We have 4 different initial array ordering situations: random, ordered, partially ordered and reversed ordered. For each type of the array, we choose 5 different lengths: 500, 1000, 2000, 4000, 8000. For each value of length, we run over 1000 times and calculate the mean running time of each experiment. And the result are shown below:

Text

Description automatically generated

\*mean time of 1000 time running

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Length=500 | Length=1000 | Length=2000 | Length=4000 | Length=8000 |
| Order | 0.031179364 | 0.034248863 | 0.019666144 | 0.045168418 | 0.083676415 |
| Random | 0.439551143 | 0.976564319 | 2.600452572 | 10.418625463 | 42.92053012 |
| Reverse | 0.823715845 | 1.308957149 | 4.975466625 | 19.703718458 | 79.164434756 |
| Partially Order | 0.121010107 | 0.173215365 | 0.651028631 | 2.561874651 | 10.403374985 |

From this table, we could clear see the result that for each array with different length, we run over 1000 times and calculate its mean time. Based on the result, we could come to some conclusions:

1. for different initial array ordering situations, sorting it by using insertion sort could run different time, for the same length, the running time:

order < partially order < random < reversed

1. for the same length, the running time of reversed array is almost equal to two times of the running time of random array
2. for partially ordered array, since the first half of the array is an already sorted array and the second half of the array is a random array. So, for example, for length=n partially ordered array, its running time close to the running time of sorting length=n/2 ordered array plus the running time of sorting length=n/2 random array. The relationship is very obvious if the length of the array n is getting bigger.

Benchmark.Test

Graphical user interface, text

Description automatically generated

TimerTest:

Text

Description automatically generated

InsertionSortTest:

Text

Description automatically generated