Complexity

Carl Jendle, cid jendle, Carl Hjalmarsson, cid carlhj

November 2018

1 Part 1

DISCLAIMER - partially written in English and partially Swedish. Sorry for potential inconvenience.
Problem statements - $T_1 = 100$ seconds. How much longer does it take for input size a) 101, b) 200, c)10000? What is maximum size input for 10^5 seconds assuming input size 1 takes 1 second?

1.1 T(n)=$log_2 n$

$log_2 101 = 6.66 \rightarrow 6$ seconds $x = \frac{log_2 101}{log_2 100} = 1.002 \rightarrow 0.2\%$ faster
$log_2 200 = 7.64 \rightarrow 7$ seconds $x = \frac{log_2 200}{log_2 100} = 1.15 \rightarrow 15\%$ faster
$log_2 100000 = 16.6 \rightarrow 16$ seconds $x = \frac{log_2 100000}{log_2 100} = 2.49 \rightarrow 249\%$ faster
$100000 = log_2 n \rightarrow n = 2^{100000} = \infty$

1.2 T(n)=n

Linear growth → a - 1%, b - 200%, $c - 10000\%$ of original time.
100000 is the maximum size due to linear growth.

1.3 T(n)=$nlog_2 n$

T(100) = 100 log2 100 664.

T(101) = 101 log2 101 672 1.012 T(100).

T(200) = 200 log2 200 1529 2.3 T(100).

T(10000) = 10000 log2 10000 = 10000 2 log2 100 = 200 T(100).

So: a) about 1.2

For d), we need to solve 100000 = n log2 n. We cannot solve this symbolically [1] but the solution is n 7740.96. Since n must be a whole number, the answer is 7740.

1.4 T(n)=n^2

$\frac{101^2}{100^2} = 1.02 = 102\%$ of original time
$\frac{200^2}{100^2} = 4 = 400\%$ of original time
$\frac{10000^2}{100^2} = 10000 = 1000000\%$ of original time
Maximum size input is $\sqrt{100000} = 316.23 \rightarrow 316$

1.5 T(n)=n^3

$\frac{101^3}{100^3} = 1.03 = 103\%$ of original time
$\frac{200^3}{100^3} = 8 = 800\%$ of original time
$\frac{10000^3}{100^3} = 1000000 = 100000000\%$ of original time
Maximum size input is $\sqrt[3]{100000} = 46.42 \rightarrow 46$

1.6 T(n)=2^n

$\frac{2^{101}}{2^{100}} = 2 = 200\%$ of original time
$\frac{2^{200}}{2^{100}} = 1.6e60 = 400\%$ of original time
$\frac{2^{10000}}{2^{100}} = \infty = \infty\%$ of original time
Maximum size input is $log_2 100000 = 16.6 \rightarrow 16$

2 Part 2

2.1 1

i loopar från 0 till n och summerar i varje steg → n operationer T(n) → O(n).

2.2 2

Yttre loopen kör n gånger, för varje värde på i kör inre loopen $n - 1$ gånger, och börjar på j=1:

$$T(n) = n(n-1) = n^2 - n \rightarrow O(n^2) \tag{1}$$

2.3 3

i loopar först en gång från 0 till n och sedan 1 till n → T(n)=2n-1, O(n)

2.4 4

$$T(n) = n^2 \cdot \log_2(n) + 1 \rightarrow O(n^2 \log_2(n)) \tag{2}$$

2.5 5

i loopar 0 till n i yttre loopen och j loopar från 0 till i i inre loopen, sedan adderas fristående for-loopen där k loopar från 0 till n $\rightarrow T(n) = \frac{n(n+1)}{2} + n \rightarrow O(n^2)$

2.6 6

$$T(n) = 1{+}T(n{-}1) = 1{+}1{+}T(n{-}2) = ... = k{+}T(n{-}k) = n{-}T(0) \rightarrow O(n) \quad (3)$$

2.7 7

i loopar från 0 till n varje runda och sedan läggs en rekursiv operation till vilken minskar n med 1 tills n=0 $\rightarrow T(n) = \sum_{k=0}^{n}(k+1) = \frac{(n+1)(n+2)}{2} \rightarrow O(n^2)$ eftersom summering uppifrån och ned är detsamma som summering nedifrån och upp.

2.8 8

$$T(n) = 1{+}T(\frac{n}{2}) = 1{+}1{+}T(\frac{n}{4}) = ... = k{+}T(\frac{n}{n^k}) = \log_2(n){-}T(0) \rightarrow O(\log_2(n)) \quad (4)$$

2.9 9

i loopar i varje steg från 0 till n och sedan görs en rekursiv operation som halverar n, vilket sätter en övre gräns på summan till $log_2 n$ då loopen bryts för $n \leq 1$. $T(n) = \sum_{k=0}^{log_2 n}(\frac{n}{2^k} + 1) = 2n + \frac{log(n)}{log(2)} \rightarrow O(n)$

2.10 10

$$T(n) = T(n{-}1){+}T(n{-}1){+}1 = 2T(n{-}1){+}1 = 2^kT(n{-}k){+}k = 2^nT(0){+}n \rightarrow O(2^n) \quad (5)$$