formally as L[i] < min(L[:i]). In that case, the second loop condition (L[i – 1] > value) will never be False. It will be the first condition (i > 0) that will eventually become False and cause the loop to terminate. Therefore, in this worst case, there will be **i** iterations of this while loop and as a result these two assignments will be executed 2 * i times.

The total number of assignment statements is: 2 * i + 2

The + 2 accounts for the two assignment statements outside the while loop:

- value = L[i]
- L[i] = value

3. For the call insertion_sort(L), in the worst case, how many comparisons are made during all the calls to insert? We assume that the list L contains n items.

For a given call to insert(L, i) there are 2 * i + 1 comparisons. We do two comparisons per iteration, plus one extra (i > 0) to realize that this while loop condition now evaluates to False. This is also the case if i is equal to 0, in which case we only do a single comparison due to Python's lazy evaluation.

Here's a breakdown of comparisons for each value of i:

- i = 0 => 2 * 0 + 1 comparisons
- i = 1 => 2 * 1 + 1 comparisons
- i = 2 => 2 * 2 + 1 comparisons
- ...
- i = n – 1 => 2 * (n – 1) + 1

Let us sum all these up:
Sum = 2 * (0 + 1 + 2 + ... + (n – 1)) + n * 1 = 2 * gauss_sum + n = n * n

We used the trick from Gauss to compute the sum of that arithmetic progression:
gauss_sum = 0 + 1 + 2 + ... + (n – 1) = n * 1/2 * (0 + (n-1)) = n * (n – 1) / 2

Having n to the power of two comparisons dominates complexity, and so insertion sort has **quadratic** running time.

Insertion Sort – Best Case Analysis

---

1. What is the best case?

We've established in class that the best case for insertion sort, the one which will cause insert(L, i) to perform the fewest number of steps is when L[i] >= max(L[:i]). Since the sublist L[:i] is sorted, the requirement becomes L[i] >= L[i-1].

In other words, the best case scenario arises when the element we are trying to insert in the sorted part of our list, L[:i], is greater than or equal to the last element in L[:i].

2. In the best case on pass i of insertion sort, how many assignment statements are executed?

In the best case scenario, there will be zero iterations of the while loop, as the second condition (L[i – 1] > value) will evaluate to False the very first time the loop condition is checked.

Only the two assignment statements outside the loop will execute.

3. For the call insertion_sort(L), in the best case, how many comparisons are made during all the calls to insert? We assume that the list L contains n items.

For a given call to insert(L, i) there will be 2 comparisons. The only exception would be when i is equal to 0 in which case, a single comparison suffices due to Python's lazy evaluation.

Here's a breakdown of comparisons for each value of i:

- i = 0 => 1 comparisons
- i = 1 => 2 comparisons
- i = 2 => 2 comparisons
- ...
- i = n – 1 => 2 comparisons

Let us sum all these up:
Sum = 2 * (n – 1) + 1 = 2 * n – 1

For the best case, insertion sort has **linear running time**.