date: 2025-10-29

1 complexity of algorithms

we mentioned that the growth functions like big-o, big-omega, and big-theta are closely related to computer science – each of them representing a upper bound, lower bound, and tight bound. in this section we will apply these growth functions to algorithms.

1.1 motivation

there are a lot of algorithms that exist to solve the same problem. usually, we want to compare the complexity between them, and figure out which one is the most efficient. take these three algorithms for example; they all solve the problem of summing the integers from 1 through n:

algorithm A

```
1 sum := 0
2 for i:= 1 to n
3 | sum := sum + 1
4 return sum
```

algorithm B

```
1 sum := 0
2 for i := 1 to n
3 | for j := 1 to i
4 | | sum := sum + 1
5 return sum
```

algorithm C

```
1 sum := n * (n + 1) / 2
2 return sum
```

1.2 analysis

to find the complexity for different algorithms, we consider the amount of operations they do. let's compare them:

	algorithm A	algorithm B	algorithm C
additions	n	$\frac{n(n+1)}{2}$	1
multiplications			1
divisions			1
total ops.	n	$\frac{n^2}{2} + \frac{n}{2}$	3

as the input gets larger, some operations might take longer than others. let's test each algorithm with for different amount of inputs:

	algorithm A	algorithm B	algorithm C
n = 1	1	1	3

	algorithm A	algorithm B	algorithm C
n = 10	10	55	3
n = 100	100	5050	3
n = 1000	1000	500500	3

we can clearly see that algorithm C performs the least amount of operations as our input grows. algorithm A and B both grows, however, at different rate. when we analyze an algorithm, we focus on the trends as the problem instances grow in size.