We will model the estimated runtime time complexity of Insertion Sort both experimentally and analytically.

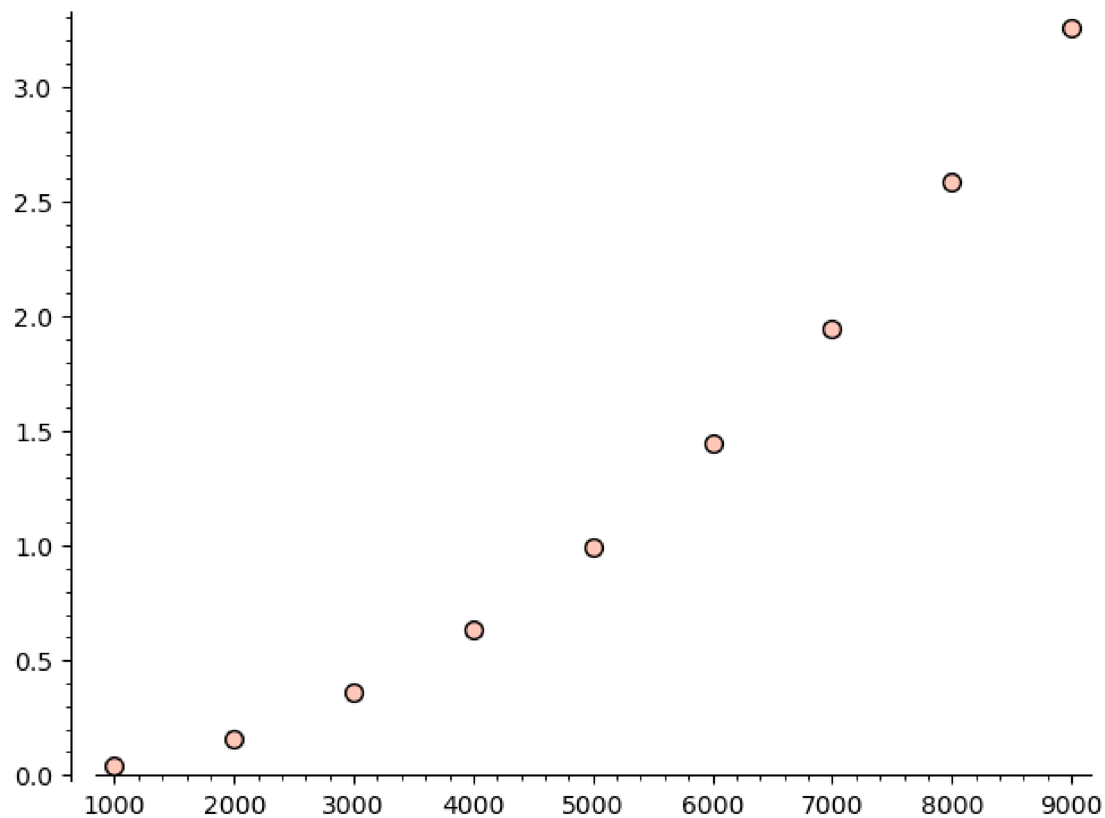
1 | **Experimental Analysis**

On a Rust script running in Debug mode (to better analyze runtime without compiler optimizations), we gather the following data regarding the runtime of insertion sort via worse-case running time (list reversed).

N	Runtime (s)
1000	0.041
2000	0.161
3000	0.361
4000	0.634
5000	0.991
6000	1.442
7000	1.943
8000	2.581
9000	3.258

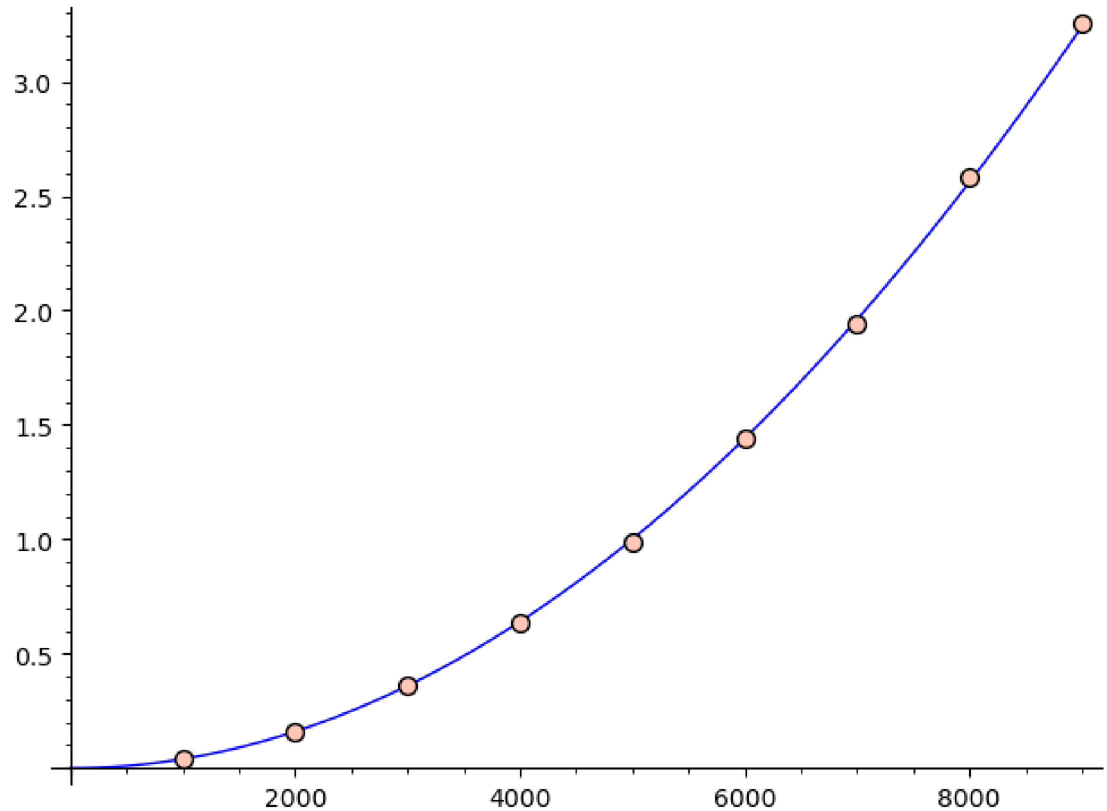
We will now plot this upon a graph to analyze the relation between N and s:

```
data = [[1000, 0.041], [2000, 0.161], [3000, 0.361], [4000, 0.634], [5000, 0.991], [6000, 1.442], [7000
scatter_plot(data)
```




```
x = var("x")
```

```
data = [[1000, 0.041], [2000, 0.161], [3000, 0.361], [4000, 0.634], [5000, 0.991], [6000, 1.442], [7000
scatter_plot(data)+plot(4*(x^2)/100000000, (x, 0, 9000))
```



After dividing by a constant scalar of $\frac{4}{100000000}$ to make the constant scaling proper, we can see that the time growth w.r.t. number of inputs is almost exactly quadratic.

2 | Theory

In the worse case scenario, we will need to swap every element backwards. Therefore, at every element i, there would need to be $N-i$ swaps to swap the element into the right place. Therefore, the total runtime would be:

$$\sum_{i=0}^{N-1} = N - i \tag{1}$$

This would be equivalent to:

$$N + (N-1) + (N-2) + \cdots + N - (N-1) \tag{2}$$

swaps. Adding this list from the two ends, each pair contains $N+1$ operations, and there are $\frac{N}{2}$ such pairs. Therefore, there is:

$$(N+1)\frac{N}{2} = \theta(N^2) \tag{3}$$

swaps.

Therefore, it is theoretically true as well that the sorting time grows quadratically.

```
s,h,l = var("s h l")

eqns = [
    (400/3)*(s/h)^(1/3) == l*20,
    (200/3)*(h^2/s^2)^(1/3) == l*170,
    170*s + 20*h - 20000 == 0
]

simplify(expand(solve(eqns, (s,h,l))))
```