Revision on Algorithms and their running time

1 Algorithms

Algorithm: Computational procedure that takes some value(s) as input, and produces some value(s) as output
Problem: Specifies desired relationship between input and output

An algorithm is **correct** if for every input instance it halts with the correct output
A correct algorithm **solves** the problem. In this course we only consider correct algorithms

2 Asymptotic Notation

For two given functions $f(n)$ and $g(n) \geqslant 0$

- $f(n) = \Omega(g(n))$ if $\exists c_1 > 0, n_0$ s.t
$$c_1 g(n) \leq f(n) \quad \forall n \geq n_0$$
- $f(n) = O(g(n))$ if $\exists c_2 > 0, n_0$ s.t.
$$f(n) \leq c_2 g(n) \quad \forall n \geq n_0$$
- $f(n) = \Theta(g(n))$ if $\exists c_1, c_2 > 0, n_0$ s.t.
$$c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \forall n \geq n_0$$
- $f(n) = \omega(g(n))$ if $\forall c_1 > 0, \exists n_0$ s.t.
$$c_1 g(n) \leq f(n) \quad \forall n \geq n_0$$
- $f(n) = o(g(n))$ if $\forall c_2 > 0, \exists n_0$ s.t.
$$f(n) \leq c_2 g(n) \quad \forall n \geq n_0$$

2.1 Relationships

For any $f(n), g(n)$

1. $f(n) = \Theta(g(n))$ iff $g(n) = \Theta(f(n))$
2. $f(n) = \Theta(g(n))$ iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$
3. $f(n) = O(g(n))$ iff $g(n) = \Omega(f(n))$
4. $f(n) = o(g(n))$ iff $g(n) = \omega(f(n))$
5. $f(n) = O(g(n))$ iff either $f(n) = \Theta(g(n))$ or $f(n) = o(g(n))$
6. $f(n) = \Omega(g(n))$ iff either $f(n) = \Theta(g(n))$ or $f(n) = \omega(g(n))$

3 Running time

The running time $T(I)$ of an algorithm α on input I is the time taken by α on I until α halts. We classify input according to their size.

Size is ideally the number of bits to represent input to a real computer program, but then encoding can be done in different ways, for example encoding a graph. Thus we let the notion of size depend on the context

3.1 Worst-case running time

We define the worst case running time $T(n)$ of algorithm α on input of size n as

$$T(n) := \max\{T(I) : I \text{ has size } n\}$$

By convention, if there are no inputs of size n, we define $T(n) := 0$

So for any input I of size n

$$T(I) \leqslant T(n)$$

Why we are interested in worst-case running time:

- It is an upper bound of the running time for any input of a given size
- For some algorithms, the worst case occurs fairly often
- The average case is often roughly as bad as the worst case

3.2 Calculating the running time

Let I be input for algorithm α. How to we compute $T(I)$?
Let the pseudocode of algorithm α consist of p lines.
Assume that the execution of line i requires time 1
Let q_i be the number of times line i is executed on input I
The running time of α on input I is

$$T(I) := \sum_{i=1}^{p} q_i$$

3.3 Example: Insertion sort

Let A be an array of n elements. For $j = 2, 3, ..., n$, we let t_j be the number of times the while loop test in line 4 is executes for that value of j

i	line i	times q_i
1:	**for** $j \leftarrow 2$ to n **do**	n
2:	$key \leftarrow A[j]$	$n-1$
3:	$i \leftarrow j-1$	$n-1$
4:	**while** $i > 0$ and $A[i] > key$ **do**	$\sum_{j=2}^{n} t_j$
5:	$A[i+1] \leftarrow A[i]$	$\sum_{j=2}^{n}(t_j - 1)$
6:	$i \leftarrow i-1$	$\sum_{j=2}^{n}(t_j - 1)$
7:	$A[i+1] \leftarrow key$	$n-1$

Then

$$T(A) = \sum_{i=1}^{7} q_i = 2n - 1 + 3\sum_{j=2}^{n} t_j$$

3.4 Worst case

The worst case maximises $\sum_{j=2}^{n} t_j$
We have $t_j \leqslant j$ for all j; can we reach this bound?

Worst case: S is in decreasing order and has distinct elements

Then for all $2 \leqslant j \leqslant n$ we have $A[i] > A[j]$ for all $1 \leqslant i \leqslant j-1$

This means that $t_j = j$ for all $2 \leqslant j \leqslant n$. Hence

$$T(n) = \frac{3}{2}n^2 + \frac{7}{2}n - 4 = \Theta\left(n^2\right)$$

4 Discussion

Reasons to use the asymptotic notation:

1. Usually difficult to give precise expressions for running times. The asymptotic notation gives (tight) upper bounds in terms of natural functions
2. From a practical point of view, we remove the dependency to specific implementation
3. We are especially interested in the performance of algorithms for very large inputs